Formalizing Henkin-Style Completeness of an Axiomatic System for Propositional Logic

Asta Halkjær From

Technical University of Denmark

WeSSLLI + ESSLLI Virtual Student Session, 13 July 2020.

Introduction

Hilbert proved the completeness of an axiomatic system for propositional logic in 1917-18 [34], Gödel proved the completeness of first-order logic in 1929 [12] and Henkin simplified this proof in 1947 [13].

We study the structure of a Henkin-style completeness proof for an axiomatic Hilbert system for propositional logic by formalizing it in the proof assistant Isabelle/HOL [21].

- A history of formalized completeness proofs.
- Isabelle primer.
- Propositional logic and axiom system.
- Henkin-style completeness.
- Conclusion.

A History of Formalized Completeness Proofs I

In 1985, Shankar formalizes propositional completeness wrt. an axiomatic proof system in the Boyer-Moore theorem prover by defining a tautology checker [28].

In 1996, Persson shows constructive completeness for intuitionistic first-order logic in Martin-Löf type theory using the proof assistant ALF [24].

By early 2000, Margetson formalizes the completeness of first-order logic and the cut elimination theorem for sequent calculus in Isabelle/HOL [18].

In 2005, Braselmann and Koepke follow in the Mizar system [6].

In 2007, Berghofer formalizes Fitting's work on natural deduction [8] in Isabelle [3].

A History of Formalized Completeness Proofs II

In 2010, Ilik investigates Henkin-style arguments for both classical and intuitionistic first-order logic in the proof assistant Coq [15].

In 2017, Michaelis and Nipkow formalize a number of proof systems for propositional logic in Isabelle/HOL: natural deduction, sequent calculus, an axiomatic system similar to ours and resolution [19,20].

Blanchette, Popescu and Traytel use codatatypes to model possibly infinite derivation trees for first-order sequent calculus and tableau systems in Isabelle [5].

Jørgensen et al. adapted the synthetic approach to a tableau system for hybrid logic [16] with a formalization in Isabelle/HOL due to the present author [10].

Isabelle Primer I

We encode our domain in higher-order logic, e.g. natural numbers:

```
datatype mynat = Zero | OnePlus mynat
```

We can then define operations on our objects:

```
primrec add :: ‹mynat ⇒ mynat ⇒ mynat› where
  ‹add Zero m = m›
  ‹add (OnePlus n) m = OnePlus (add n m)›
```

And run them!

```
value ‹add (OnePlus Zero) (OnePlus (OnePlus Zero))›
```

```
"OnePlus (OnePlus (OnePlus Zero))"
  :: "mynat"
```

Isabelle Primer II

The definition becomes simplification rules:

```
lemma ‹add Zero k = k›
  using add.simps(1) .
```

We can use induction to prove more interesting things:

```
lemma ‹add k Zero = k›
proof (induct k)
```

```
Proof outline with cases:
  case Zero
  then show ?case sorry
next
  case (OnePlus k)
  then show ?case sorry
qed
```

Isabelle Primer III

Induction splits the statement ‹add k Zero = k› into the base case:

```
 case Zero
 then show ?case (* add Zero Zero = Zero *)
  using add.simps(1) .
next
```

And the induction step:

```
 case (OnePlus k)
 have ‹add (OnePlus k) Zero = OnePlus (add k Zero)›
  using add.simps(2) .
 also have ‹… = OnePlus k›
  using OnePlus ..
 finally show ?case . (* add (OnePlus k) Zero = OnePlus k *)
qed
```

Isabelle Primer IV

The Isabelle simplifier can do the proof:

```
lemma ‹add k Zero = k›
  by (induct k) simp_all
```

Much ado about nothing?

- Proofs in higher-order logic rather than natural language.
- Machine-checked.
- Unambiguous definitions.
- Responsive to changes.
- Counter-example search:

```
lemma ‹add k Zero = Zero›
```

```
Auto Quickcheck found a counterexample:
  k = OnePlus Zero
Evaluated terms:
  add k Zero = OnePlus Zero
  Zero = Zero
```

Propositional Logic

Let us work with something more interesting than natural numbers:

```
datatype form
  = Falsity (‹⊥›)
  | Pro nat
  | Imp form form (infixr ‹⟶› 25)

abbreviation Neg (‹¬ _› [40] 40) where ‹¬ p ≡ p ⟶ ⊥›
```

And something more interesting than addition:

```
primrec semantics :: ‹(nat ⇒ bool) ⇒ form ⇒ bool› (‹_ ⊨ _› [50, 50] 50) where
  ‹(I ⊨ ⊥) = False›
| ‹(I ⊨ Pro n) = I n›
| ‹(I ⊨ (p ⟶ q)) = ((I ⊨ p) ⟶ (I ⊨ q))›
```

Where I is the interpretation of propositional symbols.

Axiom System

Church's P_1 [7] as an inductive predicate ⊢ with 1 rule and 3 axiom schemas:

```
inductive Axiomatics :: ‹form ⇒ bool› (‹⊢ _› [50] 50) where
  MP: ‹⊢ p ⟹ ⊢ (p ⟶ q) ⟹ ⊢ q›
| Imp1: ‹⊢ (p ⟶ q ⟶ p)›
| Imp2: ‹⊢ ((p ⟶ q ⟶ r) ⟶ (p ⟶ q) ⟶ p ⟶ r)›
| Neg: ‹⊢ (((p ⟶ ⊥) ⟶ ⊥) ⟶ p)›
```

The system is sound; if we can derive a formula then it is valid:

```
theorem soundness: ‹⊢ p ⟹ I ⊨ p›
  by (induct rule: Axiomatics.induct) simp_all
```

Notice that the induction is over the derivation.

Derivations

We can find small derivations with the *sledgehammer* tool:

```
lemma Imp3: ‹⊢ (p ⟶ p)›
  sledgehammer
      Sledgehammering...
      Proof found...
      "e": Try this: by (meson Imp1 Imp2 MP) (2 ms)
      "vampire": Try this: by (meson Axiomatics.simps) (68 ms)
```

Our judgment ⊢ is one-sided. We use chains of implications to mimic assumptions:

```
primrec imply :: ‹form list ⇒ form ⇒ form› where
  ‹imply [] q = q›
| ‹imply (p # ps) q = (p ⟶ imply ps q)›
```

We say that *q* can be *derived from ps* when we can derive ‹⊢ imply ps q›

Henkin-Style Completeness in One Slide

Question: Can we derive every valid formula? How do we show this?

1. Assume that a given formula *p* is valid under assumptions *ps*.
2. Assume for the sake of contradiction that there is no derivation ‹⊢ imply ps p›
3. Then there can be no derivation ‹⊢ imply ((¬ p) # ps) ⊥›
4. Therefore, ‹(¬ p) # ps› as a set is *consistent* (it does not entail falsity).
5. If we add every possible formula that preserves consistency we get a *maximal consistent set* (MCS).
6. Every MCS is a Hintikka set and formulas in such sets have a model.
7. Thus, we can construct a model for negated *p* and all of *ps*.
8. But by the validity assumption, the model satisfies *p* too.
9. This is a contradiction.

Maximal Consistent Sets I

A set of formulas is *consistent* if we cannot derive falsity from any subset:

```
definition consistent :: ‹form set ⇒ bool› where
  ‹consistent S ≡ ∄S'. set S' ⊆ S ∧ ⊢ imply S' ⊥›
```

A set of formulas is *maximal* if any proper extension makes it inconsistent:

```
definition maximal :: ‹form set ⇒ bool› where
  ‹maximal S ≡ ∀p. p ∉ S ⟶ ¬ consistent ({p} ∪ S)›
```

Given a consistent set S_0 and an enumeration of formulas, (ϕ_n), we construct a sequence of consistent sets like so:

$$S_{n+1} = \begin{cases} \{\phi_n\} \cup S_n & \text{if } \{\phi_n\} \cup S_n \text{ is consistent,} \\ S_n & \text{otherwise.} \end{cases}$$

Maximal Consistent Sets II

In Isabelle/HOL we specify a function that returns a specific element:

```
primrec extend :: ‹form set ⇒ (nat ⇒ form) ⇒ nat ⇒ form set› where
  ‹extend S f 0 = S›
| ‹extend S f (Suc n) =
    (if consistent ({f n} ∪ extend S f n)
     then {f n} ∪ extend S f n
     else extend S f n)›
```

We get the maximal consistent set in the limit:

```
definition Extend :: ‹form set ⇒ (nat ⇒ form) ⇒ form set› where
  ‹Extend S f ≡ ⋃n. extend S f n›
```

But of course we need to prove that it is maximal and consistent!

Maximal Consistent Sets III

The original set is in the limit:

```
lemma Extend_subset: ‹S ⊆ Extend S f›
  unfolding Extend_def by (metis Union_upper extend.simps(1) range_eqI)
```

Each element bounds the previous ones:

```
lemma extend_bound: ‹(⋃n ≤ m. extend S f n) = extend S f m›
  by (induct m) (simp_all add: atMost_Suc)
```

Consistency is preserved by definition:

```
lemma consistent_extend: ‹consistent S ⟹ consistent (extend S f n)›
  by (induct n) simp_all
```

Consistency in the Limit

```
lemma consistent_Extend:
  assumes ‹consistent S›
  shows ‹consistent (Extend S f)›
  unfolding Extend_def
proof (rule ccontr)
  assume ‹¬ consistent (⋃n. extend S f n)›
  then obtain S' where ‹⊢ imply S' ⊥› ‹set S' ⊆ (⋃n. extend S f n)›
    unfolding consistent_def by blast
  then obtain m where ‹set S' ⊆ (⋃n ≤ m. extend S f n)›
    using UN_finite_bound by (metis List.finite_set)
  then have ‹set S' ⊆ extend S f m›
    using extend_bound by blast
  moreover have ‹consistent (extend S f m)›
    using assms consistent_extend by blast
  ultimately show False
    unfolding consistent_def using ‹⊢ imply S' ⊥› by blast
qed
```

Assume that the starting set is consistent.
Show that the limit is too.
(by unfolding its definition)
Proof by classical contradiction.
Assume the limit is inconsistent.
Obtain some list of formulas that we can derive falsity from.
This list is a subset of some prefix of the constructed sequence.
So in particular it is a subset of a bounding element.

But such an element is consistent by construction.

So we reach a contradiction.

And the limit must be consistent.

Consistency in the Limit

```
lemma consistent_Extend:
  assumes ‹consistent S›
  shows ‹consistent (Extend S f)›
  unfolding Extend_def
proof (rule ccontr)
  assume ‹¬ consistent (⋃n. extend S f n)›
  then obtain S' where ‹⊢ imply S' ⊥› ‹set S' ⊆ (⋃n. extend S f n)›
    unfolding consistent_def by blast
  then obtain m where ‹set S' ⊆ (⋃n ≤ m. extend S f n)›
    using UN_finite_bound by (metis List.finite_set)
  then have ‹set S' ⊆ extend S f m›
    using extend_bound by blast
  moreover have ‹consistent (extend S f m)›
    using assms consistent_extend by blast
  ultimately show False
    unfolding consistent_def using ‹⊢ imply S' ⊥› by blast
qed
```

Assume the starting set is consistent.

Show the limit is too.

(by unfolding its definition)

Proof by classical contradiction.

Assume the limit is inconsistent.

Obtain some list of formulas that we can derive falsity from.

This list is a subset of some prefix of the constructed sequence.

So in particular it is a subset of a bounding element.

But such an element is consistent by construction.

So we reach a contradiction.

And the limit must be consistent.

Consistency in the Limit

```
lemma consistent_Extend:
  assumes ‹consistent S›
  shows ‹consistent (Extend S f)›
  unfolding Extend_def
proof (rule ccontr)
  assume ‹¬ consistent (⋃n. extend S f n)›
  then obtain S' where ‹⊢ imply S' ⊥› ‹set S' ⊆ (⋃n. extend S f n)›
    unfolding consistent_def by blast
  then obtain m where ‹set S' ⊆ (⋃n ≤ m. extend S f n)›
    using UN_finite_bound by (metis List.finite_set)
  then have ‹set S' ⊆ extend S f m›
    using extend_bound by blast
  moreover have ‹consistent (extend S f m)›
    using assms consistent_extend by blast
  ultimately show False
    unfolding consistent_def using ‹⊢ imply S' ⊥› by blast
qed
```

Assume the starting set is consistent.

Show the limit is too.

(by unfolding its definition)

Proof by classical contradiction.

Assume the limit is inconsistent.

Obtain some list of formulas that we can derive falsity from.

This list is a subset of some prefix of the constructed sequence.

So in particular it is a subset of a bounding element.

But such an element is consistent by construction.

So we reach a contradiction.

And the limit must be consistent.

Consistency in the Limit

```
lemma consistent_Extend:
  assumes ‹consistent S›
  shows ‹consistent (Extend S f)›
  unfolding Extend_def
proof (rule ccontr)
  assume ‹¬ consistent (⋃n. extend S f n)›
  then obtain S' where ‹⊢ imply S' ⊥› ‹set S' ⊆ (⋃n. extend S f n)›
    unfolding consistent_def by blast
  then obtain m where ‹set S' ⊆ (⋃n ≤ m. extend S f n)›
    using UN_finite_bound by (metis List.finite_set)
  then have ‹set S' ⊆ extend S f m›
    using extend_bound by blast
  moreover have ‹consistent (extend S f m)›
    using assms consistent_extend by blast
  ultimately show False
    unfolding consistent_def using ‹⊢ imply S' ⊥› by blast
qed
```

Assume the starting set is consistent.
Show the limit is too.
(by unfolding its definition)
Proof by classical contradiction.
Assume the limit is inconsistent.
Obtain some list of formulas that we can derive falsity from.
This list is a subset of some prefix of the constructed sequence.
So in particular it is a subset of a bounding element.

But such an element is consistent by construction.

So we reach a contradiction.

And the limit must be consistent.

Consistency in the Limit

lemma consistent_Extend:	
assumes ‹consistent S›	Assume the starting set is consistent.
shows ‹consistent (Extend S f)›	Show the limit is too.
unfolding Extend_def	(by unfolding its definition)
proof (rule ccontr)	Proof by classical contradiction.
assume ‹¬ consistent (⋃n. extend S f n)›	Assume the limit is inconsistent.
then obtain S' **where** ‹⊢ imply S' ⊥› ‹set S' ⊆ (⋃n. extend S f n)›	Obtain some list of formulas that we can derive
unfolding consistent_def **by** blast	falsity from.
then obtain m **where** ‹set S' ⊆ (⋃n ≤ m. extend S f n)›	This list is a subset of some prefix of the
using UN_finite_bound **by** (metis List.finite_set)	constructed sequence.
then have ‹set S' ⊆ extend S f m›	So in particular it is a subset of a bounding element.
using extend_bound **by** blast	
moreover have ‹consistent (extend S f m)›	But such an element is consistent by construction.
using assms consistent_extend **by** blast	
ultimately show False	So we reach a contradiction.
unfolding consistent_def **using** ‹⊢ imply S' ⊥› **by** blast	
qed	And the limit must be consistent.

Maximality in the Limit

lemma maximal_Extend**:**	
assumes ‹surj f›	Assume the enumeration hits every formula.
shows ‹maximal (Extend S f)›	Show that the limit is maximal.
proof (rule ccontr)	By classical contradiction.
assume ‹¬ maximal (Extend S f)›	If it is not maximal.
then obtain p **where** ‹p ∉ Extend S f› ‹consistent ({p} ∪ Extend S f)›	then some formula is absent even though it preserves
unfolding maximal_def **using** assms consistent_Extend **by** blast	consistency.
obtain k **where** n**:** ‹f k = p›	This formula is part of the enumeration.
using ‹surj f› **unfolding** surj_def **by** metis	
then have ‹p ∉ extend S f (Suc k)›	But it was not added at the corresponding step of the
using ‹p ∉ Extend S f› **unfolding** Extend_def **by** blast	construction.
then have ‹¬ consistent ({p} ∪ extend S f k)›	So, by definition, adding it breaks consistency.
using n **by** fastforce	
moreover have ‹{p} ∪ extend S f k ⊆ {p} ∪ Extend S f›	The extension is a subset of the extended limit.
unfolding Extend_def **by** blast	
ultimately have ‹¬ consistent ({p} ∪ Extend S f)›	But then the extended limit has an inconsistent subset.
unfolding consistent_def **by** fastforce	
then show False	And this is a contradiction.
using ‹consistent ({p} ∪ Extend S f)› **by** blast	
qed	So the limit must be maximal.

Hintikka Sets I

Sets that are *downwards saturated*:

```
locale Hintikka =
  fixes H :: ‹form set›
  assumes
    NoFalsity: ‹⊥ ∉ H› and
    Pro: ‹Pro n ∈ H ⟹ (¬ Pro n) ∉ H› and
    ImpP: ‹(p ⟶ q) ∈ H ⟹ (¬ p) ∈ H ∨ q ∈ H› and
    ImpN: ‹(¬ (p ⟶ q)) ∈ H ⟹ p ∈ H ∧ (¬ q) ∈ H›
```

The satisfiability of any complex formula in a Hintikka set is guaranteed by conditions on its subformulas.

This means that we can build a model based on set membership:

```
abbreviation (input) ‹model H n ≡ Pro n ∈ H›
```

Hintikka Sets II

Isabelle is powerful enough to automatically prove the *model existence* theorem*:*

```
lemma Hintikka_model:
  ‹Hintikka H ⟹ (p ∈ H ⟶ model H ⊨ p) ∧ ((¬ p) ∈ H ⟶ ¬ model H ⊨ p)›
  by (induct p) (simp; unfold Hintikka_def, blast)+
```

But we need to manually prove that maximal consistent sets are Hintikka sets:

```
lemma Hintikka_Extend:
  assumes ‹maximal S› ‹consistent S›
  shows ‹Hintikka S›
  (proof omitted)
```

Four cases, one per Hintikka condition, proofs by contradiction.

Close to 80 lines in total.

Completeness I

```
lemma imply_completeness:
  assumes valid: ‹∀I s. list_all (λq. I ⊨ q) ps ⟶ I ⊨ p›
  shows ‹⊢ imply ps p›
proof (rule ccontr)
  assume ‹¬ ⊢ imply ps p›
  then have *: ‹¬ ⊢ imply ((¬ p) # ps) ⊥›
    using Boole by blast

  let ?S = ‹set ((¬ p) # ps)›
  let ?H = ‹Extend ?S from_nat›

  have ‹consistent ?S›
    unfolding consistent_def using * imply_weaken by blast
  then have ‹consistent ?H› ‹maximal ?H›
    using consistent_Extend maximal_Extend surj_from_nat by blast+
  then have ‹Hintikka ?H›
    using Hintikka_Extend by blast
```

Strong completeness:
Assume formulas *ps* imply *p*.
Show that *p* can be derived from *ps*.
Proof by contradiction.
If there is no derivation
then adding negated *p* is consistent.

Abbreviation for the starting set.
Abbreviation for the constructed Hintikka set.

The starting set is consistent.

So the constructed set is consistent and maximal.

So it is a Hintikka set.

Completeness I

```
lemma imply_completeness:
  assumes valid: ‹∀I s. list_all (λq. I ⊨ q) ps ⟶ I ⊨ p›
  shows ‹⊢ imply ps p›
proof (rule ccontr)
  assume ‹¬ ⊢ imply ps p›
  then have *: ‹¬ ⊢ imply ((¬ p) # ps) ⊥›
    using Boole by blast

  let ?S = ‹set ((¬ p) # ps)›
  let ?H = ‹Extend ?S from_nat›

  have ‹consistent ?S›
    unfolding consistent_def using * imply_weaken by blast
  then have ‹consistent ?H› ‹maximal ?H›
    using consistent_Extend maximal_Extend surj_from_nat by blast+
  then have ‹Hintikka ?H›
    using Hintikka_Extend by blast
```

Strong completeness:
Assume formulas *ps* imply *p*.
Show that *p* derives from *ps*.
Proof by contradiction.
If there is no derivation
then adding negated *p* is consistent.

Abbreviation for the starting set.
Abbreviation for the constructed Hintikka set.

The starting set is consistent.

So the constructed set is consistent and maximal.

So it is a Hintikka set.

Completeness I

lemma imply_completeness:	Strong completeness:
assumes valid: ‹∀I s. list_all (λq. I ⊨ q) ps ⟶ I ⊨ p›	Assume formulas *ps* imply *p*.
shows ‹⊢ imply ps p›	Show that *p* derives from *ps*.
proof (rule ccontr)	Proof by contradiction.
assume ‹¬ ⊢ imply ps p›	If there is no derivation
then have *: ‹¬ ⊢ imply ((¬ p) # ps) ⊥›	then adding negated *p* is consistent.
using Boole **by** blast	
let ?S = ‹set ((¬ p) # ps)›	Abbreviation for the starting set.
let ?H = ‹Extend ?S from_nat›	Abbreviation for the constructed Hintikka set.
have ‹consistent ?S›	The starting set is consistent.
unfolding consistent_def **using** * imply_weaken **by** blast	
then have ‹consistent ?H› ‹maximal ?H›	So the constructed set is consistent and maximal.
using consistent_Extend maximal_Extend surj_from_nat **by** blast+	
then have ‹Hintikka ?H›	So it is a Hintikka set.
using Hintikka_Extend **by** blast	

Completeness II

```
 have ‹model ?H ⊨ p› if ‹p ∈ ?S› for p
   using that Extend_subset Hintikka_model ‹Hintikka ?H› by blast
 then have ‹model ?H ⊨ (¬ p)› ‹list_all (λp. model ?H ⊨ p) ps›
   unfolding list_all_def by fastforce+
 then have ‹model ?H ⊨ p›
   using valid by blast
 then show False
   using ‹model ?H ⊨ (¬ p)› by simp
qed
```

The Hintikka model satisfies any formula in the starting set.
Which includes negated *p* and all of *ps*.

So by the validity assumption, it satisfies *p*.

But this is a contradiction.

So the derivation must exist.

If we specialize to no assumptions we get the completeness theorem:

```
theorem completeness: ‹∀I. I ⊨ p ⟹ ⊢ p›
  using imply_completeness[where ps=‹[]›] by simp
```

Possible Extensions

We have extended the formalization with binary conjunction and disjunction operators and corresponding proof rules and Hintikka conditions.

The result is a strict extension: we only have to add ~130 new lines.

The model existence theorem is still completely automatic.

If we want to move to first-order logic we need to add Henkin witnesses for existential statements.

The proof assistant tells us which proofs break.

Conclusion

Proof assistants allow us to be extremely precise.

The resulting formalized proofs leave out no details.

We can use them to explicate an approach like Henkin-style completeness.

Such a formalization can serve as reference or as starting point for future work.

The resulting formalization is available at:

https://github.com/logic-tools/axiom

Abridged Bibliography

3. Berghofer, S.: First-Order Logic According to Fitting. Archive of Formal Proofs (Aug 2007), http://isa-afp.org/entries/FOL-Fitting.html

5. Blanchette, J.C., Popescu, A., Traytel, D.: Soundness and completeness proofs by coinductive methods. Journal of Automated Reasoning 58(1), 149–179 (2017)

6. Braselmann, P., Koepke, P.: Gödel's completeness theorem. Formalized Mathematics 13(1), 49–53 (2005)

7. Church, A.: Introduction to Mathematical Logic. Princeton Mathematical Series, Princeton University Press (1956)

8. Fitting, M.: First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science, Springer (1996)

10. From, A.H.: Formalizing a Seligman-style tableau system for hybrid logic. Archive of Formal Proofs (Dec 2019), http://isa-afp.org/entries/Hybrid_Logic.html, Formal proof development

12. Gödel, K.: Über die Vollständigkeit des Logikkalküls. Ph.D. thesis, University of Vienna (1929)

Abridged Bibliography

13. Henkin, L.: The Completeness of Formal Systems. Ph.D. thesis, Princeton University (1947)

14. Henkin, L.: The Discovery of My Completeness Proofs. Bulletin of Symbolic Logic 2(2), 127–158 (1996)

15. Ilik, D.: Constructive completeness proofs and delimited control. Ph.D. thesis, École polytechnique (2010)

16. Jørgensen, K.F., Blackburn, P., Bolander, T., Braüner, T.: Synthetic completeness proofs for Seligman-style tableau systems. In: Proceedings of the 11th conference on Advances in Modal Logic. pp. 302–321 (2016)

18. Margetson, J., Ridge, T.: Completeness theorem. Archive of Formal Proofs (Sep 2004), http://isa-afp.org/entries/Completeness.html, Formal proof development

19. Michaelis, J., Nipkow, T.: Propositional proof systems. Archive of Formal Proofs (Jun 2017), http://isa-afp.org/entries/Propositional_Proof_Systems.html, Formal proof development

20. Michaelis, J., Nipkow, T.: Formalized proof systems for propositional logic. In: Abel, A., Forsberg, F.N., Kaposi, A. (eds.) 23rd Int. Conf. Types for Proofs and Programs (TYPES 2017). LIPIcs, vol. 104, pp. 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2018)

Abridged Bibliography

21. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)

24. Persson, H.: Constructive completeness of intuitionistic predicate logic. Licenciate thesis, Chalmers University of Technology (1996)

28. Shankar, N.: Towards mechanical metamathematics. Journal of Automated Reasoning 1(4), 407–434 (1985)

30. Smullyan, R.M.: First-Order Logic. Springer-Verlag (1968)

34. Zach, R.: Completeness before Post: Bernays, Hilbert, and the development of propositional logic. Bulletin of Symbolic Logic 5(3), 331–366 (1999)

Note also https://www.isa-afp.org/entries/Epistemic_Logic.html:

This work is a formalization of epistemic logic with countably many agents. It includes proofs of soundness and completeness for the axiom system K. The completeness proof is based on the textbook "Reasoning About Knowledge" by Fagin, Halpern, Moses and Vardi (MIT Press 1995).