Program Analysis
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>What should hold about the program?</td>
<td>What do we want to achieve regarding the specification?</td>
<td>How will we achieve the goal?</td>
</tr>
</tbody>
</table>
Dawn of Computing
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 et seq.)

<table>
<thead>
<tr>
<th>Number of Operations</th>
<th>Variables acted upon</th>
<th>Variables receiving results</th>
<th>Indication of change in the value of any Variable</th>
<th>Statement of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data:  

<table>
<thead>
<tr>
<th>$V_1$</th>
<th>$V_2$</th>
<th>$V_3$</th>
<th>$V_4$</th>
<th>$V_5$</th>
<th>$V_6$</th>
<th>$V_7$</th>
<th>$V_8$</th>
<th>$V_9$</th>
<th>$V_{10}$</th>
<th>$V_{11}$</th>
<th>$V_{12}$</th>
<th>$V_{13}$</th>
<th>$V_{14}$</th>
<th>$V_{15}$</th>
<th>$V_{16}$</th>
<th>$V_{17}$</th>
<th>$V_{18}$</th>
<th>$V_{19}$</th>
<th>$V_{20}$</th>
<th>$V_{21}$</th>
<th>$V_{22}$</th>
<th>$V_{23}$</th>
<th>$V_{24}$</th>
<th>$V_{25}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Working Variables:  

<table>
<thead>
<tr>
<th>$V_{12}$</th>
<th>$V_{23}$</th>
<th>$V_{24}$</th>
<th>$V_{25}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Result Variables:  

<table>
<thead>
<tr>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
<th>$B_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Here follows a repetition of Operations thirteen to twenty-three.
Manual Program Analysis
Alan Turing.
"Checking a large routine."
Program Verification

Given a program and a specification, show that the program conforms to the specification by creating a formal proof.
int main()
{
    unsigned int a, b, c;
    scanf("%d %d %d", &a, &b, &c);

    if (a + b == c && c - b != a)
    
        crash();

}
int main()
{
    unsigned int a, b, c;
    scanf("%d %d %d", &a, &b, &c);

    if (a+b+c != 0 && pow(a, 3) + pow(b, 3) == pow(c, 3))
        crash();
}

Specification: The program should not crash.
Program Testing

Given a program and a specification, show that the program does not conform to the specification by providing a counterexample.
int main()
{
    unsigned int a, b, c;
    scanf("%d %d %d", &a, &b, &c);

    if (a+b+c != 0 && pow(a, 2) + pow(b, 2) == pow(c, 2))
        crash();
}

Specification: The program should not crash.

Counterexample:

a == 3
b == 4
c == 5
Program testing via "Trash Decks"

http://secretsofconsulting.blogspot.com/2017/02/fuzz-testing-and-fuzz-history.html
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Properties</td>
<td>Verification</td>
<td>Manual</td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Safety</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Need for Automated Techniques
Grace Hopper.  
"The Education of a Computer."  
Proceedings of the 1952 ACM national meeting, 1952.
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>What should hold about the program?</td>
<td>What do we want to achieve regarding the specification?</td>
<td>How will we achieve the goal?</td>
</tr>
<tr>
<td>Logical Properties</td>
<td>Verification</td>
<td>Manual</td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>Type Safety</td>
<td>Transformation</td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Robert Graham.
"Protection in an information processing utility."
Ken Thompson.
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Properties</td>
<td>What do we want to achieve regarding the specification?</td>
<td>Verification</td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td></td>
<td>Testing</td>
</tr>
<tr>
<td>Type Safety</td>
<td></td>
<td>Transformation</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Safety</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td>Manual</td>
</tr>
</tbody>
</table>

What should hold about the program?

How will we achieve the goal?
Automated Techniques
Prerequisites

Basic Block

Constraints

```
x = input()
if x == 42:
    print "Correct"
else:
    print "No"
```
x = input()
if x == 42:
    print "Correct"
else:
    print "No"
if x == 1337:
    print "Fine"
Prerequisities

Basic Block

Constraints

Control Flow Graph

Path

Path Predicates

```python
x = input()
if x == 42:
    print "Correct"
else:
    print "No"
if x == 1337:
    print "Fine"
```

```plaintext
x != 42
x == 1337
```
The Rise of Symbolic Execution

Robert Boyer, et al.
"SELECT—a formal system for testing and debugging programs by symbolic execution."
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
    print "Exiting..."
exit()
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
username = input()

if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
atoi()
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 792 et seq.)

<table>
<thead>
<tr>
<th>Number of Operations</th>
<th>Nature of Operation</th>
<th>Variables acted upon</th>
<th>Variables receiving results</th>
<th>Indication of change in the value of any Variable</th>
<th>Statement of Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$V_1 \times V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$s \times 2$</td>
<td>$s \times 2$</td>
</tr>
<tr>
<td>2</td>
<td>$V_1 - V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 - 1$</td>
<td>$2 - 1$</td>
</tr>
<tr>
<td>3</td>
<td>$V_1 + V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 + 1$</td>
<td>$2 + 1$</td>
</tr>
<tr>
<td>4</td>
<td>$V_1 = V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 = 1$</td>
<td>$2 = 1$</td>
</tr>
<tr>
<td>5</td>
<td>$V_1 - V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 - 1$</td>
<td>$2 - 1$</td>
</tr>
<tr>
<td>6</td>
<td>$V_1 - V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 - 1$</td>
<td>$2 - 1$</td>
</tr>
<tr>
<td>7</td>
<td>$V_1 = V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 = 1$</td>
<td>$2 = 1$</td>
</tr>
<tr>
<td>8</td>
<td>$V_1 + V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 + 1$</td>
<td>$2 + 1$</td>
</tr>
<tr>
<td>9</td>
<td>$V_1 = V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 = 1$</td>
<td>$2 = 1$</td>
</tr>
<tr>
<td>10</td>
<td>$V_1 - V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 - 1$</td>
<td>$2 - 1$</td>
</tr>
<tr>
<td>11</td>
<td>$V_1 = V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 = 1$</td>
<td>$2 = 1$</td>
</tr>
<tr>
<td>12</td>
<td>$V_1 = V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$V_1$, $V_2$</td>
<td>$2 = 1$</td>
<td>$2 = 1$</td>
</tr>
</tbody>
</table>

Here follows a repetition of Operations thirteen to twenty-three.
<table>
<thead>
<tr>
<th>Variables acted upon.</th>
<th>Variables receiving results.</th>
<th>Indication of change in the value on any Variable.</th>
<th>Statement of Results.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( V_2 \times V_3 )</td>
<td>( 1V_4, 1V_5, 1V_6 )</td>
<td>( \begin{align*} 1V_2 &amp;= 1V_2 \ 1V_3 &amp;= 1V_3 \ 1V_4 &amp;= 2V_4 \ 1V_5 &amp;= 1V_5 \end{align*} )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>( V_4 - V_1 )</td>
<td>( 2V_4 )</td>
<td>( \begin{align*} 1V_4 &amp;= 2V_4 \ 1V_1 &amp;= 1V_1 \ 2V_4 &amp;= 0V_4 \end{align*} )</td>
<td>( 2n - 1 )</td>
</tr>
<tr>
<td>( V_5 + V_1 )</td>
<td>( 2V_5 )</td>
<td>( \begin{align*} 1V_5 &amp;= 2V_5 \ 1V_1 &amp;= 1V_1 \end{align*} )</td>
<td>( 2n + 1 )</td>
</tr>
<tr>
<td>( V_6 \div V_4 )</td>
<td>( 1V_{11} )</td>
<td>( \begin{align*} 2V_6 &amp;= 0V_6 \ 2V_4 &amp;= 0V_4 \ 1V_{11} &amp;= 2V_{11} \end{align*} )</td>
<td>( \frac{2n - 1}{2} )</td>
</tr>
<tr>
<td>( V_{11} + V_1 )</td>
<td>( 2V_{11} )</td>
<td>( \begin{align*} 1V_{11} &amp;= 2V_{11} \ 1V_1 &amp;= 1V_2 \end{align*} )</td>
<td>( \frac{2n + 1}{2} )</td>
</tr>
<tr>
<td>( V_{13} - V_{11} )</td>
<td>( 1V_{13} )</td>
<td>( \begin{align*} 2V_{13} &amp;= 0V_{13} \ 1V_{13} &amp;= 1V_{13} \end{align*} )</td>
<td>( \frac{2n - 1}{2} )</td>
</tr>
<tr>
<td>( V_3 - V_1 )</td>
<td>( 1V_{10} )</td>
<td>( \begin{align*} 1V_3 &amp;= 1V_3 \ 1V_1 &amp;= 1V_1 \end{align*} )</td>
<td>( n - 1 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>( 1V_1 )</th>
<th>( 1V_2 )</th>
<th>( 1V_3 )</th>
<th>( 0V_4 )</th>
<th>( 0V_5 )</th>
<th>( 0V_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( V_1 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( n )</td>
<td>( 1 )</td>
<td>( 2 )</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>( n )</td>
<td>( 1 )</td>
<td>( 2 )</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>( n )</td>
<td>( 1 )</td>
<td>( 2 )</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>( n )</td>
<td>( 1 )</td>
<td>( 2 )</td>
<td>( n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
<td>( 2n )</td>
</tr>
<tr>
<td>Specification</td>
<td>Goal</td>
<td>Technique</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>----------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>What should hold about the program?</td>
<td>What do we want to achieve regarding the specification?</td>
<td>How will we achieve the goal?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical Properties</td>
<td>Verification</td>
<td>Manual</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td>Testing</td>
<td>Symbolic Execution</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type Safety</td>
<td>Transformation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Safety</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Emergence of Static Analysis

Patrick & Radhia Cousot.

"Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints"

ACM Symposium on Principles of Programming Languages, 1977,
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Properties</td>
<td>Verification</td>
<td>Manual</td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td>Testing</td>
<td>Symbolic Execution</td>
</tr>
<tr>
<td>Type Safety</td>
<td>Transformation</td>
<td>Abstract Interpretation</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Safety</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fuzzing Appears

Joe W. Duran, et al.
"A report on random testing".
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
    print "Exiting..."
exit()
<table>
<thead>
<tr>
<th>Specification</th>
<th>Goal</th>
<th>Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Properties</td>
<td>Verification</td>
<td>Manual</td>
</tr>
<tr>
<td>Absence of Crashes</td>
<td>Testing</td>
<td>Symbolic Execution</td>
</tr>
<tr>
<td>Type Safety</td>
<td>Transformation</td>
<td>Abstract Interpretation</td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td>Fuzzing</td>
</tr>
<tr>
<td>Memory Safety</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information Disclosure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What should hold about the program?</td>
<td>What do we want to achieve regarding the specification?</td>
<td>How will we achieve the goal?</td>
</tr>
</tbody>
</table>
The Program Analysis Nursery
The Program Analysis Nursery

- 249 programs
- Source code available
- Range of vulnerability classes
- Documented vulnerabilities
- Simple OS model
- Explicit security specifications!
Nursery Experiments

Symbolic Execution: 9
Optimized Symbolic Execution: 26
Symbolic Execution + Veritesting*: 31
Fuzzing (AFL): 106
username = input()
if username == "service":
    cmd_code = atoi(input())
    if cmd_code == 7:
        crash()
    else:
        print "Unknown command".
else:
    passcode = atoi(input())
    if passcode < 10000:
        print "Invalid passcode!"
    else:
        auth(username, passcode)
        print "Exiting..."
exit()
Nursery Experiments

- Symbolic Execution: 9
- Optimized Symbolic Execution: 26
- Symbolic Execution + Veritesting*: 31
- Fuzzing (AFL): 106
- Symbolically-assisted Fuzzing (Driller): 118

Total: 249
Driller Results

Applicability varies by program.

Where it was needed, Driller increased block coverage by an average of 71%.
Nursery Experiments

Symbolic Execution: 9
Optimized Symbolic Execution: 26
Symbolic Execution + Veritesting*: 31
Fuzzing (AFL): 106
Symbolically-assisted Fuzzing (Driller): 118
Join in!
Contribute to open-source frameworks!

http://anqr.io
Come do research!

I am actively looking for students, interns, etc!

Yan Shoshitaishvili
Me: @Zardus yans@yancomm.net
Team: @Shellphish team@shellphish.net
This presentation: https://goo.gl/57BAoc

Questions?