Theorem Proving in Isabelle/HOL

Matthew Doty

Outline

History

History — Logical Foundations I

1884	Gottlob Frege publishes *Die Grundlagen der Arithmetik*
1889	Giuseppe Peano axiomatizes first order arithmetic in *Arithmetices principia, nova methodo exposita*.
1900	David Hilbert presents 10 problems to the *International Congress of Mathematicians*, including his second problem: *Prove that the axioms of arithmetic are consistent*.
1901	Bertrand Russell finds a flaw in Frege's *Grundlagen*, writes a letter to Frege explaining what is now known as Russell's paradox
1903	Russell publishes *The Principles of Mathematics*. Appendix B alludes to a type hiearchy for resolving Russell's paradox. Russell and Alfred North Whitehead start collaborating on the *Principia Mathematica* shortly after.

History — Logical Foundations II

1908 Russell invents the theory of *ramified* types in his publication *Mathematical logic as based on the theory of types*.

1908 Zermelo axiomatizes set theory in *Untersuchungen über die Grundlagen der Mengenlehre*.

1910 Russell and Whitehead self-publish the *Principia Mathematica* after no publisher will print it. They take 379 pages to establish $1 + 1 = 2$ and write "The above proposition is occasionally useful."

1921 Leon Chwistek publishes *Zasady czystej teorii typów*, which criticizes Russell and Whitehead's ramified types and presents the theory of *simple* types for the first time. types.

History — Logical Foundations III

1924	Moses Schönfinkle writes *Über die Bausteine der mathematischen Logik*, where he develops *combinators* and defines logic in terms of combinator functions.
1926	David Hilbert writes *On the Infinite*, where he gives a failed proof of the continuum hypothesis. In doing so, he formulates the *axiom of choice* using an ϵ operator he invents.
1928	Hilbert authors *Grundzüge der theoretischen Logik*. This introduces the *entscheidungsproblem* (the “decision problem”), which asks if there is an algorithm for deciding for *any* mathematical statement whether it is true or false.

History — Logical Foundations IV

1930	Haskell Curry publishes *Grundlagen der Kombinatorischen Logik*, which presents (untyped) *combinator logic*. This extends Schönfinkle's work and attempts to lay a foundation for mathematics.
1931	Kurt Gödel publishes *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I*, or *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. The paper proves one theorem and makes a conjecture (ie, Gödel's *First* and *Second* incompleteness theorems respectively). Gödels theorem establishes that there is an *undecidable* proposition in *Principia Mathematica*, extended with the Peano axioms for arithmetic and *primitive recursive* functions. This provides a

History — Logical Foundations V

negative answer to Hilbert's *entscheidungsproblem*. He conjectures that "$1 \neq 2$" is undecidable (resolving Hilbert's second problem), and states he will prove so in a follow up paper. He never writes it.

1932 Alonzo Church publishes *A set of postulates for the foundation of logic*, which presents the untyped λ-calculus as a foundation for mathematics.

1933 Rosser demonstrates that Curry's combinator logic expresses the λ-calculus via the "*SKI*-embedding" in *A Mathematical Logic Without Variables. I.*

History — Logical Foundations VI

1934	Gödel and Jacques Herbrand extend primitive recursive functions to *general* recursive functions, providing a basis for computation. This development can be found in notes from Gödels' Princeton lectures from that period.
1934	Gerhard Gentzen develops the *sequent calculus* in *Untersuchungen über das logische Schließen. I*.
1935	Church writes to Gödel suggesting the λ-calculus as a basis for computation, and Gödel rejects his idea in favor of his own formulation. Gödel wrote to Church that idea of the λ-calculus as the basis for computation is “thoroughly unsatisfactory”.

History — Logical Foundations VII

1935 Kleene and Rosser publish *The inconsistency of certain formal logics*. This shows that there are terms for which logic developed in the untyped λ-calculus/combinator logic deduces are both *true* and *false* simultaneously. The untyped λ-calculus is abandoned as a foundation of mathematics.

1936 Curry publishes *Functionality in Combinatory Logic* and observes that inhabited types in simply-typed combinator logic correspond to theorems in the implicational fragement of intuitionistic logic.

History — Logical Foundations VIII

1936 Church and Rosser prove the *Church-Rosser theorem* in *Some properties of conversion*. As consequence of this is that if a term in the λ-calculus reduces to a normal form, that form is *unique*. This establishes the consistency of the functional fragment of the λ-calculus.

1936 Alonzo Church publishes *An Unsolvable Problem of Elementary Number Theory*, providing an alternate solution to the *entscheidungsproblem* to Gödels. He also demonstrates that Gödel's notion of recursive functions is equivalent to the untyped λ-calculus.

1936 Gerhard Gentzen proves that primitive recursive arithmetic with transfinite induction up to ϵ_0 is co-consistent with first order Peano arithmetic. This provides another perspective on Hilbert's second problem.

History — Logical Foundations IX

1937	Alan Turing devises *Turing Machines* as a foundation for computation in *On Computable Numbers, with an Application to the Entscheidungsproblem*. He proves no machine can compute the *halting problem*.
1937	Alan Turing proves *Turing Machines* and the λ-calculus have equivalent computational power in *Computability and λ-Definability*.
1938	Alan Turing is awarded his PhD for *Systems of Logic Based on Ordinals*.
1939	Hilbert and Bernays provide an *actual* proof of Gödel’s second incompleteness theorem in *Grundlagen der Mathematik*, vol. 2. Historians rarely acknowledge this and credit Gödel anyway.

History — Logical Foundations X

1940 Alanzo Church develops the *simply-typed* λ-calculus in *A Formulation of the Simple Theory of Types*. He provides the earliest formulation of *Higher Order Logic* and revives the research program of the *Principia Mathematica*. Church's logic contains forms of the axioms of *extensionality*, *infinity* and *choice*. Church uses ι in place of Hilbert's ϵ operator given in *On The Infinite* (1926).

Like Gentzen's arithmetical framework employing transfinite induction, Church's framework is powerful enough to prove the consistency of Peano arithmetic.

History — Logical Foundations XI

1942 Curry simplifies Kleene and Rosser's paradox from 1935 into what is now known as *Curry's Paradox*. Briefly, this is that logic formulated in the untyped λ-calculus will both prove and disprove $Y(\lambda x.\neg x)$, where $Y = \lambda f.\ (\ \lambda x.(f\ (x\ x))\ \lambda x.(f\ (x\ x))\ )$.

1955 Martin Löb adapts Curry's paradox to prove *Löb's Theorem* in Peano arithmetic. The theorem states:

$$\mathrm{PA} \vdash \mathrm{Bew}\ulcorner \mathrm{Bew}\ulcorner P \urcorner \to P \urcorner \to \mathrm{Bew}\ulcorner P \urcorner$$

An immediate consequence is Gödel's second incompleteness theorem: $\mathrm{PA} \nvdash \neg\mathrm{Bew}\ulcorner \bot \urcorner$

Proof. Suppose $\mathrm{PA} \vdash \mathrm{Bew}\ulcorner \bot \urcorner \to \bot$. Then $\mathrm{PA} \vdash \mathrm{Bew}\ulcorner \mathrm{Bew}\ulcorner \bot \urcorner \to \bot \urcorner$, hence $\mathrm{PA} \vdash \mathrm{Bew}\ulcorner \bot \urcorner$ by Löb's theorem and $\mathrm{PA} \vdash \bot$ ↯

History — Computer Proof I

1955	Findlay and Lambek remark on the connection between logic and category theory in the *Calculus of Bimodules*, but never publish.
1958	John McCarthy adapts Turing's *"Universal Turing Machine"* to the λ-calculus and calls it `eval` while writing *Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I*. Steve Russell reads his paper and implements his ideas on the IBM 704, creating *LISP 1* and the first REPL. The lisp functions CAR and CDR (**`head`** and **`tail`** in Haskell respectively) refer to the *"Content Address Register"* and the *"Content Decrement Register"* assembler macros for the 704.

History — Computer Proof II

1967	Nicolaas Govert de Bruijn invents *Automath*, one of the first computer proof checkers. *Automath* made use of the Curry-Howard isomorphism, although de Bruijn was not aware of Howard's work at the time. Theorem proving in *Automath* amounts to demonstrating type inhabitation using the λ-calculus.
1968	Joachim Lambek formally observes the isomorphism from intuitionistic logic to category theory in *Deductive systems and categories*.
1969	Howard remarks on the *Curry-Howard Isomorphism* in *The formulae-as-types notion of construction*, but doesn't publish until 1980.

History — Computer Proof III

1969	Dana Scott devises a novel framework for theorem proving using the λ-calculus in *A type-theoretical alternative to ISWIM, CUCH, OWHY*. However, he doesn't publish his manuscript until 1993.
1971	Per Martin-Löf introduces the first theory of dependent types in *An intuitionistic theory of types*. He does not publish.
1972	Jean-Yves Girard demonstrates that one can construct **`undefined`** `:: forall p . p`, in *impredicative* type theory. His construction implies all types are inhabited. Martin-Löf start to make ammedments to his logic.
1973	Per Martin-Löf revises his logic and publishes *An intuitionistic theory of types: predicative part.*

History — Computer Proof IV

1973 Robin Milner picks up Scott's idea, and revolutionizes functional programming:

- He writes ML (which stands for *"Meta Language"*) in LISP
- ML makes use of Hindley-Milner type inference
- Milner refines Scott's work is into *The Logic of Computable Functions* (LCF)
- LCF is implemented in ML. LCF is designed with a small *kernel* of axioms and rules, and while the rest of the system is untrusted.

History — Computer Proof V

- LCF makes use of ADTs for storing theorems rather than storing proofs. This is because Milner he kept running out of memory storing proofs. Except for axioms, operations for constructing theorems are operations on a *theorem* ADT.

Milner describes this work in *Models of LCF*.

1975 Diaconescu proves the axiom of choice implies the law of excluded middle in Topos theory in *Axiom of choice and complementation*.

1975 Andrzej Trybulec unveils *Mizar*, written in Algol 1204. This is the first *declarative* computer proof system. Mizar is based on first order logic and Zermelo-Fraenkel set theory. But unlike LCF, the Mizar proof system has the following flaws:

History — Computer Proof VI

- It has no small kernel of trusted computing. Instead it uses a number of ad-hoc decision algorithms for fragments of first order logic.
- It's closed source.

Rudnicki and Trybulec publish their work in *A Collection of TEXed Mizar Abstracts*

1976 Appel and Haken prove the *Four Color Theorem*, by enumerating 1,834 irreducible planar map configurations and providing a coloring for each. The proof takes a computer from that time over a thousand hours to verify. For over a decade after there are rumors of a flaw in their proof.

History — Computer Proof VII

1983 Michael Gordon embeds Church's original *Higher Order Logic* from 1940 into LCF with some simplifications. This creates the first HOL implementation. His intention is to formally verify hardware. He writes about his work in two publications: *A system for specifying and verifying hardware* and *Proving a computer correct*.

1989 Lawrence Paulson describes *Isabelle* in *The foundation of a generic theorem prover*. Isabelle is an LCF-type computer proof system written in Standard ML. Isabelle is strongly influenced by developments in *Prolog* from this period. It is originally designed for Martin-Löf's constructive type theory. Isabelle is ultimately adapted to Zermelo-Fraenkel set theory and Michael Gordon's HOL.

Pigeon Hole Proof Challenge

A Proof In Prose I

Theorem

Let L be a non-empty list of distinct, positive integers. There exists an $n \in L$ such that $n \geq |L|$ (ie, n is greater than or equal to the length of L.)

Proof. It suffices to prove that if all elements $n \in L$ are less than or equal to L then $|L|$ is a permutation of $\{1, 2, \ldots, |L|\}$. The proof proceeds by induction on the length of $|L|$.

When $|L| = 1$, it must be $L = \{1\}$ since L is a list of positive integers.

A Proof In Prose II

For the inductive step, if $|L| = k + 1$ then $L = \{n\} \cup L'$ for some positive n and L' where $|L'| = k$.

If $|L| \notin L'$ then for all $m \in L'$ we have $m < |L|$ by assumption. Hence $L' = \{1, 2, \ldots, |L'|\}$ by the inductive hypothesis. It must be then $n = |L|$ since the elements are distinct, and the result is obtained.

Otherwise if $|L| \in L'$, then $n < |L|$. However we can rewrite L as:

$$L = (\{n\} \cup (L' - \{|L|\})) \cup \{|L|\}.$$

By the inductive hypothesis $\{n\} \cup (L' - \{|L|\}) = \{1, 2, \ldots, |L'|\}$, so the result again obtains.

Q.E.D.

Overview of The Isabelle/HOL Proof

In Isabelle/HOL, our pigeon hole theorem can be expressed as:

```
⟦distinct (n # ns); ∀n'∈set (n # ns). 0 < n'⟧
⟹ ∃m∈set (n # ns). length (n # ns) ≤ m
```

The notation Isabelle/HOL uses corresponds to the following Haskell expressions:

Isabelle/HOL	Haskell
`n # ns`	`n : ns`
`length (n # ns)`	**`length`** `(n : ns)`
`x ∧ y`	`x && y`
`∀n'∈set (n # ns). p n'`	**`all`** `p (n : ns)`
`∃m∈set (n # ns). p m`	**`any`** `p (n : ns)`

Beginning The Isabelle Theory

```
theory Pigeon
  imports "~~/src/HOL/Library/Permutation"
begin
```

Lemma — Statement

To prove this theorem we first establish a lemma:

```
⟦distinct ns; ∀n∈set ns. 0 < n ∧ n ≤ length ns⟧ ⟹ ns ≃ [1..<length ns + 1]
```

Where:

- `[1..<m]` is equivalent to `[1..m]` in Haskell
- `ns ≃ [1..<m]` denotes that the list `ns` is a *permutation* of `[1..<m]`

The proof is a bit long. but showcases a number of features of Isabelle/HOL.

Lemma — Starting The Proof

```
lemma bounded_above_perm:
  fixes ns :: "nat list"
  assumes "distinct ns"
  and "∀ n ∈ set ns. 0 < n ∧ n ≤ length ns"
  shows "ns ≃ [1 ..< length ns + 1]"
using assms
proof (induct "length ns" arbitrary: ns)
```

Lemma — Base Case

```
    case 0
    then show ?case by auto
```

Note that the prose proof started with 1 rather than 0.

This is because it is more convenient to prove something even stronger than the sub-result in the prose proof.

Lemma — Inductive Step † Sub-Lemma I

```
next
  case (Suc m)
  have †: "    ∀ns' n. ns ≃ n # ns'
          ⟶ length ns ∉ set ns'
          ⟶ ns ≃ [1..<length ns + 1]"
  proof (rule allI, rule allI,
         rule impI, rule impI)
    fix n :: nat
    fix ns' :: "nat list"
    assume "ns ≃ n # ns'"
       and "length ns ∉ set ns'"
```

Lemma — Inductive Step † Sub-Lemma II

- `(`$\bigwedge$`x. P x)` $\Longrightarrow$ $\forall$`x. P x`

  Universal Quantifier Introduction — “If we can prove `P x` for any fixed `x`, then we can prove for all $\forall$`x. P x`”

  Note that $\bigwedge$ here is not the same as in System-F.

  For example, consider the System-F term $\Lambda\alpha.\lambda x^{\alpha}.x : \forall\alpha.\alpha \to \alpha$.

  Here $\Lambda\alpha.P\alpha$ is a type-level functional abstraction.

  On the other hand, the term $\bigwedge$`x. P x` involves a universal quantifier in Isabelle/Pure, the base logic for Isabelle/HOL (and other logics Isabelle supports).

Lemma — Inductive Step † Sub-Lemma III

- $(\mathtt{P} \Longrightarrow \mathtt{Q}) \Longrightarrow \mathtt{P} \longrightarrow \mathtt{Q}$

  Implication Introduction — “If we can deduce `Q` when assuming `P`, we can deduce $\mathtt{P} \longrightarrow \mathtt{Q}$”

Together these two rules allow us to introduce fixed variables and assumptions in order to prove

$\forall$`ns' n. ns` $\simeq$ `n # ns'` $\longrightarrow$ `length ns` $\notin$ `set ns'` $\longrightarrow$ `ns` $\simeq$ `[1 ..< length ns + 1]`

Lemma — † Sub-Lemma Proof I

```
    hence "∀n∈set ns'. 0 < n ∧ n ≤ m"
      by (metis Suc.hyps(2) Suc.prems(2)
                le_SucE notin_set_remove1
                perm_set_eq remove_hd)
    hence "ns' ≃ [1..<length ns]"
      by (metis ‹ns ≃ n # ns'›
                One_nat_def Suc.hyps(1)
                Suc.hyps(2) Suc.prems(1)
                Suc_inject distinct.simps(2)
                length_Cons list.size(4)
                perm_distinct_iff
                perm_length)
```

Lemma — † Sub-Lemma Proof II

The `metis` resolution based theorem prover.

It was written in ML [2], and ported to Isabelle/HOL.

Isabelle's kernel checks the proof `metis` generates to make sure they are correct.

Lemma — † Sub-Lemma Proof III

All of the theorems provided to `metis` are discovered by Isabelle/HOL's *Sledgehammer* tool [3, 4].

This tool transforms the current proof goal and theorems known to Isabelle into a form suitable for SMT solvers.

Proofs are reconstructed using `metis`, using the relevant theorems the solvers used to derive their proofs.

Note that results from earlier in the proof will be used along with library theorems. Earlier results are denoted by "$\langle \cdots \rangle$".

Recently a version of hammer has been ported to *Coq* [1].

Lemma — † Sub-Lemma Proof IV

```
have "n ∉ set ns'"
  by (meson ‹ns ≃ n # ns'›
            Suc.prems(1)
            distinct.simps(2)
            perm_distinct_iff)
hence "n ≥ length ns"
  by (metis ‹ns ≃ n # ns'›
            ‹ns' ≃ [1..<length ns]›
            One_nat_def
            Suc.prems(2) Suc_leI
            atLeastLessThan_iff
            leI list.set_intros(1)
            perm_set_eq set_upt)
```

Lemma — † Sub-Lemma Proof V

```
      hence "n = length ns"
        using ‹ns ≃ n # ns'›
              Suc.prems(2)
              perm_set_eq
        by force
      hence "n # ns' ≃ [1..<length ns + 1]"
        by (metis ‹ns' ≃ [1..<length ns]›
                  Suc.hyps(2) Suc_eq_plus1
                  Suc_eq_plus1_left
                  add.right_neutral
                  add_le_cancel_right
                  leI not_less0 perm.Cons
                  perm.trans perm_append_single
                  perm_sym upt_Suc)
      thus "ns ≃ [1..<length ns + 1]"
        using ‹ns ≃ n # ns'› by blast
    qed
```

Lemma — † Sub-Lemma Proof VI

Just as **`proof`** denotes the start of a proof-block, **`qed`** denotes the end.

We have now established

```
∀ns' n. ns ≃ n # ns' ⟶ length ns ∉ set ns' ⟶ ns ≃ [1..<length ns + 1]
```

...using the inductive hypothesis.

Lemma — Proof Conclusion By Case Reasoning I

```
    from Suc obtain n ns'
      where "ns = n # ns'"
      by (meson Suc_length_conv)
```

Just as the pattern

```
proof (rule allI, rule impI)
fix x
assume "P x"
```

…converts universal quantifiers into fixed variables with assumptions, existential quantifiers are converted with

```
obtain … where …
```

Lemma — Proof Conclusion By Case Reasoning II

```
    have "distinct ns'"
         "∀n∈set ns'. 0 < n ∧ n ≤ length ns"
         "length ns' = m"
      using Suc(2) Suc(3) Suc(4)
      unfolding ‹ns = n # ns'›
      by auto
    show ?case
    proof (cases "length ns ∈ set ns'")
```

Lemma — Proof Conclusion By Case Reasoning III

The proof proceeds in a *non-constructive* manner, checking the two possible cases wehre `length ns ∈ set ns'` and `length ns ∉ set ns'`.

Non-constructive proofs are often simpler than constructive proofs. Consider the statement “There exists two irrational numbers x and y such that x^y is rational.”

The non-constructive proof is simple. Let $x = y = \sqrt{2}$. We know $\sqrt{2}$ is irrational due to Hippasus the Pythagorean’s argument from circa 500 BC. If $\sqrt{2}^{\sqrt{2}}$ is rational we are done.
If not, let $x = \sqrt{2}^{\sqrt{2}}$ and $y = \sqrt{2}$. In this case we are done too since $(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = 2$.

Lemma — Proof Conclusion By Case Reasoning IV

For a more satisfying proof, consider $x = \sqrt{10}$ and $y = \log(4)$. Since $\sqrt{10}^{\log(4)} = 10^{\log(2)} = 2$ we are done.

But this constructive proof relies on defining logarithms, as well as two lemmas regarding the irrationality of $\sqrt{10}$ and $\log(4)$ respectively.

Formalizing it in Isabelle/HOL would be a lot more work!

Lemma — Proof Conclusion By Case Reasoning V

```
    case False
    thus ?thesis
      using † ‹ns = n # ns'› by blast
```

Lemma — Proof Conclusion By Case Reasoning VI

```
  next
    let ?ns'' = "remove1 (length ns) (n # ns')"
    case True
    hence "(n # ns') ≃ length ns # ?ns''"
      using perm_remove by force
    moreover have "length ns ∉ set ?ns''"
      using True Suc ‹ns = n # ns'› by simp
    ultimately show ?thesis
      using † ‹ns = n # ns'› by blast
  qed
qed
```

Lemma — Proof Conclusion By Case Reasoning VII

Here

```
remove1 :: 'a ⇒ 'a list ⇒ 'a list
```

...is akin to...

```
Data.List.delete :: Eq a ⇒ a → [a] → [a]
```

Main Proof I

```
lemma pigeon_hol:
  fixes n  :: nat
  fixes ns  :: "nat list"
  assumes "distinct (n # ns)"
      and "∀ n' ∈ set (n # ns). 0 < n'"
    shows "∃ m ∈ set (n # ns).
             length (n # ns) ≤ m"
proof (cases "∀ n' ∈ set (n # ns).
                n' ≤ length (n # ns)")
  case False
  then show ?thesis
    using nat_le_linear by blast
```

Main Proof II

```
next
  case True
  hence "(n # ns) ≃ [1..<length (n # ns) + 1]"
    using assms bounded_above_perm by metis
  moreover have
    "length (n # ns)
       ∈ set [1..<length (n # ns) + 1]"
    by (induct ns, auto)
  ultimately show ?thesis
    using perm_set_eq by blast
qed
```

Bibliography

[1] L. Czajka and C. Kaliszyk, *Hammer for Coq: Automation for Dependent Type Theory*, 61, pp. 423–453.

[2] J. Hurd, *First-order proof tactics in higher-order logic theorem provers*, in Design and Application of Strategies/Tactics in Higher Order Logics, Number NASA/CP-2003-212448 in NASA Technical Reports, pp. 56–68.

[3] J. Meng, C. Quigley, and L. C. Paulson, *Automation for interactive proof: First prototype*, 204, pp. 1575–1596.

[4] L. C. Paulson and K. W. Susanto, *Source-Level Proof Reconstruction for Interactive Theorem Proving*, in Theorem Proving in Higher Order Logics, K. Schneider and J. Brandt, eds., Lecture Notes in Computer Science, Springer, pp. 232–245.