Software Testing, Quality Assurance & Maintenance—Lecture 12

Patrick Lam
University of Waterloo

January 30, 2015

Last Time

- iComment/aComment
- FindBugs
- Java Path Finder
- Korat
- Randoop

Daikon: Dynamic Invariant Detection ([other] UW)

Goal: recover invariants from programs.

Technique: run program, examine the values it computes

Results:

- formal specs
- bugs

Can use Daikon on Java, C, C++, Lisp.

`plse.cs.washington.edu/daikon/`

ESC/Java (Compaq)

Statically checks Java programs against specifications written in JML (Java Modelling Language).

Will see more examples later. Here's one:

```
//@ public invariant balance >= 0
    && balance <= MAX_BALANCE;
```

www.hpl.hp.com/downloads/crl/jtk/index.html

Tools you can Download

We'll survey some tools for:

- Java
- C/C++

(or, use e.g. a programming language with stronger types!)

cppcheck

Open-source tool that statically checks for:

- out-of-bounds errors;
- memory leaks;
- division by zero;
- null pointer dereferences;
- calls to obsolete functions;
- uses of uninitalized variables
- etc.

`sourceforge.net/projects/cppcheck`

Valgrind

memcheck is a Valgrind tool to
detect memory errors at runtime.
Helps make your programs more correct, by finding:

- illegal reads & writes
- uses of uninitialized values
- double frees
- copies with overlapping sources and destinations
- memory leaks

helgrind is a Valgrind tool to detect thread errors.

`valgrind.org`

Flawfinder

grep++: identifies non-comment calls to bad functions:

- buffer overflow risks:
  `strcpy()`, `strcat()`, `gets()`, `sprintf()`, `scanf()`
- format string problems:
  `[v][f]printf()`, `[v]snprintf()`, `syslog()`
- file system race conditions:
  `access()`, `chown()`, `tmpnam()`, etc.

`www.dwheeler.com/flawfinder`

Clang Static Analyzer (U Illinois)

(Extensible) C/C++ compiler front-end with static analyzer.

The compiler: `clang-analyzer.llvm.org`

`clang-analyzer.llvm.org/available_checks.html`:

- label function arguments as “nonnull”, check for violations;
- the usual: memory leaks, division by 0, null pointer dereferences.

Sparse (Linux)

A “semantic parser” which finds faults in Linux kernel.

- mixing userspace and kernelspace pointers
- the usual: null pointer dereferences etc

`https://sparse.wiki.kernel.org/index.php/Main_Page`
`linux.die.net/man/1/sparse`

Splint (U Virginia)

```
/*@falsewhennull@*/ bool isNonEmpty
                              (/*@null@*/ char *x) {
  return (x != NULL && *x != '\0');
}
```

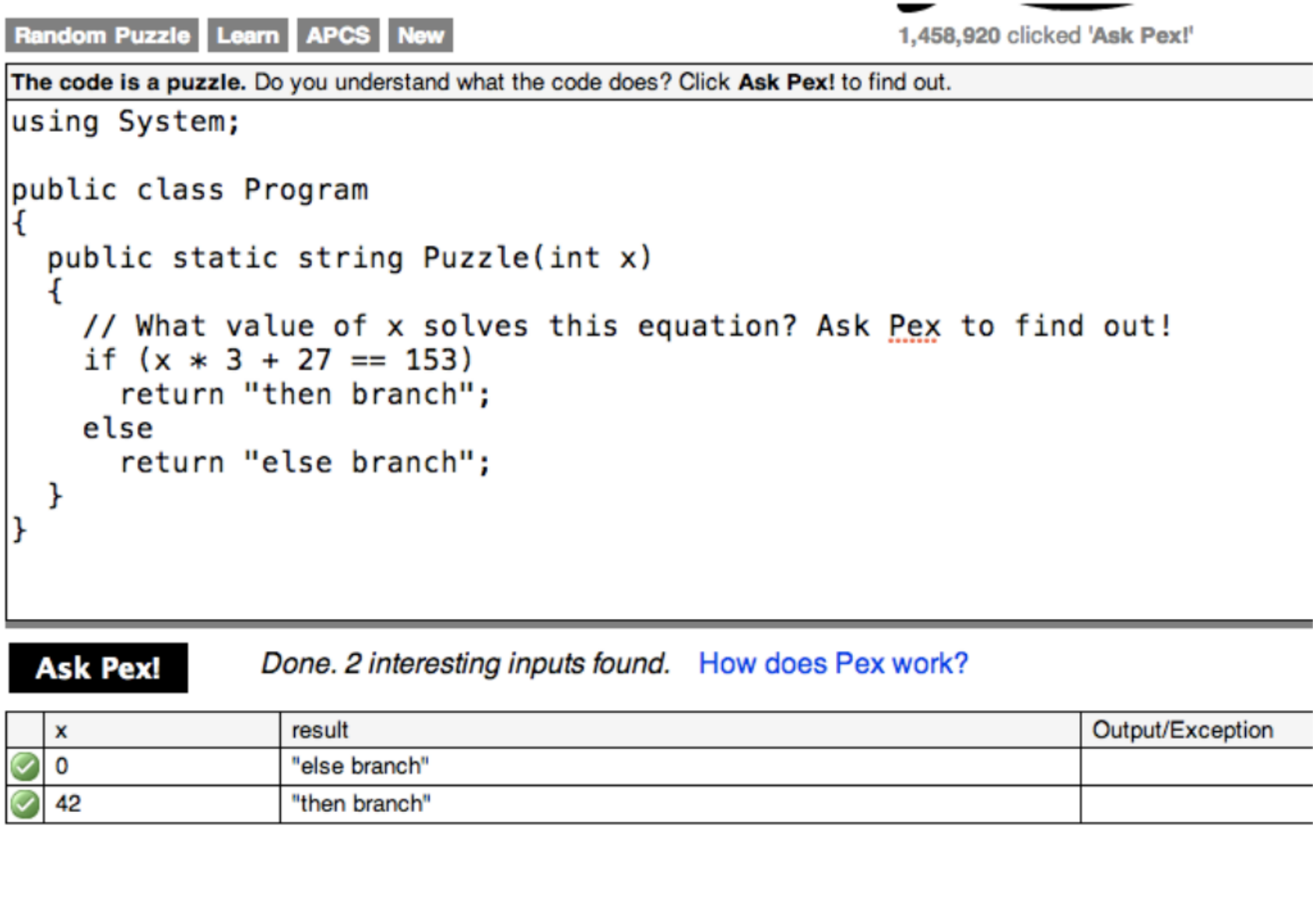
- lint: the original (somewhat lame) static analyzer for C.
- splint: a better lint (for C, also).
  Checks for security vulnerabilities and coding mistakes.

www.splint.org

Pex (Microsoft)

White Box Unit Testing!

`pexforfun.com`



KLEE: Symbolic Execution Engine (Stanford)

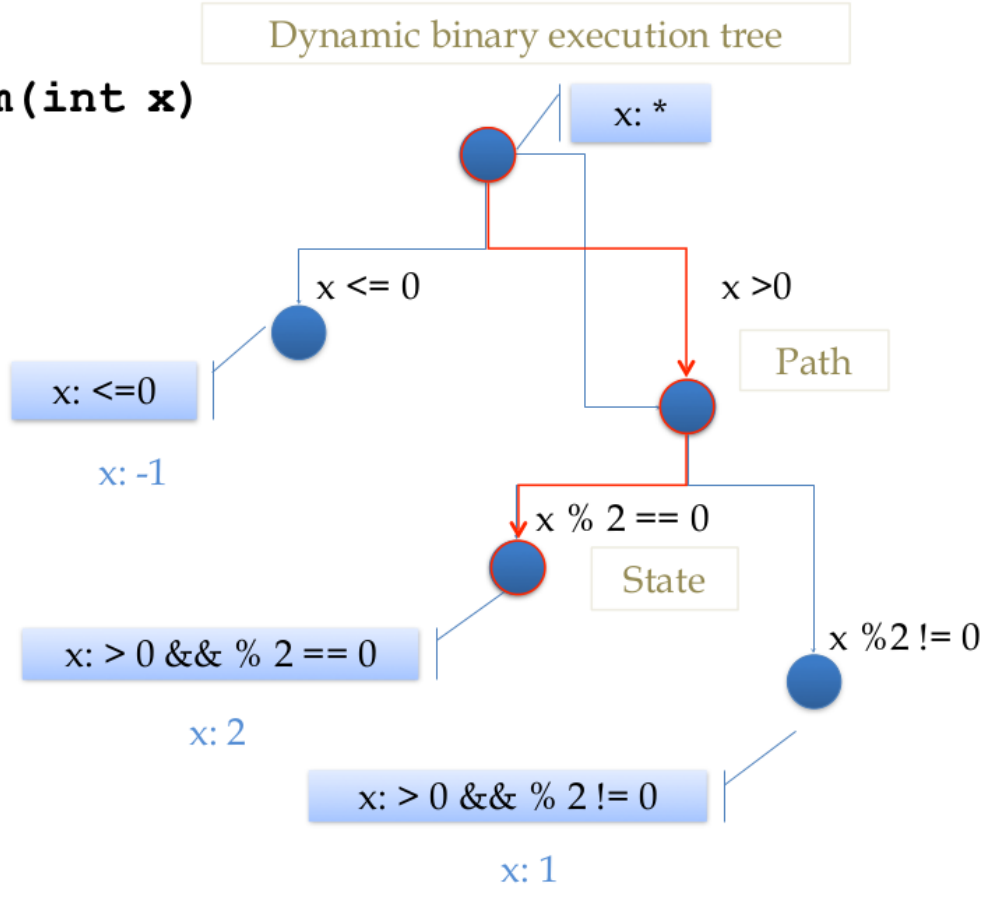
Key idea: Use symbolic execution to automatically generate high-coverage test suites.

Challenge: zillions of program paths, find the interesting ones.

Download KLEE: `klee.github.io`
Research paper:
`www.doc.ic.ac.uk/~cristic/papers/klee-osdi-08.pdf`

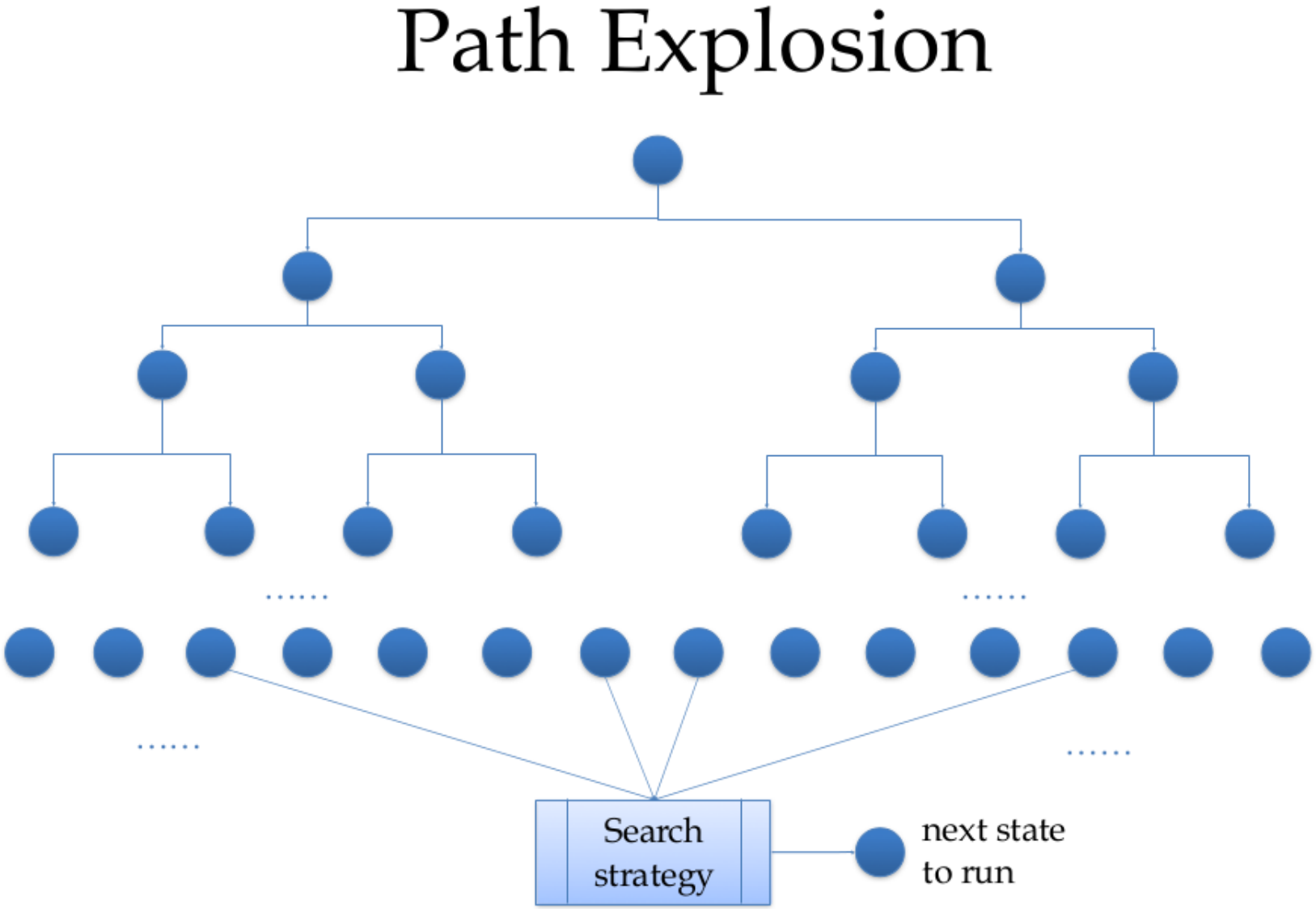
Symbolic Execution

```
bool isPositiveEvenNum(int x)
{
  if (x <= 0)
    return false;
  else {
    if (x % 2 == 0)
      return true;
    else
      return false;
  }
}
```



DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing.
ICSE '15: Wong, Zhang, Wang, Liu, & Tan.

Symbolic Execution's Problem



DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing. ICSE '15: Wong, Zhang, Wang, Liu, & Tan.

DASE

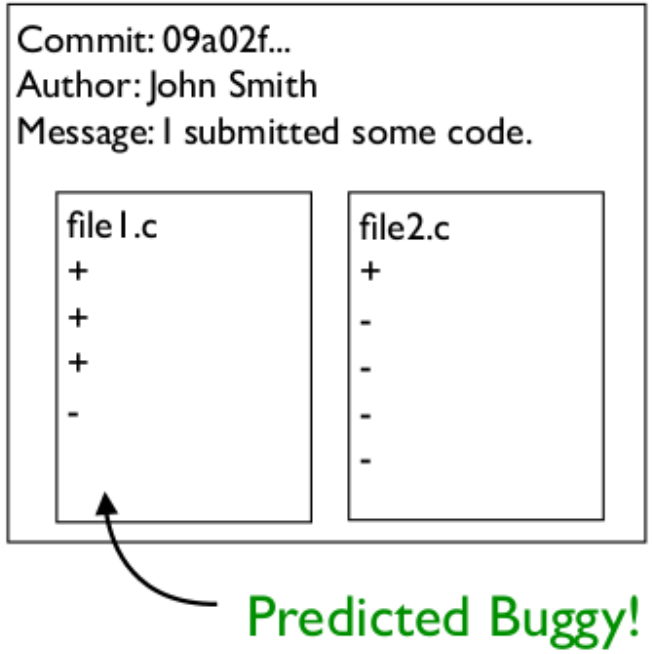
Use input constraints automatically extracted from documents to guide symbolic execution to test more effectively:

DASE: Document-Assisted Symbolic Execution for Improving Automated Software Testing.
ICSE ’15: Edmund Wong, Lei Zhang, Song Wang, Taiyue Liu, and Lin Tan.

Defect Prediction

Source Control

Code
Comments,
Documentation



Personalized Defect Prediction (ASE '13), Jiang, Tan and Kim.

(picture courtesy A. Hassan)

Part I

Other Tools from Industry

Coverity Static Analyzer

Identifies bugs in C/C++, Java, and C# codebases.

Scales to “hundreds of users, thousands of defects, and millions of lines of code in a single analysis.”

Infers must-beliefs and may-beliefs.

Aims for low false positives.

`www.coverity.com`

GrammaTech CodeSonar

Static analysis tool for C, C++, and Java.

CodeSonar is very good at C/C++.

Java bug finding performance similar to FindBugs.
(CodeSonar has better user interface)

Aims for high recall (find all the things!)

`www.grammatech.com/products/codesonar`

Visual Studio (Microsoft)

```
opensolaris/intel/io/acpica/resources/rscalc.c:
/* … AmlBufferLength - Size of AmlBuffer … */
ACPI_STATUS AcpiRsGetListLength (
    UINT8                           *AmlBuffer,
    UINT32  __ecount(AmlBuffer)     AmlBufferLength, ...)
```

Can write constraints that are related to e.g. buffer length, which are checked for bugs.

Other Commercial Tools

PCLint:
fast, naïve. `www.gimpel.com/html/pcl.htm`

PVS-Studio: `www.viva64.com/en/b/0149`

Fortify: helps find security vulnerabilities
in a wide variety of languages + config files

Intel Parallel Studio XE:
Static Security Analysis for C++, Fortran

Klocwork Insight:
finds security issues & bugs in C/C++, Java, C#

Tools, more for Development

ScalaTest: flexible testing framework for Scala

ScalaCheck: random test generators,
property-based testing

Jacoco: code coverage
(many others also, eg EclEmma)

Atlassian Bamboo: continuous integration server

suggested by Michael Viana

Homework

Draw a decision tree to help a software tester/developer select an appropriate tool (if any) for a given project.

- more than one tool may be appropriate.

Pick two tools, use them, and compare them.