Why is software always crashing?
Are we lazy or just not clever enough to code?

Achim D. Brucker
a.brucker@sheffield.ac.uk https://www.brucker.ch/
Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/
May 16, 2018
We build software since over 50 years and still do not get it right.

Why?

A small example: what triangle do I have?

Our program

Given: The length of three lines

Answer: Do the three lines form a triangle?

```ocaml
> testTriangle(1,2,3);
val it = Error : triangle

> testTriangle(2,2,2);
val it = Equilateral : triangle

> testTriangle(1,2,2);
val it = Isosceles : triangle

> testTriangle(2,4,5);
val it = Scalene : triangle
```

© 2018 LogicalHacking.com.
Is our program correct?

We tested 4 different inputs ...
The program has 3 inputs, each can take

\[2^{64} = 1'844'6744'073'709'551'616\]
different values

Assume we can test 1'000'000 per second
it takes 584'942 to test them all!

But we have three inputs:

\[3^{64} = 1'790'184'577'738'583'171'520'872'861'\]
\[412'518'665'678'211'592'275'841'099'096'961\]
A small example: what triangle do I have?

Let's have a look at our program

datatype triangle = Equilateral | Scalene | Isosceles | Error

fun isTriangle (x:int , y: int , z: int )
  = ( (z < (x+y)) andalso (x < (x+z)) andalso (y < (x+z)))

fun testTriangle (x:int , y:int , z: int )
  = if isTriangle (x,y,z) then
    if x=y then if y=z then Equilateral
    else Isosceles
    else if y=z then Isosceles
    else if x=z then Isosceles
    else Scalene
  else Error

Can we do better?

We can prove the correctness mathematically!

Lemmas:
- axioms: "x = y" and "x = z"
- assumptions: "isTriangle x y z"
- shows: "testTriangle x y z = isosceles"
- using: assume testTriangle_def by auto
Ensuring correctness, security, and safety

(Inductive) Verification
- Formal (mathematical) proof
- Can show absence of all failures relative to specification

Testing
- Execution of test cases
- Can show failures on real system

My vision

Combining testing and verification to ensure the security, safety, reliability, and correctness of (software) systems.

Is testing a "poor man's verification?"
Or: Why should I test if I verified my program (and vice versa)

- Fully formally verified
- Total number of flights: 0
- Fully tested
- Total number of flights: 1,000

Can we do better?
We can prove the correctness mathematically!

Verification can show the correctness (for all possible inputs)!
Any questions or remarks?

Contact: Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP, UK

a.brucker@sheffield.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/

© 2018 LogicalHacking.com, A.D. Brucker.
This presentation is classified as Public (CC BY-ND 4.0).
Except where otherwise noted, this presentation is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International Public License (CC BY-ND 4.0).