A Succinct Formalization of the Completeness of First-Order Logic

Asta Halkjær From
Technical University of Denmark, Kongens Lyngby, Denmark

Abstract

I succinctly formalize the soundness and completeness of a small Hilbert system for first-order logic in the proof assistant Isabelle/HOL. The proof combines and details ideas from de Bruijn, Henkin, Herbrand, Hilbert, Hintikka, Lindenbaum, Smullyan and others in a novel way, and I use a declarative style, custom notation and proof automation to obtain a readable formalization. The formalized definitions of Hintikka sets and Herbrand structures allow open and closed formulas to be treated uniformly, making free variables a non-concern. This paper collects important techniques in mathematical logic in a way suited for both study and further work.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases First-Order Logic, Completeness, Isabelle/HOL

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.8

Supplementary Material
Software (Formalization (stable)): `https://isa-afp.org/entries/FOL_Axiomatic.html`
Software (Formalization (latest)): `https://devel.isa-afp.org/entries/FOL_Axiomatic.html`

Acknowledgements Thanks to Frederik Krogsdal Jacobsen, Alexander Birch Jensen and Jørgen Villadsen for their useful comments. I am especially grateful to the anonymous reviewers for their insightful remarks which have improved both the formalization and the paper.

1 Introduction

The completeness of first-order logic has been known since Gödel's work in 1929 [19]. Proof assistants too have a long history [18], with de Bruijn initiating the Automath project in 1967 and the first system of LCF, an Isabelle/HOL predecessor, being developed in 1972. Despite of this, I am unaware of a formalization of completeness in a proof assistant with a focus on explaining the core techniques. The ideas involved in such a proof deserve to be documented and detailed in a formalization where the proof assistant guarantees precision and correctness. This effort benefits students trying to understand mathematical logic and researchers looking for a base for their own work. I start from a Hilbert system, partly because I am unaware of a formalization which does the same, and partly because its simplicity allows me to focus on the ideas of the completeness proof itself. While other deduction systems may be more popular for first-order logic, Hilbert systems are still prominent in areas like modal logic.

This paper builds especially on work by Smullyan [40] and Henkin [21]. The Hilbert system of choice is Smullyan's System Q1 [40, p. 81] and the completeness proof resembles the "more direct construction" near the end of his book [40, p. 96] (a construction that was pointed out to him by Henkin himself). This paper formalizes ideas by de Bruijn, Henkin, Herbrand [23], Hilbert, Hintikka, Lindenbaum and Smullyan in an attempt to give a "strikingly direct" [40, p. 96] completeness proof formalized in a modern proof assistant.

Smullyan includes a generalization rule for the universal quantifier that works under an assumption (i.e. to the right of an implication) rather than on a standalone formula. This extra generality makes it very suited for my purposes, where I always work under a number

of assumptions. Smullyan does not consider function symbols as part of his syntax, but his System Q1 is easily extended to cover these: simply allow for any term in the instantiation axiom. Barwise [1] makes the same modification.

I use the proof assistant Isabelle/HOL [34]. Isabelle is a generic proof assistant and Isabelle/HOL is the version based on higher-order logic. This paper includes a number of Isabelle listings, all taken from the formalization after it has been exported to LaTeX. These listings appear either in bulleted lists or prefixed by an Isabelle command in **bold** and should therefore be clearly distinguishable from the surrounding text. Any such listing has been checked and verified by the proof assistant. I sometimes use these listings to explain proofs. In these cases, I do not include the commands that justify each step of reasoning, but only the intermediate statements themselves. For clarity, I have omitted many types from the main text. Some of these can be found in Table 1 on page 6. The full formalization (under 700 lines) is available in the Archive of Formal Proofs [17], which referees Isabelle formalizations and, when accepted, keeps them up to date with the latest version of the proof assistant.

Contributions

The main contribution of this paper is a succinct formalization of the definitions and proofs that make up the synthetic style, a widely applicable method of proving completeness.

As a smaller contribution, this is, to my knowledge, the first formalization of completeness for classical first-order logic that starts from a Hilbert system. However, several formalizations that start from other proof systems are available (cf. Section 2) and the relations between various proof systems have also been formalized, see for instance recent work by Laurent [27] in Coq on translating between Hilbert systems and natural deduction for first-order logic.

On the more technical side, I formalize a Herbrand universe which, like in Herbelin and Ilik's [22] unformalized proof, consists of all terms, not just those without variables. Combined with a Hintikka set in the style of Forster et al. [11] in Coq, based on the absence of formulas rather than the presence of their negations, but which, unlike theirs, contains open formulas as well as closed, I naturally formalize completeness for all valid formulas.

Isabelle/HOL Overview

This section seeks to give a quick overview of the Isabelle/HOL features used later. Nipkow and Klein [33, Part 1] give a more complete introduction.

The higher-order logic of Isabelle/HOL can be reasonably understood as functional programming + logic [33]. The **datatype** command defines a new type from a series of constructors, where each can be given custom syntax. The natural numbers are built from the nullary constructor *0* and unary *Suc*. The constructors *True* and *False* belong to the built-in type *bool*. The usual connectives and quantifiers from first-order logic ($\longrightarrow$, $\forall$, etc.) are available for *bool*, as well as *if-then-else* expressions. We can write total functions over datatypes using **primrec** for primitive recursive functions and **fun** for more advanced definitions. The type constructor $A \Rightarrow B$ denotes a function from A to B. Instead of concrete types, we can also use type variables $'a$, $'b$, etc. These stand in the place of other types. The term *UNIV* stands for the set of all values of a given type.

Another important built-in type is $'a$ *list*, the type of lists whose elements are of type $'a$. These are built from [], the empty list, and #, an infix constructor that adjoins an element to an existing list. The notation [*a, b, c*] is shorthand for these primitive operations. The function *set* turns a list into a set of its elements. The higher-order function *map* applies a given function to every element of a list.

Function application resembles functional programming languages in that $f(x, y)$ is written as $f\ x\ y$. The function $f(x := y)$ maps x to y and every other element x' to $f\ x'$. Anonymous functions can be built using λ-expressions, e.g. $\lambda n.\ n + n$ for $f(n) = n + n$.

In proofs, the meta-logical implication $\Longrightarrow$ separates assumptions from conclusions. These can also be distinguished using the **assumes** and **shows** keywords, using **and** as a separator when there are multiple assumptions or conclusions. The keyword **have** states an intermediate fact in a proof and the keywords **then**, **moreover** and **ultimately** bind these statements together in different ways. The keyword **let** introduces a local abbreviation and **obtain** eliminates an existential statement; **for** quantifies a statement universally.

The command **definition** introduces a new definition that is hidden behind a name and must be explicitly unfolded, while an **abbreviation** is unfolded by the parser. The **inductive** command also makes use of the meta-logical implication. This command allows us to specify the least predicate closed under some given rules. For instance a predicate that denotes whether a formula can be derived, specified by axioms and inference rules. A **locale** in Isabelle, as used here, names an association between terms and assumptions about them. We could, for instance, specify a group as a set and a binary operation coupled with the group axioms. To instantiate the locale we must give concrete terms and show that they satisfy the assumptions. When assuming a locale, we assume the conditions hold for the terms.

The axiom of choice is available as Hilbert's choice operator: the expression *SOME x. P x* returns some element x that satisfies the predicate P, when such an element exists.

Overview of Paper

I discuss related work next (Section 2). In Section 3, I formalize the syntax of first-order logic in Isabelle/HOL, including the idea of de Bruijn indices. This idea is developed further in Section 4 on the semantics of terms and formulas. Section 5 formalizes the proof system and its soundness, including all the operations necessary to do so. This includes the instantiation of universal quantifiers, propositional tautologies and a range of lemmas. I prove the completeness in Section 6 where I introduce the notion of a maximal consistent set, the Lindenbaum construction and the model existence theorem for Hintikka sets. I discuss challenges and choices in Section 7 and conclude with thoughts on future work in Section 8.

2 Related Work

The completeness of first-order logic itself has a long history, starting with Gödel's proof [19] and Henkin's later simplification [21]. Smullyan [40], Barwise [1] and Fitting [10] all include completeness proofs in their texts. Smullyan's main completeness proof is an "analytic" proof for a tableau system. He devotes only two pages to the "synthetic" (also called Henkin-style) completeness proof of System Q1 [40, pp. 96–97] that I formalize a version of here. Barwise [1] considers System Q1 extended with axioms for equality and gives a quite different proof that relies on a reduction to propositional logic (and the completeness of propositional logic). Fitting [10] proves completeness for tableaux and resolution similarly to Smullyan and leaves the completeness of his Hilbert system as an exercise for the reader. This paper spells out the synthetic completeness proof for first-order logic, starting from a Hilbert system rather than from tableaux, resolution or similar.

The synthetic style has been successfully applied in several formalizations lately. From [13] used it to formalize the completeness of a Hilbert system for propositional logic in Isabelle/HOL. Berghofer [3] formalized natural deduction for first-order logic in Isabelle/HOL

following the work by Fitting [10]. My formalization of the syntax and semantics of first-order logic and the Lindenbaum construction is inspired by his work. My formalization of Hintikka sets and proof of the model existence theorem, however, differ from his and is inspired by Herbelin and Ilik [22] and Forster et al. [11]. In particular, I prove completeness for open and closed formulas together, where Berghofer's completeness proof only covers closed formulas and is extended to cover open formulas afterwards. From, Schlichtkrull and Villadsen [14, 16] built on Berghofer's work to formalize the completeness of both a sequent calculus and tableau system for first-order logic. They also described Berghofer's formalization in detail. Bentzen [2] formalized the completeness of a Hilbert system for the modal logic S5 in Lean. Jørgensen et al. [26] gave a synthetic completeness proof for a tableau system for basic hybrid logic, which From [12, 15] formalized in Isabelle/HOL.

I am far from the first to formalize the completeness of first-order logic, but my formalization is the only one that proves completeness for a Hilbert system for classical first-order logic. Shankar [39] formalized a tautology checker for first-order logic in the Boyer-Moore theorem prover, but notably did not formalize first-order completeness. Harrison [20] also formalized first-order logic in higher-order logic (but HOL rather than Isabelle/HOL). He did not formalize a proof system but rather model-theoretic results like compactness and the Löwenheim-Skolem theorem. Margetson and Ridge [29] formalized the completeness of first-order logic without functions in Isabelle/HOL via a sequent calculus. Braselmann and Koepke [7] did the same in their Mizar formalization. Schlichtkrull [37, 38] formalized the completeness of first-order resolution in Isabelle/HOL. Michaelis and Nipkow [30, 31] did not formalize first-order logic, but did formalize a Hilbert system for propositional logic in Isabelle/HOL. They proved completeness via translation from a sequent calculus with an analytic completeness proof. Blanchette, Popescu and Traytel [5, 6] formalized analytic completeness of abstract sequent calculus and tableau systems for first-order logic in Isabelle/HOL. Blanchette [4] outlines formalizations of logical meta-theory in Isabelle/HOL.

The completeness proof presented here relies on Lindenbaum's lemma: that any consistent set of formulas has a maximal consistent extension. The proof is non-constructive since, for the given semantics, Lindenbaum's lemma is equivalent to Weak König's Lemma [22, 24]. There are, however, a number of formalizations of completeness in other meta-theories (and necessarily using other semantics). Veldman [43] gave an intuitionistic completeness theorem for intuitionistic predicate logic in 1976. Persson [35] formalized the completeness of sequent calculus and natural deduction for intuitionistic first-order logic in the ALF proof assistant, but only defined a Hilbert system without further proof. His models are based on formal topology. Constable and Bickford [8] constructively proved completeness for intuitionistic first-order logic in the proof assistant Nuprl. Ilik [25] formalized multiple constructive proofs of first-order completeness in the proof assistant Coq using novel variants of Kripke models for full classical and intuitionistic first-order logic. Forster et al. [11] recently analyzed the computational content of completeness theorems for various semantics and for natural deduction and sequent calculus systems. They mechanized their results in constructive type theory using Coq.

3 Syntax

The following syntax will be our starting point.

A term t is either a variable x or a function symbol f applied to a number of other terms:

$$s, t ::= x \mid f(t_1, \ldots, t_n)$$

A function symbol applied to zero terms is called a constant and is denoted by a.

A formula p is either falsity (denoted $\bot$), a predicate symbol P applied to a list of terms, an implication ($\longrightarrow$) between two formulas or a universally quantified formula:

$$p, q ::= \bot \mid P(t_1, \ldots, t_n) \mid p \longrightarrow q \mid \forall x.\ p(x)$$

I apply a number of techniques to make this syntax suitable for formalization. First, I represent the variables with de Bruijn indices [9]. Instead of connecting quantifiers and variables by using the same variable symbol x, each variable is a natural number n that is understood to be connected to the nth quantifier, starting from the innermost. For example, the formula $\forall x.\ \forall y.\ P(x, y)$ is represented as $\forall\ \forall\ P(1, 0)$. This makes it simpler to define capture-avoiding instantiation, which we need for the proof system.

Second, to distinguish variables, function symbols and predicate symbols in the proof assistant, I prefix each kind by a distinct symbol: † for function symbols, ‡ for predicate symbols and # for variables. The formula $\forall\ P(f(0))$ is then written (for now) as $\forall\ ‡P(†f(\#0))$.

Finally, lists of argument terms are represented as proper Isabelle/HOL lists, so the term $f(a)$ becomes $†f\ [a]$.

The (parameterized) datatype *'f tm* of terms with function symbols of type *'f* becomes:

```
datatype ('f) tm
  = Var nat (#)
  | Fun 'f ('f tm list) (†)
```

The annotation *params-tm* generates a function of that name from terms to *'f* sets: it collects all values of type *'f* in a given term. I discuss its properties in Section 5.1.

The following abbreviates a constant, as I use these frequently:

```
abbreviation Const (⋆) where ⋆a ≡ †a []
```

The datatype *('f, 'p) fm* of formulas with functions symbols of type *'f* and predicate symbols of type *'p* becomes:

```
datatype ('f, 'p) fm
  = Falsity (⊥)
  | Pre 'p ('f tm list) (‡)
  | Imp (('f, 'p) fm) (('f, 'p) fm) (infixr ⟶ 55)
  | Uni (('f, 'p) fm) (∀)
```

The custom notation for each syntactic case is enclosed in parentheses (**infixr** specifies right associativity and 55 specifies parsing priority). I use bold symbols to avoid conflicts with existing syntax. The notation *params-fm*, similarly to for terms, generates a function which produces a set of all function symbols in a given formula.

The Isabelle command **term** checks the type of an expression. Given the above definitions, we can try our syntax, here with strings for the types of function and predicate symbols:

```
term ∀(⊥ ⟶ ‡''P'' [†''f'' [#0]])
```

In regular notation this would be $\forall x.\ \bot \longrightarrow P(f(x))$.

The following abbreviation for negation will ease notation: $\neg\ p \equiv p \longrightarrow \bot$.

Similar notations can easily be introduced for conjunction, disjunction, the existential quantifier etc. since in classical logic, these can be defined using the given syntax.

It should be noted that since arities are implicit in the datatypes above, we unfortunately cannot represent finite signatures. The awarded benefit is that we do not need a predicate to distinguish between correct and incorrect syntax.

Table 1 Type signatures for selected functions.

semantics-tm	$(nat \Rightarrow 'a) \Rightarrow ('f \Rightarrow 'a\ list \Rightarrow 'a) \Rightarrow 'f\ tm \Rightarrow 'a$
semantics-fm	$(nat \Rightarrow 'a) \Rightarrow ('f \Rightarrow 'a\ list \Rightarrow 'a) \Rightarrow ('p \Rightarrow 'a\ list \Rightarrow bool) \Rightarrow ('f, 'p)\ fm \Rightarrow bool$
shift	$(nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow nat \Rightarrow 'a$
boolean	$('p \Rightarrow 'f\ tm\ list \Rightarrow bool) \Rightarrow (('f, 'p)\ fm \Rightarrow bool) \Rightarrow ('f, 'p)\ fm \Rightarrow bool$
Axiomatic	$('f, 'p)\ fm \Rightarrow bool$
imply	$('f, 'p)\ fm\ list \Rightarrow ('f, 'p)\ fm \Rightarrow ('f, 'p)\ fm$
consistent	$('f, 'p)\ fm\ set \Rightarrow bool$
extend	$('f, 'p)\ fm\ set \Rightarrow (nat \Rightarrow ('f, 'p)\ fm) \Rightarrow nat \Rightarrow ('f, 'p)\ fm\ set$
witness	$'f\ set \Rightarrow ('f, 'p)\ fm \Rightarrow ('f, 'p)\ fm\ set$
hmodel	$('f, 'p)\ fm\ set \Rightarrow ('f, 'p)\ fm \Rightarrow bool$

4 Semantics

The semantics of first-order logic has two parts: one for terms and one for formulas. I formalize both as functions.

4.1 Terms

A term evaluates to an element of the *domain*. It does so under an *environment* (a mapping from variables to the domain) and a *function denotation* (a mapping from function symbols to functions on the domain).

In Isabelle, I represent the domain as a type (variable) and the environment as a function E from the natural numbers (the de Bruijn indices) to that type. Similarly, the function denotation becomes the function F from function symbols to functions on the domain. This results in the following definition:

```
primrec semantics-tm (⦇-, -⦈) where
  ⦇E, F⦈ (#n) = E n
| ⦇E, F⦈ (†f ts) = F f (map ⦇E, F⦈ ts)
```

The semantics of a variable is given by the environment and in the case of a function application $\dagger f\ ts$, we must first evaluate all the argument terms *ts* (using *map*) and then apply the function denoted by *f*.

Here $\llparenthesis E, F \rrparenthesis$ denotes the function from terms to the domain, given by the environment E and function denotation F. As seen in the clause above for functions, this notation lets me conveniently "bundle" a given E and F so they can be applied to any term without the need for anonymous functions. I use a similar notation $[\![E, F, G]\!]$ for the semantics of formulas.

4.2 Formulas

I use a deep embedding where formulas evaluate to a truth value under an environment E, a function denotation F and a *predicate denotation*, dubbed G, that maps predicate symbols to predicates on the domain. I formalize it as follows:

```
primrec semantics-fm (⟦-, -, -⟧) where
  ⟦-, -, -⟧ ⊥ = False
| ⟦E, F, G⟧ (‡P ts) = G P (map ⦇E, F⦈ ts)
| ⟦E, F, G⟧ (p ⟶ q) = (⟦E, F, G⟧ p ⟶ ⟦E, F, G⟧ q)
| ⟦E, F, G⟧ (∀ p) = (∀ x. ⟦E⟨0:x⟩, F, G⟧ p)
```

The formula $\bot$ is always *False* and the semantics of a predicate is similar to that of a function application. For implication each subformula is evaluated to a truth value and the connective is interpreted using the built-in implication. Similarly, I use the built-in universal quantifier to interpret the object-level quantifier. The notation $E\langle 0{:}x\rangle$ is explained next.

4.3 Shifting

The expression $E\langle n{:}x\rangle$ modifies the environment E such that variable n is assigned x, any smaller variable m is assigned $E\ m$ and any larger variable m is assigned $E\ (m-1)$. This *shift* operation has the following definition:

definition *shift* $(\text{-}\langle\text{-}{:}\text{-}\rangle)$ **where**
$E\langle n{:}x\rangle\ m \equiv \mathit{if}\ m < n\ \mathit{then}\ E\ m\ \mathit{else\ if}\ m = n\ \mathit{then}\ x\ \mathit{else}\ E\ (m{-}1)$

To understand the shifting operation on larger variables, consider the following:

$$[\![E, F, G]\!]\ (\forall\ \forall\ \ddagger P\ [\#0, \#1])$$

By the semantics, this reduces to:

$$\forall x.\ [\![E\langle 0{:}x\rangle, F, G]\!]\ (\forall\ \ddagger P\ [\#0, \#1])$$

where the outer quantifier comes from the meta-logic. This again reduces to:

$$\forall x.\ \forall y.\ [\![E\langle 0{:}x\rangle\langle 0{:}y\rangle, F, G]\!]\ (\ddagger P\ [\#0, \#1])$$

Thus, the terms are evaluated by $(\!|E\langle 0{:}x\rangle\langle 0{:}y\rangle, F|\!)$. This is clearly correct for variable $\#0$ since $E\langle 0{:}x\rangle\langle 0{:}y\rangle\ 0 = y$ as desired. We also want that $\#1$ corresponds to the outer meta-logic quantifier, namely that $E\langle 0{:}x\rangle\langle 0{:}y\rangle\ 1 = x$. This is exactly what happens since $E\langle 0{:}x\rangle\langle 0{:}y\rangle\ 1 = E\langle 0{:}x\rangle\ (1-1) = x$. Thus, the semantics reduces to the expected:

$$\forall x.\ \forall y.\ G\ P\ [y, x]$$

Notice that any *free* variable in a formula (those with no corresponding quantifier) are not affected by this shifting when it is coupled with the removal of an outer quantifier: they are mapped to whatever E originally assigned them to. In this sense they behave like constants.

The following four lemmas will be used implicitly.

▶ **Lemma 1** (Shifting). *The first three results characterize the function and the last one commutes a shift of variable 0 with another shift.*

- $n = m \Longrightarrow E\langle n{:}x\rangle\ m = x$
- $m < n \Longrightarrow E\langle n{:}x\rangle\ m = E\ m$
- $n < m \Longrightarrow E\langle n{:}x\rangle\ m = E\ (m{-}1)$
- $(E\langle n{:}y\rangle\langle 0{:}x\rangle) = (E\langle 0{:}x\rangle\langle n{+}1{:}y\rangle)$

Proof. Immediate from the definition. ◀

5 Proof System

To define the proof system I must first define a number of operations needed to state the side conditions and transformations of formulas.

5.1 Parameters

The proof rule for the universal quantifier will generalize a *fresh* constant to a quantified variable. To perform this freshness check, I use the functions *params-tm* and *params-fm* that Isabelle generates automatically from the datatype declarations above. These collect all function symbols in terms and formulas, respectively, and would also be easy to define recursively. Similarly to Smullyan [40], I abbreviate function symbol to *parameter*.

The following definition generalizes *params-fm* to a set of formulas:

abbreviation *params* $S \equiv \bigcup p \in S.$ *params-fm* p

I need a few lemmas about parameters for later.

▶ **Lemma 2** (Finite parameters). *Terms and formulas contain only finitely many parameters:*
- *finite* (*params-tm* t)
- *finite* (*params-fm* p)

Proof. By simple inductions. ◀

▶ **Lemma 3** (Unused parameters). *The denotation of an unused parameter does not affect the semantics of either terms or formulas:*
- $f \notin$ *params-tm* $t \Longrightarrow (\!|E,\ F(f := x)|\!)\ t = (\!|E,\ F|\!)\ t$
- $f \notin$ *params-fm* $p \Longrightarrow [\![E,\ F(f := x),\ G]\!]\ p = [\![E,\ F,\ G]\!]\ p$

Proof. By simple inductions. ◀

5.2 Instantiation

I will need to instantiate a universally quantified formula with a concrete term: to go from $\forall\ p$ to "p with t inserted for variable 0 and free variables in p adjusted." I will denote this formula by $\langle t/0\rangle p$. Note that when instantiating for n in $\forall\ p$, we need to then instantiate for $n+1$ in p, since we enter the scope of another quantifier (the formula $\forall x.\, \forall y.\, P(x, y)$ becomes $\forall\forall P(1, 0)$ with de Bruijn indices, so to instantiate for x we must actually replace variable 1).

There are two additional considerations. Consider first why we need to adjust the free variables in p. Say that we are instantiating $\forall\ P\ [\#0, \#3]$ with the term t. When evaluating $\forall\ P\ [\#0, \#3]$ under an environment E, the free variable 3 will be interpreted as $(E\langle 0{:}x\rangle)\ 3 = E\ 2$. We expect that the interpretation of free variables under the same environment does not change when we instantiate a quantifier. However, when evaluating the naively instantiated formula $P\ [t, \#3]$, the free variable 3 will be evaluated as $E\ 3$, which might be a different value than $E\ 2$. Therefore, we should decrement any free variables we encounter during the instantiation. The result here should then be $P\ [t, \#2]$.

Second, it is important that any free variable in t remains free in $\langle t/0\rangle p$ (i.e. that the instantiation avoids capturing a free variable). With named variables we would ensure this by renaming any bound variables in p that would conflict. By using de Bruijn indices we are free from having to come up with fresh names for such an operation. Instead, we increment every variable in t by one whenever we pass under a quantifier. Thus $\langle \dagger f\ [\#0]/0\rangle(\forall\ (\ddagger P)) = \forall\ (\langle \dagger f\ [\#1]/1\rangle(\ddagger P))$.

I call this last operation *lifting* the term:

primrec *lift-tm* $(\uparrow)$ **where**
  $\uparrow(\#n) = \#(n{+}1)$
| $\uparrow(\dagger f\ ts) = \dagger f\ (map \uparrow ts)$

While this terminology is common (cf. Nipkow [32], Berghofer [3]) it unfortunately conflicts with the terminology in explicit substitutions (cf. Lescanne [28]) where *lifting* and *shifting* have roughly opposite meanings compared to this paper.

With the above considerations in mind, we can now define instantiation on terms:

primrec *inst-tm* $(\langle\!\langle \text{-}'/\text{-}\rangle\!\rangle)$ **where**
  $\langle\!\langle s/m\rangle\!\rangle(\#n) = (\mathit{if}\ n < m\ \mathit{then}\ \#n\ \mathit{else}\ \mathit{if}\ n = m\ \mathit{then}\ s\ \mathit{else}\ \#(n-1))$
| $\langle\!\langle s/m\rangle\!\rangle(\dagger f\ ts) = \dagger f\ (\mathit{map}\ \langle\!\langle s/m\rangle\!\rangle\ ts)$

The notation $\langle\!\langle s/m\rangle\!\rangle$ "bundles" an instantiation of term s for variable m, ready to be applied to a term. For formulas, the only interesting case is for the universal quantifier, where we lift the term we are instantiating with and increment the variable we are instantiating for:

primrec *inst-fm* $(\langle \text{-}'/\text{-}\rangle)$ **where**
  $\langle \text{-}/\text{-}\rangle\bot = \bot$
| $\langle s/m\rangle(\ddagger P\ ts) = \ddagger P\ (\mathit{map}\ \langle\!\langle s/m\rangle\!\rangle\ ts)$
| $\langle s/m\rangle(p \longrightarrow q) = \langle s/m\rangle p \longrightarrow \langle s/m\rangle q$
| $\langle s/m\rangle(\forall\, p) = \forall\, (\langle{\uparrow}s/m{+}1\rangle p)$

Despite the complexity of instantiation when using de Bruijn indices, it can be captured in the three simple definitions above that involve little more than simple arithmetic.

A more standard name for $\langle t/n\rangle p$ is substitution, but I prefer instantiation since it potentially does more than simple syntactic substitution of term t for variable n: namely lifts t and decrements variables in p.

The only results about instantiation that I need for the formalization are the following.

▶ **Lemma 4** (Lifting and shifting). *Lifting cancels out with shifting the environment at 0.*
- $(\!|E\langle 0{:}x\rangle,\ F|\!)\ ({\uparrow}t) = (\!|E,\ F|\!)\ t$

Proof. By structural induction. ◀

▶ **Lemma 5** (Instantiation and shifting). *Instantiating with a term at m is the same as shifting the environment at m with the value denoted by that term.*
- $(\!|E,\ F|\!)\ (\langle\!\langle s/m\rangle\!\rangle t) = (\!|E\langle m{:}(\!|E,\ F|\!)\ s\rangle,\ F|\!)\ t$
- $[\![E,\ F,\ G]\!]\ (\langle t/m\rangle p) = [\![E\langle m{:}(\!|E,\ F|\!)\ t\rangle,\ F,\ G]\!]\ p$

Proof. By structural induction, using Lemma 4. ◀

5.3 Size

To prove the model existence theorem, I will need to do induction on formulas. However, structural induction does not work, since in the case for $\forall\, p$, the induction hypothesis must be applied to the instance $\langle t/0\rangle p$, for some term t, rather than simply to p. This calls for induction on the size of the formula. Unfortunately, the pre-defined *size* measure for our datatype takes the size of terms into account and is therefore not invariant under instantiation. The following definition suffices:

primrec *size-fm* **where**
  size-fm $\bot = 1$
| *size-fm* $(\ddagger\text{-}\ \text{-}) = 1$
| *size-fm* $(p \longrightarrow q) = 1 + \mathit{size\text{-}fm}\ p + \mathit{size\text{-}fm}\ q$
| *size-fm* $(\forall\, p) = 1 + \mathit{size\text{-}fm}\ p$

▶ **Lemma 6** (Size). *Instantiation preserves size.*
- *size-fm* $(\langle t/m\rangle p) = \mathit{size\text{-}fm}\ p$

Proof. By structural induction. ◀

5.4 Propositional Semantics

Instead of picking a suitable set of propositional axioms, Smullyan [40], Barwise [1] and others simply include all tautologies as one of their axioms. I follow their lead and need a suitable way to express which formulas are tautologies. Smullyan [40, p. 51] extends his notion of a *Boolean valuation* from propositional logic to the syntax of first-order logic by treating quantified formulas as another sort of propositional symbols. A *tautology* is then a formula that is true under all Boolean valuations.

The following definition uses the same principle, where G is a predicate denotation as before and A is a special "universally quantified formula denotation."

primrec *boolean* **where**
  boolean - - $\bot$ = *False*
| *boolean* G - ($\ddagger P$ *ts*) = G P *ts*
| *boolean* G A ($p \longrightarrow q$) = (*boolean* G A $p \longrightarrow$ *boolean* G A q)
| *boolean* - A ($\forall\, p$) = A ($\forall\, p$)

The hyphens stand for ignored arguments. Compare this semantics to the first-order one: it is indeed a Boolean valuation [40] of first-order logic. We can now take Smullyan's notion of tautology as definition:

abbreviation *tautology* $p \equiv \forall\, G\ A.$ *boolean* G A p

Smullyan gives no details on his extension of Boolean valuations to first-order logic. The way I set it up, with a separate denotation for the quantified formulas, it can be directly related to the first-order semantics.

▶ **Lemma 7** (Boolean semantics). *The Boolean and first-order semantics coincide when G matches the first-order predicate semantics and A is the first-order semantics itself.*
boolean ($\lambda a.$ G $a \circ$ *map* ($\lVert E,\ F \rVert$)) $[\![E,\ F,\ G]\!]$ = $[\![E,\ F,\ G]\!]$

Proof. By structural induction. ◀

▶ **Lemma 8** (Tautologies). *All tautologies are valid.*
tautology $p \Longrightarrow [\![E,\ F,\ G]\!]\ p$

Proof. Since a tautology holds for any choice of G and A it holds in particular for those that coincide with the first-order semantics (cf. Lemma 7). ◀

For reassurance, Isabelle easily verifies that not all first-order validities are propositional tautologies (e.g. $(\forall x.\, P(x)) \longrightarrow P(a)$ is only the former):

proposition $\exists\, p.$ ($\forall\, E\ F\ G.$ $[\![E,\ F,\ G]\!]\ p$) $\wedge\ \neg$ *tautology* p

5.5 The Inductively Defined Calculus

Finally, we are ready to define the calculus itself. I define it as an inductive predicate $\vdash$ that holds exactly when a formula can be derived from the given axioms and rules. The previous work has made the definition simple:

inductive *Axiomatic* ($\vdash$ - [*50*] *50*) **where**
  TA: *tautology* $p \Longrightarrow \vdash p$
| *IA*: $\vdash \forall\, p \longrightarrow \langle t/0 \rangle p$
| *MP*: $\vdash p \longrightarrow q \Longrightarrow \vdash p \Longrightarrow \vdash q$
| *GR*: $\vdash q \longrightarrow \langle \star a/0 \rangle p \Longrightarrow a \notin$ *params* $\{p,\ q\} \Longrightarrow \vdash q \longrightarrow \forall\, p$

The Tautology Axiom (TA) derives any tautology. The Instantiation Axiom (IA) states that a quantified formula implies its instantiation with any term. The Modus Ponens (MP) rule is stated as usual and lifts an implication between formulas to an implication between derivations. Finally, the Generalization Rule (GR) works under assumptions q and generalizes from an instance to a quantified formula, given that the witness (the constant) is fresh.

▶ **Theorem 9** (Soundness). *Any derivable formula is valid:*

$\vdash p \Longrightarrow [\![E, F, G]\!]\ p$

Proof. By induction over the inductive definition of the axiomatic system for arbitrary function denotation F.

All cases except for GR can be proven automatically, with the case for TA relying on Lemma 8 about tautologies. In the GR case I apply the induction hypothesis not just once at plain F but at $F(a := x)$ for every element x of the domain:

have $[\![E, F(a := x), G]\!]\ (q \longrightarrow \langle \star a/0 \rangle p)$ **for** x

This is enough help for Isabelle to prove the case. ◀

▶ **Corollary 10.** *Falsity cannot be derived:*

$\neg\ (\vdash \bot)$

5.5.1 Notation

For the proof of completeness I need to express that a formula can be derived from a set of *assumptions*. Instead of building this notion into the definition of the proof system, I am going to simulate it using chains of implications. The expression $[p_1, p_2, \ldots, p_n] \rightsquigarrow q$ expands to $p_1 \longrightarrow p_2 \longrightarrow \ldots \longrightarrow p_n \longrightarrow q$. It is defined by recursion on the list of assumptions:

primrec *imply* (**infixr** $\rightsquigarrow$ *56*) **where**
$([] \rightsquigarrow q) = q$
$\mid (p \# ps \rightsquigarrow q) = (p \longrightarrow ps \rightsquigarrow q)$

I then write $ps \vdash q$ to abbreviate $\vdash ps \rightsquigarrow q$:

When I talk about *assumptions* in a derivation I will always mean a finite list of formulas.

5.6 Derived Formulas

Due to my semantic characterization of the Tautology Axiom, the automation in Isabelle can easily prove that various propositional formulas (schemas) can be derived.

▶ **Lemma 11** (Derivations). *The S and K combinators, double negation elimination and contraposition in both directions can all be derived:*

$\vdash (p \longrightarrow q \longrightarrow r) \longrightarrow (p \longrightarrow q) \longrightarrow p \longrightarrow r$

$\vdash q \longrightarrow p \longrightarrow q$

$\vdash \neg \neg p \longrightarrow p$

$\vdash (p \longrightarrow q) \longrightarrow \neg q \longrightarrow \neg p$

$\vdash (\neg q \longrightarrow \neg p) \longrightarrow p \longrightarrow q$

Proof. By the Tautology Axiom. ◀

5.6.1 Generalization Rule

My use of chains of implications is a disadvantage to the *GR* rule since it works on the consequent but implication is right associative. Consider the following: we know that $ps \vdash \langle \star a/0 \rangle p$, for fresh a and want to use *GR* to derive $ps \vdash \forall\, p$. We can only do so if ps consists of exactly one formula q, as $ps \vdash p$ is short for $\vdash ps \rightsquigarrow q$. To circumvent this restriction, I derive the following variant of the rule.

▶ **Lemma 12** (*GR'* rule). *The following rule is derivable:*

$$GR'\text{:} \vdash \neg \langle \star a/0 \rangle p \longrightarrow q \Longrightarrow a \notin \mathit{params}\ \{p, q\} \Longrightarrow \vdash \neg\, (\forall\, p) \longrightarrow q$$

Proof. Follows from the *GR* rule, modus ponens and the derivations in Lemma 11. ◀

Since this rule works on the left-hand side of the implication, the right-hand side can, without issues, be an arbitrarily long chain of implications. Smullyan [40, p. 83] himself uses this version of the rule in his System Q1' (but for notational reasons).

An alternative is to start from the existential quantifier, $\exists$, as primitive, rather than $\forall$, as the generalization rule for $\exists$ works on the left-hand side of the implication [40]. However, it is less immediately clear why this rule for $\exists$ can be called a generalization rule.

5.6.2 Working with Assumptions

The following is an assortment of useful lemmas for working with assumptions.

▶ **Lemma 13** (Assumptions). *The following are derivable: modus ponens under assumptions, derivation of any assumption, the deduction theorem in both directions, a cut rule, classical reasoning and finally a structural rule encompassing weakening, contraction and exchange:*

- $ps \vdash p \longrightarrow q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$
- $p \in \mathit{set}\ ps \Longrightarrow ps \vdash p$
- $ps \vdash p \longrightarrow q \Longrightarrow p \mathbin{\#} ps \vdash q$
- $p \mathbin{\#} ps \vdash q \Longrightarrow ps \vdash p \longrightarrow q$
- $p \mathbin{\#} ps \vdash r \Longrightarrow q \mathbin{\#} ps \vdash p \Longrightarrow q \mathbin{\#} ps \vdash r$
- $(\neg\, p) \mathbin{\#} ps \vdash \bot \Longrightarrow ps \vdash p$
- $ps \vdash q \Longrightarrow \mathit{set}\ ps \subseteq \mathit{set}\ ps' \Longrightarrow ps' \vdash q$

Proof. By a mix of induction over the list of assumptions and propositional reasoning. ◀

6 Completeness

We are now ready to delve into the completeness proof itself. The plan is as follows. If we cannot derive a formula p under any assumptions from X then we cannot derive falsity from $\neg\, p$ and any assumptions from X either. Sets like $\{\neg\, p\} \cup X$ are *consistent* with respect to the proof system, as we cannot derive a contradiction from them. I formalize them in Section 6.1. These sets are defined based on the proof system but we will use them to build a model that contradicts the validity of p under X. For this purpose we must prove that two important types of formulas preserve consistency: fresh witnesses of existential formulas (*Henkin witnesses*) and instances of universal formulas.

Lindenbaum (according to Tarski [41]) showed how to extend a consistent set into a *maximal consistent set (MCS)*. Any proper superset of a maximal consistent set is inconsistent. In particular this means that for any formula p, an MCS contains exactly p or $\neg\, p$. Henkin [21],

showed the utility of adding the Henkin witnesses for existential formulas during Lindenbaum's construction. I formalize the construction and its consistency in Section 6.2 and prove that the result is maximal in Section 6.3.

The addition of Henkin witnesses ensures that our MCSs are *saturated*. Section 6.4 outlines the benefits of ensuring this by construction.

Instead of building a model directly from a maximal consistent saturated set, I introduce a standard layer of abstraction. In Section 6.5, I formalize the notion of a *Hintikka* set [40] using three simple conditions and prove a model existence theorem: given a Hintikka set H, I build a model from a Herbrand structure [10, 22] that satisfies exactly the formulas in H. I then prove that maximal consistent saturated sets are Hintikka sets.

In Section 6.6, I put all the pieces together. The model existence theorem gives us a model for $\neg\, p$ and all of X. Therefore, if p is in fact valid under assumptions from X, then it must be derivable or we have a contradiction.

6.1 Consistent Sets

The definition of consistency is straightforward. The set of formulas S is consistent when there is no list of assumptions S', coming from S, that can be used to derive falsity:

definition *consistent* $S \equiv \nexists S'.$ *set* $S' \subseteq S \wedge S' \vdash \bot$

The following lemma will be useful.

▶ **Lemma 14** (Inconsistent addition). *Assume that S is consistent on its own but becomes inconsistent with the addition of a formula p. Then there exists a concrete list of assumptions S', coming from S, such that $p \# S' \vdash \bot$:*

assumes *consistent* S **and** $\neg$ *consistent* $(\{p\} \cup S)$
obtains S' **where** *set* $S' \subseteq S$ **and** $p \# S' \vdash \bot$

Proof. It follows from consistency and the structural lemma for assumptions (Lemma 13). ◀

It is important to prove that two types of formulas preserve consistency. The first type is fresh witnesses for existential formulas.

▶ **Lemma 15** (Consistency of fresh witnesses). *If a consistent set contains an existential formula $\neg\, (\forall\, p)$ then adding a witness $\neg\, \langle \star a/0 \rangle p$, for a fresh a, preserves consistency:*

assumes *consistent* S **and** $\neg\, (\forall\, p) \in S$ **and** $a \notin$ *params* S
shows *consistent* $(\{\neg\, \langle \star a/0 \rangle p\} \cup S)$

Proof. We need to show that there is no finite subset from which we can derive falsity, so assume that indeed there is one. From Lemma 14 we can name the problematic assumptions:

then obtain S' **where** *set* $S' \subseteq S$ **and** $(\neg\, \langle \star a/0 \rangle p) \# S' \vdash \bot$

After showing that the side conditions are fulfilled, we can apply the *GR'* rule:

then have $\neg\, (\forall\, p) \# S' \vdash \bot$

But every assumption is now in S, which we assumed to be consistent, so we have reached the desired contradiction and proved the lemma. ◀

We shall also need that instantiating a universally quantified formula preserves consistency.

▶ **Lemma 16** (Consistency of instantiation). *If a consistent set contains a universal formula* $\forall\, p$ *then adding an instance* $\langle t/0\rangle p$*, for any term* t*, preserves consistency:*

assumes *consistent* S **and** $\forall\, p \in S$
shows *consistent* $(\{\langle t/0\rangle p\} \cup S)$

Proof. The proof proceeds as before and we start by naming the problematic assumptions from an assumed inconsistency (Lemma 14):

then obtain S' **where** *set* $S' \subseteq S$ **and** $\langle t/0\rangle p \mathbin{\#} S' \vdash \bot$

This time we make use of the Instantiation Axiom, instantiated to p and t:

moreover have $\vdash \forall\, p \longrightarrow \langle t/0\rangle p$

With the deduction theorem, the cut rule and the structural lemma (Lemma 13), we can apply this implication to weaken the derivation of falsity:

ultimately have $\forall\, p \mathbin{\#} S' \vdash \bot$

But again, these assumptions are all in S, which we assumed to be consistent, so this is a contradiction and adding the instance must also be consistent. ◀

6.2 Lindenbaum Extension

We turn now to a central construction. Note first that if the sets of variable, function and predicate symbols are countable, so too are the sets of terms and formulas (formalized in Section 6.6). Thus, we can enumerate the formulas as $p_0, p_1, \ldots$ and so on. Starting from a consistent set S_0, which leaves infinitely many parameters unused, we then build a sequence of consistent sets in the following way. Given S_n, construct S_{n+1} as:

$$S_{n+1} = \begin{cases} w(*, p_n) \cup \{p_n\} \cup S_n & \text{if } \{p_n\} \cup S_n \text{ is consistent} \\ S_n & \text{otherwise} \end{cases}$$

where $*$ is the set of parameters in $\{p_n\} \cup S_n$.

The function w returns a singleton set with a fresh witness when p_n is an existential formula and the empty set otherwise. Usually, the availability of such fresh witnesses is guaranteed by extending the set of function symbols. I assume instead that the set of function symbols is infinite from the start and that S_0 leaves infinitely many parameters unused. I pass the parameters of $\{p_n\} \cup S_n$ to w. It can then pick a parameter that has not been used already. This is simpler than dealing with two sorts of function symbols.

In the Isabelle formalization, the enumeration of formulas is represented by a (surjective) function f from the set of natural numbers to the set of formulas (cf. Section 6.6). The expression *extend* $S\ f\ n$ constructs the set S_n starting from $S_0 = S$:

```
primrec extend where
  extend S f 0 = S
| extend S f (Suc n) =
  (let Sn = extend S f n in
    if consistent ({f n} ∪ Sn)
    then witness (params ({f n} ∪ Sn)) (f n) ∪ {f n} ∪ Sn
    else Sn)
```

The function *witness* is simple:

fun *witness* **where**
witness used $(\neg\ (\forall\, p)) = \{\neg\ \langle\star(SOME\ a.\ a \notin used)/0\rangle p\}$
| *witness* - - = {}

Its definition uses Hilbert's choice operator to pick a fresh parameter.

The maximal consistent set is given by taking the union of this sequence of sets: $\bigcup_{n\in\mathbb{N}} S_n$. In Isabelle, it becomes:

definition *Extend S f* $\equiv \bigcup n.$ *extend S f n*

The following lemmas are needed later.

▶ **Lemma 17** (Lindenbaum bounds). *The starting set is included in its maximal extension and each set in the constructed sequence bounds the previous sets:*

- $S \subseteq$ *Extend S f*
- $(\bigcup n \leq m.$ *extend S f n*$) =$ *extend S f m*

Proof. By definition and by induction on m, respectively. ◀

▶ **Lemma 18** (Lindenbaum parameters). *A witness includes only finitely many parameters and each set S_n contains finitely many more parameters than the starting set S_0:*

- *finite* (*params* (*witness used p*))
- *finite* (*params* (*extend S f n*) − *params S*)

Proof. Since p contains finitely many parameters and by induction on n, respectively. ◀

6.2.1 Consistency

The consistency of each constructed set S_n is apparent.

▶ **Lemma 19** (Consistency of S_n). *When starting from a consistent S_0 with infinitely many unused parameters, any constructed S_n is consistent:*

assumes *consistent S* **and** *infinite* (*UNIV* − *params S*)
shows *consistent* (*extend S f n*)

Proof. By induction on n. The consistency of adding the witness follows from Lemma 15. The only complication is to prove that there are indeed always fresh parameters available and therefore that the parameter given by Hilbert's choice operator is usable, but this follows from Lemma 18. ◀

The consistency of the union $\bigcup_n S_n$ is more interesting.

▶ **Lemma 20** (Consistency of $\bigcup_n S_n$). *The maximal extension of a consistent set S with infinitely many unused parameters is consistent:*

assumes *consistent S* **and** *infinite* (*UNIV* − *params S*)
shows *consistent* (*Extend S f*)

Proof. Assume towards a contradiction that we can derive falsity from some finite subset:

then obtain S' **where** $S' \vdash \bot$ **and** *set* $S' \subseteq$ *Extend S f*

Since this subset is finite, it must be a subset of some initial segment of the union:

then obtain *m* **where** *set* $S' \subseteq (\bigcup n \leq m.$ *extend S f n*$)$

But, by Lemma 17, each such segment is bounded by its last element:

then have *set* $S' \subseteq$ *extend S f m*

And since we have already shown the consistency of each S_n (Lemma 19), we reach our desired contradiction. ◀

6.3 Maximal Sets

A maximal set is inconsistent under any proper extension:

definition *maximal* $S \equiv \forall\, p.\ p \notin S \longrightarrow \neg$ *consistent* $(\{p\} \cup S)$

Maximal consistent sets are truly maximal:

▶ **Lemma 21** (Maximality of Maximal Consistent Sets)**.** *If S is a maximal consistent set, then for every formula p, $p \in S$ if and only if $\neg\, p \notin S$.*

assumes *consistent* S **and** *maximal* S
shows $p \in S \longleftrightarrow (\neg\, p) \notin S$

Proof. The left-to-right direction follows from consistency alone and the right-to-left direction follows from consistency and maximality. ◀

That the Lindenbaum extension results in a maximal set is very easy to see.

▶ **Lemma 22** (Maximality of $\bigcup_n S_n$)**.** *Given a surjective enumeration f, $\bigcup_n S_n$ is maximal:*

assumes *surj* f
shows *maximal* (*Extend* S f)

Proof. Assume towards a contradiction that some formula p is not included even though its inclusion preserves consistency:

assume $p \notin$ *Extend* S f **and** *consistent* $(\{p\} \cup$ *Extend* S $f)$

Say that p is formula number k in the enumeration. Since p is not in the result, it must be inconsistent with S_k:

then have $\neg$ *consistent* $(\{p\} \cup$ *extend* S f $k)$

And this set is a subset of the final result:

moreover have $\{p\} \cup$ *extend* S f $k \subseteq \{p\} \cup$ *Extend* S f

Ultimately, this contradicts the assumption that adding p preserves consistency. ◀

6.4 Saturation

We shall need saturation to show that our constructed sets are Hintikka sets:

definition *saturated* $S \equiv \forall\, p.\ \neg\ (\forall\, p) \in S \longrightarrow (\exists\, a.\ (\neg\ \langle \star a/0 \rangle p) \in S)$

So, in a saturated set there is a corresponding Henkin witness for each existential formula.

▶ **Lemma 23** (Saturation of $\bigcup_n S_n$)**.** *A consistent Lindenbaum extension is saturated:*

assumes *consistent* (*Extend* S f) **and** *surj* f
shows *saturated* (*Extend* S f)

Proof. Guaranteed by construction. ◀

If we only constructed our set to be maximal consistent and tried to show that it was also saturated, we would run into trouble [40, p. 96]. First, given an arbitrary maximal consistent set S, it might be that a Henkin witness is missing because S includes every parameter available and every reuse of a parameter results in an inconsistency. Second, we might be unlucky and enumerate the negation of every suitable witness before enumerating the witness itself: we might always add the negation and never the witness. Following Henkin [21], I ensure saturation by adding the Henkin witnesses together with the existential formulas.

6.5 Hintikka Sets

Instead of showing the model existence theorem directly for maximal consistent saturated sets, it will be cleaner to show that Hintikka sets induce a model for their formulas and that our sets are in fact Hintikka sets.

The following definition characterizes a Hintikka set H over our syntax:

locale *Hintikka* =
  fixes $H :: ('f, 'p)\ fm\ set$
  assumes
    FlsH: $\bot \notin H$ **and**
    ImpH: $(p \longrightarrow q) \in H \longleftrightarrow (p \in H \longrightarrow q \in H)$ **and**
    UniH: $(\forall\, p \in H) \longleftrightarrow (\forall\, t.\ \langle t/0\rangle p \in H)$

Hintikka sets are sets that are *saturated downwards* [40, p. 27] and induce a model for the formulas in them. Since the set should induce a model, $\bot$ should never be present (*FlsH*). Following Forster et al. [11, Lemma 11], I enforce that the set *respects* both implication (*ImpH*) and universal quantification (*UniH*): a formula is in the Hintikka set if and only if the "evidence" for that formula is also present. Here, evidence is to be understood in terms of the Herbrand model given below.

6.5.1 Model Existence

The model induced by a Hintikka set H is very simple. It consists of a Herbrand structure [10] and a predicate denotation based on H itself:

Domain Herbrand universe: the universe of terms.
Function denotation The constructor †, i.e. every function symbol evaluates to itself.
Predicate denotation Predicate P is true for terms ts exactly when $‡P\ ts \in H$.

Like in the work by Herbelin and Ilik [22], but unlike for instance the formalizations by Berghofer [3] and Forster et al. [11], the Herbrand universe includes all terms, not just those with no variables. I never formalize what it means for a formula to be closed. The Herbrand structure famously evaluates any term without variables to itself [10]. Or in this case:

▶ **Lemma 24** (Herbrand semantics). *Under any Herbrand structure and the specific environment #, every term evaluates to itself:*
- $(\!|\#, †|\!)\ t = t$

Proof. By structural induction. ◀

I reuse the notation for semantics and abbreviate the model induced by H as $[\![H]\!]$:

abbreviation *hmodel* ($[\![\text{-}]\!]$) **where** $[\![H]\!] \equiv [\![\#, †, \lambda P\ ts.\ ‡P\ ts \in H]\!]$

We now reach the model existence theorem.

▶ **Theorem 25** (Model existence). *When H is a Hintikka set, $[\![H]\!]$ satisfies exactly the formulas in H.*

assumes *Hintikka H*
shows $p \in H \longleftrightarrow [\![H]\!]\ p$

Proof. By well-founded induction on the size of the formula as given by *size-fm*. Thus the induction hypothesis applies to any formula that is smaller by this measure, i.e. to subformulas and to instances of universally quantified formulas (cf. Lemma 6). These are exactly the

formulas that appear in the Hintikka conditions. The proof proceeds by considering each type of formula. Since there is a Hintikka condition for every type, which corresponds exactly to the semantics of the induced model, Isabelle automatically proves each case. For instance, a universal formula p is in the Hintikka set iff every instance $\langle t/0\rangle p$ is in the Hintikka set (*UniH*) iff every instance $\langle t/0\rangle p$ holds in the induced model (by the induction hypothesis). ◀

6.5.2 Saturated MCSs are Hintikka Sets

Consider first the following correspondence between derivability and MCSs.

▶ **Lemma 26** (Derivability and MCSs). *A formula p is derivable from an MCS S iff p is in S:*

assumes *consistent S* **and** *maximal S*
shows $(\exists ps.\ set\ ps \subseteq S \wedge ps \vdash p) \longleftrightarrow p \in S$

Proof. The left to right direction follows from the maximality of MCSs. The right to left direction follows trivially from the derivability of any assumption (Lemma 13). ◀

I now show that maximal consistent saturated sets are Hintikka sets.

▶ **Lemma 27** (Saturated MCSs are Hintikka sets). *If the set H is consistent, maximal and saturated, it is a Hintikka set:*

assumes *consistent H* **and** *maximal H* **and** *saturated H*
shows *Hintikka H*

Proof. We need to prove each case of the Hintikka definition. Take first the *FlsH* case:

show $\bot \notin H$

We need to show that falsity does not appear in our set. This follows directly from Lemma 26 and the assumed consistency of H.

Consider next the *ImpH* case:

show $(p \longrightarrow q) \in H \longleftrightarrow (p \in H \longrightarrow q \in H)$

From left to right, by using Lemma 26 this simply becomes modus ponens: if both $p \longrightarrow q$ and p are derivable from H then q must be derivable from H. The right to left direction is similar. It relies on Lemma 26, contraposition and Lemma 21: that exactly one of a formula and its negation is present in an MCS.

Consider next the *UniH* case:

show $(\forall p \in H) \longleftrightarrow (\forall t.\ \langle t/0\rangle p \in H)$

One direction follows directly from consistency of instantiation (Lemma 16) and the maximality of H. The other direction follows from saturation (and Lemma 21). ◀

6.6 Completeness Theorem

Isabelle can automatically prove the countability of our syntax:

instance *tm* :: (*countable*) *countable*
instance *fm* :: (*countable*, *countable*) *countable*

These commands provide instances of the surjective function *from-nat* that takes natural numbers and returns terms and formulas, respectively. I state the main theorem as follows.

▶ **Theorem 28** (Completeness). *Assume that formula p is valid under assumptions X and that X leaves infinitely many parameters unused. Then we can derive p from X.*

fixes $p :: ('f :: countable, 'p :: countable)\ fm$
assumes $\forall (E :: - \Rightarrow 'f\ tm)\ F\ G.\ (\forall q \in X.\ [\![E, F, G]\!]\ q) \longrightarrow [\![E, F, G]\!]\ p$
and $infinite\ (UNIV - params\ X)$
shows $\exists ps.\ set\ ps \subseteq X \wedge ps \vdash p$

Proof. By contradiction:

assume $\nexists ps.\ set\ ps \subseteq X \wedge ps \vdash p$
then have $*$: $\nexists ps.\ set\ ps \subseteq X \wedge ((\neg\ p)\ \#\ ps \vdash \bot)$

If no such list of assumptions exists, then (by classical reasoning on the object level) there is also no list that allows us to derive falsity when assuming $\neg\ p$.

I introduce some local abbreviations *?S* and *?H* (where ? is required by Isabelle):

let $?S = \{\neg\ p\} \cup X$
let $?H = Extend\ ?S\ from\text{-}nat$

It is easy to see from $*$ above that *?S* must be consistent and the extension *?H* is therefore maximal consistent (Lemmas 20, 22):

have $consistent\ ?S$
moreover have $infinite\ (UNIV - params\ ?S)$
ultimately have $consistent\ ?H$ **and** $maximal\ ?H$

?H is saturated (Lemma 23) and Hintikka (Lemma 27):

moreover from *this* **have** $saturated\ ?H$
ultimately have $Hintikka\ ?H$

The model induced by *?H* satisfies any formula in *?H* (Theorem 25), including the starting set *?S* (Lemma 17):

have $[\![?H]\!]\ p$ **if** $p \in ?S$ **for** p
then have $[\![?H]\!]\ (\neg\ p)$ **and** $\forall q \in X.\ [\![?H]\!]\ q$

But this includes all formulas in X so by the assumed validity, $[\![H]\!]$ must also satisfy p and we reach our contradiction:

moreover from *this* **have** $[\![?H]\!]\ p$
ultimately show *False*

The proof system is complete. ◀

The following abbreviation of validity in a specific Herbrand universe, with countably infinite function and predicate symbols, makes the result simpler to state:

abbreviation $valid :: (nat, nat)\ fm \Rightarrow bool$ **where**
$valid\ p \equiv \forall (E :: nat \Rightarrow nat\ tm)\ F\ G.\ [\![E, F, G]\!]\ p$

I fix the function and predicate symbols to be natural numbers but any countably infinite type works. One thing to note is that I only assume validity in one domain (the Herbrand universe), as I cannot quantify over the type I use to represent the domain. This is, however, a weaker assumption than assuming validity in all domains as is usually done.

▶ **Theorem 29** (Soundness and completeness). *Exactly the valid formulas are derivable:*

theorem *main*: $valid\ p \longleftrightarrow (\vdash p)$

Proof. By Theorems 9, 28. ◀

Only the definitions in Sections 3, 4 and Sections 5.1 to 5.5 must be inspected to trust the result. The definitions in this Section are only used for the proof.

7 Discussion

There are many choices to make in a formalization like this one. I choose to work with de Bruijn indices rather than named variables or Nominal Isabelle [42], which provides automation for this situation. While this choice makes it more complicated to explain the formalizations of e.g. semantics and quantifier instantiation, it makes the formalization self-contained. I hope to have demonstrated that the definitions themselves are simple, the functions are short and only a few simple lemmas are needed about them.

Recall the *GR* rule which is used in Lemma 15 to justify the consistency of fresh witnesses:

$$GR\text{:} \vdash q \longrightarrow \langle \star a/0 \rangle p \Longrightarrow a \notin \mathit{params}\ \{p, q\} \Longrightarrow \vdash q \longrightarrow \forall\, p$$

Since I use de Bruijn indices, this could also be formalized without the use of a parameter a by *lifting* q, in the sense of $\uparrow$, to ensure that variable 0 in p is safe to generalize directly:

$$\vdash \uparrow q \to p \Longrightarrow \vdash q \to \forall\, p$$

However, we would then need to ensure that the entire set S' in Lemma 15 is *lifted* in order to apply the rule. With the present *GR* rule, we simply ensure that a is chosen to be fresh. It would be interesting to try Laurent's anti-locally nameless approach to quantifiers [27] and see whether this would yield a simpler formalization.

Another choice has been to simulate assumptions in derivations by a chain of implications. This trick applies directly to a one-sided calculus and makes it a lot simpler to work with, especially with some custom notation. It works especially well with Smullyan's System Q1 where the generalization rule (*GR*) works under an implication. The semantic characterization of the tautology axiom, which works well with Isabelle's proof automation, makes it even smoother since propositional reasoning becomes a non-issue.

One challenge was the realization that the variant *GR'* is more suitable than *GR*. Isabelle cannot tell us something like this, nor is the proof automation powerful enough to derive the rule automatically. The insight comes from experimenting with the formalization and proofs.

Some of these issues could also be resolved by starting from a natural deduction system rather than Smullyan's Hilbert system. Natural deduction systems have a context built in, where I must simulate it with implications, and more *natural* rules for the connectives, which could be used instead of the semantic characterization of tautologies. It remains future work to adapt the formalization to this setting and review the potential benefits.

At this point in time there is a large body of formalizations to draw on. I am inspired by Berghofer's formalization [3] of the completeness of natural deduction for first-order logic. Berghofer also formalizes Lindenbaum's construction and my definition is close to his. My formalization of Hintikka sets and the model existence theorem, however, is both shorter (due to Forster et al. [11]) and, unlike Berghofer's, works directly for open formulas (cf. Herbelin and Ilik [22]). As such, even though some notion has already been formalized, it can be beneficial to revisit it.

8 Conclusion and Future Work

I have used techniques from computer science like de Bruijn indices and functional programming to work in the meta-logic of the proof assistant Isabelle/HOL. Here, I have formalized the syntax and semantics of first-order logic and defined a simple axiomatic proof system for it. This definition has included careful considerations of the interplay between syntax and semantics, a semantic characterization of tautologies suitable for formalization and notational tricks like the use of implications to simulate assumptions.

I have then carried out a completeness proof for the Hilbert system in the style of Henkin, and using ideas from Lindenbaum, Hintikka and Herbrand along the way. The proof is direct: use Lindenbaum's construction to extend a consistent set to a maximal consistent set, add Henkin witnesses of existential formulas during this construction, notice that the result is a Hintikka set and build a model in the Herbrand universe. Section 2 demonstrated the usefulness of this style in the formalization of other logics and proof systems. My formalization may serve as starting point for such endeavors: researchers can modify the existing definitions and proofs rather than start from scratch. Isabelle/HOL ensures that such modifications are correct and can help fill in gaps in the proofs when they arise. This provides an entry point to formalizing such a completeness proof.

In the future, however, I want to abstract this proof along several dimensions. First, the entire construction outlined above could potentially be given in the abstract and instantiated with a concrete proof system, witnessing function, notion of saturation, etc. Then it might be shared among the several formalizations of this method, and potential new ones. Popescu and Traytel [36] have already developed some syntax-independent logical infrastructure in their formal verification of an abstract account of Gödel's incompleteness theorems. This future work could potentially build on theirs, extending it with the model existence theorem and more. Second, Smullyan gives many constructions in his uniform notation that abstracts over the concrete choice of syntax. I would like to abstract this formalization in a similar way: witnesses could be added for "δ-formulas", which might happen to be of the form $\neg\ (\forall\ p)$ like here or maybe of the form $\Diamond\ p$ as seen in hybrid logic [26].

I also want to extend the syntax, semantics, proof system and completeness proof to first-order logic with equality. I already handle function symbols, unlike Smullyan, but to get on par with Barwise, equality needs to be considered too. The Henkin style should scale well for this extension. The current formalization does, however, have the benefit of outlining the fundamental ideas of the completeness proof without too many auxiliary considerations. This is an advantage for adapting it to other logics.

I hope this formalization will serve as inspiration, and perhaps as a starting point, for further formalizations of logic.

References

1 Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 5–46. Elsevier, 1977. doi:10.1016/S0049-237X(08)71097-8.

2 Bruno Bentzen. A Henkin-style completeness proof for the modal logic S5. In Pietro Baroni, Christoph Benzmüller, and Yì N. Wáng, editors, *Logic and Argumentation - 4th International Conference, CLAR 2021, Hangzhou, China, October 20-22, 2021, Proceedings*, volume 13040 of *Lecture Notes in Computer Science*, pages 459–467. Springer, 2021. doi:10.1007/978-3-030-89391-0_25.

3 Stefan Berghofer. First-order logic according to Fitting. *Archive of Formal Proofs*, August 2007. Formal proof development. URL: https://isa-afp.org/entries/FOL-Fitting.html.

4 Jasmin Christian Blanchette. Formalizing the metatheory of logical calculi and automatic provers in Isabelle/HOL (invited talk). In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019*, pages 1–13. ACM, 2019.

5 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Abstract completeness. *Archive of Formal Proofs*, April 2014. Formal proof development. URL: https://isa-afp.org/entries/Abstract_Completeness.html.

6 Jasmin Christian Blanchette, Andrei Popescu, and Dmitriy Traytel. Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58(1):149–179, 2017. doi:10.1007/s10817-016-9391-3.

7 Patrick Braselmann and Peter Koepke. Gödel's completeness theorem. *Formalized Mathematics*, 13(1):49–53, 2005.

8 Robert L. Constable and Mark Bickford. Intuitionistic completeness of first-order logic. *Annals of Pure and Applied Logic*, 165(1):164–198, 2014. doi:10.1016/j.apal.2013.07.009.

9 N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. In R.P. Nederpelt, J.H. Geuvers, and R.C. de Vrijer, editors, *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 375–388. Elsevier, 1994. Reprinted from: Indagationes Math, 34, 5, p. 381-392, by courtesy of the Koninklijke Nederlandse Akademie van Wetenschappen, Amsterdam. doi:10.1016/S0049-237X(08)70216-7.

10 Melvin Fitting. *First-Order Logic and Automated Theorem Proving, Second Edition*. Graduate Texts in Computer Science. Springer, 1996.

11 Yannick Forster, Dominik Kirst, and Dominik Wehr. Completeness theorems for first-order logic analysed in constructive type theory. *Journal of Logic and Computation*, 31(1):112–151, 2021. doi:10.1093/logcom/exaa073.

12 Asta Halkjær From. Synthetic completeness for a terminating Seligman-style tableau system. In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TYPES.2020.5.

13 Asta Halkjær From. Formalizing Henkin-style completeness of an axiomatic system for propositional logic. In *WESSLLI + ESSLLI Virtual Student Session*, 2021. Accepted for post-proceedings.

14 Asta Halkjær From, Anders Schlichtkrull, and Jørgen Villadsen. A sequent calculus for first-order logic formalized in Isabelle/HOL. In Stefania Monica and Federico Bergenti, editors, *Proceedings of the 36th Italian Conference on Computational Logic, Parma, Italy, September 7-9, 2021*, volume 3002 of *CEUR Workshop Proceedings*, pages 107–121. CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-3002/paper7.pdf.

15 Asta Halkjær From. Formalizing a Seligman-style tableau system for hybrid logic. *Archive of Formal Proofs*, December 2019. Formal proof development. URL: http://isa-afp.org/entries/Hybrid_Logic.html.

16 Asta Halkjær From. A sequent calculus for first-order logic. *Archive of Formal Proofs*, July 2019. Formal proof development. URL: https://isa-afp.org/entries/FOL_Seq_Calc1.html.

17 Asta Halkjær From. Soundness and completeness of an axiomatic system for first-order logic. *Archive of Formal Proofs*, September 2021. Formal proof development. URL: https://isa-afp.org/entries/FOL_Axiomatic.html.

18 Herman Geuvers. Proof assistants: History, ideas and future. *Sadhana*, 34(1):3–25, 2009. doi:10.1007/s12046-009-0001-5.

19 Kurt Gödel. Die Vollständigkeit der Axiome des logischen Funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37(1):349–360, 1930.

20 John Harrison. Formalizing basic first order model theory. In Jim Grundy and Malcolm C. Newey, editors, *Theorem Proving in Higher Order Logics, 11th International Conference, TPHOLs'98, Canberra, Australia, September 27 - October 1, 1998, Proceedings*, volume 1479 of *Lecture Notes in Computer Science*, pages 153–170. Springer, 1998. doi:10.1007/BFb0055135.

21 Leon Henkin. The discovery of my completeness proofs. *Bulletin of Symbolic Logic*, 2(2):127–158, 1996. doi:10.2307/421107.

22 Hugo Herbelin and Danko Ilik. An analysis of the constructive content of Henkin's proof of Gödel's completeness theorem. *Manuscript available online*, 2016.

23 Jacques Herbrand. *Recherches sur la théorie de la démonstration*. Number 110 in Thèses de l'entre-deux-guerres. Faculté des Sciences de L'Université de Paris, 1930.

24 Denis R. Hirschfeldt. *Slicing the Truth - On the Computable and Reverse Mathematics of Combinatorial Principles*, volume 28 of *Lecture Notes Series / Institute for Mathematical Sciences / National University of Singapore*. World Scientific, 2014. doi:10.1142/9208.

25 Danko Ilik. *Constructive Completeness Proofs and Delimited Control. (Preuves constructives de complétude et contrôle délimité)*. PhD thesis, École Polytechnique, Palaiseau, France, 2010. URL: https://tel.archives-ouvertes.fr/tel-00529021.

26 Klaus Frovin Jørgensen, Patrick Blackburn, Thomas Bolander, and Torben Braüner. Synthetic completeness proofs for Seligman-style tableau systems. In Lev D. Beklemishev, Stéphane Demri, and András Maté, editors, *Advances in Modal Logic 11, proceedings of the 11th conference on "Advances in Modal Logic," held in Budapest, Hungary, August 30 - September 2, 2016*, pages 302–321. College Publications, 2016. URL: http://www.aiml.net/volumes/volume11/Joergensen-Blackburn-Bolander-Brauner.pdf.

27 Olivier Laurent. An anti-locally-nameless approach to formalizing quantifiers. In Catalin Hritcu and Andrei Popescu, editors, *CPP '21: 10th ACM SIGPLAN International Conference on Certified Programs and Proofs, Virtual Event, Denmark, January 17-19, 2021*, pages 300–312. ACM, 2021. doi:10.1145/3437992.3439926.

28 Pierre Lescanne. From lambda-sigma to lambda-upsilon: a journey through calculi of explicit substitutions. In Hans-Juergen Boehm, Bernard Lang, and Daniel M. Yellin, editors, *Conference Record of POPL'94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon, USA, January 17-21, 1994*, pages 60–69. ACM Press, 1994. doi:10.1145/174675.174707.

29 James Margetson and Tom Ridge. Completeness theorem. *Archive of Formal Proofs*, September 2004. Formal proof development. URL: http://isa-afp.org/entries/Completeness.html.

30 Julius Michaelis and Tobias Nipkow. Propositional proof systems. *Archive of Formal Proofs*, June 2017. Formal proof development. URL: http://isa-afp.org/entries/Propositional_Proof_Systems.html.

31 Julius Michaelis and Tobias Nipkow. Formalized Proof Systems for Propositional Logic. In Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi, editors, *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*, volume 104 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.TYPES.2017.5.

32 Tobias Nipkow. More Church-Rosser proofs. *Journal of Automated Reasoning*, 26(1):51–66, 2001. doi:10.1023/A:1006496715975.

33 Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014. doi:10.1007/978-3-319-10542-0.

34 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45949-9.

35 Henrik Persson. Constructive completeness of intuitionistic predicate logic. *Licenciate thesis, Chalmers University of Technology*, 1996.

36 Andrei Popescu and Dmitriy Traytel. A formally verified abstract account of Gödel's incompleteness theorems. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 442–461. Springer, 2019. doi:10.1007/978-3-030-29436-6_26.

37 Anders Schlichtkrull. The resolution calculus for first-order logic. *Archive of Formal Proofs*, June 2016. Formal proof development. URL: http://isa-afp.org/entries/Resolution_FOL.html.

38 Anders Schlichtkrull. Formalization of the resolution calculus for first-order logic. *Journal of Automated Reasoning*, 61(1-4):455–484, 2018. doi:10.1007/s10817-017-9447-z.

39 Natarajan Shankar. Towards mechanical metamathematics. *Journal of Automated Reasoning*, 1(4):407–434, 1985.

40 Raymond M. Smullyan. *First-Order Logic*. Springer-Verlag, 1968.

41 Alfred Tarski. *Logic, Semantics, Metamathematics: Papers from 1923 to 1938*. Hackett Publishing, 1983.

42 Christian Urban. Nominal techniques in Isabelle/HOL. *Journal of Automated Reasoning*, 40(4):327–356, 2008. doi:10.1007/s10817-008-9097-2.

43 Wim Veldman. An intuitionistic completeness theorem for intuitionistic predicate logic. *Journal of Symbolic Logic*, 41(1):159–166, 1976. doi:10.1017/S0022481200051859.