

Hardware & Software Verification

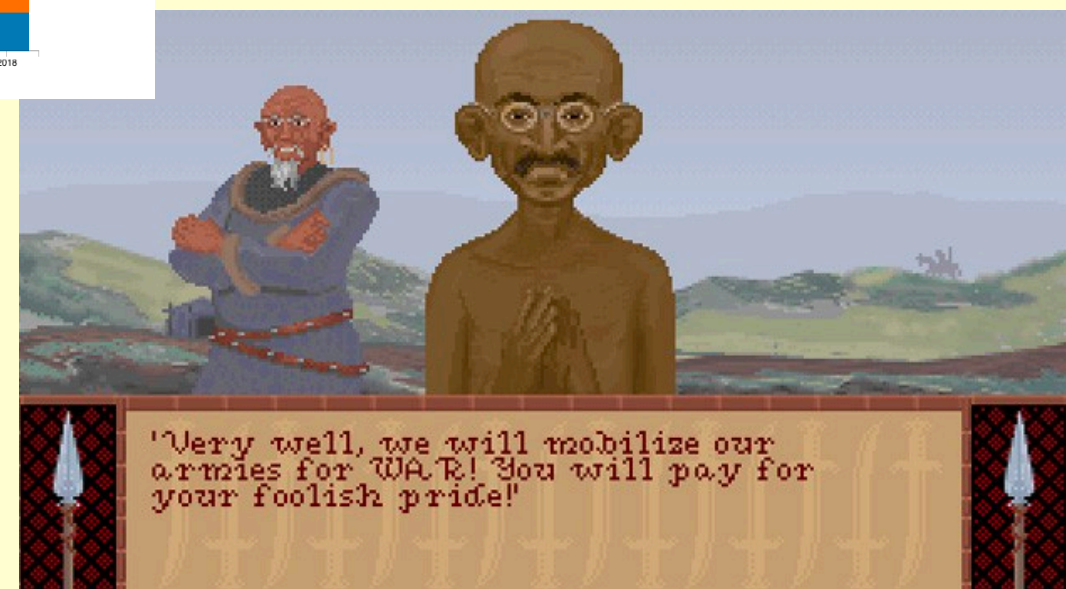
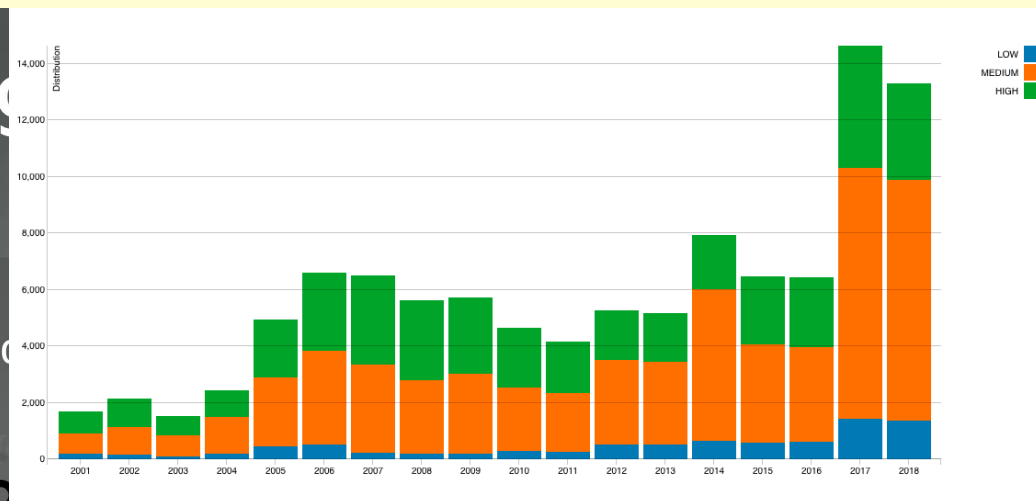
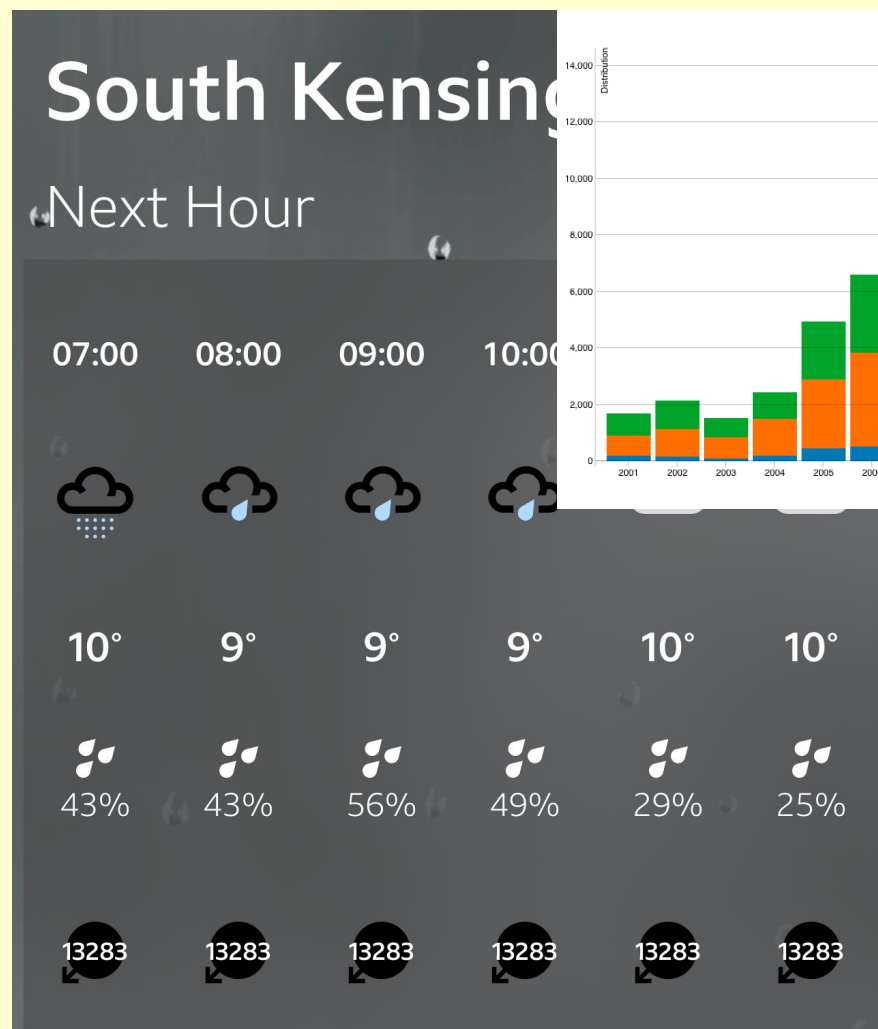
John Wickerson

Lecture 1
10 October 2024

The correctness problem



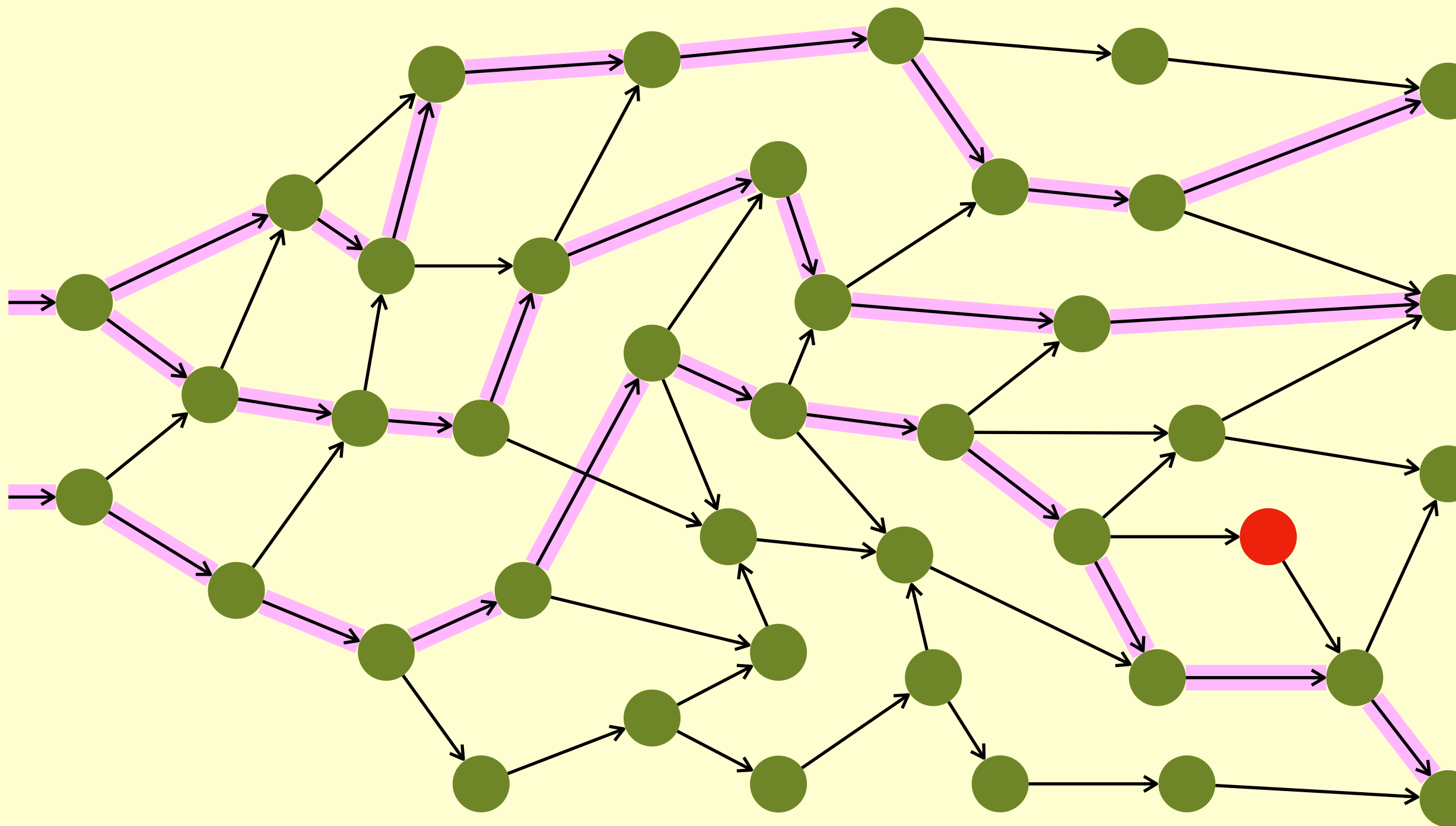
```
int main () {  
    double a = 4195835;  
    double b = 3145727;  
    printf ("%f\n", a/b);  
}
```



The correctness problem

- Computer systems are becoming more **complicated** and more **trusted**.
- This means that being confident that they are correct is increasingly **difficult** and increasingly **important**.
- Traditional **testing** is no longer enough, especially in our manycore era.
- Fortunately there are techniques and tools for **verifying** that a computer system is correct.

The problem with testing



The problem with testing

```
#include <stdlib.h>
#include <stdio.h>

int main(int argc, char **argv) {
    if (argc > 1 && atoi(argv[1]) == 4242) {
        printf ("KABOOM!\n");
        return 1;
    }
}
```

The problem with testing

```
int main() {  
    int x = 0, y = 0;  
    int r1 = 0, r2 = 0;  
  
    x = 1;    ||    r1 = y;  
    y = 1;    ||    r2 = x;  
  
    if (r1==1 && r2==0) {  
        printf ("KABOOM!\n");  
        return 1;  
    } else {  
        printf ("r1=%d, r2=%d\n", r1, r2);  
        return 0;  
    }  
}
```


Can we do better?

- Rather than testing on many *particular* inputs, construct a general argument for why the program is correct on *all* inputs.



"There is no solution to $a^n + b^n = c^n$ when $n > 2$."

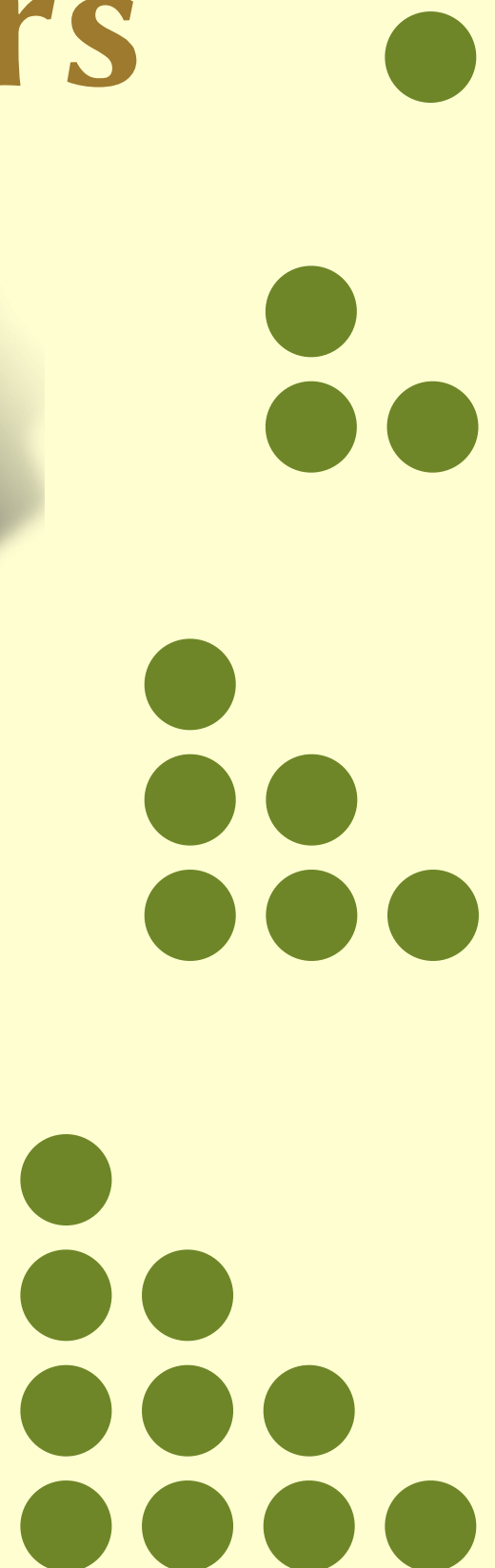


Triangle numbers

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

int triangle(int n) {
    int t = 0, i = 0;
    while (i < n) {
        i = i+1;
        t = t+i;
    }
    assert(t == n * (n+1) / 2);
    return t;
}

int main(int argc, char **argv) {
    int n = atoi(argv[1]);
    printf("%d\n", triangle(n));
}
```



before first iteration

```
assert(t == 0 && i == 0);
```



```
assert(t == i * (i+1) / 2);
```

arbitrary iteration

```
assert(t == i * (i+1) / 2);
```

```
assert(i < n);
```

```
i = i+1;
```

```
t = t+i;
```



```
assert(t == i * (i+1) / 2);
```

after last iteration

```
assert(t == i * (i+1) / 2);
```

```
assert(i == n);
```



```
assert(t == n * (n+1) / 2);
```

Dafny

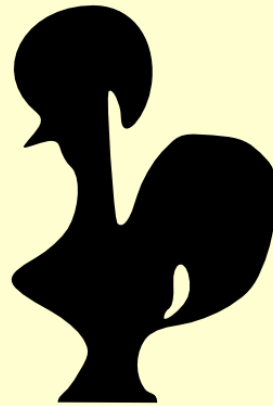
- Developed since 2008 by Rustan Leino and others at Microsoft Research, and nowadays mainly at Amazon.



Verifying with Dafny

```
method triangle(n:int) returns (t:int)
  requires 0 <= n
  ensures t == n * (n+1) / 2
{
  t := 0;
  var i := 0;
  while i < n
    invariant t == i * (i+1) / 2
    invariant i <= n
  {
    i := i+1;
    t := t+i;
  }
  return t;
}
```

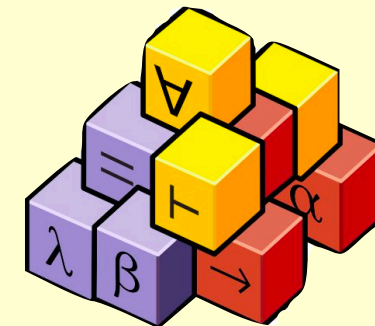
Interactive theorem proving



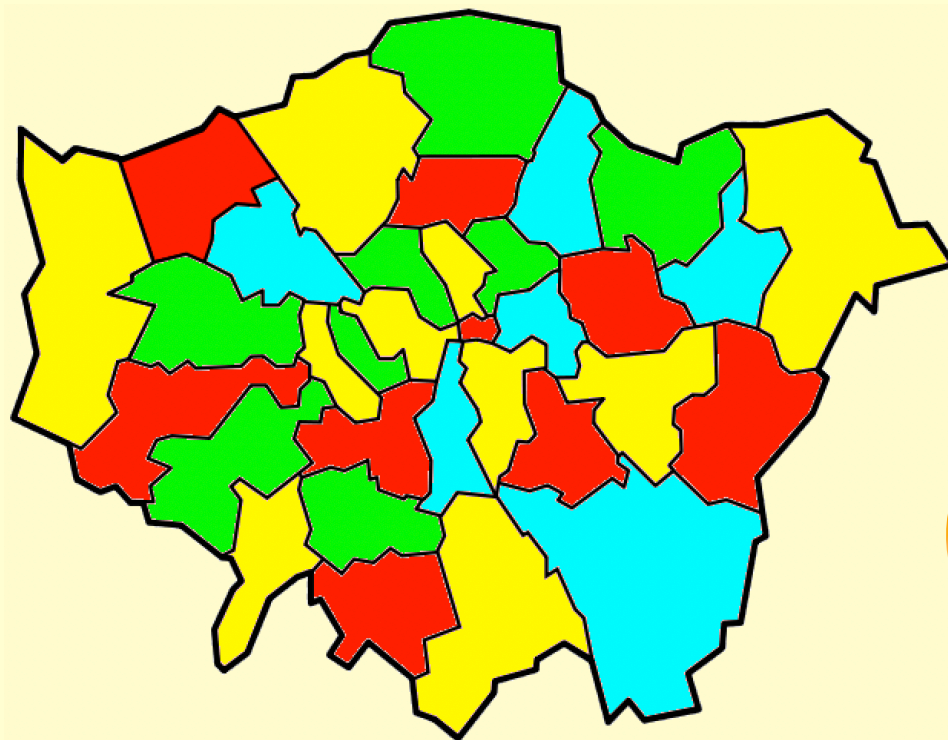
Coq



HOL

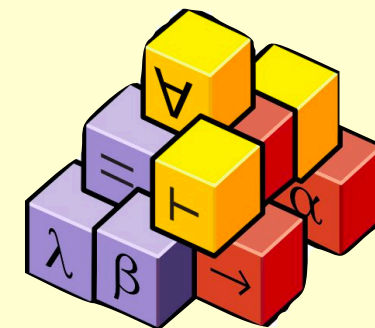


Isabelle

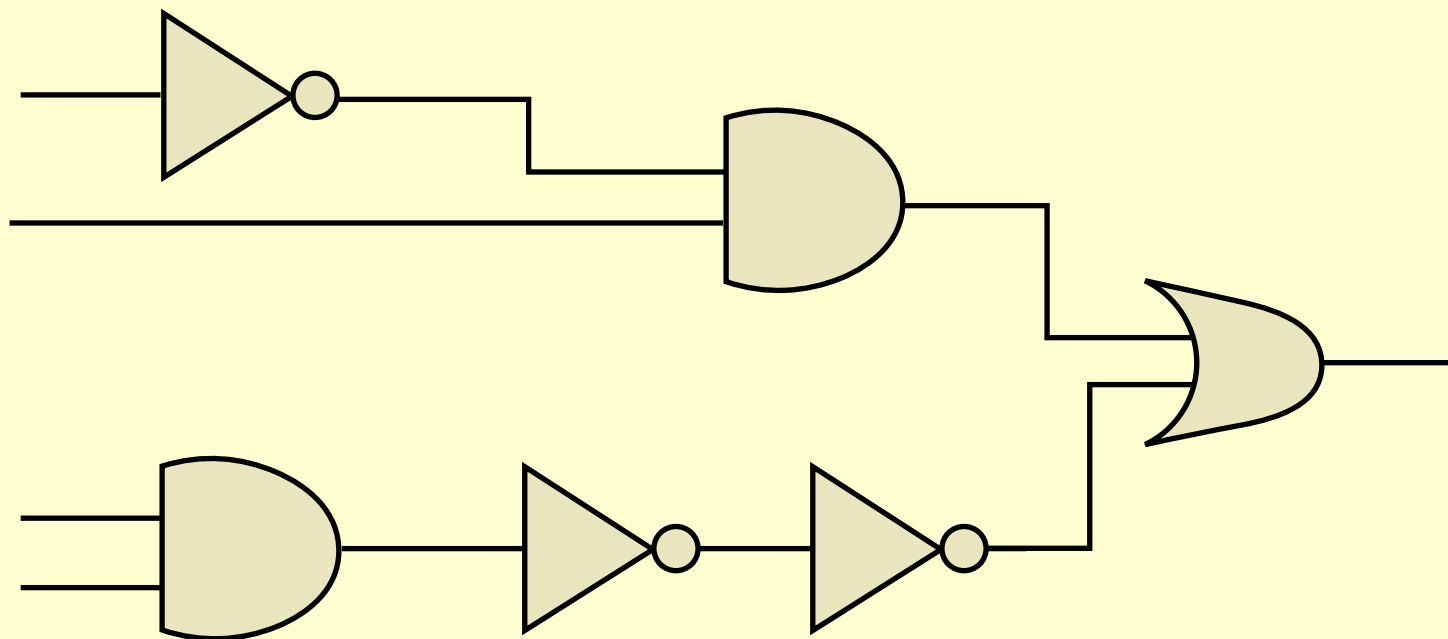


Interactive theorem proving

- *Your task:* prove a correctness theorem about a little logic synthesiser.



Isabelle



Computer-aided proof

Fully
manual

Fully
automatic

Coq

SymbiYosys

Facebook Infer

HOL

Dafny

Astrée

Isabelle

VCC

Model checking

Aims of the module

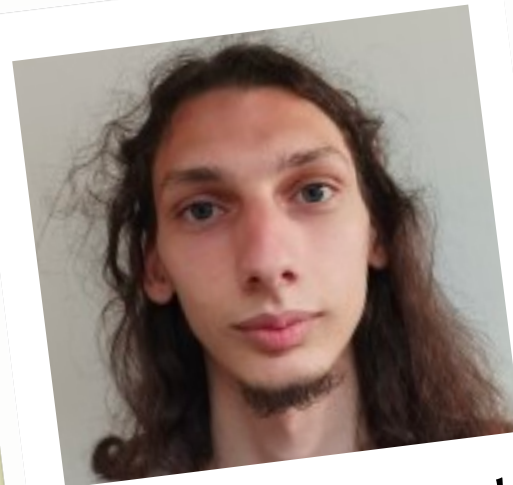
1. Be able to use Dafny to verify simple programs.
 - **Assessment:** I will give you several (increasingly difficult) programs, and you have to verify them using Dafny.
2. Be able to use Isabelle to conduct simple mathematical proofs.
 - **Assessment:** I will give you several (increasingly difficult) theorems, and you have to prove them using Isabelle.
3. Be able to use SymbiYosys to verify simple Verilog designs.
 - **Assessment:** I will give you several (increasingly difficult) Verilog designs, and you have to verify them using SymbiYosys.

Lecture plan

- Introduction
- Proving theorems using Isabelle
- Verifying software using Dafny
- Verifying hardware using SymbiYosys
- Principles of verification

Teaching support

- Lectures: 2pm every Thursday.
- Labs: 11am-1pm every Thursday.
- Work in pairs.
- Two teaching assistants:



Quentin Corradi



Michalis Pardalos

A first proof

- **Theorem.** $\sqrt{2}$ is irrational.

