Formal Methods for Software Assurance

CITS5501/CITS3501 - Software Testing and Quality Assurance

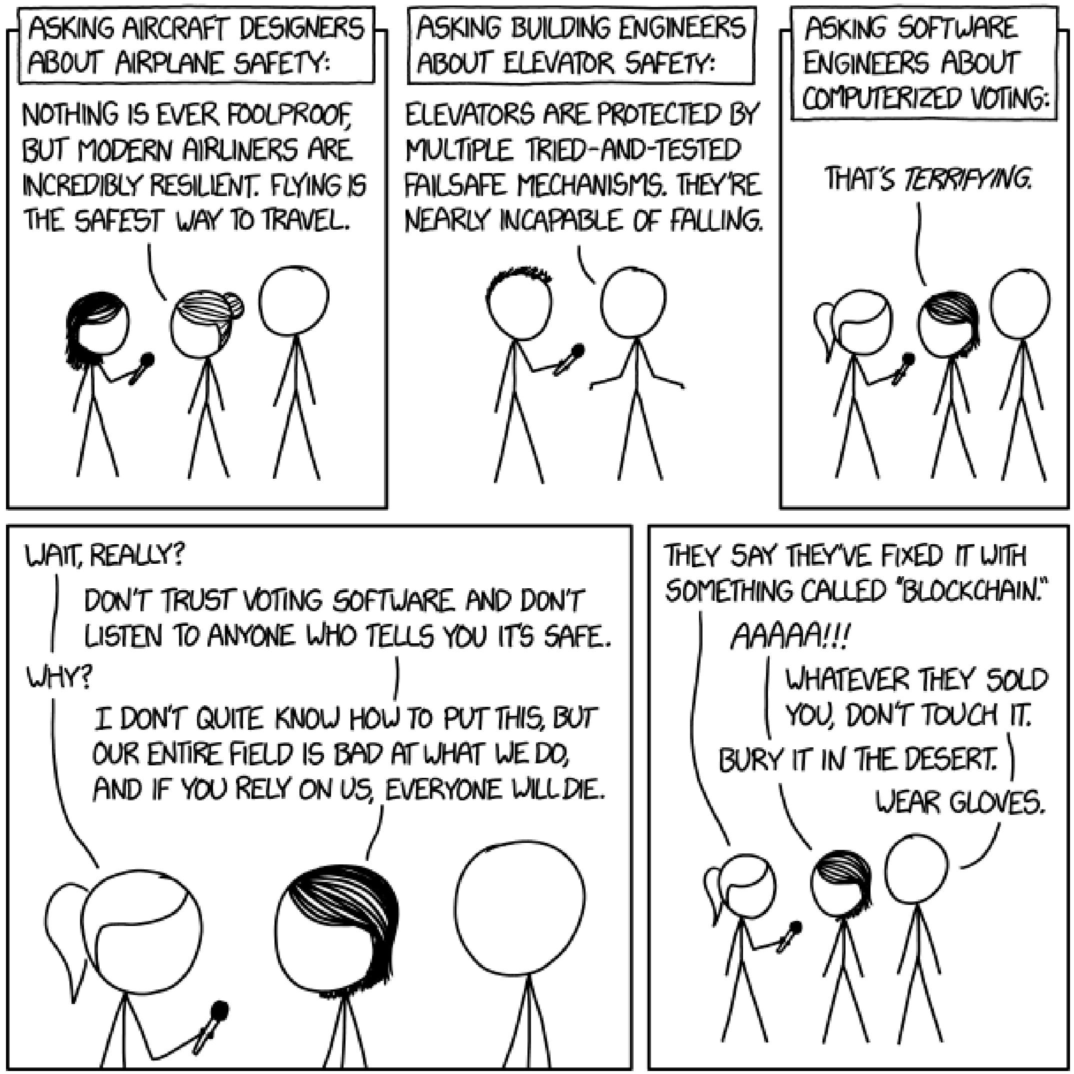
Guest lecturer: Matthew Daggitt

2025 - Semester 2

Motivation

Motivation

- Software engineers are objectively bad at their jobs compared to other fields.
- Pretty much every piece of non-trivial software has bugs in.
- How bad can bugs be?



www.xkcd.com/2030

Example 1: Knight Capital spending spree

THE UNIVERSITY OF WESTERN AUSTRALIA

Knight®

- In 2012, one of the biggest trading companies in the world pushed an update to their automated trading algorithm.
- A bug resulted in effectively an infinite loop of buying stocks.
- Lost $400 million in 28 minutes and the company went bankrupt.

https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/

Example 2: Intel Pentium processor

- In 1994, Intel released a processor which didn't always divide numbers correctly.
- Recall of the chips cost $500 million USD (in 1994 money!).

https://en.wikipedia.org/wiki/Pentium_FDIV_bug

Example 3: Boeing 737 crash

- In 2018, a Lion Air Boeing 737 Max 8 jetliner crashed into the Java Sea off Indonesia, killing all 189 passengers and crew
- Investigators described the cause as a “glitch” in the plane’s flight-control software.

https://www.henricodolfing.com/2019/06/project-failure-case-study-knight-capital.html

What can we do about it?

Our goal

- Consider the a Python function that takes two numbers and returns a result:

```
def calculate(a, b):
```

- Assume that is ***incredibly*** important that for any values of a and b then:

```
calculate(a, b) == calculate(b, a)
```

- If this property doesn't hold then all kitten pictures are going to be deleted from the internet... **forever**!

Formal methods

In testing the approach is to think of corner cases that might break, e.g.

- calculate(0,1) vs calculate(1,0)
- calculate(-1,2) vs calculate(-1,2)
- calculate(0,0) vs calculate(0,0)

However, this doesn't give us a **guarantee** that the property holds.

Formal methods are a collection of methods by which we can take a computer programs and a set of properties that we know should hold of it and obtain strong guarantees that the program satisfies those properties.

Formal verification

THE UNIVERSITY OF WESTERN AUSTRALIA

- All software is really very complicated mathematical functions.

- Formal verification consists of:

  1. Make a mathematical model of our software
     - ideally automatically from your program

  2. Write down the property that you want to hold
     - using some form of logic (e.g. predicate logic)

  3. Proving that the model satisfies the property
     - either done manually or automatically or some form of both.

Formal verification spectrum

THE UNIVERSITY OF WESTERN AUSTRALIA

- Formal methods exists broadly on the following spectrum.
    1. Testing – no property, no proof
    2. Property-based testing – property but no proof
    3. Model checking – property and unknown proof
    4. Interactive Theorem Provers – property and known proof

- In this lecture we will look at the last three types.

Property-based testing

Property-based testing

Key idea:

- You write down the property.
- The computer generates test cases automatically!

Property-based testing was introduced by researchers in the Haskell community in 1999.

Haskell
QuickCheck

There are now property-based checking libraries available in almost all major programming languages.

Property-based testing

Steps

1. <u>User</u>: Define a property P(x).

2. <u>User</u>: Choose a strategy for generating values for `x`.

3. <u>Library</u>: Use the strategy to generate x_1, x_2, … and test $P(x_1)$, $P(x_2)$, …
   - The strategy may use the results of the previous tests to inform the generation of the next test.

4. <u>Library</u>: If a counter-example x_i is found, automatically shrink x_i to be as simple as possible, and then fail the test.

Industrial applications

Correctness of key algorithms supporting cloud computing

Correctness of financial trading algorithms

Correctness of reference implementation for operating systems for vehicles

Correctness of file synchronisation algorithms

Property-based testing

- **Pros**:

  1. Write a single test and test many values at the same time.
  2. Intelligently test a much larger range of values and find many more bugs
  3. Often quicker to write than normal tests.

- **Cons**:

  1. For complex inputs, defining a strategy for generating data can be complicated.
  2. Still no hard guarantees!

Model-checking tools

Model-checking

Key idea:

- You write down the property
- The computer proves it automatically!

How does model-checking work?

- There used to be many different domain-specific model checkers.

- About 10-20 years ago, researchers found that a family of tools called SMT solvers could solve problems from almost all domains.

- Many mainstream SMT solvers accept queries in a standard format called **SMTLIB**, e.g.
  1. Z3
  2. CVC5
  3. Yices2

```
(set-logic QF_LIA)
(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (= (+ (* 6 x) (* 2 y) (* 12 z)) 30))
(assert (= (+ (* 3 x) (* 6 y) (* 3 z)) 12))
(check-sat)
```

How does model-checking work?

- Procedure for a domain-specific tool:

  1. User: writes down their property in a high-level language.
  2. Tool: compiles down the property to a set of SMTLIB queries.
  3. Tool: calls an SMT solver to answer the queries.
  4. Tool: converts any counter-example found back into a form understandable by the user.

Industrial applications

AMD

Correctness of computer-chip design.

AIRBUS

Correctness of aeroplane control systems.

esa

European Space Agency

Correctness of reference implementation for operating systems for vehicles.

Microsoft

Correctness of Windows hardware drivers

https://github.com/ligurio/practical-fm

Pros and cons of model checking

- **Pros**:

    1. Formal guarantee of correctness.

    2. (Sometimes) don't have to alter your program.

- **Cons**:

    1. Cannot prove more complicated properties
        - see CITS2211 for non-computable problems, e.g. the halting problem.

    2. Sometimes you are forced to rewrite your program to make life easier for the model checker.

    3. A counter-example doesn't immediately tell you *why* your program has gone wrong.

Interactive Theorem Provers

Interactive Theorem Provers

Key idea:

- You provide the proof.
- The computer checks it.

Problem: Standard programming languages not designed to write proofs!

Solution: Custom program languages called theorem provers in which you can write both proofs

Industrial applications

Correctness of computer-chip design.

Correctness of military control systems.

Correctness of cryptographic protocols.

Correctness of search algorithms

https://github.com/ligurio/practical-fm

Pros and cons of ITPs

- **Pros**:

  1. Formal guarantee of correctness.
  2. Can represent pretty much any proof or argument.
  3. Can be used to prove mathematical theorems as well!

- **Cons**:

  1. Writing down a correct proof is 100 times more time-consuming than writing the program!
  2. You must write your program in specialised languages.

Advice on using Formal Methods

Comparison of formal methods

Method	Guarantees	Property	Proof	Applicability	Difficulty
Testing	No	No	No	High	Low
Property-based testing	No	Yes	No	Medium	Low
Model checking	Yes	Yes	Yes (unknown)	Medium	High
Interactive Theorem Provers	Yes	Yes	Yes	High	Extremely High

Takeaways

THE UNIVERSITY OF WESTERN AUSTRALIA

1. **There is no good reason not to use property-based testing!**
   - Quick to setup and run!
   - Much more powerful than traditional tests!

2. **Model checking is situational**
   - Can be extremely powerful for relatively simple code and properties.
   - Finicky to use with more complicated properties.

3. **Interactive theorem provers are rarely the right answer (but very cool!)**
   - If either human lives or hundreds of millions of dollars depend on your code running correctly, then you can justify the cost.

Limitations of formal methods!

“All mathematical models are wrong. Some are useful.”

1976, George Box, British Statistician

- All these methods check whether the program obeys your property....
- None of them guarantee that the property itself is correct!

Large Language Models and Formal Methods

Uses of LLMs in Formal methods

What not to do:

- Ask the LLM "Is this code correct?"

Hot research fields:

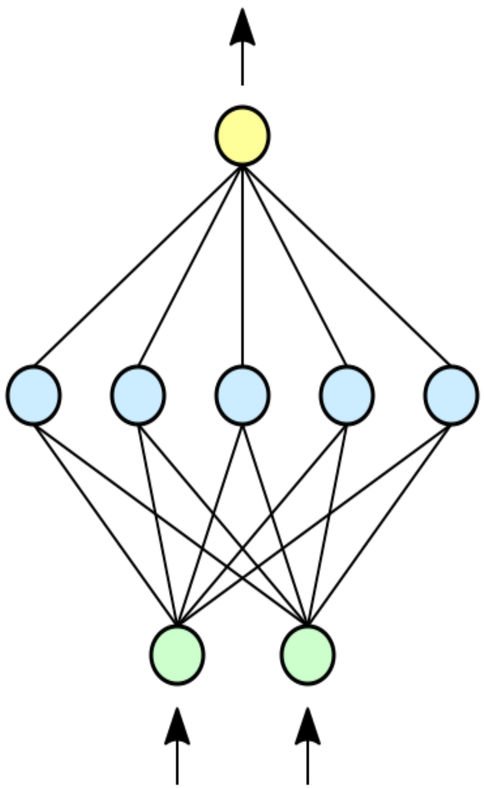
- Using LLMs to generate formal properties from human text.
- Automatic program repair based on formal properties.
- Using LLMs as generating strategies in property-based testing.

The Holy Grail in Formal Methods

When you ask an LLM to "generate code that does X", the LLM should:

1. Translate X into a formal specification.
2. Generate code that does X.
3. Generate a proof that the code satisfies the specification.
4. Give the proof to an Interactive Theorem Prover to check.
5. (Optional but recommended!) The user checks that the specification means X.

Formal methods for Neural networks



Software engineers: build neural networks as non-deterministic black-box models

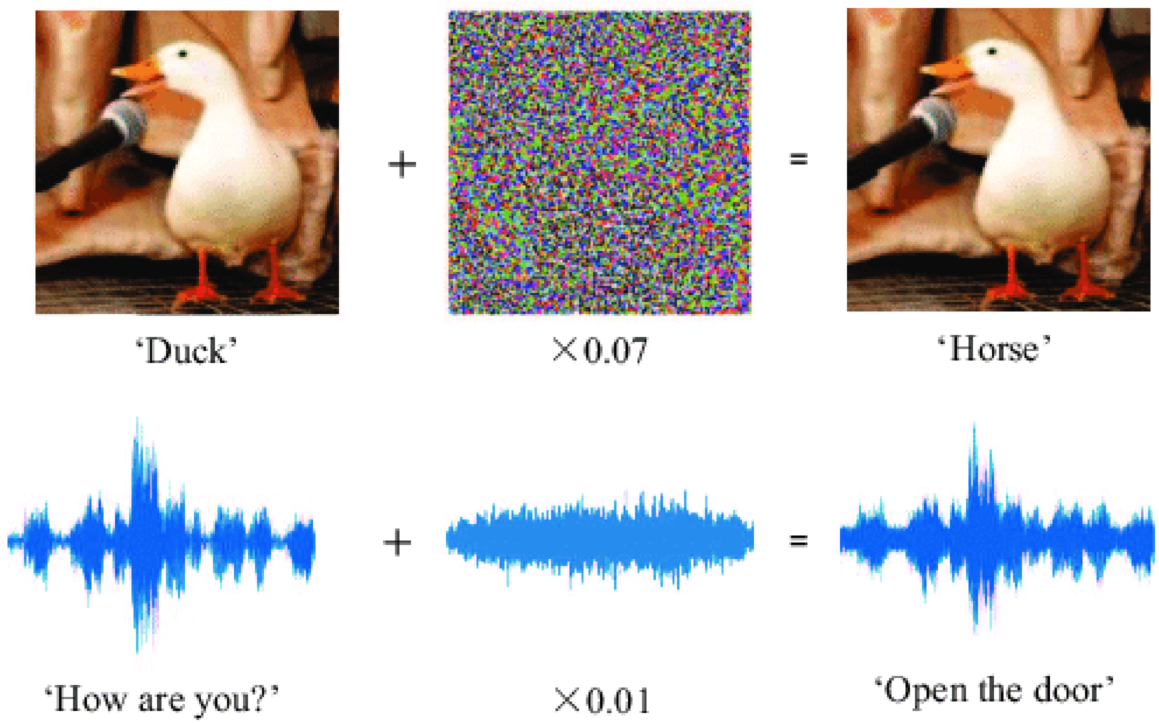
Formal methods: can't prove anything about them

Software engineers:

Self-promotion time!

My research interest: Can we use formal methods to improve neural networks?

Adversarial Robustness



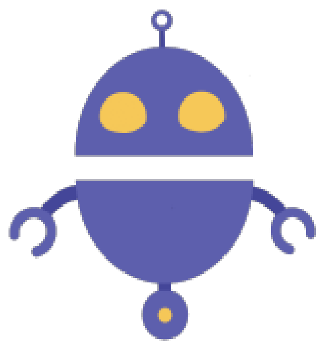
My research

THE UNIVERSITY OF WESTERN AUSTRALIA

I'm developing **Vehicle** - a tool for checking properties of neural networks.

Users can write their property in Vehicle and:

- Use model-checking to prove that a trained network obeys the property.
- (End of this year) Use the property to train the network!
- (Early next year) Use property-based testing to find counter-examples.



https://github.com/vehicle-lang/vehicle

Lots of open research questions

(For CITS4011, CITS5505, GENG4412, GENG5512)

Some research problems I'm interested in!

1. Evaluating the effectiveness of property-based testing with gradient descent.
2. The theory behind property-based testing with gradient descent.
3. Supporting properties that involve derivatives of functions.
4. Model checkers for neural networks using quantum computing.

(For CITS5551, CITS5552)

Many implementation-based software projects available as well!