Automatic Theorem Proving: Impressions from the *Interactive* World

Lawrence C Paulson / Computer Laboratory / University of Cambridge

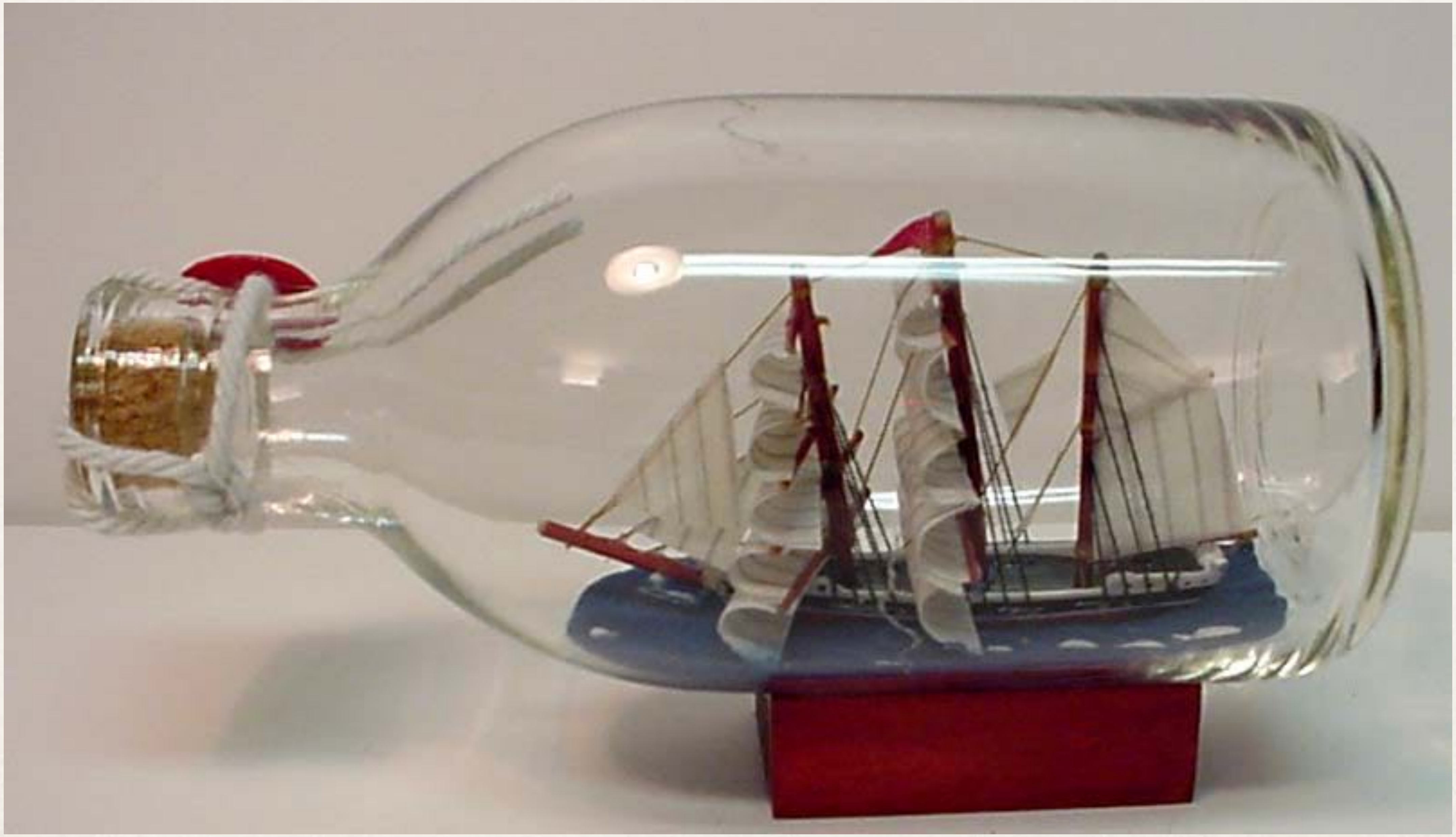
CADE-26, Gothenburg, 11 August 2017

The Great Divide

- *Automatic Theorem Provers*
  - Put in your conjecture and axioms
  - Full automation!
  - First-order logic (+T)
  - Careful about correctness

- *Interactive Proof Assistants*
  - Create big specification hierarchies
  - *You* do the hard work
  - Nice rich logics
  - *Neurotic about correctness*

But interactive proof is like building one of these…



So *everybody* wanted automation!

- LCF: conditional rewriting (as in Boyer/Moore, 1977!)
- PVS: various decision procedures, BDDs, etc (1995)
- HOL: decision procedures, resolution provers (1996–)
- Coq: decision procedures, reflection

Isabelle, in the beginning (1985)

Based on a higher-order logical framework, but with

- *unification* (even though it had to be higher-order)
- *backtracking* primitives via **lazy lists**

because I assumed these were
necessary for automation

- so, something like a higher-order Prolog

Sequent calculi in Isabelle (1986)

$$\frac{\Gamma_1, A[t/x], \Gamma_2 \Rightarrow \Delta}{\Gamma_1, \forall x\, A, \Gamma_2 \Rightarrow \Delta}$$

- using *associative unification* (via a higher-order trick) to support sequent calculus rules directly
- some automation using backtracking
- the equivalent of old-style “semantic tableaux”

A sequent calculus for *set theory*

It was easy to derive a proof calculus of high-level rules for set theory, and prove many facts automatically:

$$A \neq \emptyset \,\&\, B \neq \emptyset \quad \rightarrow \quad \bigcap(A \cup B) = (\bigcap A) \cap (\bigcap B)$$

$$C \neq \emptyset \quad \rightarrow \quad \bigcap_{x \in C}(A(x) \cap B(x)) = (\bigcap_{x \in C} A(x)) \cap (\bigcap_{x \in C} B(x))$$

(From a system description published at CADE-9 in 1988)

The push for more power

The discovery that this automation could make a difference in *real proof developments*

… and that it was far inferior to even *quite basic* automatic provers …

led to the perusal of *this paper*:

F. J. Pelletier, Seventy-five Problems for Testing Automatic Theorem Provers, *JAR* 2 (1986), 191–216

Pelletier's problem #43

$$\forall xy\,(\psi(x,y) \leftrightarrow \forall z\,(\phi(z,x) \leftrightarrow \phi(z,y)))$$
$$\rightarrow \forall xy\,(\psi(x,y) \leftrightarrow \psi(y,x))$$

requires a reasonably sophisticated treatment of quantifiers

Trivial? Not using sequent methods…

Time to try a *good* proof strategy?

M.E. Stickel. A Prolog technology theorem prover: implementation by an extended Prolog compiler. *JAR* 4 (1988), 353–380

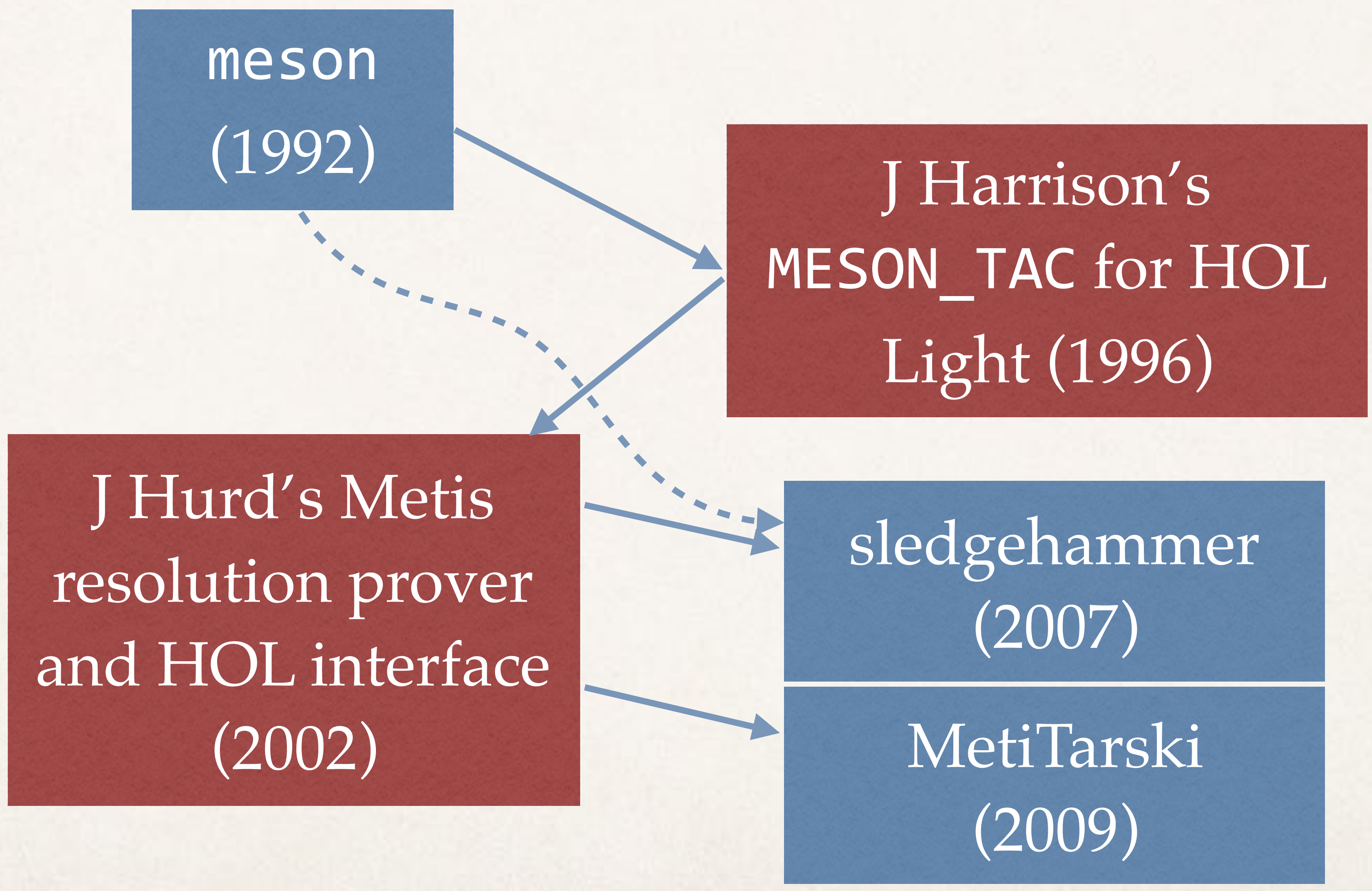
D.A. Plaisted. A sequent-style model elimination strategy and a positive refinement. *JAR* 6 (1990), 389–402

`meson`: The world's slowest model elimination theorem prover (1992)

- An obscure Isabelle tactic, inspired by Stickel's PTTP
- Runs on Isabelle's "Prolog" engine (**so no trust issues**)
- Far better than naive methods for first-order logic

But not generic — pure FOL only — so a dead end… ?

Spinoffs from Isabelle's ME tactic



Cryptographic protocol verification

- Based on operational semantics
- Inductive definitions and proofs in Isabelle
- Rewriting with respect to a formal theory of messages
- … followed by first-order reasoning (mainly *forward and backward chaining*)

Painful proofs
despite partial automation

… versus Ernie Cohen's TAPS

E. Cohen. TAPS: A first-order verifier for cryptographic protocols. *IEEE Comp. Security Foundations Workshop* (2000).

- *Automatic, deductive* verification of crypto protocols!
- Couldn't figure out how it worked *except*
  - everything was translated to FOL
  - … and proved using SPASS!

The key to better automation??

```
prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
    prove(A,[B|UnExp],Lits,FreeV,VarLim).
prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
    prove(A,UnExp,Lits,FreeV,VarLim),
    prove(B,UnExp,Lits,FreeV,VarLim).
prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
    \+ length(FreeV,VarLim),
    copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
    append(UnExp,[all(X,Fml)],UnExp1),
    prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).
prove(Lit,_,[L|Lits],_,_) :-
    (Lit = -Neg; -Lit = Neg) ->
    (unify(Neg,L); prove(Lit,[],Lits,_,_)).
prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
    prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```

leanT^A^P: simple; surprisingly good

B. Beckert & J. Posegga. leanT^A^P: Lean, tableau-based deduction. *JAR* 15 (1995), 339–358

It could prove Problem 43!

Could it be the inspiration for a better prover

… that was **still generic**?

The "blast" proof method (1998)

- Like leanT^A^P, but 1300 lines instead of 15
- *Generic*: forward and backward chaining without explicit quantifiers
- Runs in Standard ML; afterwards, successful proofs given to Isabelle's "Prolog" engine
- Now central to Isabelle's automation

But what about using *real* ATP in an interactive prover?

- Had been attempted many times (e.g. Ωmega, KIV)
- J Hurd: Integrating Gandalf and HOL (1999); Metis prover for the *ordered paramodulation calculus*

Joe Hurd. An LCF-style interface between HOL and first-order logic. In A. Voronkov, editor, CADE-18 (2002), 134–138.

Automation for interactive proof

Key technical problems

- **usability** for both novices and pros
- not burying the ATPs
- higher-order & types
- trust issues

Solutions

- ✓ 1-click invocation using *all* known facts
- ✓ relevance filtering
- ✓ a range of translations
- ✓ proof reconstruction

Sledgehammer: key points

Proofs are thrown away!
(*ATPs used as relevance filters*)

completely recoded at Munich by Blanchette et al

now the main source of resolution problems

that old "meson" method is still used for reconstruction

One more thing…

Gödel's incompleteness theorems

1. Every reasonable* formal calculus is *incomplete*: at least one formula can neither be proved nor disproved.

2. No reasonable formal system proves its own **consistency**.

**reasonable* = consistent and capable of expressing a certain amount of elementary arithmetic

Stages of the proofs

- The *syntax* of a first-order theory is formalised: terms, formulas, substitution…
- A *deductive calculus* for sequents of the form $\Gamma \vdash \alpha$ (typically for Peano arithmetic)
- *Meta-theory* to relate truth and provability. E.g. "all true Σ formulas are theorems". *(The set of Σ formulas is built using $\vee$ $\wedge$ $\exists$ and bounded $\forall$.)*
- A *system of coding* to formalise the calculus within itself. The code of α is a term, written $\ulcorner\alpha\urcorner$.
- *Syntactic predicates* to recognise codes of terms, substitution, axioms, etc.
- (and *correctness proofs* for them)
- Finally the *provability predicate* Pf, such that $\vdash \alpha \iff \vdash \mathrm{Pf}\ulcorner\alpha\urcorner$.

First incompleteness theorem

- Construct δ to express "δ is not provable" (¬Pf ⌜ δ ⌝).
- It follows (*provided* the calculus is consistent) that neither δ nor its negation can be proved, and that δ is true.
- Need to show that *substitution behaves like a function*.
  - Requires a lengthy, low-level proof in the calculus
  - [… or other intricate calculations, to do with bounded quantifiers]

Second incompleteness theorem

$$\text{If } \alpha \text{ is a } \Sigma \text{ sentence, then } \vdash \alpha \to \mathrm{Pf}\,\ulcorner\alpha\urcorner.$$

- A crucial lemma! Proved by induction over the construction of α *as a Σ formula*.
- It requires generalising the statement above to allow the formula α to contain free variables.
    - complex technicalities
    - lengthy deductions in the formal calculus

Defining the deductive calculus

```
inductive hfthm :: "fm set ⇒ fm ⇒ bool" (infixl "⊢" 55)
  where
    Hyp:    "A ∈ H ⟹ H ⊢ A"
  | Extra:  "H ⊢ extra_axiom"
  | Bool:   "A ∈ boolean_axioms ⟹ H ⊢ A"
  | Eq:     "A ∈ equality_axioms ⟹ H ⊢ A"
  | Spec:   "A ∈ special_axioms ⟹ H ⊢ A"
  | HF:     "A ∈ HF_axioms ⟹ H ⊢ A"
  | Ind:    "A ∈ induction_axioms ⟹ H ⊢ A"
  | MP:     "H ⊢ A IMP B ⟹ H' ⊢ A ⟹ H ∪ H' ⊢ B"
  | Exists: "H ⊢ A IMP B ⟹
             atom i ♯ B ⟹ ∀C∈H. atom i ♯ C ⟹ H ⊢ (Ex i A) IMP B"
```

Two dozen predicates formalising logical syntax

```
definition MakeForm :: "hf ⇒ hf ⇒ hf ⇒ bool"
where "MakeForm y u w ≡
      y = q_Disj u w ∨ y = q_Neg u ∨
      (∃v u'. AbstForm v 0 u u' ∧ y = q_Ex u')"
```

$y = u \vee w$, or $y = \neg\, u$, or $y = (\exists v)\, u$

with an *explicit* abstraction step on u

```
nominal_primrec MakeFormP :: "tm ⇒ tm ⇒ tm ⇒ fm"
where "⟦atom v ♯ (y,u,w,au); atom au ♯ (y,u,w)⟧ ⟹
  MakeFormP y u w =
    y EQ Q_Disj u w OR y EQ Q_Neg u OR
    Ex v (Ex au (AbstFormP (Var v) Zero u (Var au) AND y EQ Q_Ex (Var au)))"
```

The "official" version as a formula, not a boolean

Steps to the first theorem

- We need a function K such that $\vdash K(\ulcorner\phi\urcorner) = \ulcorner\phi(\ulcorner\phi\urcorner)\urcorner$
- … but we have no function symbols. Instead, define a relation, *KRP*:

  lemma *prove_KRP: "{} $\vdash$ KRP ⌜Var i⌝ ⌜A⌝ ⌜A(i::=⌜A⌝)⌝"*

- Proving that it *behaves like a function* takes 600 formal proof steps.

  lemma *KRP_unique: "{KRP v x y, KRP v x y'} $\vdash$ y' EQ y"*

- Finally, the *diagonal lemma*:

  lemma *diagonal:*
  obtains δ **where** *"{} $\vdash$ δ IFF α(i::=⌜δ⌝)" "supp δ = supp α - {atom i}"*

```
theorem Goedel_I:
  assumes Con: "¬ {} ⊢ Fls"
  obtains δ where "{} ⊢ δ IFF Neg (PfP ⌜δ⌝)"
                  "¬ {} ⊢ δ"  "¬ {} ⊢ Neg δ"
                  "eval_fm e δ"  "ground_fm δ"
proof -
  obtain δ where          "{} ⊢ δ IFF Neg ((PfP (Var i))(i::=⌜δ⌝))"
            and [simp]: "supp δ = supp (Neg (PfP (Var i))) - {atom i}"
    by (metis SyntaxN.Neg diagonal)
  hence diag: "{} ⊢ δ IFF Neg (PfP ⌜δ⌝)"
    by simp
  hence np: "¬ {} ⊢ δ"
    by (metis Con Iff_MP_same Neg_D proved_iff_proved_Pf)
  hence npn: "¬ {} ⊢ Neg δ" using diag
    by (metis Iff_MP_same NegNeg_D Neg_cong proved_iff_proved_Pf)
  moreover have "eval_fm e δ" using hfthm_sound [where e=e, OF diag]
    by simp (metis Pf_quot_imp_is_proved np)
  moreover have "ground_fm δ"
    by (auto simp: ground_fm_aux_def)
  ultimately show ?thesis
    by (metis diag np npn that)
qed
```

sledgehammer proofs!

Steps to the Second Theorem

- Coding must be generalised to allow *variables* in codes.
  - $\ulcorner x \lhd y \urcorner = \langle \ulcorner \lhd \urcorner, \ulcorner x \urcorner, \ulcorner y \urcorner \rangle$
  - $\lfloor x \lhd y \rfloor_V = \langle \ulcorner \lhd \urcorner, x, y \rangle$

codes of variables are integers

- Variable renaming is needed, with the aim of creating "pseudo-terms" like $\langle \ulcorner \lhd \urcorner, \mathrm{Q}\, x, \mathrm{Q}\, y \rangle$.
- Q is a magic "name of" function: $\mathrm{Q}\, x = \ulcorner t \urcorner$ where t is some canonical term denoting the set x.

One of the Final Lemmas

$$\mathrm{QR}(x,x'),\mathrm{QR}(y,y') \vdash x \in y \to \mathrm{Pf}\,\lfloor x' \in y' \rfloor_{\{x',y'\}}$$
$$\mathrm{QR}(x,x'),\mathrm{QR}(y,y') \vdash x \subseteq y \to \mathrm{Pf}\,\lfloor x' \subseteq y' \rfloor_{\{x',y'\}}$$
$$\mathrm{QR}(x,x'),\mathrm{QR}(y,y') \vdash x = y \to \mathrm{Pf}\,\lfloor x' = y' \rfloor_{\{x',y'\}}$$

- The first two require *simultaneous induction,* yielding the third.
- Similar proofs for the symbols ∨ ∧ ∃ and bounded ∀.
- The proof in the formal predicate calculus needs under 450 lines.

```
theorem Goedel_II:
  assumes Con: "¬ {} ⊢ Fls"
    shows      "¬ {} ⊢ Neg (PfP ⌜Fls⌝)"
proof -
  from Con Goedel_I obtain δ
    where diag: "{} ⊢ δ IFF Neg (PfP ⌜δ⌝)"  "¬ {} ⊢ δ"
      and gnd:  "ground_fm δ"
    by metis
  have "{PfP ⌜δ⌝} ⊢ PfP ⌜PfP ⌜δ⌝⌝"
    by (auto simp: Provability ground_fm_aux_def supp_conv_fresh)
  moreover have "{PfP ⌜δ⌝} ⊢ PfP ⌜Neg (PfP ⌜δ⌝)⌝"
    apply (rule MonPon_PfP_implies_PfP [OF _ gnd])
    apply (auto simp: ground_fm_aux_def supp_conv_fresh) using diag
    by (metis Assume ContraProve Iff_MP_left Iff_MP_left' Neg_Neg_iff)
  moreover have "ground_fm (PfP ⌜δ⌝)"
    by (auto simp: ground_fm_aux_def supp_conv_fresh)
  ultimately have "{PfP ⌜δ⌝} ⊢ PfP ⌜Fls⌝" using PfP_quot_contra
    by (metis (no_types) anti_deduction cut2)
  thus "¬ {} ⊢ Neg (PfP ⌜Fls⌝)"
    by (metis Iff_MP2_same Neg_mono cut1 diag)
qed
```

sledgehammer proofs!

Nearly 25% of the proof lines in the Gödel proof come from sledgehammer!

Where are we now?

we can use automation from the world's best ATPs

it's ***frequently*** successful, returning *surprising* proofs

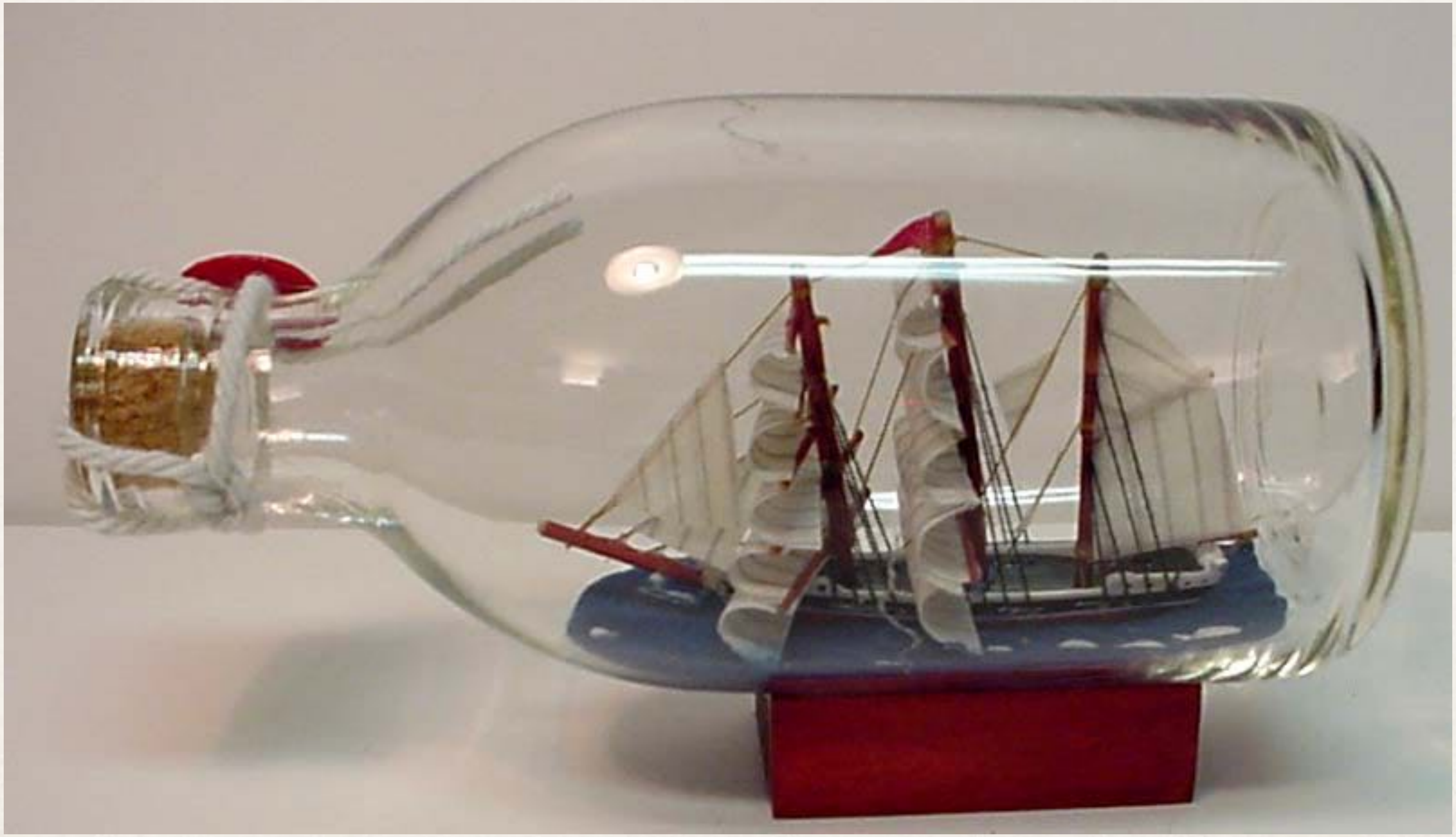
no longer need to understand the material, e.g. while porting 50,000 lines of HOL Light

Jordan curve theorem, Cauchy's integral formula

What's still needed?

- combined first-order logic + arithmetic reasoning
- automatic suggestions for *parts* of proofs
- higher-order reasoning

From this…



… to this!

Essential contributors

Tobias Nipkow

Strategic direction

- type system
- simplifier
- countless projects

Makarius Wenzel

- type classes
- structured proofs
- user interfaces
- multicore tech

Financial support from the UK's EPSRC

Thank You!