Computational Logic and the Quest for Greater Automation

Lawrence C Paulson, Distinguished Affiliated Professor for Logic in Informatics

Technische Universität München

(and Computer Laboratory, University of Cambridge)

Themes of This Lecture

I. Logic and the Real World

II.Computers, Logic and Mathematical Proofs

III. Achievements

IV. The Quest for Greater Automation

I. Logic and the Real World

The Two Forms of Logical Reasoning

- *Inductive* logic draws *general* conclusions from observations
  - It is the basis of science, and it concerns the real world
  - ... but it never gives an absolute YES or NO.
- *Deductive* logic draws *specific* conclusions by “pure reasoning” from *axioms*
  - It is the basis of mathematics
  - ... and is 100% certain, except for human error.

Chickens Can Reason Inductively!

The man is our friend!

...or is he?

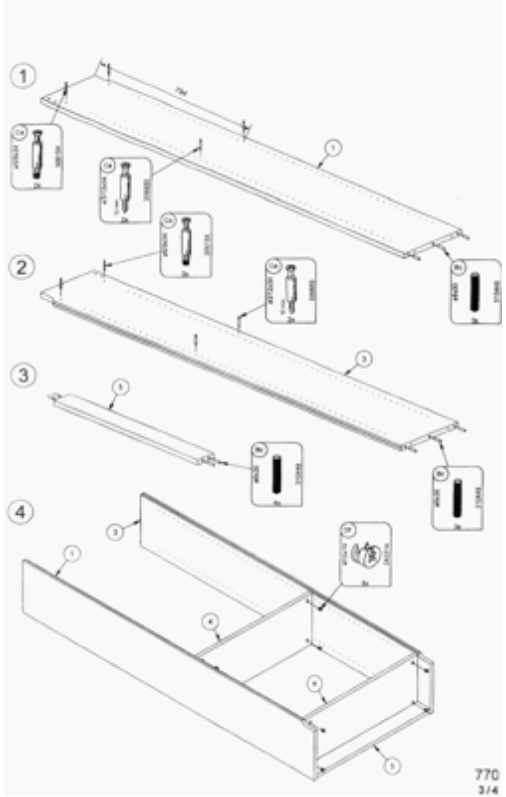
Can People Reason Deductively?

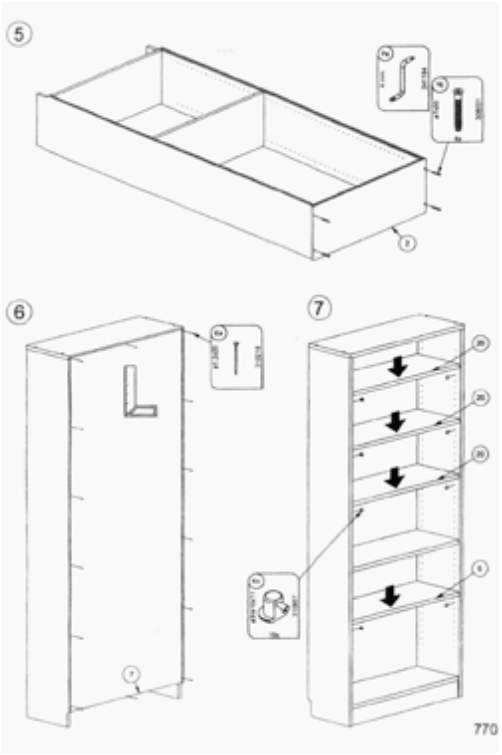
- A Sudoku has just one solution.
- An answer is right or wrong—the rules are simple and clear.
- Chickens can’t solve this, and neither can most people.
- However, as a logic problem, it’s trivial!

				5	1	4		
			7				2	
		8						3
	1							8
5				4				9
7							6	
4						7		
	2				6			
		3	9	8				

What Can Deduction Say About the Real World?

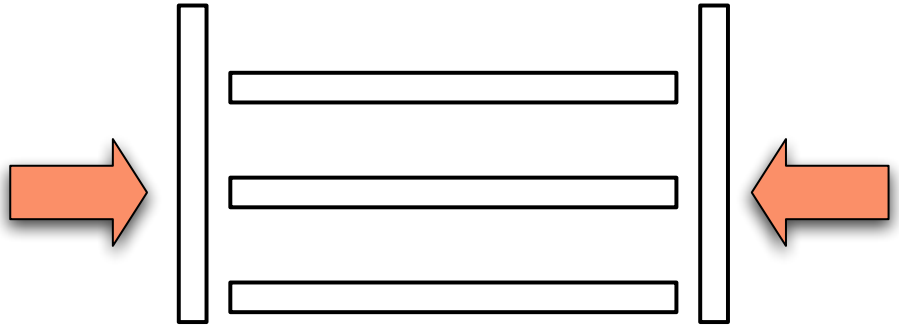
- Self-assembly furniture is like a Sudoku:
- millions of combinations, but only one solution.
- Chickens can't solve them, and neither can most people.
- Deduction can help solve real-world problems, if we can find the right *mathematical model*.

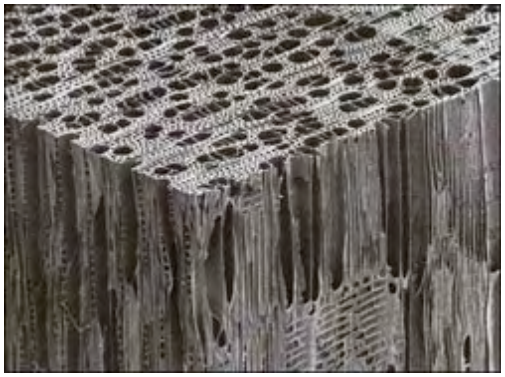




What is the Right Type of Model?

Too simple: deductions about this say nothing about how to build the real bookshelf.





Too complicated: do we really need to understand the fine structure of the wood?

A good model will identify the problem's important features.

II. Computers, Logic and Mathematical Proofs

Myth: “Computers Can’t Do Logic”

- Reason 1: Gödel proved it was impossible.

He didn’t.

- Reason 2: Church *did* prove it was impossible. (No computer can answer all problems in first-order logic.)

Computers can still help in many cases.

- Reason 3: Whitehead and Russell needed 362 pages to prove 1+1=2!

There are other ways of formalizing mathematics—
or, would you fly in a 100-year-old aeroplane?

A Formal Proof of 1+1=2 (Back in 1910)

$$*54{\cdot}42.\quad \vdash :: \alpha \in 2 . \supset :. \beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv . \beta \in \iota``\alpha$$

Dem.

$$\vdash . *54{\cdot}4 . \quad \supset \vdash :: \alpha = \iota`x \cup \iota`y . \supset :.$$

$$\beta \subset \alpha . \exists ! \beta . \equiv : \beta = \Lambda . \vee . \beta = \iota`x . \vee . \beta = \iota`y . \vee . \beta = \alpha : \exists ! \beta :$$

$$[*24{\cdot}53{\cdot}56.*51{\cdot}161] \qquad \equiv : \beta = \iota`x . \vee . \beta = \iota`y . \vee . \beta = \alpha \qquad (1)$$

$$\vdash . *54{\cdot}25 . \text{Transp} . *52{\cdot}22 . \supset \vdash : x \neq y . \supset . \iota`x \cup \iota`y \neq \iota`x . \iota`x \cup \iota`y \neq \iota`y :$$

$$[*13{\cdot}12] \quad \supset \vdash : \alpha = \iota`x \cup \iota`y . x \neq y . \supset . \alpha \neq \iota`x . \alpha \neq \iota`y \qquad (2)$$

$$\vdash . (1) . (2) . \supset \vdash :: \alpha = \iota`x \cup \iota`y . x \neq y . \supset :.$$

$$\beta \subset \alpha . \exists ! \beta . \beta \neq \alpha . \equiv : \beta = \iota`x . \vee . \beta = \iota`y :$$

$$[*51{\cdot}235] \qquad \equiv : (\exists z) . z \in \alpha . \beta = \iota`z :$$

$$[*37{\cdot}6] \qquad \equiv : \beta \in \iota``\alpha \qquad (3)$$

$$\vdash . (3) . *11{\cdot}11{\cdot}35 . *54{\cdot}101 . \supset \vdash . \text{Prop}$$

$$*54{\cdot}43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$$

Dem.

$$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$$

$$[*51{\cdot}231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$$

$$[*13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$$

$$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$$

$$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$$

$$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash . \text{Prop}$$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

What's In a Formal Logic?

- *Syntax*: a grammar for logical statements
- *Semantics*: a definition of what each grammar element means
- *Proof theory*: mechanisms for transforming problems into simpler problems (typically consisting of *axioms* and *inference rules*)

★ Computers handle the syntax and proof theory. The semantics is for us.

A Simple Formal Logic: Boolean Satisfiability

- A problem is a list of OR-statements. Can they all be true at the same time?

rainy | cloudy | sunny
–sunny | hot *“if sunny, then hot”*
–rainy | wet *“if rainy, then wet”*
–hot
–wet

- We conclude that it must be cloudy. A SAT-solver can handle problems 100,000 times this size.

- Is this logic trivial? No, it has endless applications, such as finding bugs in Microsoft device drivers—or solving Sudokus!

The Classic Formalism: First-Order Logic (FOL)

- for all X, Y and Z, if father(X,Y) and father(Y,Z) then grandfather(X,Z)

$$\forall X\ Y\ Z.\ father(X,Y) \,\&\, father(Y,Z) \rightarrow grandfather(X,Z)$$

- for all X, there exists y such that father(X,Y)

$$\forall X.\ \exists Y.\ father(X,Y)$$

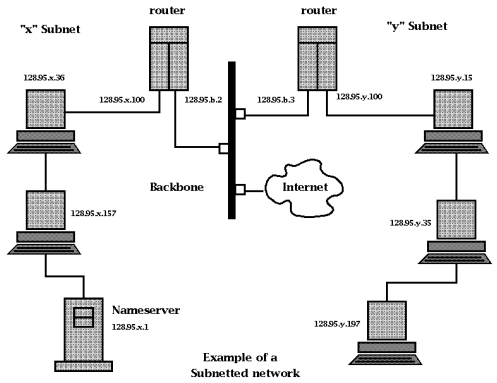
- Much of mathematics can be expressed in FOL.
- Powerful automatic provers exist. *(Unlike SAT-solvers, they are hardly used.)*

Limitations of Automatic Proof Tools

- If automatic software is so powerful, why do we need anything else?

Because it restricts us to small problems and simple models.

- With richer formalisms, we could model almost anything:
  - computer processors
  - networked systems
  - security environments
  - advanced mathematics

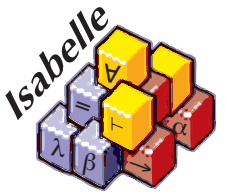


$$L_0 = 0$$
$$L_{\alpha+1} = \mathcal{D}(L_\alpha)$$
$$L_\alpha = \bigcup_{\xi<\alpha} L_\xi$$
$$\mathbf{L} = \bigcup_{\alpha\in\mathbf{ON}} L_\alpha$$

Interactive Theorem Proving

- Formal logics with deeper concepts: functions, sets, induction, recursion.
- Hierarchies of mathematical *books* (or “theories”):
  - Each book defines some concept, such as cryptography.
  - Books can build on other books, so developments can be huge.
- *Users* prove the theorems:
  - The software knows what proof steps are legal at a given point.
  - It helps by doing basic steps automatically.
  - We are constantly adding to this automation.

Isabelle: A Generic Interactive Prover

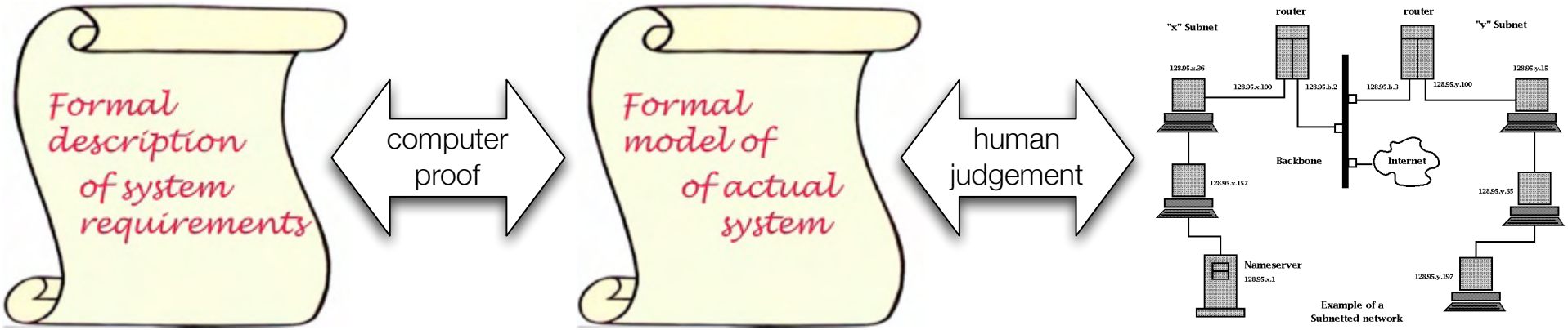


- *Generic* means the user can introduce new logical formalisms, in addition to the standard ones.
- The syntax and axioms can simply be listed.
- A general mechanism, called *higher-order unification*, combines separate proof steps.

- Over the years, researchers at TUM have given Isabelle...
  - an elaborate formalization of *higher-order logic;*
  - a structured language, *Isar,* for writing proofs in traditional style;
  - automatic typesetting of mathematical proofs.

III. Achievements

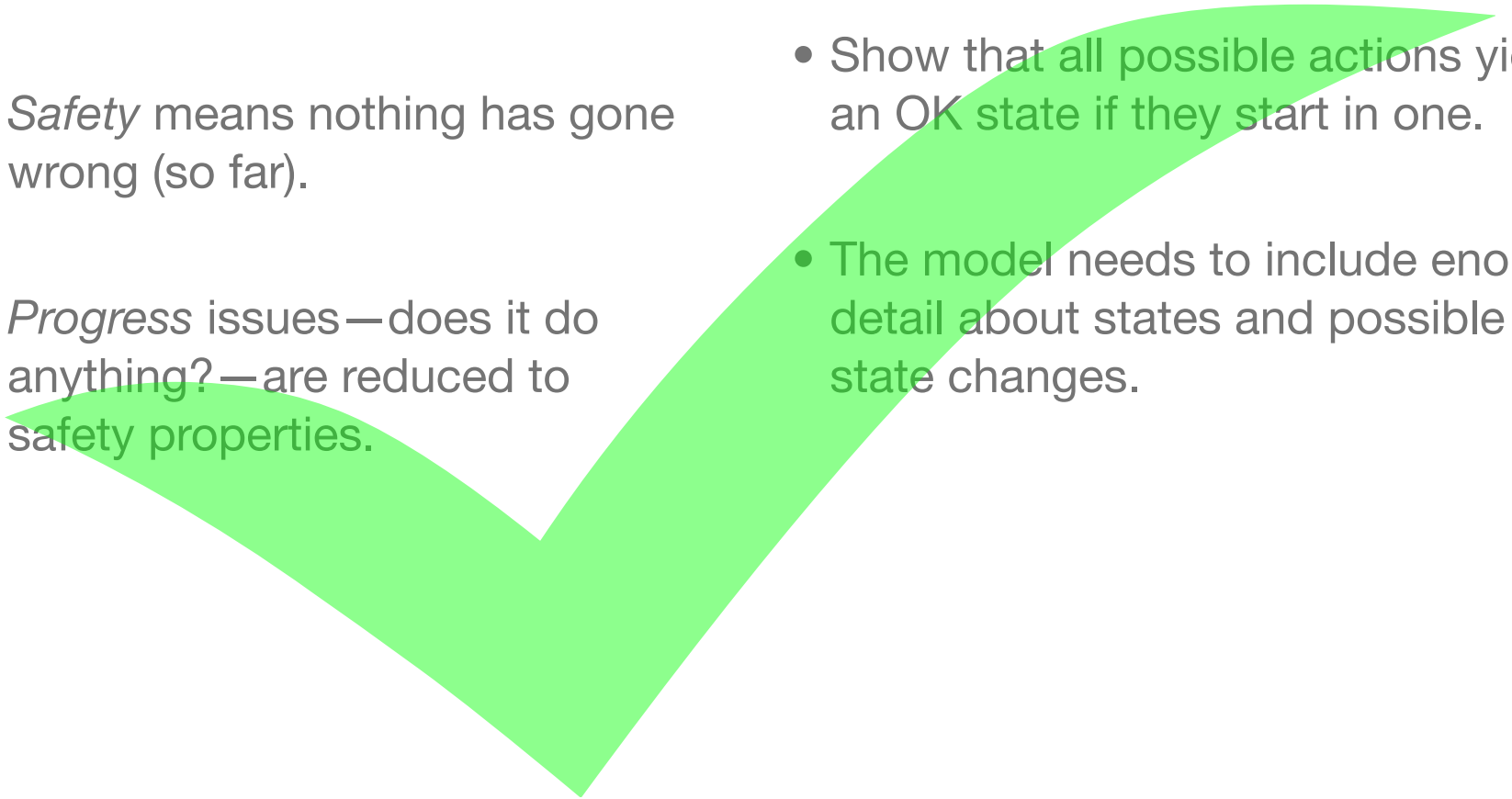
Verified Computer Systems



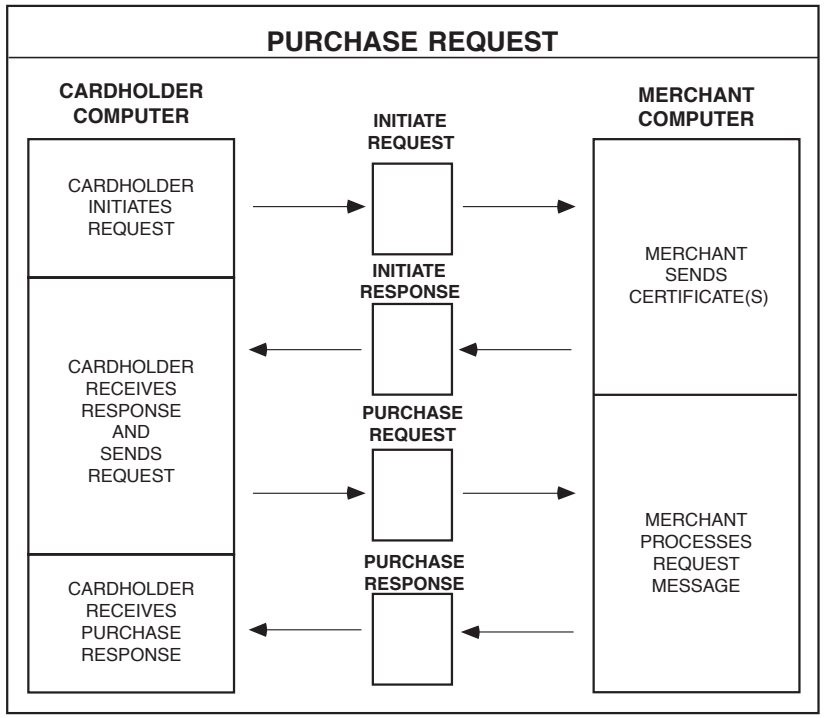
- *Goal*: to increase reliability through a mathematical proof of correctness
- Results are only as accurate as the formal model.
- Traditional testing guards against errors in the model.

How to Verify a Computer System

- Define what it means for a computation to be OK.
  - *Safety* means nothing has gone wrong (so far).
  - *Progress* issues—does it do anything?—are reduced to safety properties.
- Show all initial states to be OK.
- Show that all possible actions yield an OK state if they start in one.
- The model needs to include enough detail about states and possible state changes.



Verification Example: Security Protocols



These message exchanges use encryption to keep data *secret* while *authenticating* the remote computer

- *Whom* are you talking to really?
- Can a spy on the Internet trick your bank into revealing your details?
- In Isabelle, realistic models of protocols can be formalized.
- Industrial protocols such as SET (Secure Electronic Transaction) can be proved correct.

Verified Mathematics

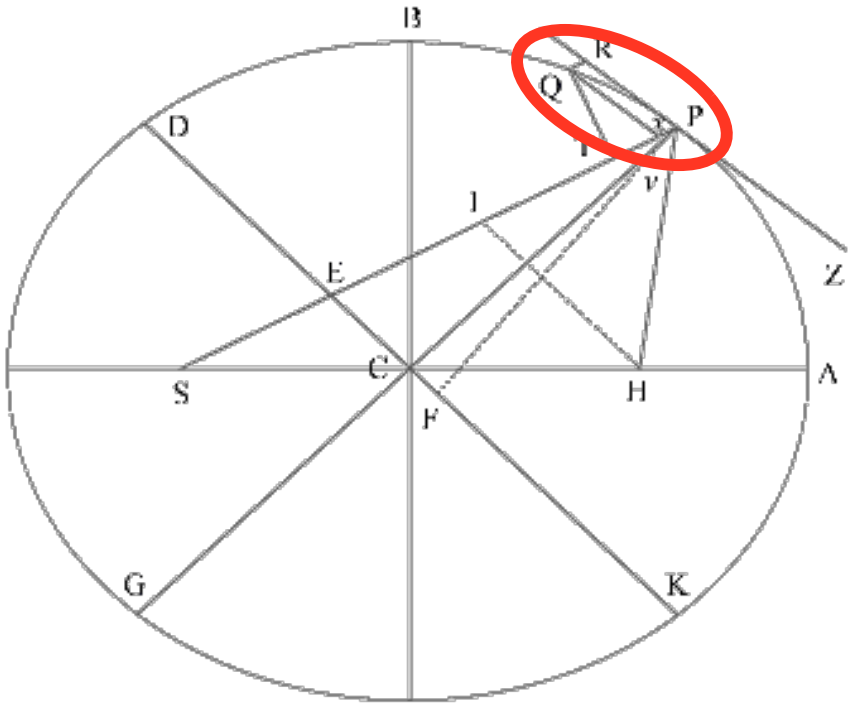
- Mathematical techniques are of unlimited variety and sophistication.
- Verifying known mathematics helps us improve our tools.
  - *Algebra* requires a flexible treatment of abstractions.
  - *Real analysis* requires special solvers for inequalities.
- Formalization finds exceptional cases and can yield historical insights.
- Known proofs of the Four Colour Theorem and of the Kepler Conjecture are too complicated for manual methods.

Isabelle Milestones in Verified Mathematics

- *Equivalents of the Axiom of Choice*: Grąbczewski (1996) verified two chapters of this famous book by Rubin and Rubin.
- Newton's *Principia*: Fleuriot (1998) combined geometry with non-standard analysis to formalize Newton’s logic and check some proofs.
- *The relative consistency of the axiom of choice*: was Gödel right to claim that his proof did not require meta-mathematical reasoning?
- The *prime number theorem*: Avigad (2004) formalized this landmark of number theory.
- *Tame graphs*: Nipkow and colleagues (2006) are contributing to the effort to prove the Kepler Conjecture formally.

Verifying Newton's *Principia*

- Newton's great book on motion and gravity did *not* use the calculus.
- In this proof of the inverse-square law, he merely asks what happens when "the points *P* and *Q* coincide."
- Fleuriot formalized Newton's *infinitesimal geometry* using non-standard analysis.
- He found an error in this proof, but found an alternative way to the result.

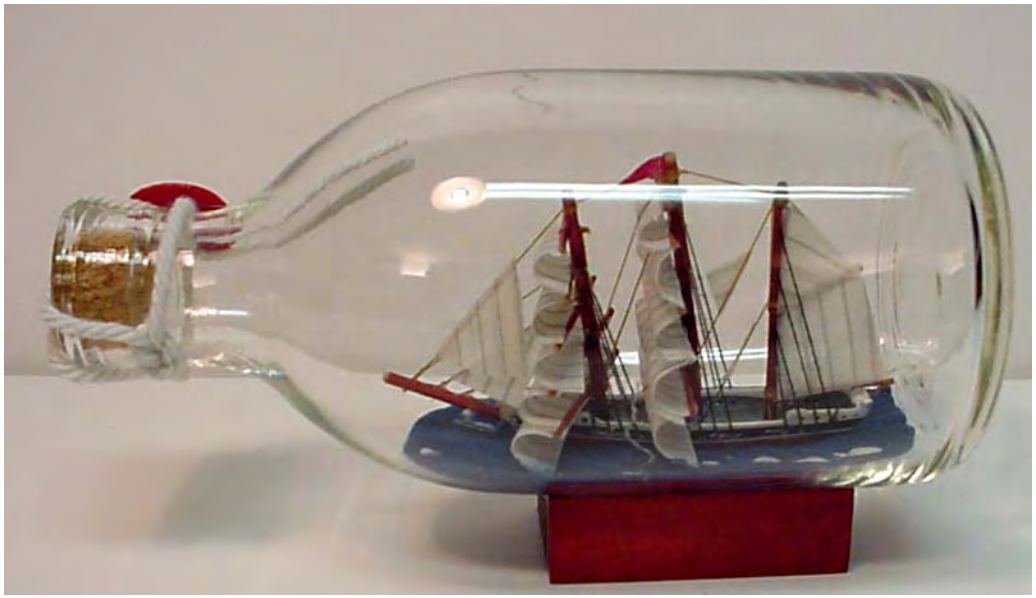


Proposition XI: a body in elliptical orbit

IV. The Quest for Greater Automation

Interactive Proof: What's the Catch?

- Proving theorems interactively is like building one of these.
- Even obvious facts can be difficult to prove.
- Legal proof steps are tiny, so the proofs are long.
- The work can be tiresome and frustrating...
- and only experts can do it.



Greater Automation: What's Been Done?

- Most proof tools can use equations like this one to simplify formulas.

$$x \neq 0 \Longrightarrow \frac{x}{x} = 1$$

- Isabelle can also use implications like these to search for proofs, using forward and backward chaining ...

$$x < y, y < z \Longrightarrow x < z$$

$$xz < yz, 0 \leq z \Longrightarrow x < y$$

- thanks to which it can prove complicated things automatically.

$$\bigcup_{i \in I} (A_i \cup B_i) = \left(\bigcup_{i \in I} A_i \right) \cup \left(\bigcup_{i \in I} B_i \right)$$

Still we need *more automation*!

Idea 1: Combine Isabelle with Automatic Provers

- The best automatic provers—E, SPASS, Vampire—do much more than chaining.
- To use them requires encoding Isabelle's rich formalism into the spartan language of first-order logic.
- They can beat Isabelle's built-in provers, but not always: they will require tuning before they become effective on problems generated by Isabelle.
- *Parallelism* can be exploited: we can call multiple provers, take the best result and record it permanently.

Idea 2: Proving Inequalities

- We often need to prove formulas involving functions like sin, cos, exp, log. There is no general solution procedure...

$$0 \leq x \leq 1 \implies e^x \leq 1 + x + x^2$$

- but often the function is bounded above or below by a *polynomial* (at least in a finite range).

- Inequalities involving polynomials can be solved using a procedure for *real closed fields.*

$$\left(1 + (-x) + \frac{(-x)^2}{2} + \frac{(-x)^3}{6}\right)^{-1} \leq 1 + x + x^2$$

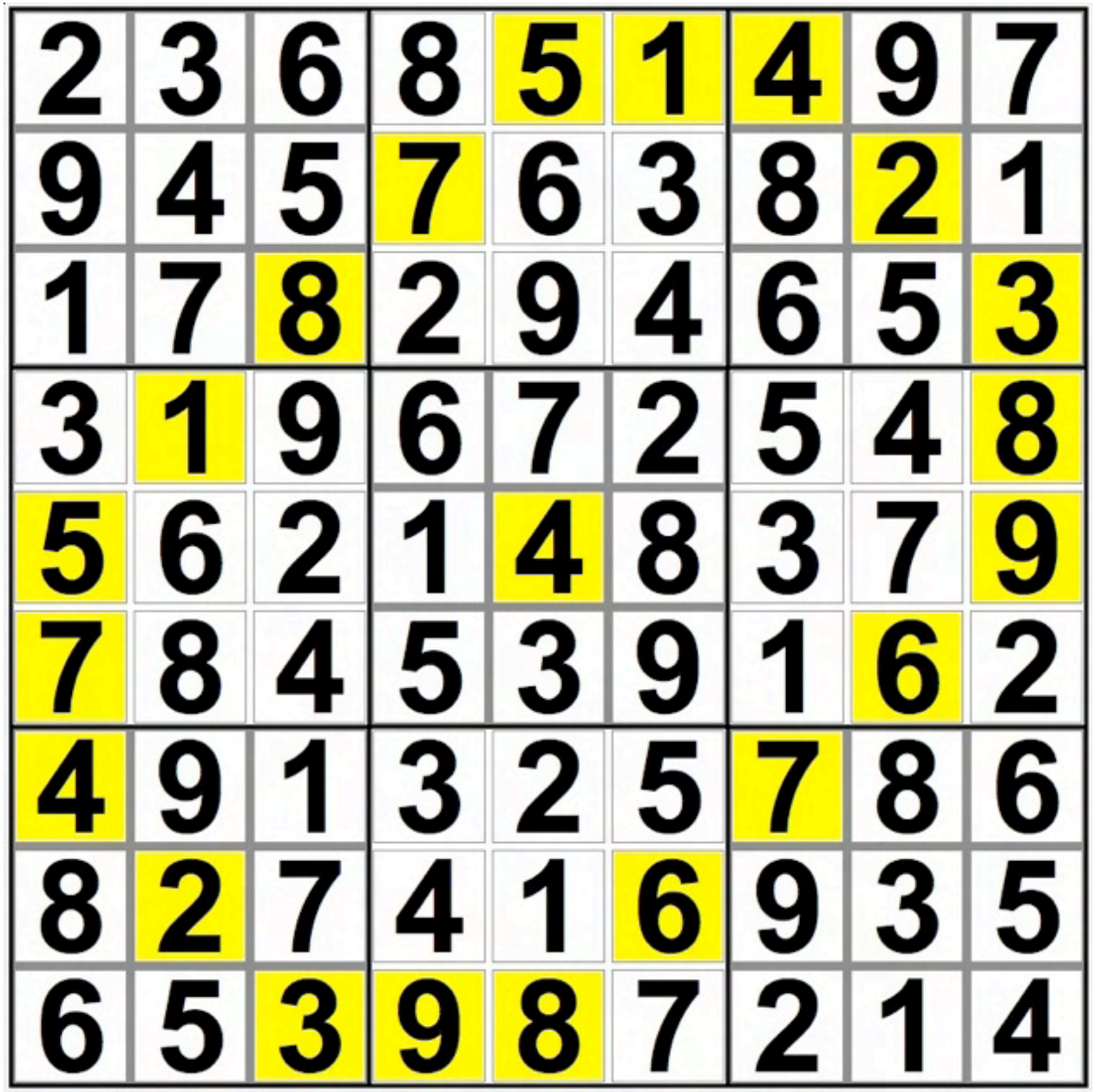
Idea 3: Telling the User When to Give Up

- Often our problem has no solution:
  - the chip design *isn't* correct, or
  - we have expressed it wrongly, or
  - forgot to mention essential facts.
- Trying to solve impossible problems is a terrible waste of time!

- People at TUM have built tools for warning Isabelle users of this situation.
- One way to do this is simply by testing the problem on a few values. (More complicated than it sounds!)

The Future?

- Proof technologies of all sorts have developed rapidly.
- In the past, we have benefitted from faster processors,
- ... and now we are ready to exploit the new multi-core computers.
- Large-scale trials are under way, such as the VeriSoft project,
- ... with other applications in the U. S. and Australia,
- ... and research projects in many countries.


2 3 6 8 5 1 4 9 7
9 4 5 7 6 3 8 2 1
1 7 8 2 9 4 6 5 3
3 1 9 6 7 2 5 4 8
5 6 2 1 4 8 3 7 9
7 8 4 5 3 9 1 6 2
4 9 1 3 2 5 7 8 6
8 2 7 4 1 6 9 3 5
6 5 3 9 8 7 2 1 4