Verified completeness in Henkin-style for intuitionistic propositional logic

Huayu Guo [1] Dongheng Chen [2] Bruno Bentzen [3]

School of Philosophy, Zhejiang University
Hangzhou
China

Abstract

This paper presents a formalization of the classical proof of completeness in Henkin-style developed by Troelstra and van Dalen for intuitionistic logic with respect to Kripke models. The completeness proof incorporates their insights in a fresh and elegant manner that is better suited for mechanization. We discuss details of our implementation in the Lean theorem prover with emphasis on the prime extension lemma and construction of the canonical model. Our implementation is restricted to a system of intuitionistic propositional logic with implication, conjunction, disjunction, and falsity given in terms of a Hilbert-style axiomatization. As far as we know, our implementation is the first verified Henkin-style proof of completeness for intuitionistic logic following Troelstra and van Dalen's method in the literature. The full source code can be found online at https://github.com/bbentzen/ipl.

Keywords: Intuitionistic propositional logic, Henkin completeness, Formal proofs, Lean.

1 Introduction

Troelstra and van Dalen [17] propose a completeness proof in Henkin-style for full intuitionistic predicate logic with respect to Kripke models. Despite being a fairly standard result in the literature, this completeness proof has yet to be formally verified in a proof assistant. In this paper, we describe a formalization for intuitionistic propositional logic using the Lean theorem prover [13].

Our main goal is to document some challenges encountered along the way and the design choices made to overcome them to obtain a formalized proof that is elegant, intuitive, and better suited for mechanization using the specific techniques available in the Lean programming language, in particular, the `encodable.decode` and `insert_code` methods developed by Bentzen [1].

[1] guohuayu@zju.edu.cn
[2] chen_dongheng@zju.edu.cn
[3] bbentzen@zju.edu.cn

To the best of our knowledge, our implementation is the first verified Henkin-style proof of strong completeness for intuitionistic logic following Troelstra and van Dalen's method in the literature. As far as its propositional fragment is concerned, the main ingredient of Troelstra and van Dalen's Henkin-proof is a model construction based on a consistent extension of sets of formulas, which is achieved by going through all disjunctions of the language [17, lem 6.3]. To carry out this extension, they assume an enumeration of disjunctions with infinite repetitions, also remarking that an alternative approach in which at each stage we treat the first disjunction not yet treated. This variant appears in Van Dalen [5, lem 5.3.8]. Our implementation is based on a third variant of the consistent extension method, which we developed to better suit our needs of formalization. Each propositional formula is only listed once in the enumeration, but we carry out the extension for each of them infinitely many times. The formalization consists of roughly 800 lines of code and encompasses the syntax and semantics of intuitionistic propositional logic, along with the soundness and strong completeness theorems. We adopt a Hilbert-style proof system due to its simplicity. The full source code can be found online at https://github.com/bbentzen/ipl.

1.1 Related work

The formal verification of completeness proofs for intuitionistic logic can be traced back to Coquand's [3] use of ALF to mechanize a constructive proof of soundness and completeness with respect to Kripke models for the simply typed lambda-calculus with explicit substitutions. Heberlin and Lee [9] give a constructive completeness proof of Kripke semantics with constant domain for intuitionistic logic with implication and universal quantification in Coq. Recently, Hagemeier and Kirst [8] formalize a constructive proof of completeness for intuitionistic epistemic logic based on a natural deduction system. They also provide a classical Henkin proof using methods similar to those in Bentzen [1], but they do not present a formalization of the approach of Troelstra and van Dalen [17] as is done in this paper. Bentzen [1] formalizes the Henkin-style completeness method for modal logic S5 using Lean and From formalizes in Isabelle/HOL a Henkin-style completeness proof for both classical propositional logic [6] and classical first-order logic [7]. Maggesi and Brogi [12] give a formal completeness proof for provability logic in HOL Light. The formalization presented here is inspired by the work of Bentzen [1], but makes a few improvements regarding design choices, in particular, the use of Prop in the definition of the semantics and the indexing of models to arbitrary types.

1.2 Lean

Lean [13] is an interactive theorem prover based on the version of dependent type theory known as the calculus of constructions with inductive types [15,4]. Users can construct proof terms directly as in Agda [14], using tactics as in Coq [16] or both proof terms and tactics simultaneously. Lean's built-in logic is constructive, but it supports classical reasoning as well. In fact, our Henkin-style proof is classical since it relies on a nonconstructive use of contraposition.

Therefore, we do not worry about any complexity and computational aspects related to our proof. Our implementation makes use of some results from Lean's standard library and the user-maintained mathematical library `mathlib` [2].

Throughout the remainder of this paper, Lean code will be used to showcase some design decisions in our formalization. The syntax and semantics of intuitionistic propositional logic that is the starting point of our formalization is described in Section 2. We also describe our formalization of a countermodel for the law of excluded middle and sketch a proof of soundness. Then, an informal overview of the Henkin-style proof method as well as a description of our implementation is provided in Section 3. Finally, some concluding remarks are given in Section 4.

2 Intuitionistic Logic

2.1 The language

The intuitionistic propositional language considered here contains implication, conjunction, disjunction, and falsity as the only primitive logical connectives. The language is defined using inductive types with one constructor for propositional letters, falsum, implication, conjunction, and disjunction, respectively:

```
inductive form : Type
| atom : ℕ → form
| bot  : form
| impl : form  → form → form
| and  : form  → form → form
| or   : form  → form → form
```

This code can be found in `language.lean` file.

Since our language contains countably many propositional letters $p_0, p_1, \ldots$ we use the type ℕ of natural numbers to define the constructor `atom` of propositional letters. The only way to construct a term of type `form` is using this atomic constructor(`atom`) and the constructors for falsum (`bot`), implication (`impl`), conjunction (`and`), disjunction (`or`).

The elimination rule is an operation that allows us to define functions by recursion from it to any other types, including also the type of propositions `Prop`, in which case, this elimination rule is an instance of the principle of induction on the structure of the formula.

Constructors are displayed in Polish notation by default, but we define some custom infix notation with the usual Unicode characters for better readability:

```
prefix   `#`       := form.atom
notation `⊥`       := form.bot
infix    `⊃`       := form.impl
notation p `&` q   := form.and p q
notation p `∨` q   := form.or p q
notation `~`:40 p  := form.impl p (form.bot )
```

Contexts are just sets of formulas. In Lean sets are defined as functions of type `A` → `Prop`. As usual in logic textbooks, we display the formulas in a context in list notation separated by a comma instead of using unions of singletons. We introduce the following notation to make this possible:

```
notation Γ ` 、 ` p :=set.insert p Γ
```

The formalization of the language can be found in the `language.lean` file.

2.2 The proof system

We define a Hilbert-style system for intuitionistic propositional logic that is best described as a refinement of Heyting's original axiomatization [10, §2]. The proof system is implemented with a type of proofs, which is inductively defined as follows:

```
inductive prf : set form → form → Prop
| ax {Γ} {p} (h : p ∈ Γ) :prf Γ p
| k {Γ} {p q} : prf Γ (p ⊃ (q ⊃ p))
| s {Γ} {p q r} : prf Γ ((p ⊃ (q ⊃ r)) ⊃ ((p ⊃ q) ⊃ (p ⊃ r))
  )
| exf {Γ} {p} : prf Γ (⊥ ⊃ p)
| mp {Γ} {p q} (hpq: prf Γ (p ⊃ q)) (hp :prf Γ p) : prf Γ q
| pr1 {Γ} {p q} : prf Γ ((p & q) ⊃ p)
| pr2 {Γ} {p q} : prf Γ ((p & q) ⊃ q)
| pair {Γ} {p q} : prf Γ (p ⊃ (q ⊃ (p & q)))
| inr {Γ} {p q} : prf Γ (p ⊃ (p ∨ q))
| inl {Γ} {p q} : prf Γ (q ⊃ (p ∨ q))
| case {Γ} {p q r} : prf Γ ((p ⊃ r) ⊃ ((q ⊃ r) ⊃ ((p ∨ q) ⊃ r
  )))
```

Again, the elimination rule for this type generalizes definition by recursion and induction on the structure of proofs. To follow the usual logical notation, we abbreviate `prf Γ p` with $\Gamma \vdash_i p$ as follows:

```
notation Γ ` ⊢ᵢ ` p :=prf Γ p
notation Γ ` ⊬ᵢ ` p :=prf Γ p → false
```

To illustrate, we compare a mechanized formal Hilbert-style proof of the identity of implication $p \supset p$ in our implementation:

```
lemma id {p : form } {Γ : set form } :
| Γ ⊢ᵢ p ⊃ p :=
mp (mp (@s  Γ p (p ⊃ p) p) k) k
```

with a non-mechanized formal proof written in Lemmon style:

1	$p \supset ((p \supset p) \supset p) \supset (p \supset (p \supset p)) \supset (p \supset p)$	S
2	$p \supset ((p \supset p) \supset p)$	K
3	$(p \supset (p \supset p)) \supset (p \supset p)$	MP 1, 2
4	$(p \supset (p \supset p))$	K
5	$p \supset p$	MP 3, 4

Notice that the proof structure in our term proof is actually clearer since it indicates how the axiom schemes should be instantiated.

The formalization of the proof system can be found in the `theory.lean` file.

2.3 Semantics

2.3.1 Kripke models

We define the semantics for intuitionistic propositional logic in terms of Kripke semantics as usual [17,5]. A model $\mathcal{M}$ is a triple $\langle \mathcal{W}, \leq, \mathsf{v} \rangle$ where $\mathcal{W}$ is a set of possible worlds of type A, $\leq$ is a reflexive, symmetric and monotonic binary relation on A, and v specifies the truth value of a formula at a world.

In Lean, Kripke models can be defined as inductive types having just one constructor using the structure command. We define it not as a triple but as a 6-tuple, composed of a domain `W`, an accessibility relation `R`, a valuation function `val`, and proofs of reflexivity, transitivity, and monotonicity for the accessibility relation `R`, denoted as `refl`, `trans`, and `mono`:

```
structure model (A : Type) :=
| (W : set A)
| (R : A  → A → Prop)
| (val : ℕ → A → Prop)
| (refl : ∀ w ∈ W, R w w)
| (trans : ∀ w ∈ W, ∀ v ∈ W, ∀ u ∈ W, R w v → R v u → R w u)
| (mono : ∀ p, ∀ w1 w2 ∈ W, val p w1 → R w1 w2 → val p w2)
```

In our case, a possible world is a term of type A. This allows for more generality in the construction of a model unlike in [1]. What is more, the type of propositions Prop is used to encode our truth values `true` or `false`.

2.3.2 Semantic consequence

To formalize the notion of truth at a type, we define a forcing relation $w \Vdash_{\mathcal{M}} p$ that takes as arguments a model $\mathcal{M}$, a formula p, and a type A and returns a term of type Prop. As usual, falsity, conjunction, and disjunction are defined truth-functionally and an implication $p \supset q$ is true at a world w iff if $\mathcal{R}(w, v)$ then p is true implies q is true at v, for all $v \in \mathcal{W}$. We also introduce the familiar notation for this forcing relation:

```
def forces_form {A : Type} (M : model A) : form → A → Prop
| (#p)      := λv, M.val p v
| (bot)     := λv, false
| (p ⊃ q)   := λv, ∀ w ∈ M.W, v ∈ M.W → M.R v w
→ forces_form p w → forces_form q w
| (p & q)   := λv, forces_form p v ∧ forces_form q v
| (p ∨ q)   := λv, forces_form p v ∨ forces_form q v

notation w `⊩ ` `{` M `} ` p := forces_form M p w
```

To formalize the intuitionistic notion of semantic consequence $\Gamma \vDash_i p$ we first extend this forcing relation to contexts pointwise and then we stipulate that $\Gamma \vDash_i p$ iff for all types A, models $\mathcal{M}$ and possible worlds $w \in \mathcal{W}$, Γ being true at w in $\mathcal{M}$ implies p being true at w in $\mathcal{M}$:

```
def forces_ctx {A : Type} (M : model A) (Γ : set form) : A →
    Prop :=
λw, ∀ p, p ∈ Γ → forces_form M p w

notation w `⊩` `{` M `} ` Γ :=forces_ctx M Γ w
```

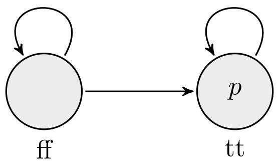
```
def sem_csq (Γ : set form) (p : form) :=
∀ {A :Type} (M : model A) (w ∈ M.W), (w ⊩ {M} Γ) → (w ⊩ {M} p
    )

notation Γ `⊨ᵢ` p :=sem_csq Γ p
```

It is worth noting that we are overloading the forcing relation notation for formulas `w ⊩ {M}` p and contexts `w ⊩ {M}` Γ. There is no ambiguity because Lean will delay the choice until elaboration and determine how to disambiguate the notations depending on the relevant types.

The formalization of the Kripke semantics described above can be found in the `semantics.lean` file.

2.3.3 The failure of the law of excluded middle

Before proceeding to prove completeness, it will be helpful to see how we can build models in our implementation. To give a concrete example, let us show how to build the following countermodel for the law of excluded middle [11, p.99] using the type of booleans true `tt` and false `ff`:



Since our possible worlds are always booleans, the domain, accessibility relation, and valuation function are formalized in Lean in a slightly different way. The reflexivity, transitivity, and monotonicity proofs are straightforward, so we shall omit them:

```
def W : set bool := {ff, tt}

def R : bool → bool → Prop :=λ w v, w =v ∨ w =ff

@[simp]
def val : nat → bool → Prop :=λ _ w, w =tt
```

Using this countermodel, we assume that the law of excluded middle holds, that is for any formula p, either $\emptyset \models_i p$ or $\emptyset \models_i \neg p$, and then derive a contradiction. This allows us to prove that the law of excluded middle fails in general:

```
lemma no_lem: ¬ ∀ p, (∅ ⊨ᵢ p ∨ ~p)
```

The mechanization of the countermodel can be found in the `nolem.lean` file.

2.3.4 Soundness

The soundness theorem asserts that if a formula p can be derived from a set of assumptions Γ using the inference rules of the logical system, then p is logically valid under any interpretation that satisfies Γ.

```
theorem soundness {Γ : set form} {p : form} :
(Γ ⊢ᵢ p) → (Γ ⊨ᵢ p)
```

The code for proof of soundness can be found in `soundness.lean`.

The proof proceeds by using induction to perform case analysis for each inference rule. For each rule, the proof provides a way to derive the conclusion based on the rule and a way to show that the conclusion is logically valid based on the interpretation and the premises.

3 The completeness theorem

Now that we have presented the implementation of the syntax and semantics of intuitionistic propositional logic in the previous section, we are prepared to undertake a formal proof of completeness. The strong completeness theorem, which states that every semantic consequence is a syntactic consequence, can be stated in Lean using our custom notation as follows:

```
theorem completeness {Γ : set form} {p : form} :
(Γ ⊨ᵢ p) → (Γ ⊢ᵢ p)
```

Our implementation follows the original Henkin-style completeness proof given by Troelstra and van Dalen [17] with some small modifications. The main proof argument runs as follows.

(i) Assume that $\Gamma \vDash_i p$ and $\Gamma \nvdash_i p$ hold;

(ii) Build a model $\mathcal{M}$ such that $w \Vdash_{\mathcal{M}} p$ iff $w \vdash_i p$ for all worlds $w \in \mathcal{W}$, where we have sets of formulas as possible worlds;

(iii) Show that there is a world $w \in \mathcal{W}$ such that $w \Vdash_{\mathcal{M}} \Gamma$ but $w \nVdash_{\mathcal{M}} p$;

(iv) Establish a contradiction from our assumption that $\Gamma \vDash_i p$.

Our proof appeals to classical reasoning at the metalevel of Lean's logic on two occasions [17, p.87], namely, in our proof of $\Gamma \vdash_i p$ where we assume double negation elimination and in our proof of $w \Vdash_{\mathcal{M}} p$ iff $w \vdash_i p$.

The reader can refer to the `completeness.lean` file for the full details of our implementation of the completeness proof.

3.0.1 Consistent prime extensions

The first step of Troelstra and van Dalen's proof is the definition of what they call a "saturated theory" [17, def.6.2]. We shall make use of the equivalent concept of prime theory instead [5, def.5.3.7], in which the disjunction property is expressed in terms of the membership relation. We say that a set of formulas Γ is a prime theory if Γ is closed under derivability and if $p \vee q \in \Gamma$ implies $p \in \Gamma$ or $q \in \Gamma$. In `completeness.lean` file, we write:

```
def is_closed (Γ : set form) :=
∀ {p :form}, (Γ ⊢ᵢ p) → p ∈ Γ

def has_disj (Γ : set form) :=
∀ {p q :form}, ((p ∨ q) ∈ Γ) → ((p ∈ Γ) ∨ (q ∈ Γ))
```

```
def is_prime (Γ : set form) :=
is_consist Γ ∧ has_disj Γ
```

The second step of Troelstra and van Dalen's completeness proof is the proof of a prime extension lemma [17, lem 6.3], which states that if $\Gamma \nvdash r$ then there is a prime theory $\Gamma' \supseteq \Gamma$ such that $\Gamma' \nvdash r$. Assuming that they have a list of disjunctions $\langle \varphi_{i,1} \vee \varphi_{i,2} \rangle_i$ with infinite repetitions, they define

$$\Gamma' = \bigcup_{i \in \mathbb{N}} \Gamma_i,$$

where $\Gamma_0 = \Gamma$ and Γ_{k+1} is defined inductively as follows:

- Case 1: $\Gamma_k \vdash \varphi_{k,1} \vee \varphi_{k,2}$. Put
  - $\Gamma_{k+1} = \Gamma_k \cup \{\varphi_{k,2}\}$ if $\Gamma_k, \varphi_{k,1} \vdash r$, and
  - $\Gamma_{k+1} = \Gamma_k \cup \{\varphi_{k,1}\}$ otherwise
- Case 2: $\Gamma_k \nvdash \varphi_{k,1} \vee \varphi_{k,2}$. Put
  - $\Gamma_{k+1} = \Gamma_k$

Since we want to extend Γ to a prime theory Γ', we want to ensure the disjunctive property that if $\phi \vee \psi \in \Gamma'$ then $\phi \in \Gamma'$ or $\psi \in \Gamma'$. If there were no infinite repetitions in the list, we could never be sure that we have treated all disjunctions in Case 1, for, at step $k+1$, its disjuncts only get added to the set when Γ_k proves the disjunction. It is possible that later the disjunction becomes provable from Γ_{k+m}, but, we will never go back to it again.

Troelstra and van Dalen mention a simpler variant of the construction that uses an enumeration of disjunctions without requiring infinite repetitions. At stage $k+1$ we simply treat the first disjunction not yet treated. This proof is spelled out by van Dalen in [5, lem 5.3.8]. However, the proof method is less suitable for mechanization given that it is difficult to tell a proof assistant how exactly they should find the first disjunction not yet treated. We implement a simplified version of this method where at each step $k+1$ we always treat all disjunctions in the language once more. The following Lean code encapsulates the idea of the construction sketched above:

```
def insert_form (Γ :  set form) (p q r : form) :  set form :=
if (Γ ˎ p ⊢ᵢ r) then Γˎ q else Γˎ p

def insert_code (Γ : set form) (r : form) (n : nat) :  set form
     :=
match encodable.decode (form) n with
| none   := Γ
| some (p ∨ q) :=if Γ ⊢ᵢ p ∨ q then insert_form Γ p q r else Γ
| some _ := Γ
end
```

```
def insertn (Γ : set form) (r : form) : nat → set form
| 0     := Γ
| (n+1) := insert_code (insertn n) r n

def primen (Γ : set form) (r : form) : nat → set form
| 0     := Γ
| (n+1) := ⋃ i, insertn (primen n) r i

def prime (Γ: set form) (r : form) :  set form :=
⋃ n, primen Γ r n
```

Unlike in Troesltra and van Dalen [17] and van Dalen [5], the enumeration in our formalization lists not just all disjunctions but all propositional formulas in the language. When a formula is not a disjunction we simply ignore it just as in Case 2 above. We follow Bentzen [1] in using `encodable` types to enumerate the language. In Lean, a type α is encodable if there is an encoding function `encode :`α `→ nat` and a (partial) inverse `decode :nat → option` α that decodes the encoded term of α.

Now that we extended Γ to Γ', which we denote as `prime Γ r`, we have to prove it is indeed a prime extension of Γ. First, we show that $\Gamma \subseteq \Gamma'$. But this is easy, since for every Γ'_n `n` in the family of sets, $\Gamma \subseteq \Gamma'_n$ `n`. Therefore, Γ must also be included in the union of all Γ'_n `n`, which is Γ'_n.

```
lemma primen_subset_prime {Γ : set form} {r : form} (n):
primen Γ r n ⊆ prime Γ r

lemma subset_prime_self {Γ : set form} {r : form} :
Γ ⊆ prime Γ r
```

The next step is to prove that the Γ' also has the disjunction property and it is closed under derivability. Let us focus on the former first.

We need to show that $p \vee q \in \Gamma'$ implies $p \in \Gamma'$ or $q \in \Gamma'$. If $p \vee q \in \Gamma'$ then there is some $n \in \mathbb{N}$ such that $p \vee q \in \Gamma'_n$. But then since $\Gamma'_n \vdash p \vee q$, then we know that $p \in \Gamma'_{n+1}$ or $q \in \Gamma'_{n+1}$ because the disjunction was treated at some point. Thus, $p \in \Gamma'$ or $q \in \Gamma'$.

```
def prime_insertn_disj {Γ: set form} {p q r : form} (h : (p ∨
    q)  ∈ prime Γ r) :
∃ n, p ∈ (insertn (primen Γ r n) r (encodable.encode (p  q)+1))
     ∨ q ∈ (insertn (primen Γ r n) r (encodable.encode (p ∨ q)
    +1))

lemma insertn_to_prime {Γ : set form} {r : form} {n m : nat} :
insertn (primen Γ r n) r m ⊆ prime Γ r

def prime_has_disj {Γ : set form} {p q r : form} :
((p ∨ q) ∈ prime Γ r) → p ∈ prime Γ r ∨ q ∈ prime Γ r
```

Saying that Γ' is closed under derivability means that if we can deduce a formula from Γ', it is an element of Γ'. We use a lemma that states that if we can prove $r \vee p$ from Γ', then there exists an n such that $p \in \Gamma_{n+1}$. We use the above lemma `insertn_to_prime` to deduce that $p \in \Gamma'$:

```
lemma prime_prf_disj_self {Γ : set form} {p r : form} :
(prime Γ r ⊢ᵢ r ∨ p) → ∃ n, p ∈ (insertn (primen Γ r n) r (
    encodable.encode (r ∨ p)+1))

def prime_is_closed {Γ : set form} {p q r : form} :
(prime Γ r ⊢ᵢ p) → p ∈ prime Γ r
```

At this moment, we need to prove that Γ' still remains consistent. First, we by structural induction on the derivation that if $\Gamma' \vdash r$ then there is some n such that $\Gamma_n \vdash r$. Then we prove by induction on n that if $\Gamma_n \vdash r$ then $\Gamma \vdash r$. The base case is trivial. In the inductive case, we complete the proof by unfolding the definition of Γ_n and manipulating the inductive hypothesis. Putting both lemmas together, we prove that $\Gamma' \vdash r$ implies $\Gamma \vdash r$:

```
def primen_not_prfn {Γ : set form} {r : form} {n} :
(primen Γ r n ⊢ᵢ r) → (Γ ⊢ᵢ r)

def prime_not_prf {Γ : set form} {r : form} :
(prime Γ r ⊢ᵢ r) → (Γ ⊢ᵢ r)
```

3.0.2 The canonical model construction

Given a set of formulas Γ and ϕ such that $\Gamma \nvdash \phi$, the next step is to build a canonical Kripke model $\mathcal{M}$ such that with $w \Vdash_{\mathcal{M}} \Gamma$ and $w \nVdash_{\mathcal{M}} \phi$ for some possible world. We build this model by letting $\mathcal{W}$ be the set of all consistent prime theories; $w \leq v$ iff $w \subseteq v$ for $w, v \in \mathcal{W}$; and $\mathsf{v}(w, p) = 1$ iff $w \in \mathcal{W}$ and $p \in w$, for a propositional letter p. The following Lean code reflects the model construction:

```
def domain : set (set form) := {w | is_consist w ∧ ctx.
    is_prime w}

def access : set form → set form → Prop :=λ w v, w ⊆ v

def val : ℕ → set form → Prop :=λ q w, w ∈ domain ∧ (#q) ∈ w
```

The accessibility relation $\leq$ is clearly reflexive and transitive since so is $\subseteq$. Monotonicity is easy to see since $p \in w$ and $w \subseteq v$ means that $q \in v$. We prove these lemmas by straightforward unfolding the definition of `access`.

Our model is integrated into Lean's code as follows:

```
def M : model (set form):=
begin
fapply model.mk,
apply domain,
apply access,
apply val,
apply access.refl,
apply access.trans,
apply access.mono
end
```

3.0.3 Truth and derivability

It turns out that a formula is true at a world in the canonical model if and only if it can be proved from that world:

```
lemma model_tt_iff_prf {p : form} :
∀ (w ∈ domain), (w ⊨ {M} p) ↔ (w ⊢ᵢ p)
```

We mechanize the proof employing the `induction` tactic, which allows us to use the elimination rule of a type. This approach yields five goals, namely, to prove the case where a formula is a propositional letter, falsity, implication, conjunction, or disjunction. The proof of implication and disjunction deserve some mention.

The disjunction case is simpler, so we shall discuss it first. Lean gives us a biconditional in the following goal:

```
⊢ ∀ (w :set form),
w ∈ domain → (w ⊨ {M} (p ∨ q)) ↔ (w ⊢ᵢ p ∨ q))
```

The proof in the forward direction starts with the introduction of assumptions and then splits the proof into two cases. In the first case, we assume that $w \models_{\mathcal{M}} p \vee q$ and our goal is $w \vdash_i p \vee q$. Through the tactic `cases`, which expresses case reasoning, we can finish our goal using some basic facts about disjunctions and the inductive hypotheses in both cases.

In the backward direction, we assume that $w \vdash_i p \vee q$. Since w is a prime theory and thus enjoys the disjunctive property, we can reason by cases depending on whether $w \vdash_i p$ or $w \vdash_i q$. The result follows the inductive hypothesis.

Now we proceed to the implication case. Using the `intro` tactic, we begin by assuming the inductive hypothesis for p. If w is a world and it is a prime theory, then by unfolding the true definition of a formula in the model's world, we arrive at a biconditional goal that can be expressed as follows.

```
⊢ ∀ (w :set form),
w ∈ domain → (w ⊨ᵢ {M} (p ⊃ q)) ↔ (w ⊢ᵢ p ⊃ q))
```

We split the biconditional proof into two smaller conditionals using the `split` tactic. In the forward direction, we first assume that $w \Vdash_{\mathcal{M}} p \supset q$. We reason by cases depending on whether $w \vdash_i p \supset q$ or not, therefore invoking the law of excluded middle. If that is the case, we are done. If not, then we know that $w, p \nvdash q$. We want to derive a contradiction. We extend the context w, p to a prime theory $(w, p)'$ that still does not prove q. By our inductive hypothesis, since $(w, p)'$ is in the domain, we know that $(w, p)' \Vdash_{\mathcal{M}} q \leftrightarrow (w, p)' \vdash_i q$.

To derive a contradiction, we just have to show that $(w, p)' \Vdash_{\mathcal{M}} q$. Recall that our assumption $w \Vdash_{\mathcal{M}} p \supset q$ states that for all $v \in \mathcal{W}$ such that $w \leq v$, if $v \Vdash_{\mathcal{M}} p$ then $v \Vdash_{\mathcal{M}} q$. But, clearly, $w \leq (w, p)'$. To complete the proof, we just have to show that $(w, p)' \Vdash_{\mathcal{M}} p$. By our inductive hypothesis, it suffices to show that $(w, p)' \vdash_i p$. But this is clearly true, since the original set w, p is contained in the prime extension $(w, p)'$ and $w, p \vdash_i p$.

For the backward direction, what we have to prove is $w \Vdash_{\mathcal{M}} p \supset q$. This means for all $v \in \mathcal{W}$ such that $w \leq v$, if $v \Vdash_{\mathcal{M}} p$ then $v \Vdash_{\mathcal{M}} q$. We assume that

$v \in \mathcal{W}$ such that $w \leq v$, $v \Vdash_{\mathcal{M}} p$ then we have to show $v \Vdash_{\mathcal{M}} q$. Using our inductive hypothesis, we just have to show that $v \vdash_i q$.

Since we know $w \vdash_i p \supset q$ and $w \subseteq v$, by weakening, we will have $v \vdash_i p \supset q$. We complete the proof by noting that $v \vdash_i p$ by our inductive hypothesis and assumption that $v \Vdash_{\mathcal{M}} p$. The result follows from modus ponens.

We have finished the proof of implication.

3.0.4 The completeness proof

To finish our completeness proof we just have to put together all the above pieces into 27 lines of code. We assume that $\Gamma \nvdash_i p$ and $\Gamma \models_i p$, we just need to arrive at a contradiction. We extend Γ to a prime theory Γ' such that $\Gamma' \nvdash_i p$. Since we know $\Gamma' \Vdash_{\mathcal{M}} q \iff \Gamma' \vdash_i q$ for every formula q, we can conclude that $\Gamma' \nVdash_{\mathcal{M}} p$. Thus, we contradict our assumption that $\Gamma \models_i p$, given that $\Gamma' \Vdash_{\mathcal{M}} \Gamma$ but $\Gamma' \nVdash_{\mathcal{M}} p$.

4 Conclusion

We have used Lean to formally verify the Henkin-style completeness proof for intuitionistic logic proposed by Troesltra and van Dalen [17] restricted to a propositional fragment with implication, falsity, conjunction, disjunction. The propositional proof system we implement is based on a Hilbert-style axiomatization. In future work, we hope to expand our implementation to full intuitionistic first-order logic with existential and universal quantifiers and thus complete the formalization of Troesltra and van Dalen's proof. Our implementation also includes a mechanized proof of soundness and a countermodel for the general validity of the law of excluded middle in intuitionistic propositional logic.

Acknowledgments This research was supported in part by the Zhejiang Federation of Humanities and Social Sciences grant 23YJRC04ZD.

References

[1] Bentzen, B.: A Henkin-style completeness proof for the modal logic S5. In: Logic and Argumentation: 4th International Conference, CLAR 2021, Hangzhou, China, October 20–22, 2021, Proceedings 4. pp. 459–467. Springer (2021)

[2] Carneiro, M.: The Lean 3 Mathematical Library (mathlib). URL: https://robertylewis.com/files/icms/Carneiro_mathlib.pdf (2018), international Congress on Mathematical Software

[3] Coquand, C.: A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. Higher-Order and Symbolic Computation **15**(1), 57–90 (2002), uRL: https://doi.org/10.1023/A:1019964114625

[4] Coquand, T., Huet, G.: The Calculus of Constructions. Information and Compututation **76**(2-3), 95–120 (1988), uRL: https://hal.inria.fr/inria-00076024/document

[5] van Dalen, D.: Logic and structure, vol. 5. Springer (2013)

[6] From, A.H.: Formalizing Henkin-style completeness of an axiomatic system for propositional logic. Proceedings of the ESSLLI & WeSSLLI Student Session pp. 1–12 (2020)

[7] From, A.H.: A succinct formalization of the completeness of first-order logic. In: 27th International Conference on Types for Proofs and Programs (TYPES 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2022)

[8] Hagemeier, C., Kirst, D.: Constructive and mechanised meta-theory of intuitionistic epistemic logic. In: Logical Foundations of Computer Science: International Symposium, LFCS 2022, Deerfield Beach, FL, USA, January 10–13, 2022, Proceedings. pp. 90–111. Springer (2022)

[9] Herbelin, H., Lee, G.: Forcing-based cut-elimination for Gentzen-style intuitionistic sequent calculus. In: H., O., M., K., de Queiroz R. (eds.) International Workshop on Logic, Language, Information, and Computation. pp. 209–217. Springer, Berlin, Heidelberg (2009), uRL: https://doi.org/10.1007/978-3-642-02261-6_17

[10] Heyting, A.: Die formalen Regeln der intuitionistischen Logik. Sitzungsbericht PreuBische Akademie der Wissenschaften Berlin, physikalisch-mathematische Klasse II pp. 42–56 (1930)

[11] Kripke, S.A.: Semantical analysis of intuitionistic logic I. In: Studies in Logic and the Foundations of Mathematics, vol. 40, pp. 92–130. Elsevier (1965)

[12] Maggesi, M., Brogi, C.P.: A formal proof of modal completeness for provability logic. arXiv preprint arXiv:2102.05945 (2021)

[13] de Moura, L., Kong, S., Avigad, J., Van Doorn, F., von Raumer, J.: The Lean theorem prover (system description). In: Felty, A., Middeldorp, A. (eds.) International Conference on Automated Deduction. pp. 378–388. Springer, Cham (2015), uRL: https://doi.org/10.1007/978-3-319-21401-6_26

[14] Norell, U.: Dependently typed programming in Agda. In: Koopman, P., Plasmeijer, R., Swierstra, D. (eds.) International School on Advanced Functional Programming. pp. 230–266. Springer, Berlin, Heidelberg (2008), uRL: https://doi.org/10.1007/978-3-642-04652-0_5

[15] Pfenning, F., Paulin-Mohring, C.: Inductively defined types in the Calculus of Constructions. In: International Conference on Mathematical Foundations of Programming Semantics. pp. 209–228. Springer (1989)

[16] The Coq project: The Coq proof assistant. URL: http://www.coq.inria.fr (2017)

[17] Troelstra, A.S., van Dalen, D.: Constructivism in mathematics. Vol. I, Studies in Logic and the Foundations of Mathematics, vol. 121. North-Holland, Amsterdam (1988)