THE REVIEW OF SYMBOLIC LOGIC
Volume 0, Number 0, Month 2014

A Machine-Assisted Proof of Gödel's Incompleteness Theorems for the Theory of Hereditarily Finite Sets

LAWRENCE C. PAULSON
University of Cambridge

Abstract. A formalisation of Gödel's incompleteness theorems using the Isabelle proof assistant is described. This is apparently the first mechanical verification of the second incompleteness theorem. The work closely follows Świerczkowski (2003), who gave a detailed proof using hereditarily finite set theory. The adoption of this theory is generally beneficial, but it poses certain technical issues that do not arise for Peano arithmetic. The formalisation itself should be useful to logicians, particularly concerning the second incompleteness theorem, where existing proofs are lacking in detail.

§**1. Introduction.** Gödel's incompleteness theorems (Feferman, 1986; Gödel, 1931) are undoubtedly the most misunderstood results in mathematics. Franzén (2005) has written an entire book on this phenomenon. One reason is they have attracted the attention of a great many non-mathematicians, but even specialists who should know better have drawn unfounded conclusions. One of the main obstacles to understanding these theorems is the great technical complexity of their proofs, and indeed of their very statements.

Świerczkowski (2003) claims that the theory of hereditarily finite sets (HF) is more suitable than the usual Peano Arithmetic (PA) as a basis for proving the incompleteness theorems. The coding of terms and formulas can be done directly using traditional set-theoretic constructions, without referring to prime factorisation or the Chinese remainder theorem. As evidence, he gives a detailed presentation of the proofs of these theorems, along with a development of the HF theory itself. He also states a theorem saying that the theories HF and PA are definitionally equivalent.

The present paper describes a formalisation of Świerczkowski's development using the interactive theorem prover Isabelle/HOL. This formalisation makes some of the advantages and drawbacks of his approach very clear, and these will be discussed below. Moreover, the availability of this formal proof (which can be surveyed by anybody who has a suitable computer and a copy of the Isabelle software) can help to demystify the incompleteness theorems.

Boolos (1993) devotes more than two pages (pp. 33–34) to an explanation of how coding syntax using integers differs from using PA to reason about addition and multiplication. As a computer scientist, I do not see the need for such lengthy explanations: coding one thing in another is how computers work on every architectural level. Coming from that perspective, it isn't obvious that representing the ordered pair $\langle x,y\rangle$ set-theoretically as $\{\{x\},\{x,y\}\}$ is more natural than representing it arithmetically as 2^x3^y, for example. What we can objectively say is that the former approach is likely to save effort, eliminating the need to formalise the fundamental theorem of arithmetic or the Chinese remainder theorem explicitly in PA.

doi:

It's clear that Gödel regarded the need to construct explicit formal proofs as highly undesirable. We can regard the proof of a sentence A on three levels: informally, as a proof of $\vdash A$ in a suitable formal calculus, or as a proof of $\vdash \mathrm{Pf}\ulcorner A\urcorner$, given a suitable coding system defining $\ulcorner A\urcorner$ and a provability predicate Pf corresponding to the formal calculus and coding system. Obviously, the effort required to prove A increases hugely as we move up from one level to the next, but one could argue that the intrinsic complexity does not increase at all; the additional effort is essentially mechanical and bureaucratic. Nevertheless, Gödel's treatment makes strenuous efforts to minimise the need to construct formal proofs.

Gödel describes a relation $R(x_1,\ldots,x_n)$ as *entscheidungsdefinit* (the modern term is *numeralwise expressible*) provided there is a formula $\mathrm{R}(x_1,\ldots,x_n)$ such that, for each x_1, $\ldots$, x_n,

$$R(x_1,\ldots,x_n) \quad \text{implies} \quad \vdash \mathrm{R}(\boldsymbol{x_1},\ldots,\boldsymbol{x_n}) \tag{1}$$

$$\overline{R}(x_1,\ldots,x_n) \quad \text{implies} \quad \vdash \neg\mathrm{R}(\boldsymbol{x_1},\ldots,\boldsymbol{x_n}) \tag{2}$$

Here, $\overline{R}$ means "not R" and $\boldsymbol{x_1},\ldots,\boldsymbol{x_n}$ denotes the numerals expressing the values of x_1, $\ldots$, x_n (Feferman, 1986, p. 130). This technique shows that $\vdash \mathrm{R}(\boldsymbol{x_1},\ldots,\boldsymbol{x_n})$ is a theorem of the formal calculus without requiring an explicit proof. Unfortunately, the price is a considerable increase in intrinsic complexity: explicit numerical bounds have to be given for all quantifiers, and the proofs that these bounds are sufficiently large can be very complicated. These proofs refer to the coding functions and require detailed reasoning about primes, lowest common multiples, etc.

A Σ_1 formula in PA is logically equivalent to one of the form $\exists x_1 \ldots \exists x_n \phi$, where ϕ is a primitive recursive formula. Based on this concept (henceforth simply "Σ formulas"), one can eliminate the need for bounded existential quantifiers. Σ formulas turn out to be sufficient to express the provability predicate Pf and the syntactic concepts underlying it: terms, formulas, substitutions, etc. They satisfy property (1) above but not (2). To recover the latter property, Boolos (1993) uses the concept of a Δ formula: a Σ formula whose negation is also a Σ formula. Unfortunately, this approach again requires bounds for existential quantifiers. Boolos (1993) devotes more than a page (page 41) to a "grisly" proof of one of these bounds, concerned with the coding of terms. The very statement of the theorem (which replaces one unbounded existential quantifier by three bounded quantifiers) is highly technical. As there are a great many other existential quantifiers in the definition of the provability predicate, this approach cannot lead to an intelligible proof of the incompleteness theorems.

Świerczkowski (2003) confines himself to Σ formulas. Since property (2) does not hold, it is necessary to perform some proofs in the HF formal calculus. He presents detailed proofs that the coded substitution operations on coded terms and formulas are single-valued. These proofs are as long as the one given in Boolos (1993), but conceptually they are simple; their purpose is to demonstrate that the proof of the single-valued property is elementary enough to be proved in the HF calculus.

To actually exhibit a formal proof, some elementary concepts and lemmas in the theory of HF have to be developed formally: the principle of mathematical induction, the linear ordering for the natural numbers, etc. But to reach the first incompleteness theorem, these formal developments do not even need to define addition. To reach the second theorem, we require a few addition laws and some basic properties of finite sequences, but nothing more: certainly, not multiplication. This is the main benefit of using HF, since $\langle x,y\rangle$ is simply $\{\{x\},\{x,y\}\}$, and coding is no longer arithmetisation.

Świerczkowski (2003) quotes Boolos (1993), who describes his proofs as "incomplete" and "irremediably messy" (page 16). Świerczkowski's proof of the second incompleteness theorem is certainly less messy, because he eliminates virtually all arithmetical arguments. The Isabelle/HOL proofs are of course complete, and represent the first machine-assisted proof of the second incompleteness theorem. The explicit derivations in the HF calculus are necessarily messy, because they are strings of low-level logical inferences. But with few exceptions, the statements actually proved are straightforward; generally, they prove that various coded operations do exactly what they are supposed to do.

The rest of the paper discusses Isabelle/HOL (§2.) and the fundamental definitions underlying the proofs (§3.). Techniques used to formalise Gödel-numbering are briefly sketched (§4.). The steps leading to the first incompleteness theorem is then described (§5.). One small but interesting finding concerns the technique for proving the second incompleteness theorem. The descriptions given by both Boolos (1993) and Świerczkowski (2003) are potentially misleading, if not actually wrong (§6.). Another finding is that Świerczkowski's proof is actually incomplete, with a significant gap which I have closed using methods quite different from the ones he outlined (§7.). A brief section concludes the paper (§8.).

Note that this paper contains no definitions or proofs as conventionally understood in mathematics; rather, it describes definitions and formal proofs that have been conducted in Isabelle/HOL, and lessons learned from them. Our focus below concerns such logical issues revealed by the Isabelle/HOL development. Technological aspects of this development are discussed in a companion paper Paulson (2013). In order to save space, standard definitions involving the incompleteness theorems are not presented below except where they need to be discussed specifically. This material is widely available, and Świerczkowski (2003) can be downloaded from an Internet archive.[1]

§**2. Background.** These proofs were conducted using Isabelle/HOL, an interactive theorem prover (Nipkow et al., 2002). Therefore *all* proofs are conducted in a formal calculus: higher-order logic. Nevertheless, there is an enormous difference between proofs carried out Isabelle/HOL's native logic and those carried out in a formal calculus specified within Isabelle/HOL. Interactive theorem provers typically hide the underlying calculus as much as possible through automatic simplifiers and other tools, trying to create the illusion that the user is writing a rigorous but flexible mathematical document. A logical calculus formalised within Isabelle/HOL is an inductively defined set, and a proof within this calculus is a demonstration that a particular object (representing a formula) belongs to that set. Isabelle's automation assists with such demonstrations, but they are nevertheless long and all but incomprehensible.

Before formalising the logical calculus, we must formalise the syntax of terms and formulas. A crucial question is the treatment of bound variables. The names of bound variables are typically regarded as significant, so that $\exists xy\,[x > y]$ and $\exists vw\,[v > w]$ are distinct (albeit logically equivalent) formulas. With such an approach, renaming a bound variable is an explicit step. Gödel's proofs make heavy use of explicit formulas with many quantifiers, and also require induction over the structure of formulas. Having to rename bound variables complicates proofs considerably.

[1] `http://journals.impan.gov.pl/dm/Inf/422-0-1.html`

Nominal Isabelle is a formal theory developed within Isabelle/HOL in order to support reasoning about named bound variables (Urban & Kaliszyk, 2012). Variable names are significant where they appear free, but variable binding constructions are quotiented with respect to the bound variable names, so that $\exists xy\,[x > y]$ and $\exists vw\,[v > w]$ denote the same formula exactly as $\{0,1\}$ and $\{1,0\}$ denote the same set. Permutations on names are the key underlying mechanism, for which can be derived the function $\mathrm{supp}(\alpha)$, which coincides with the set of free variables in α when α is something like a term or formula. When performing induction on a formula, these mechanisms can ensure that any bound variables inside the formula are distinct from those of any other formulas that we are interested in. Thus we can avoid the many problems reported by O'Connor (2005), who formalised the first incompleteness theorem using Coq.

One penalty that must be paid in exchange for these advantages is that any function defined on formulas must use bound variables sensibly (for example, we may not define the set of variables *bound* in a formula). While the formal definition of "sensibly" admits all the definitions required for the incompleteness theorems, proving this property required specialised skills (I frequently called upon Christian Urban for assistance), and they can be very demanding of processor time.

For the coding of formulas, bound variables can be formalised using the nameless approach of de Bruijn (1972). Bound variable occurrences are designated by non-negative integers: 0 for the innermost bound variable and increasing for each intervening quantifier. Substitution and abstraction can be defined easily. The main drawback of eliminating bound variable names in this manner is a complete loss of readability, but that is of no importance for coding. The Isabelle/HOL development proves an exact correspondence between the syntax of terms and formulas defined using Nominal Isabelle and the codes of terms and formulas. This correspondence extends to syntactic operations, such as substitution, encoded using a combination of Świerczkowski's and de Bruijn's techniques. There is no need to formalise the nominal theory in the HF calculus, and the complications would be considerable.

A sceptical reader is entitled to ask why we should trust this complicated software and the mysterious nominal theory. We gain confidence in it—as with all human artefacts—through a combination of personal experience, its reputation and an understanding of its design. Isabelle/HOL has now been used in a great many substantial projects by hundreds of users, giving strong reasons to accept that it is a correct implementation of higher-order logic. The nominal theory is a definitional extension of this logic, all concepts ultimately reducible to HOL primitives. The formally verified correspondence between nominal syntax and de Bruijn syntax, mentioned above, is further evidence for its correctness. The formal development itself presents a proof of the incompleteness theorems at a level of detail vastly greater than can be found in any published account. Moreover, this formal development is a live document: our sceptic can load it into Isabelle/HOL, point to any part of any proof, and quickly see what has to be proved at that point. Transparency is the best response to scepticism.

§3. The Isabelle/HOL formalisation: fundamentals.

Let us see what typical definitions and proofs look like in Isabelle/HOL. One claim for this work is that the machine proofs are readable, at least to a limited extent, allowing this very lengthy and complicated series of definitions and proofs to be examined.

The hereditarily finite sets are recursively defined as finite sets of hereditarily finite sets. Świerczkowski (2003) presents a first-order theory having a constant 0 (the empty set), a

binary operation symbol $\lhd$ (augmentation, or "eats"), a relation symbol $\in$ (membership) as well as equality, satisfying the following axioms:

$$z = 0 \leftrightarrow \forall x\,[x \notin z] \tag{HF1}$$

$$z = x \lhd y \leftrightarrow \forall u\,[u \in z \leftrightarrow u \in x \vee u = y] \tag{HF2}$$

$$\phi(0) \wedge \forall xy\,[\phi(x) \wedge \phi(y) \rightarrow \phi(x \lhd y)] \rightarrow \forall x\,[\phi(x)] \tag{HF3}$$

The third axiom expresses induction. Świerczkowski (2003) develops the necessary elements of this set theory, including functions, ordinals (which are simply the natural numbers) and definitional principles. Kirby (2007) presents an elegant generalisation of ordinal addition to the universe of sets. Formalising such material in Isabelle/HOL is routine.

The first milestone in proving the incompleteness theorems is to formalise the syntax of the HF calculus. Remember, in Isabelle/HOL, mathematics is expressed in higher-order logic. This is a typed formalism, and the following declaration establishes a recursive type `tm` of HF terms. The type `name` has already been established, using the nominal framework, as the type of variable names for this calculus.

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

This declares that a term is either `Zero` or has the form `Var i`, where `i` is a name, or has the form `Eats t1 t2` for terms `t1` and `t2`.

It is now possible to define the type `fm` of HF formulas.

```
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
```

The HF calculus includes an existential quantifier, denoted `Ex`, which involves variable binding via the nominal framework. The **infixr** declarations provide an alternative syntax for the membership relation, the equality relation, and disjunction. A formula can also be a negation. The other logical connectives are introduced later as abbreviations.

Substitution is often problematical to formalise, but here it is straightforward. Substitution of a term `x` for a variable `i` is defined as follows:

```
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)"
```

For substitution within a formula, we normally expect issues concerning the capture of a bound variable. Note that the result of substituting the term `x` for the variable `i` in the formula `A` is written `A(i::=x)`.

```
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))"
```

Substitution is again straightforward in the first four cases (membership, equality, disjunction, negation). In the existential case, the precondition `atom j ♯ (i, x)` (pronounced "`j` is fresh for `i` and `x`") essentially says that `i` and `j` must be different names with `j` not free in `x`. We do not need to supply a mechanism for renaming the bound variable, as that is part of the nominal framework, which in most cases will choose a sufficiently fresh bound variable at the outset. The usual properties of substitution (commutativity, for example) have simple proofs by induction on formulas. In contrast, O'Connor (2009) needed to combine three substitution lemmas in a simultaneous proof by induction, a delicate argument involving 1900 lines of Coq.

The HF proof system is an inductively defined predicate, where `H ⊢ A` means that the formula `A` is provable from the set of formulas `H`.

```
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"
```

Note that the existential rule is subject to the condition that the bound variable, `i`, is fresh with respect to `B` and the formulas in `H`. The definitions of `boolean_axioms`, etc., are taken from Świerczkowski (2003). He formalised a simpler inference system, with theorems of the form `⊢ A`. Introducing `H` allows a proof of the deduction theorem and the derivation of a sort of sequent calculus, a practical necessity if we are to conduct proofs in this formal calculus.

Another deviation from Świerczkowski (2003) is the inclusion of `extra_axiom`. It is a parameter of the entire development; it can be any formula that is true under the Tarski truth-definition.[2] Its purpose is to generalise the statements of the incompleteness theorems, which Świerczkowski proved only for one specific calculus. O'Connor (2005) has gone further to prove the first incompleteness theorem even for infinite extensions of the calculus.

The incompleteness theorems require the definition of a great many predicates, mostly for coding the syntax of terms and formulas, and operations on them. It may be instructive to look at a very simple definition, namely of the subset relation:

```
nominal_primrec Subset :: "tm ⇒ tm ⇒ fm"   (infixr "SUBS" 150)
  where "atom z ♯ (t, u) ⟹ t SUBS u = All2 z t ((Var z) IN u)"
```

This introduces `SUBS` as the name of the subset relation, which is defined using a bounded quantifier by $t \subseteq u \iff \forall (z \in t)\, [z \in u]$. Note that `All2` is our syntax for a bounded universal quantifier. The condition `atom z ♯ (t,u)` states that the quantified variable (`z`) must be fresh for the terms `t` and `u`. In other words, and in contrast to some treatments, the

[2] This is formalised as the function `eval_fm`, which is presented in the companion paper (Paulson, 2013, section 3.1). The constraint that `extra_axiom` must be true is not shown here.

bound variable is a parameter of the definition rather than being fixed; however, the choice of `z` cannot affect the denotation of the right-hand side, thanks to quotienting.

Proving the elementary properties of the subset relation within the HF calculus is extremely tedious, over 200 lines of proof script. Extensionality must be proved by induction within the calculus:

```
lemma Extensionality: "H ⊢ x EQ y IFF (x SUBS y AND y SUBS x)"
```

The length of these trivial proofs might be taken as a sign that mechanising the incompleteness theorems is infeasible. It is fortunate that proofs of apparently more advanced properties do not get longer and longer, even when we come to prove the Hilbert-Bernays derivability conditions.

Świerczkowski (2003) discusses Σ formulas, constructed from atomic formulas using conjunction, disjunction, existential quantification and bounded universal quantification. *Strict* Σ formulas contain no terms other than variables, and the bound j in $\forall(i \in j)A$ must not be free in the quantified body, A.

```
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)"
```

One advantage of formal proof is that these conditions are immediately evident, when they may not be clear from an informal presentation. Świerczkowski (2003) does not impose the last condition (on the bound of a universal quantifier), but it greatly simplifies the main induction needed to reach the second incompleteness theorem. (If we are only interested in formalising the first incompleteness theorem, we can use a more generous notion of Σ formula, allowing atomic formulas and their negations over arbitrary terms.) Formally, a Σ formula is defined to be any formula that can be proved equivalent (in the HF calculus) to a strict Σ formula:

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

The condition `supp B ⊆ supp A` essentially means that every variable free in `B` must also be free in `A`. After a certain amount of effort, it is possible to derive the expected properties of Σ formulas and ultimately to reach a key result based on this concept:

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

If `A` is a true Σ sentence, then $\vdash$`A`. This result reduces the task of proving $\vdash$ `A` in the formal calculus to proving that `A` holds (written `eval_fm e0 A`) in Isabelle/HOL's native higher-order logic.

§**4. The Isabelle/HOL formalisation: The coding of syntax.** The coding of terms, formulas, substitution, the HF axioms and ultimately the provability predicate is straightforward to formalise. Gödel (1931) and Świerczkowski (2003) present full details. Many other authors prefer to simplify matters via repeated appeals to Church's thesis. Even the detailed presentations mentioned above omit any demonstration that the definitions are correct. The *proof formalisation condition* for the provability predicate (written `PfP` below) is typically stated with a minimum of justification:

theorem *proved_iff_proved_Pf: "{} ⊢ A ⟷ {} ⊢ PfP ⌜A⌝"*

One could argue that there is no need for the definitions to be correct in every detail, provided they convince the reader that correct and suitable definitions exist. However, only correct definitions can be verified in Isabelle/HOL. Most of these proofs are indeed routine, though in places (for example, in the specification of an instance of the HF induction axiom) extremely tedious.

The de Bruijn (1972) representation of variable binding requires new versions of the syntactic predicates for "formula", "substitution", etc. The coding of terms and formulas is done by first translating them from nominal syntax to de Bruijn syntax. In verifying the coding predicates, we also verify this translation.

A standard treatment of de Bruijn syntax requires defining two operations: abstraction and substitution. Abstraction replaces free occurrences of a given term by a new bound variable, represented by a numeric index; the resulting formula is ill-formed until a matching quantifier is prefixed to it. Substitution is the inverse of abstraction, replacing the outermost bound variable (after a quantifier has been stripped from a formula) by some given term. For the incompleteness theorems, both operations can be simplified: abstraction replaces a free variable by a bound variable, and substitution replaces a free variable by a given term. Abstraction is needed to formalise the construction of a formula, because it is a necessary step before a quantifier can be attached.

The interplay of these various points can be seen below:

```
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')"
```

Thus y is the code of a formula constructed from existing formulas u and v provided y codes the disjunction $u \vee v$, the negation $\neg u$ or the existential formula $\exists(u')$, where u' has been obtained by abstracting u over some variable, v. The predicate *AbstForm* performs de Bruijn abstraction over a formula; its definition is complicated, and omitted here. Note that the codes of quantified formulas do not mention the names of bound variables.

This predicate is given by a higher-order logic formula, and therefore at the level of the meta-theory. Working at this level eliminates the need to construct HF proofs, and most of the correctness properties we need can be proved in this manner. However, in order to perform the diagonalisation argument and exhibit the undecidable formula, we need a version of every coding predicate as an HF formula. Therefore, each predicate must be defined on both levels:

```
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)))"
```

As we saw above in the definition of *Subset*, constraints are required on all quantified variables. Here there are only two, but to define *AbstForm* requires 12 bound variables. The necessary declarations are lengthy and messy, and put a heavy burden on the nominal package (proofs run very slowly), but the alternative of having to rename explicit bound variables is also unattractive.

§**5. The Isabelle/HOL formalisation: first incompleteness theorem.** The diagonalisation theorem is now easily reached. Continuing to follow Świerczkowski (2003), the next step is to define a function K such that $\vdash K(\ulcorner\phi\urcorner) = \ulcorner\phi(\ulcorner\phi\urcorner)\urcorner$. Formally, K is a *pseudo-function*, represented by the single-valued relation `KRP`, taking two arguments. The following result is not difficult to obtain, given the existing coding of substitution, and some other steps that will be discussed later. This theorem does not require a proof within the HF calculus, but follows from `Sigma_fm_imp_thm` because it is a sentence (coded syntax contains no free variables) and a Σ formula.

lemma `prove_KRP: "{} ⊢ KRP ⌜Var i⌝ ⌜A⌝ ⌜A(i::=⌜A⌝)⌝"`

The property of being single-valued is easily stated, but it is neither a sentence nor a Σ formula. Proving this result requires about 600 lines of explicit reasoning steps in the HF calculus, verifying that substitution over terms or formulas yields a unique result.

lemma `KRP_unique: "{KRP v x y, KRP v x y'} ⊢ y' EQ y"`

The diagonal lemma is now reached by the standard argument. It concerns an arbitrary formula, α, presumably containing `i` as a free variable. Note that `α(i::=⌜δ⌝)` denotes the result of replacing `i` by $\ulcorner\delta\urcorner$. The **obtains** syntax represents a form of existential quantification, here asserting the existence of an HF formula δ satisfying the two properties shown.

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

The second part of the conclusion, namely `supp δ = supp α - {atom i}`, states that the free variables of the formula δ are those of α with the exception of `i`; it is necessary in order to show that the undecidable formula is actually a sentence.

The first incompleteness theorem itself can now be proved. Figure 1 presents the full text. Even a reader who is wholly unfamiliar with Isabelle/HOL should be able to see something intelligible in this proof script. Assuming consistency of the calculus, formalised as `¬ {} ⊢ Fls` (falsity is not provable), we obtain a formula δ satisfying the properties shown, in particular `¬ {} ⊢ δ` and `¬ {} ⊢ Neg δ`. Lines beginning with commands such as **obtain**, **hence**, **show** introduce assertions to be proved. The details of the reasoning may be unclear, but milestones such as `"{} ⊢ δ IFF Neg (PfP ⌜δ⌝)"` and `"¬ {} ⊢ δ"` are visible, as references to previous named results. This legibility, however limited, is possible because the entire Isabelle/HOL proof is written in the structured Isar language (Wenzel, 2007). Only the HF calculus proofs remain unintelligible: it is not easy to impose structure on those.

§**6. Issues involving the second incompleteness theorem.** My object in writing this paper is not to discuss the formalisation in general, but to examine the specific consequences of basing the development on HF set theory rather than Peano arithmetic. A further aim is to look at a crucial step in the proof of the second incompleteness theorem that is not described especially well in other presentations.

It is well-known that the theorem follows easily from the Hilbert-Bernays derivability conditions (Boolos, 1993, p. 15), one of which is $\vdash \mathrm{Pf}(\ulcorner\phi\urcorner) \to \mathrm{Pf}(\ulcorner\mathrm{Pf}(\ulcorner\phi\urcorner)\urcorner)$. This result is a consequence of the theorem

$$\text{if } \alpha \text{ is a } \Sigma \text{ sentence, then } \vdash \alpha \to \mathrm{Pf}(\ulcorner\alpha\urcorner), \tag{3}$$

```
theorem Goedel_I:
  assumes "¬ {} ⊢ 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
```

Fig. 1. Proof of the first incompleteness theorem

which can be proved by a tricky induction on the construction of α as a strict Σ formula.

For this proof, the system of coding is extended to allow variables in codes. If we regard variables as indexed by positive integers, then the variable x_i is normally coded by the term $\mathrm{SUCC}^i(0)$, where $\mathrm{SUCC}(x) = x \lhd x$ is the usual successor function. Similarly, the formula $x_1 = x_2$ is normally coded by the term $\langle \ulcorner = \urcorner, \ulcorner x_1 \urcorner, \ulcorner x_2 \urcorner \rangle$. If variables are preserved rather than coded, we instead get the term $\langle \ulcorner = \urcorner, x_1, x_2 \rangle$. In general, $\lfloor \alpha \rfloor_V$ designates the coding of α where all variables from the set V are preserved as variables in the result, while all other variables are coded by constant terms. Świerczkowski (2003) calls this *pseudo-coding*.

Imagine that we could define in HF a function Q such that

$$Q(0) = \ulcorner 0 \urcorner = 0 \tag{4}$$

$$Q(x \lhd y) = \langle \ulcorner \lhd \urcorner, Q(x), Q(y) \rangle \tag{5}$$

Then we would have $Q(x) = \ulcorner t \urcorner$, where t is some canonical term denoting the set x. [Świerczkowski (2003) introduces a total ordering on HF to make this possible, as discussed below.] Suppose that α is a formula whose set of free variables is $V = \{x_1, \ldots, x_n\}$. Given the theorem $\vdash \alpha$, obtain $\vdash \mathrm{Pf}(\ulcorner \alpha \urcorner)$ by the proof formalisation condition, then successively replace x_i by $Q(x_i)$, for $i = 1, \ldots, n$. The replacements are possible because the HF calculus includes a rule to substitute a term t for a variable x in the formula ϕ:

$$\frac{H \vdash \phi}{H \vdash \phi(x/t)}$$

Performing the replacements requires the analogue of this substitution rule as encoded in the provability predicate, Pf. For example, we can obtain the following series of theorems:

$$\begin{array}{ll}
\vdash y \in (x \lhd y) & \\
\vdash \mathrm{Pf}\ulcorner y \in (x \lhd y)\urcorner & \text{proof formalisation condition} \\
\vdash \mathrm{Pf}\langle\ulcorner\in\urcorner,\ulcorner y\urcorner,\langle\ulcorner\lhd\urcorner,\ulcorner x\urcorner,\ulcorner y\urcorner\rangle\rangle & \text{definition of coding} \\
\vdash \mathrm{Pf}\langle\ulcorner\in\urcorner,\ulcorner y\urcorner,\langle\ulcorner\lhd\urcorner,Q(x),\ulcorner y\urcorner\rangle\rangle & \text{replacement of } x \\
\vdash \mathrm{Pf}\langle\ulcorner\in\urcorner,Q(y),\langle\ulcorner\lhd\urcorner,Q(x),Q(y)\rangle\rangle & \text{replacement of } y
\end{array}$$

To simplify the notation, let $\alpha(V/Q)$ abbreviate $\alpha(x_1/Q(x_1),\ldots,x_n/Q(x_n))$, the result of simultaneously replacing every free variable x_i in α by the term $Q(x_i)$. As a further simplification, let us write $\lfloor t\rfloor_V(Q)$ instead of $\lfloor t\rfloor_V(V/Q)$. Then the sequence of steps above can also be written

$$\begin{array}{ll}
\vdash \mathrm{Pf}\ulcorner y \in (x \lhd y)\urcorner & \\
\vdash \mathrm{Pf}\lfloor y \in (x \lhd y)\rfloor_{\{x\}}(Q) & \text{replacement of } x \\
\vdash \mathrm{Pf}\lfloor y \in (x \lhd y)\rfloor_{\{x,y\}}(Q) & \text{replacement of } y
\end{array}$$

A crucial part of the reasoning is that the replacement of $\ulcorner y\urcorner$ by $Q(y)$ leaves the occurrences of $Q(x)$ unchanged. That holds because $Q(x)$ is always the code of a *constant* term, as can trivially be proved from (4) and (5) by induction on x. A constant term is unaffected by substitutions.

The difficulty with this sketch is that no function $Q(x)$ can exist, because the HF language has only one function symbol, $\lhd$. Extending this language with the function symbol Q would require redoing all the coding and syntactic functions; Q would also need to encode references to itself. Instead, $Q(x)$ is typically regarded as a "pseudo-function": it must be defined in the form of a relation $\mathrm{QR}(x,y)$ for which $\forall x\,[\exists! y\,\mathrm{QR}(x,y)]$ can be proved. We must modify the transformations above accordingly. Boolos (1993) and Świerczkowski (2003) both state that the formula $A(Q(x))$ is an abbreviation for $\exists x'\,[\mathrm{QR}(x,x')\wedge A(x')]$; the latter author describes a detailed procedure for replacing occurrences of pseudo-functions from the inside out (Świerczkowski, 2003, p. 47). This suggests the following modified sequence:

$$\begin{array}{ll}
\vdash \mathrm{Pf}\ulcorner y \in (x \lhd y)\urcorner & \\
\vdash \mathrm{Pf}\lfloor \exists x'\,[\mathrm{QR}(x,x')\wedge y \in (x' \lhd y)]\rfloor_{\{x\}} & \text{replacement of } x \\
\vdash \mathrm{Pf}\lfloor \exists y'\,[\mathrm{QR}(y,y')\wedge \exists x'\,[\mathrm{QR}(x,x')\wedge y' \in (x' \lhd y')]]\rfloor_{\{x,y\}} & \text{replacement of } y
\end{array}$$

Further evidence that this is the intended transformation is the remark (Boolos, 1993, p. 45) that the transformed formula, $\mathrm{Pf}(\lfloor\alpha\rfloor_V(Q))$ in our notation, "has the same variables free as" the original formula, α. The difficulty is that this modified sequence does not work, and neither can any other that leaves the original variables free in the transformed formula. The explanation is simple: these variables (in particular x above) range over all values, including the codes of all possible formulas. There is no reason why $\mathrm{QR}(x,x')$ should be left unchanged after the substitution for y: there is nothing to exclude the possibility that $x = \ulcorner y\urcorner$, for example. One could argue that the remarks and explanations that I have cited are true in spirit if not in fact, but they are misleading. We even see a detailed proof that $Q(x_i)$ is correctly substituted for x_i with reference to the definitions of the syntactic substitution predicates (Świerczkowski, 2003, p. 25), but there is no such term as $Q(x)$.

The correct sequence of steps introduces new free variables in the transformed formula, while simultaneously constraining them as constants on the left-hand side of the $\vdash$ symbol.

$$\begin{aligned} &\vdash \mathrm{Pf}\ulcorner y \in x \lhd y\urcorner \\ \mathrm{QR}(x,x') &\vdash \mathrm{Pf}\lfloor y \in x' \lhd y\rfloor_{\{x'\}} && \text{replacement of } x \\ \mathrm{QR}(y,y'),\mathrm{QR}(x,x') &\vdash \mathrm{Pf}\lfloor y' \in x' \lhd y'\rfloor_{\{x',y'\}} && \text{replacement of } y \end{aligned}$$

Here, x is replaced by x', constrained by the new assumption $\mathrm{QR}(x,x')$ and then y is replaced by y'. Now x' is unaffected by substitutions because (given the assumption $\mathrm{QR}(x,x')$) it can be shown to contain no variables. This reasoning is straightforward enough to conduct formally in the HF calculus.

This may seem to be a small detail, but as can be seen, it is not difficult to explain correctly. One could argue that the correct version is actually simpler to explain than the traditional version involving the pseudo-function Q: the notation $\lfloor\alpha\rfloor_V(Q)$ is no longer necessary. Eliminating the pseudo-functions from the presentation actually simplifies it.

§**7. Issues connected with the use of HF sets.** The motivation for using hereditarily finite sets rather than Peano arithmetic is that it allows more natural and simpler proofs. But it appears to complicate the definition of the function $Q(x)$ mentioned above, which is needed to prove both incompleteness theorems. In PA, the analogous function $Z(n)$ is trivial to define (Feferman, 1986, p. 165): there is only one way to write a natural number in the form $\mathrm{SUCC}^n(0)$.

Świerczkowski (2003) eliminates the ambiguity implicit in (5) above by appealing to a total ordering, $<$, on the HF universe. The difficulty is how to define this ordering within the HF calculus. Świerczkowski develops the theory, including a definition by recursion on the rank of a set, but it does not look easy to formalise in HF. Another approach is to define the function $f : \mathrm{HF} \to \mathbb{N}$ such that $f(x) = \sum\{2^{f(y)} \mid y \in x\}$. Then we can define $x < y \iff f(x) < f(y)$. Again, the effort to formalise this theory in HF may be simpler than that needed to formalise the Chinese remainder theorem, but it is still considerable.

The alternative is to eliminate the need for this ordering. Świerczkowski has already completed part of this task. In his proof of the first incompleteness theorem, he introduces a function H such that $H(\ulcorner\phi\urcorner) = \ulcorner\ulcorner\phi\urcorner\urcorner$. This function is recursively defined on valid codes, that is, on terms recursively built over natural numbers using ordered pairing. In fact, H is identical to Q but with a restricted domain, ensuring that it can easily be proved to be a function.

For the second incompleteness theorem, the solution to our conundrum is again to focus on the corresponding relation, QR. There is no need to prove that this relation describes a function. All that is necessary in order to prove (3) is the property

$$\mathrm{QR}(x,x'),\mathrm{QR}(y,y') \vdash x \in y \to \mathrm{Pf}\lfloor x' \in y'\rfloor_{\{x',y'\}}, \tag{6}$$

Świerczkowski shows that this follows from the lemma

$$\mathrm{QR}(x,x'),\mathrm{QR}(y,y') \vdash x = y \to \mathrm{Pf}\lfloor x' = y'\rfloor_{\{x',y'\}}, \tag{7}$$

which clearly holds even if QR does not describe a functional relationship. A way to prove both (6) and (7) can be seen from the following elementary set-theoretic equivalences,

which connect the relations $\in$, $\subseteq$ and $=$:

$$\begin{aligned} z \in \emptyset &\iff \bot \\ z \in x \lhd y &\iff z \in x \vee z = y \\ \emptyset \subseteq z &\iff \top \\ x \lhd y \subseteq z &\iff x \subseteq z \wedge y \in z \\ x = y &\iff x \subseteq y \wedge y \subseteq x \end{aligned}$$

The point of all this is that (6) and (7) can be proved by a simultaneous induction:

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

The induction is on the sum of the lengths of the derivations of $\mathrm{QR}(x,x')$ and $\mathrm{QR}(y,y')$. Like most of the syntactic predicates used in the incompleteness theorems, $\mathrm{QR}(x,x')$ is defined to hold provided there exist k and s such that s is a k-element sequence representing the conditions (4) and (5). Induction on the sum of the lengths allows us to prove

$$x \in y \to \mathrm{Pf}\lfloor x' \in y' \rfloor_{\{x',y'\}}$$

by case analysis on the form of y, while proving

$$x \subseteq y \to \mathrm{Pf}\lfloor x' \subseteq y' \rfloor_{\{x',y'\}}$$

by case analysis on the form of x. One case of the reasoning is as follows:

$$\begin{aligned} x_1 \lhd x_2 \subseteq y &\iff x_1 \subseteq y \wedge x_2 \in y \\ &\implies \mathrm{Pf}\lfloor x_1' \subseteq y' \rfloor_{\{x_1',y'\}} \wedge \mathrm{Pf}\lfloor x_2' \in y' \rfloor_{\{x_2',y'\}} \\ &\iff \mathrm{Pf}\lfloor x_1' \lhd x_2' \subseteq y' \rfloor_{\{x_1',x_2',y'\}} \end{aligned}$$

The formalisation of the entire mutually inductive argument in the HF calculus requires under 450 lines of Isabelle/HOL. The need to define an ordering on the HF universe has disappeared.

The mechanised proof requires only the simplest induction principles throughout. The basic principle of the hereditarily finite sets (HF3) is used eight times, mostly to develop the fundamentals of the HF set theory itself. Complete induction on the natural numbers is used ten times, while ordinary mathematical induction is used eleven times. No other form of induction is necessary. Świerczkowski (2003) frequently sketches proofs by induction on terms or formulas. He suggests induction on the HF ordering, $<$, to prove (6) above and also to prove the bounded quantifier case of the main theorem:

$$\vdash \forall (j \in i)\, \alpha(j) \to \mathrm{Pf}(\lfloor \forall (j' \in i)\, \alpha(j') \rfloor)$$

Each of these theorems concerns syntactic predicates defined by the existence of a k-element sequence, and is more directly proved by complete induction on k, or rarely (where there are two sequences, as above) on the sum $k_1 + k_2$.

§**8. Discussion and conclusions.** The first mechanised formalisation of Gödel's (first) incompleteness theorem is due to Shankar (1986). It was an astonishing accomplishment given the technology of the 1980s. An interesting technical note is that Shankar (2013) found de Bruijn indices indispensable in a companion proof (of the Church-Rosser theorem), but not in his formalisation of the logical calculus. He also used HF set theory, but using a different axiom system (Shankar, 1994, p. 12) that he attributes to Cohen. Nineteen

years later, O'Connor (2005) mechanised the first theorem using quite different methods and the Coq proof assistant. Another proof, by John Harrison, can be downloaded with his HOL Light proof assistant, `http://code.google.com/p/hol-light/`. There appears to exist no other machine proof of the second incompleteness theorem.

The mechanised incompleteness theorems described above were difficult chiefly because of their sheer size, and because of the presentational issues discussed from §6. onwards, which resulted in a great deal of wasted work. But we now have a complete, transparent and machine-checked formalisation of these landmark results.

Acknowledgement Jesse Alama drew my attention to Świerczkowski (2003), which was the source material for this project. Christian Urban assisted with some proofs and wrote some code involving his nominal package. Brian Huffman assisted with the formalisation of the HF sets. Dana Scott offered advice and drew my attention useful related work, for example Kirby (2007). Matt Kaufmann made insightful comments on a draft of this paper. The referee made a great many constructive remarks.

Bibliography

Boolos, G. S. (1993). *The Logic of Provability*. Cambridge University Press.

de Bruijn, N. G. (1972). Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indagationes Mathematicae* **34**, 381–392.

Feferman, S., editor (1986). *Kurt Gödel: Collected Works*, Volume I. Oxford University Press.

Franzén, T. (2005). *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse*. A K Peters.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik* **38**(1), 173–198.

Kirby, L. (2007). Addition and multiplication of sets. *Mathematical Logic Quarterly* **53**(1), 52–65.

Nipkow, T., Paulson, L. C., & Wenzel, M. (2002). *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer. LNCS Tutorial 2283.

O'Connor, R. (2005). Essential incompleteness of arithmetic verified by Coq. In Hurd, J. & Melham, T., editors, *TPHOLs*, LNCS 3603, pp. 245–260. Springer.

O'Connor, R. S. S. (2009). *Incompleteness & Completeness: Formalizing Logic and Analysis in Type Theory*. Ph. D. thesis, Radboud University Nijmegen.

Paulson, L. C. (2013). A mechanised proof of Gödel's incompleteness theorems using Nominal Isabelle. Submitted for publication.

Shankar, N. (1986). *Proof-checking Metamathematics*. Ph. D. thesis, University of Texas at Austin.

Shankar, N. (1994). *Metamathematics, Machines, and Gödel's Proof*. Cambridge University Press.

Shankar, N. (2013). Shankar, Boyer, Church-Rosser and de Bruijn indices. E-mail.

Świerczkowski, S. (2003). Finite sets and Gödel's incompleteness theorems. *Dissertationes Mathematicae* **422**, 1–58. `http://journals.impan.gov.pl/dm/Inf/422-0-1.html`.

Urban, C., & Kaliszyk, C. (2012). General bindings and alpha-equivalence in Nominal Isabelle. *Logical Methods in Computer Science* **8**(2:14), 1–35.

Wenzel, M. (2007). Isabelle/Isar — a generic framework for human-readable proof documents. In Matuszewski, R. & Zalewska, A., editors, *From Insight to Proof —*

Festschrift in Honour of Andrzej Trybulec. University of Białystok. Studies in Logic, Grammar, and Rhetoric 10(23).

COMPUTER LABORATORY
UNIVERSITY OF CAMBRIDGE
CAMBRIDGE, CB3 0FD, UK
E-mail: lp15@cam.ac.uk