Under-Approximation

- Any behaviour analysed is **genuine**
- Promises fewer false alarms
- But may **miss some bugs**

- Much like testing!
- But automatic
- Can still deal with partial systems
Symbolic Execution

- Run program, but write down formula instead of program state
- Get program inputs from constraint solver
- View as “solver-guided” fuzz-testing
Path-based Symbolic Execution

```c
if ( (0 <= t) && (t <= 79) )
    switch ( t / 20 )
    {
        case 0:
            TEMP2 = ( (B AND C) OR (~B AND D) );
            TEMP3 = ( K-1 );
            break;
        case 1:
            TEMP2 = ( (B XOR C XOR D) );
            TEMP3 = ( K-2 );
            break;
        case 2:
            TEMP2 = ( (B AND C) OR (B AND D) OR (C AND D) );
            TEMP3 = ( K-3 );
            break;
        case 3:
            TEMP2 = ( B XOR C XOR D );
            TEMP3 = ( K-4 );
            break;
    default:
        assert(0);
    }

(from an implementation of SHS)
```
Path-based Symbolic Execution

\[
0 \leq t \leq 79 \\
\land t/20 \neq 0 \\
\land t/20 \neq 1 \\
\land t/20 \neq 2 \\
\land t/20 \neq 3 \\
\land TEMP2 = B \oplus C \oplus D \\
\land TEMP3 = K \cdot 2
\]
Path-based Symbolic Execution

We pass

\[ 0 \leq t \leq 79 \]
\[ \land \quad t/20 \neq 0 \]
\[ \land \quad t/20 = 1 \]
\[ \land \quad TEMP2 = B \oplus C \oplus D \]
\[ \land \quad TEMP3 = K \_2 \]

to a decision procedure, and obtain a satisfying assignment, say:

\[ t \mapsto 21, \ B \mapsto 0, \ C \mapsto 0, \ D \mapsto 0, \ K \_2 \mapsto 10, \]
\[ TEMP2 \mapsto 0, \ TEMP3 \mapsto 10 \]

✓ It provides the values of any inputs on the path.
Path-based Symbolic Execution

If $0 \leq t \leq 79$

Switch

Case 0
$t/20 \neq 0$

Case 1
$t/20 \neq 1$

Case 2
$t/20 \neq 2$

Case 3
$t/20 \neq 3$

Default

That is UNSAT, so the assertion is unreachable.
Path-based Symbolic Execution

What if variable is assigned twice?

```c
x=0;
if (y >= 0)
  x++;  
```

Rename appropriately:

\[
x_1 = 0 \\
\land y_0 \geq 0 \\
\land x_2 = x_1 + 1
\]

This is a special case of SSA (static single assignment)
Symbolic Execution: Advantages

- Can look for very specific things
  - Look for user-specified events
  - Constrain with partial inputs
  - Constrain with observations from logs
    (e.g.: NASA uses this for probe logs)

- Only needs an operational model, and thus has been done for wide range of languages
  (including JavaScript and x86 assembler)
Prominent Tools

- SAGE, PEX, CodeDigger (Microsoft)
- KLEE
- Verisoft (concurrency)
- Romano: Linux Bug Release
  http://www.bugsdujour.com/release/
  30k binaries,
  5 min symbolic execution per binary
Path-based Symbolic Exection: Scalability

This is a loop with an if inside.

Q: how many paths for \( n \) iterations?
Improving Scalability

- The SAT problems are too easy!

<table>
<thead>
<tr>
<th>Total programs</th>
<th>33,248</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total SMT queries</td>
<td>15,914,407,892</td>
</tr>
<tr>
<td>Queries hitting cache</td>
<td>12,307,311,404</td>
</tr>
<tr>
<td>Symbolic intrs</td>
<td>71,025,540,812</td>
</tr>
<tr>
<td>Run time</td>
<td>235,623,757s</td>
</tr>
<tr>
<td>Symb exec time</td>
<td>125,412,247s</td>
</tr>
<tr>
<td>SAT time</td>
<td>40,411,781s</td>
</tr>
<tr>
<td>Model gen time</td>
<td>30,665,881s</td>
</tr>
<tr>
<td># test cases</td>
<td>199,685,594</td>
</tr>
<tr>
<td># crashes</td>
<td>2,365,154</td>
</tr>
<tr>
<td># unique bugs</td>
<td>11,687</td>
</tr>
<tr>
<td># fixed bugs</td>
<td>162</td>
</tr>
<tr>
<td>Confirmed control flow hijack</td>
<td>152</td>
</tr>
</tbody>
</table>
Path Merging

- Idea: use SSA $\phi$-nodes when paths meet
- Much like $\phi$-folding in compilers
Merge _All: BMC

- Also called Bounded Model Checking
- Builds one big formula

- Users are primarily in the automotive domain
  - Toyota
  - BTC-ES
  - TCS
Example

Program

```c
void my_func(int param_0) {
    int x, y;

    y1 = 0xabc;
    x1 = param_0 + 1;
    if (x1 > 10)
        x2 = 10;
    x3 = \phi(x1, x2);

    y2 = y1 + x3;
}
```

Formula

\[
\begin{align*}
    y_1 &= 2748 \\
    x_1 &= \text{param}_0 + 1 \\
    \text{cond}_1 &= (x_1 > 10) \\
    x_2 &= 10 \\
    x_3 &= \text{ITE}(\text{cond}_1, x_2, x_1) \\
    y_2 &= y_1 + 1
\end{align*}
\]
CBMC is by far the best bug finder in SV-COMP's category Falsification.\url{http://ift.tt/2jm2jfo} Quantile Plot:

\url{http://ift.tt/2kkjr6D}

CBMC is by far the best bug finder in SV-COMP's category Falsification....

plus.google.com

Zvonimir Rakamaric and 5 others

2 Shares
Information Leakage: Heartbleed

- OpenSSL is widely used
- In April 2014 an implementation error was found in OpenSSL through manual code review
- Because of this error, passwords and other private information could be extracted from a remote system
- As the error was in the heartbeat protocol, it was named “Heartbleed”
- Tautschnig/Malacharia: CBMC for measuring leakage
Information Leakage: Heartbleed

$ time cbmc -DTLS1_process_heartbeat d1_both_harness.c --unwind 50 --all-properties -DBUG
Applications of CBMC

- [http://www.cprover.org/cbmc/applications/](http://www.cprover.org/cbmc/applications/)
- Error explanation and localization
- Concurrency
- Equivalence Checking
- Cyper-physical Systems and Control
- Test-vector generation
- Worst-case execution time
- Security
Download me!

- Available as package in most Linux distributions
- Now a regression test in the Linux mainline kernel
- Get sources via www.github.com/diffblue/cbmc
- Preliminary support for Java