A Machine-Assisted Proof of Gödel's Incompleteness Theorems

Lawrence C. Paulson, Computer Laboratory, University of Cambridge

The most misunderstood theorems in mathematics

- Gödel's theorems have highly technical, syntactic proofs.

  1. Every "strong enough" formal system is *incomplete*, in that at least one formula can neither be proved nor disproved.

  2. And if such a formal system admits a proof of its own consistency, then it is actually *inconsistent*.

- For the first time, *both* of Gödel's proofs have been mechanised, following a paper by Świerczkowski (2003)

- The machine proof, in the structured Isar language, is complete, almost readable, and can be perused interactively.

Hereditarily finite set theory

- A *hereditarily finite set* is a finite set of HF sets.
- Many mathematical constructions, including natural numbers and sequences, can be defined as in standard set theory.
- HF set theory is equivalent to Peano Arithmetic via the mapping

$$f(x) = \sum \{2^{f(y)} \mid y \in x\}$$

Benefits of Using HF Set Theory

- Can use standard definitions of pairing and sequences.
- The first incompleteness theorem requires an HF development of the natural numbers, induction, etc., but not addition.
- The second incompleteness theorem requires operations on sequences and addition, but not multiplication.
- No need for least common multiples, prime numbers or the Chinese remainder theorem.

The Axioms of HF Set Theory

$$z = 0 \leftrightarrow \forall x\,[x \notin z]$$

$$z = x \lhd y \leftrightarrow \forall u\,[u \in z \leftrightarrow u \in x \vee u = y]$$

$$\phi(0) \wedge \forall xy\,[\phi(x) \wedge \phi(y) \rightarrow \phi(x \lhd y)] \rightarrow \forall x\,[\phi(x)]$$

- 0 denotes the empty set
- $x \lhd y$ denotes the set x extended with the element y.
- There are **no other function symbols**.
- Union, intersection, etc can be shown to exist by induction.

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, but here HF)
- *Meta-theory* to relate truth and provability. E.g. "all true Σ formulas are theorems". Σ formulas are 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.
- Finally the predicate Pf, such that $\vdash \alpha \iff \vdash \mathrm{Pf} \ulcorner \alpha \urcorner$.

First Incompleteness Theorem

- To prove Gödel's *first* incompleteness theorem, construct δ that expresses that δ is not provable.
- It follows (*provided* the calculus is consistent) that neither δ nor its negation can be proved.
- Need to show that substitution behaves like a function.
  - Requires a detailed proof in the calculus,
  - … alternatively, other detailed calculations.

Second Incompleteness Theorem

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

- This crucial lemma for Gödel's *second* incompleteness theorem is 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 calculus

Proving Theorems in the Calculus

- Gödel knew that formal proofs were difficult. They could be eliminated, but at what cost?
- By coding all predicates as executable functions, and proving a meta-theorem, Gödel reduced provability to truth.
- But then only bounded quantifiers can be used, with tricky arithmetical proofs that the bounds are adequate.
- With Σ formulas, provability is reduced to truth for most desired properties, with no tricky proofs about bounds.
- Instead, some straightforward inductions need to be formalised in the calculus.
- The second theorem requires working in the calculus anyway.

Isabelle/HOL and Nominal

- a proof assistant for higher-order logic
- much automation to hide the underlying proof calculus
- support for recursive functions and inductive sets
- the *nominal package*, for working with named variables
- Free names are significant, but *not* once they are bound.
- Syntax involving variable binding can be defined using recursion, provided variables are used "sensibly".
- During proof by induction, bound variable names can be *guaranteed not to clash* with specified other terms.

De Bruijn Indexes

- This approach to variable binding replaces names by numbers.
- 0 denotes the innermost bound variable, 1 for the next, etc.
- This approach destroys readability, but substitution and abstraction are very easy to define.
- During coding, formulas are translated into the de Bruijn format.
- And so there is no need to formalise the nominal theory within HF.

Defining Terms and Formulas

```
nominal_datatype tm = Zero | Var name | Eats tm tm

nominal_datatype fm =
    Mem tm tm      (infixr "IN" 150)
  | Eq tm tm       (infixr "EQ" 150)
  | Disj fm fm     (infixr "OR" 130)
  | Neg fm
  | Ex x::name f::fm binds x in f
```

Variable binding formalised using nominal

Defining Substitution

```
nominal_primrec subst :: "name ⇒ tm ⇒ tm ⇒ tm"
   where
    "subst i x Zero         = Zero"
  | "subst i x (Var k)      = (if i=k then x else Var k)"
  | "subst i x (Eats t u) = Eats (subst i x t) (subst i x u)"
```

```
nominal_primrec  subst_fm :: "fm ⇒ name ⇒ tm ⇒ fm"
  where
    Mem:  "(Mem t u)(i::=x)  = Mem (subst i x t) (subst i x u)"
  | Eq:   "(Eq t u)(i::=x)   = Eq  (subst i x t) (subst i x u)"
  | Disj: "(Disj A B)(i::=x) = Disj (A(i::=x)) (B(i::=x))"
  | Neg:  "(Neg A)(i::=x)    = Neg (A(i::=x))"
  | Ex:   "atom j ♯ (i, x) ⟹ (Ex j A)(i::=x) = Ex j (A(i::=x))"
```

The variable *j* must be fresh for *i* and *x*

Properties of substitution have simple proofs.

Defining the HF 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"
```

The variable *i* must be fresh for *B* and *H*

Early Steps in the HF Calculus

- the *deduction theorem* (yielding a sequent calculus)
- *derived rules* to support explicit formal proofs
  - for defined connectives, including $\wedge \rightarrow \forall$
  - for equality, set induction, …
- definitions and proofs for *subsets*, *extensionality*, *foundation* and natural number *induction*

Σ Formulas

Strict Σ formulas only contain variables and are the basis for the main induction of the second incompleteness theorem. We can still derive the general case of Σ formulas.

```
inductive ss_fm :: "fm ⇒ bool" where
    MemI:  "ss_fm (Var i IN Var j)"
  | DisjI: "ss_fm A ⟹ ss_fm B ⟹ ss_fm (A OR B)"
  | ConjI: "ss_fm A ⟹ ss_fm B ⟹ ss_fm (A AND B)"
  | ExI:   "ss_fm A ⟹ ss_fm (Ex i A)"
  | All2I: "ss_fm A ⟹ atom j ♯ (i,A) ⟹ ss_fm (All2 i (Var j) A)"
```

```
"Sigma_fm A ⟷ (∃B. ss_fm B & supp B ⊆ supp A & {} ⊢ A IFF B)"
```

```
theorem "⟦Sigma_fm A; ground_fm A; eval_fm e0 A⟧ ⟹ {} ⊢ A"
```

True Σ formulas are theorems!

Coding Terms and Formulas

- must first translate from nominal to de Bruijn format
  - the actual coding is a simple recursive map:
  - $\ulcorner 0 \urcorner = 0$, $\ulcorner x_k \urcorner = k$, $\ulcorner x \lhd y \urcorner = \langle \ulcorner \lhd \urcorner, \ulcorner x \urcorner, \ulcorner y \urcorner \rangle$, …
- also define (in HF) *predicates to recognise codes*
  - *abstraction* over a variable — needed to define Form(x), the predicate for formulas
  - *substitution* for a variable

Example: Making a Formula

```
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 an HF formula, not a boolean

Those Coding Predicates

SeqTerm Term SeqConst Const

SeqStTerm AbstTerm SubstTerm

AbstAtomic SeqAbstForm AbstForm

SubstAtomic SeqSubstForm SubstForm

Atomic MakeForm SeqForm Form

VarNonOccTerm VarNonOccAtomic SeqVarNonOccForm VarNonOccForm

… And Proof Predicates

Sent Equality_ax HF_ax Special_ax Induction_ax

Axiom ModPon Exists Subst

Prf (a sequence of proof steps, and finally…)

Pf

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 $\mathtt{\mathit{prove\_KRP}}$: $\mathtt{"\{\} \vdash KRP\ \ulcorner Var\ i\urcorner\ \ulcorner A\urcorner\ \ulcorner A(i::=\ulcorner A\urcorner)\urcorner"}$

- Proving its functional behaviour takes 600 HF proof steps.

  lemma $\mathtt{\mathit{KRP\_unique}}$: $\mathtt{"\{KRP\ v\ x\ y,\ KRP\ v\ x\ y'\} \vdash y'\ EQ\ y"}$

- Finally, the diagonal lemma:

  lemma *diagonal*:
  obtains δ **where** $\mathtt{"\{\} \vdash \delta\ IFF\ \alpha(i::=\ulcorner\delta\urcorner)"}$ $\mathtt{"supp\ \delta = supp\ \alpha - \{atom\ i\}"}$

```
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
```

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

- Variables must be renamed, with the intent of creating "pseudo-terms" like $\langle \ulcorner \lhd \urcorner, \mathrm{Q}\,x, \mathrm{Q}\,y \rangle$.
- Q is a magic function: $\mathrm{Q}\,x = \ulcorner t \urcorner$ where t is some canonical term denoting the set x.

Complications

- Q must be a *relation*.
  - Function symbols cannot be added…
  - Sets do not have an easily defined canonical ordering.
- QR(0,0)
- $\mathrm{QR}(x,x'), \mathrm{QR}(y,y') \Longrightarrow \mathrm{QR}(x \lhd y, \langle \ulcorner \lhd \urcorner, x', y' \rangle)$

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 HF calculus needs under 450 lines.
- Fills a major gap in various presentations, including Świerczkowski's.

```
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
```

What Did We Learn?

- Some highly compressed proofs were finally made explicit.
- The entire proof development can be examined interactively.
- The nominal package can cope with very large developments…

  (BUT: performance issues, some repetitive notation, complications in accepting function definitions)
- <9 months for the first theorem, a further 4 for the second
- Under 16 000 lines of proof script in all.

Conclusions

- the first-ever machine formalisation of Gödel's second incompleteness theorem
- using both nominal and de Bruijn syntax for bound variables
- within an axiom system for hereditarily finite set theory
- conducted using Isabelle/HOL.

Many thanks to Christian Urban for help at critical points!

Also Jesse Alama, Dana Scott.