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

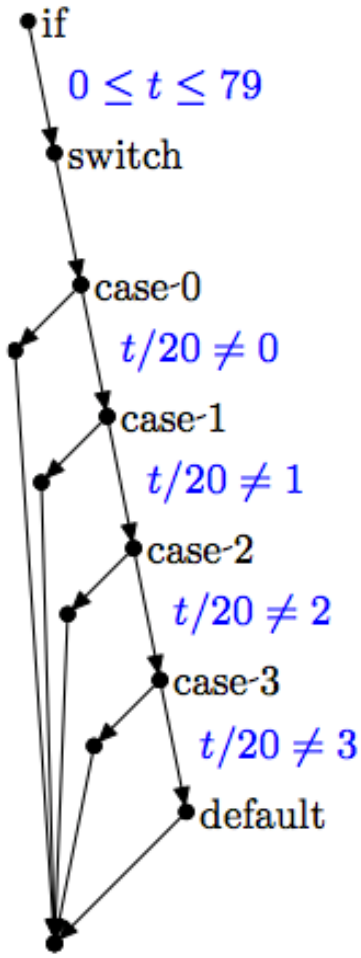
Symbolic Execution

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

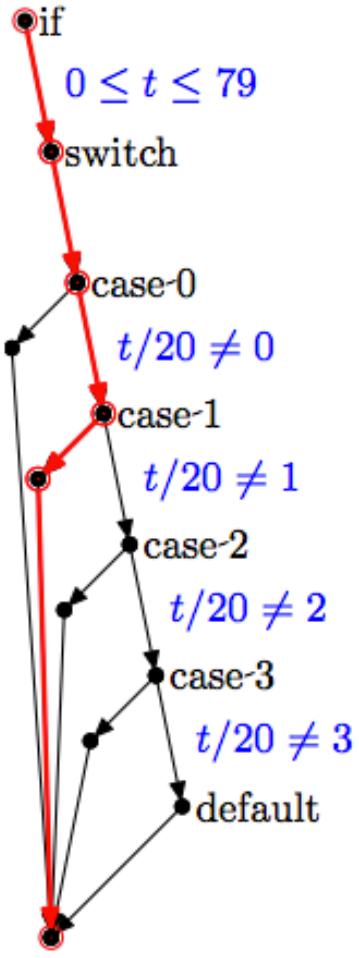
Path-based Symbolic Execution

```
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



$$
\begin{array}{ll}
 & 0 \leq t \leq 79 \\
\wedge & t/20 \neq 0 \\
\wedge & t/20 = 1 \\
\wedge & TEMP2 = B \oplus C \oplus D \\
\wedge & TEMP3 = K\_2
\end{array}
$$

Path-based Symbolic Execution

We pass

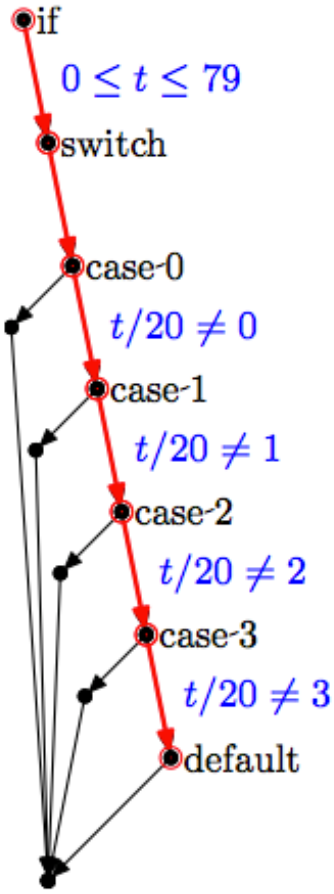
$$
\begin{aligned}
& 0 \leq t \leq 79 \\
\wedge \quad & t/20 \neq 0 \\
\wedge \quad & t/20 = 1 \\
\wedge \quad & TEMP2 = B \oplus C \oplus D \\
\wedge \quad & TEMP3 = K\_2
\end{aligned}
$$

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

$$
\begin{gathered}
t \mapsto 21,\ B \mapsto 0,\ C \mapsto 0,\ D \mapsto 0,\ K\_2 \mapsto 10, \\
TEMP2 \mapsto 0,\ TEMP3 \mapsto 10
\end{gathered}
$$

✔ It provides the values of any inputs on the path.

Path-based Symbolic Execution



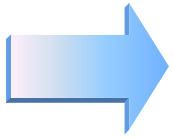
$$
\begin{aligned}
& 0 \leq t \leq 79 \\
\wedge \quad & t/20 \neq 0 \\
\wedge \quad & t/20 \neq 1 \\
\wedge \quad & t/20 \neq 2 \\
\wedge \quad & t/20 \neq 3
\end{aligned}
$$

That is UNSAT, so the assertion is unreachable.

Path-based Symbolic Execution

What if variable is assigned twice?

```
x=0;

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



Rename appropriately:

$$
\begin{array}{ll}
 & x_1 = 0 \\
\wedge & y_0 \geq 0 \\
\wedge & x_2 = x_1 + 1
\end{array}
$$

This is a special case of SSA (static single assignment)

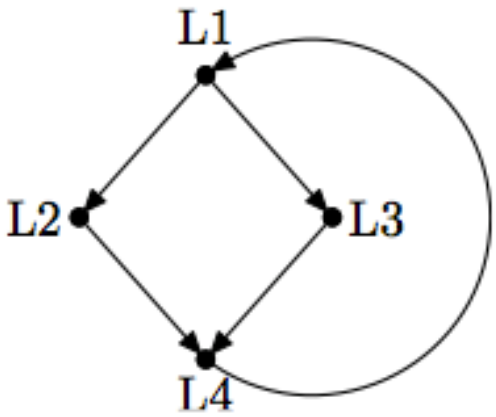
Symbolic Execution: Advantages

- 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



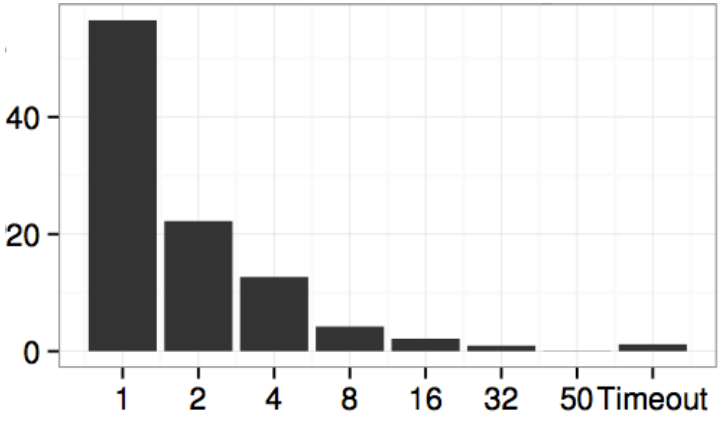
This is a loop with an `if` inside.

Q: how many paths for n iterations?

Improving Scalability

- The SAT problems are <u>too easy</u>!

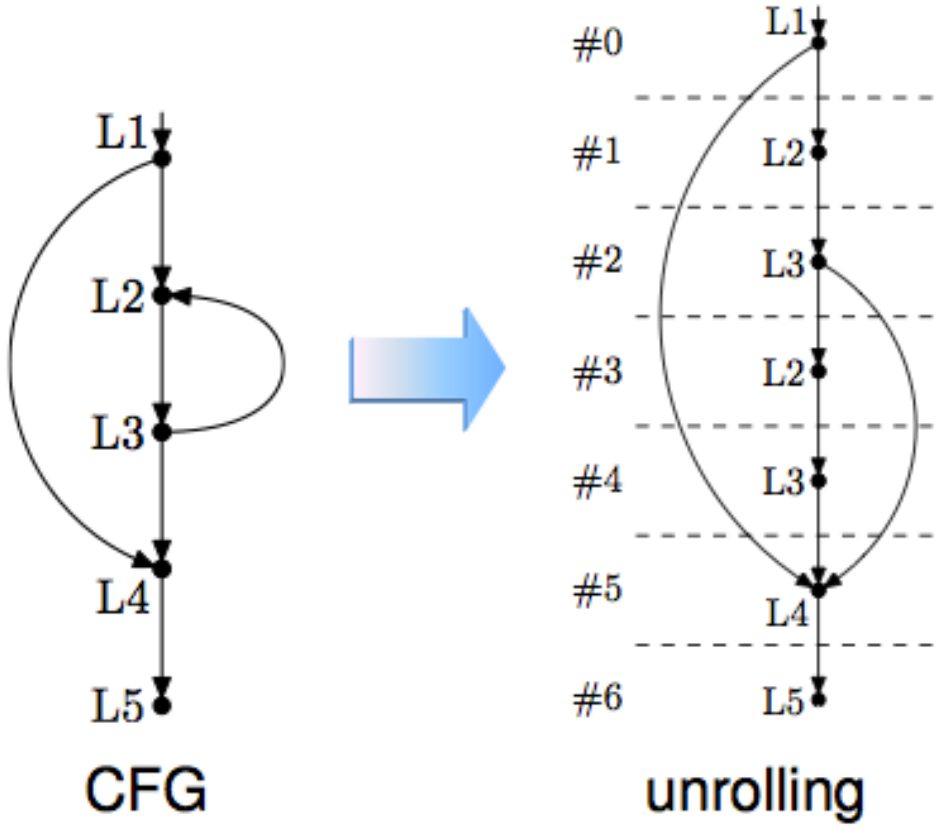
Total programs	33,248
Total SMT queries	15,914,407,892
Queries hitting cache	12,307,311,404
Symbolic instrs	71,025,540,812
Run time	235,623,757s
Symb exec time	125,412,247s
SAT time	40,411,781s
Model gen time	30,665,881s
# test cases	199,685,594
# crashes	2,365,154
# unique bugs	11,687
# fixed bugs	162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



[ICSE 2014]

Path Merging

- Idea: use SSA ϕ-nodes when paths meet
- Much like ϕ-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

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

  y_1 = 0xabc;
  x_1 = param_0 + 1;
  if (x_1 > 10)
    x_2 = 10;
  x_3 = φ(x_1, x_2);

  y_2 = y_1 + x_3;
}
```

Formula

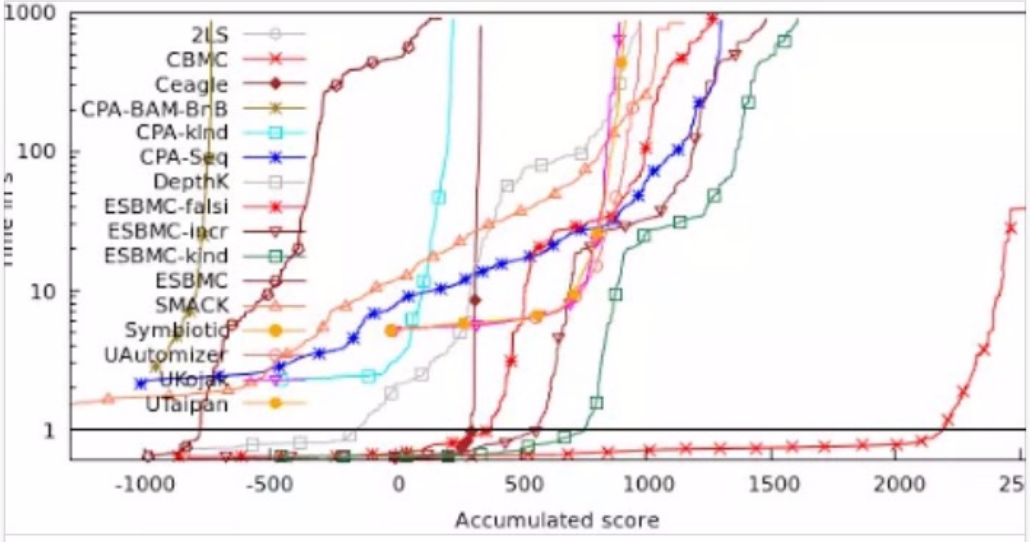
$$
\begin{aligned}
& y_1 = 2748 \\
\wedge\ & x_1 = param_0 + 1 \\
\wedge\ & cond_1 = (x_1 > 10) \\
\wedge\ & x_2 = 10 \\
\wedge\ & x_3 = \mathsf{ITE}(cond_1, x_2, x_1) \\
\\
\wedge\ & y_2 = y_1 + 1
\end{aligned}
$$

Dirk Beyer

9 hrs · IFTTT ·

CBMC is by far the best bug finder in SV-COMP's category Falsification.http://ift.tt/2jm2jfo Quantile Plot:

http://ift.tt/2kkrj6D



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

plus.google.com

Zvonimir Rakamaric and 5 others

2 Shares

UNIVERSITY OF OXFORD

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/Malacaria: CBMC for measuring leakage

Information Leakage: Heartbleed

3. bash

```
$ time cbmc -DTEST_dtls1_process_heartbeat d1_both_harness.c --unwind 50 --all-properties -DBUG
```

Applications of CBMC

- 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

CBMC