

# Hardware & Software Verification

John Wickerson

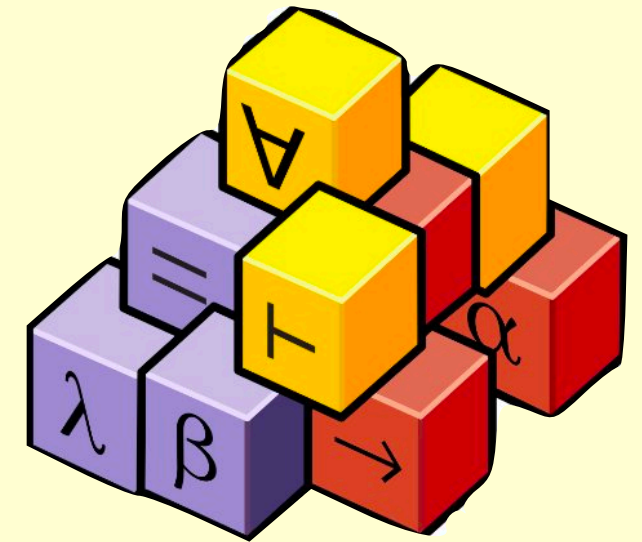
Lecture 2: Isabelle  
14 October 2025

# Lecture Outline

- Proving simple theorems by hand.
- Proving simple theorems using Isabelle.
- **Next lecture:** proving the correctness of a logic synthesiser.

# Isabelle

- Invented by Lawrence Paulson around 1986. Developed ever since at the University of Cambridge and at TU München.
- Has been used for large mathematical proofs, such as the Kepler conjecture.
- Has been used to build a verified operating system! The OS implementation is about 7.5k lines of C, the proof has about 200k steps, and it uncovered hundreds of bugs in the initial implementation.





# Observations

- Use `sorry` to skip a proof.
- Use `find_theorems` to search Isabelle's database of theorems.
- CTRL+click (or CMD+click) on a name to jump to its definition.
- Use `thm` to print out a theorem. Use `thm[of x]` or `thm[OF f]` to print out an instantiated theorem.
- Refer to facts using ``backticks`` or by naming them.
- Use `try` to invoke the Sledgehammer.

# A simple proof

- **Theorem.** There is no greatest even number.
- **Proof.** To show that the greatest number does *not* exist, we shall assume that it *does*, and deduce a contradiction. To this end, suppose there *is* a greatest even number, and call it  $n$ . But if  $n$  is even, then so is  $n+2$ , which is greater than  $n$ . This contradicts the assumption that  $n$  is the greatest even number. Therefore, the greatest even number does not exist.



# Observations

- Use `moreover..ultimately` to avoid labelling each fact.
- Isabelle proofs can use the "structured" style or the "procedural" style.
- The procedural style offers various low-level commands like `defer` and `prefer`, and low-level methods like `thin_tac` and `rename_tac`.
- There are a range of automated methods: `auto`, `simp`, `clarify`, `clarsimp`, `blast`, etc.



# Some constructions

- `fix` *<variable name>*
- `assume` *<new fact>*
- `have` *<new fact>* `by` *<method>*
- ~~`from this`~~<sup>hence</sup> `have` *<new fact>* `by` *<method>*
- `with` *<name of old fact>* `have` *<new fact>* `by` *<method>*
- `have` *<new fact>* `using` *<name of old fact>* `by` *<method>*
- `show` *<thesis>* `by` *<method>*
- ~~`from this`~~<sup>thus</sup> `show` *<thesis>* `by` *<method>*
- `moreover..ultimately`

# Meta vs Object logic

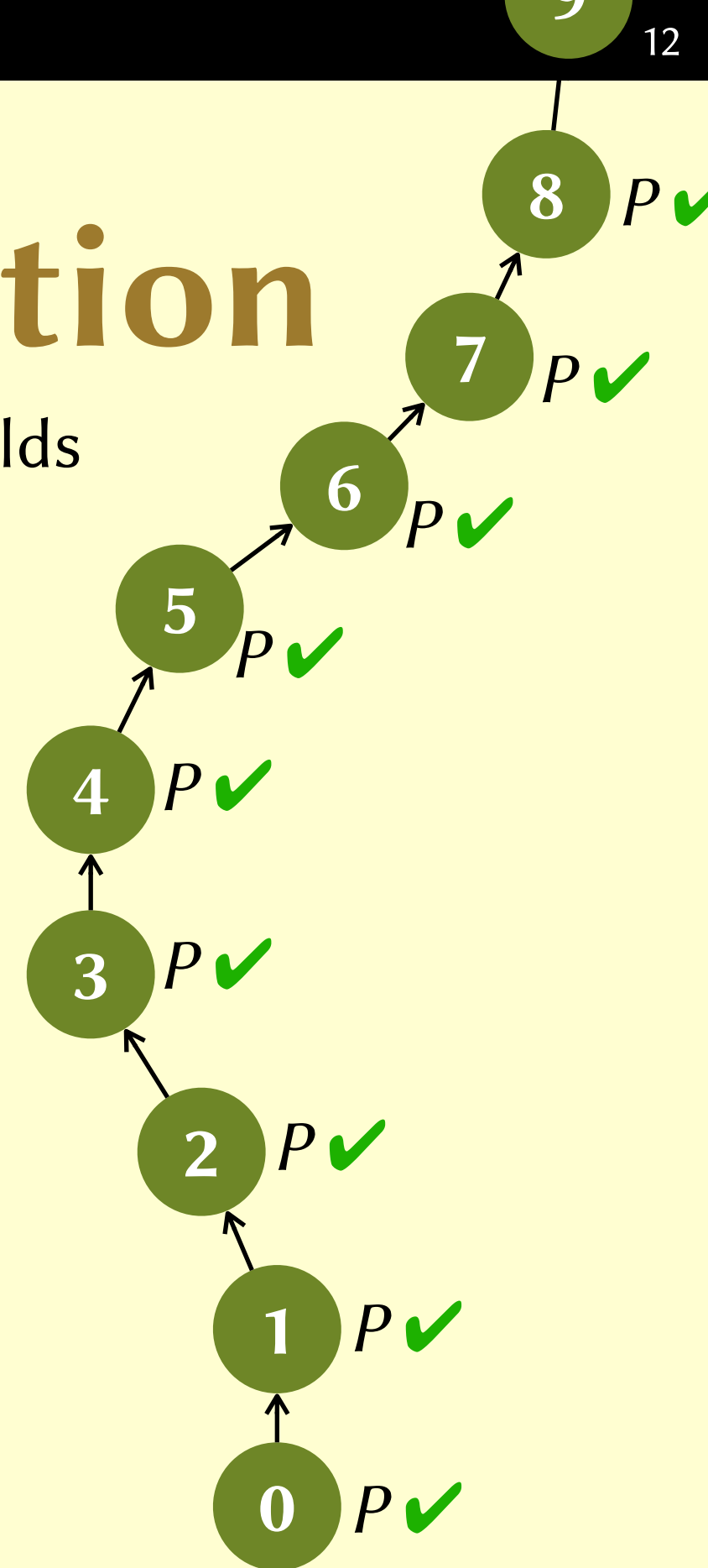
- This is the difference between making a judgement about a logical statement and the logical statement itself.
- Examples:
  - For every  $x$ , if it is the case that  $\text{even}(x)$  holds and it is the case that  $\text{odd}(x)$  holds then it is the case that  $x=0$  holds.
  - For every  $x$ , if it is the case that  $\text{even}(x) \wedge \text{odd}(x)$  holds then it is the case that  $x=0$  holds.
  - For every  $x$ , it is the case that  $(\text{even}(x) \wedge \text{odd}(x)) \rightarrow x=0$  holds.
  - It is the case that  $\forall x. (\text{even}(x) \wedge \text{odd}(x)) \rightarrow x=0$  holds.

# Meta vs Object logic

- This is the difference between making a judgement about a logical statement and the logical statement itself.
- Examples:
  - $\wedge x. \llbracket \text{even}(x); \text{odd}(x) \rrbracket \Rightarrow x=0$
  - $\wedge x. \text{even}(x) \wedge \text{odd}(x) \Rightarrow x=0$
  - $\wedge x. \text{even}(x) \wedge \text{odd}(x) \rightarrow x=0$
  - $\forall x. (\text{even}(x) \wedge \text{odd}(x)) \rightarrow x=0$

# Proof by induction

- Suppose we want to show that property  $P$  holds for all natural numbers.
- To do this, it suffices to prove two things:
  - $P$  holds for 0 (this is called the **base case**), and
  - for all  $k$ , if  $P$  holds for  $k$ , then  $P$  also holds for  $k+1$  (this is called the **inductive step**).



# Triangle numbers

**Define:**  $\text{triangle}(n) = \text{if } n=0 \text{ then } 0 \text{ else } n + \text{triangle}(n-1)$

**Theorem 1.**  $\text{triangle}(n) = (n+1)n/2$ .

**Proof.** We proceed by mathematical induction.

*Base case.*  $\text{triangle}(0) = 0 = (0+1)0/2$

definition of  $\text{triangle}$

algebraic manipulation

Thus  $\text{triangle}(0) = (0+1)0/2$ .

*Inductive step.* Pick arbitrary  $k$ .

Assume  $\text{triangle}(k) = (k+1)k/2$  as the *induction hypothesis* (IH).

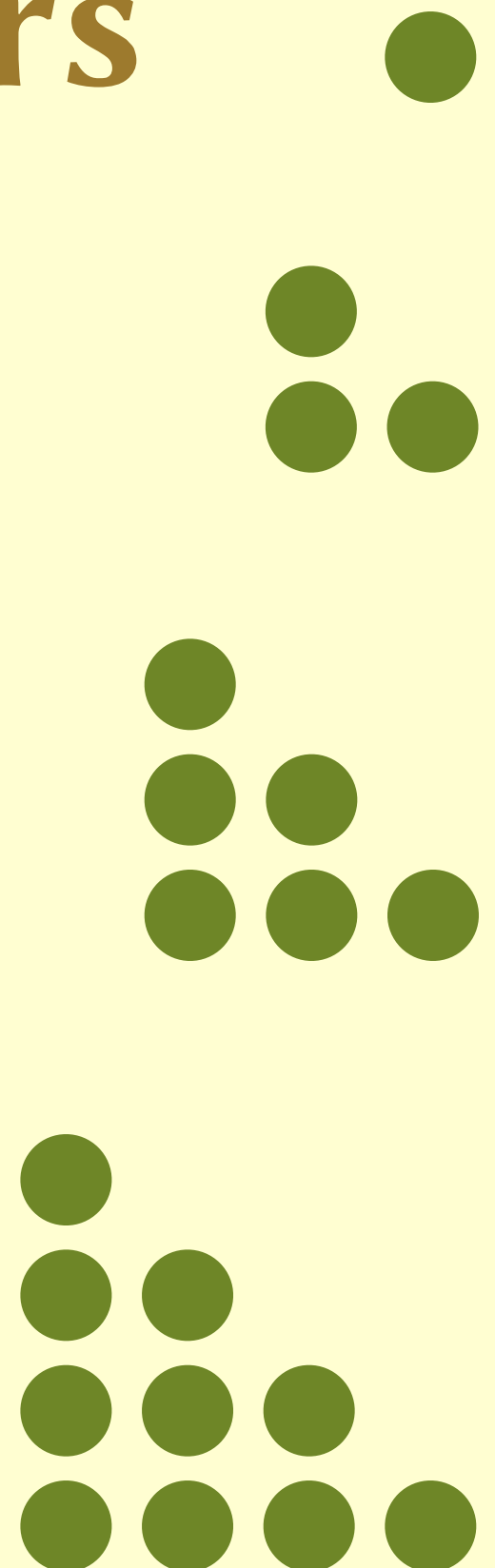
$\text{triangle}(k+1) = k + 1 + \text{triangle}(k) = k + 1 + (k+1)k/2 = (k+2)(k+1)/2$

definition of  $\text{triangle}$

IH

algebraic manipulation

Thus  $\text{triangle}(k+1) = (k+2)(k+1)/2$ .





# Tetrahedral numbers

**Define:**  $\text{tet}(n) = \text{if } n=0 \text{ then } 0 \text{ else } \text{triangle}(n) + \text{tet}(n-1)$

**Theorem 2.**  $\text{tet}(n) = (n+2)(n+1)n/6$ .

**Proof.** We proceed by mathematical induction.

*Base case.*  $\text{tet}(0) = 0 = (0+2)(0+1)0/6$

definition of  $\text{tet}$

algebraic manipulation

Thus  $\text{tet}(0) = (0+2)(0+1)0/6$ .

*Inductive step.* Pick arbitrary  $k$  and assume

$\text{tet}(k) = (k+2)(k+1)k/6$  as the induction hypothesis (IH).

$\text{tet}(k+1) = \text{triangle}(k+1) + \text{tet}(k) = (k+2)(k+1)/2 + \text{tet}(k) = (k+2)(k+1)/2 + (k+2)(k+1)k/6$

definition of  $\text{tet}$

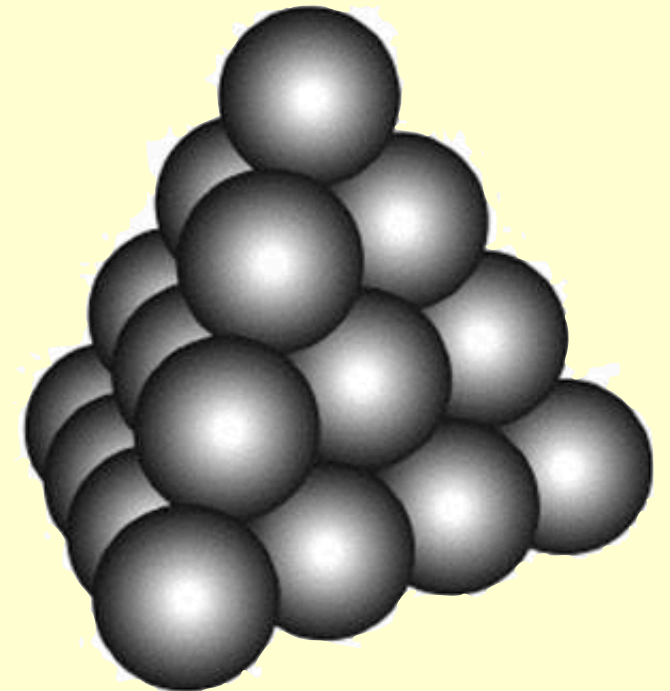
Theorem 1

IH

$= (k+3)(k+2)(k+1)/6$

algebraic manipulation

Thus  $\text{tet}(k+1) = (k+3)(k+2)(k+1)/6$ .







# Observations

- Use `also..finally` for chains of equational reasoning.
- Isabelle will provide a bare-bones induction proof for you when you type `proof (induct ...)`.
- Use `{` braces `}` to delimit the scope of a local assumption.

# Summary

- **This lecture:** how to conduct some basic proofs in Isabelle.
- **Next lecture:** How to implement a (small) logic synthesiser in Isabelle and verify that it is correct.