Soundness and Completeness of Implicational Logic

Asta Halkjær From   Jørgen Villadsen

October 27, 2022

Contents

Abstract

This work is a formalization of soundness and completeness of the Bernays-Tarski axiom system for classical implicational logic. The completeness proof is constructive following the approach by László Kalmár, Elliott Mendelson and others. The result can be extended to full classical propositional logic by uncommenting a few lines for falsehood.

1 Formalization of the Bernays-Tarski Axiom System for Classical Implicational Logic

1.1 Syntax, Semantics and Axiom System

theory *Implicational-Logic* **imports** *Main* **begin**

datatype *form* =

Pro nat (‹·›) |
Imp form form (**infixr** ‹→› *55*)

primrec *semantics* (**infix** ‹$\models$› *50*) **where**

‹$I \models \cdot\ n = I\ n$› |
‹$I \models p \rightarrow q = (I \models p \longrightarrow I \models q)$›

inductive *Ax* (‹$\vdash$ -› *50*) **where**

Simp: ‹$\vdash p \rightarrow q \rightarrow p$› |
Tran: ‹$\vdash (p \rightarrow q) \rightarrow (q \rightarrow r) \rightarrow p \rightarrow r$› |
MP: ‹$\vdash p \rightarrow q \Longrightarrow \vdash p \Longrightarrow \vdash q$› |
PR: ‹$\vdash (p \rightarrow q) \rightarrow p \Longrightarrow \vdash p$›

1.2 Soundness and Derived Formulas

theorem *soundness*: ‹$\vdash p \Longrightarrow I \models p$›
by (*induct p rule*: *Ax.induct*) *auto*

lemma *Swap*: ‹$\vdash (p \rightarrow q \rightarrow r) \rightarrow q \rightarrow p \rightarrow r$›
proof –
have ‹$\vdash q \rightarrow (q \rightarrow r) \rightarrow r$›
using *MP PR Simp Tran* **by** *metis*
then show *?thesis*
using *MP Tran* **by** *meson*
qed

lemma *Peirce*: ‹$\vdash ((p \rightarrow q) \rightarrow p) \rightarrow p$›
using *MP PR Simp Swap Tran* **by** *meson*

lemma *Hilbert*: ‹$\vdash (p \rightarrow p \rightarrow q) \rightarrow p \rightarrow q$›
using *MP MP Tran Tran Peirce* **.**

lemma *Id*: ‹$\vdash p \rightarrow p$›
using *MP Hilbert Simp* **.**

lemma *Tran'*: ‹$\vdash (q \rightarrow r) \rightarrow (p \rightarrow q) \rightarrow p \rightarrow r$›
using *MP Swap Tran* **.**

lemma *Frege*: ‹$\vdash (p \rightarrow q \rightarrow r) \rightarrow (p \rightarrow q) \rightarrow p \rightarrow r$›
using *MP MP Tran MP MP Tran Swap Tran' MP Tran' Hilbert* **.**

lemma *Imp1*: ‹$\vdash (q \rightarrow s) \rightarrow ((q \rightarrow r) \rightarrow s) \rightarrow s$›
using *MP Peirce Tran Tran'* **by** *meson*

lemma *Imp2*: ‹$\vdash ((r \rightarrow s) \rightarrow s) \rightarrow ((q \rightarrow r) \rightarrow s) \rightarrow s$›
using *MP Tran MP Tran Simp* **.**

lemma *Imp3*: ‹$\vdash ((q \rightarrow s) \rightarrow s) \rightarrow (r \rightarrow s) \rightarrow (q \rightarrow r) \rightarrow s$›
using *MP Swap Tran* **by** *meson*

1.3 Completeness and Main Theorem

```
fun pros where
  ‹pros (p → q) = remdups (pros p @ pros q)› |
  ‹pros p = (case p of (· n) ⇒ [n] | - ⇒ [])›

lemma distinct-pros: ‹distinct (pros p)›
  by (induct p) simp-all

primrec imply (infixr ‹⇝› 56) where
  ‹[] ⇝ q = q› |
  ‹p # ps ⇝ q = p → ps ⇝ q›

lemma imply-append: ‹ps @ qs ⇝ r = ps ⇝ qs ⇝ r›
  by (induct ps) simp-all

abbreviation Ax-assms (infix ‹⊢› 50) where ‹ps ⊢ q ≡ ⊢ ps ⇝ q›

lemma imply-Cons: ‹ps ⊢ q ⟹ p # ps ⊢ q›
proof –
  assume ‹ps ⊢ q›
  with MP Simp have ‹⊢ p → ps ⇝ q› .
  then show ?thesis
    by simp
qed

lemma imply-head: ‹p # ps ⊢ p›
  by (induct ps) (use MP Frege Simp imply.simps in metis)+

lemma imply-mem: ‹p ∈ set ps ⟹ ps ⊢ p›
  by (induct ps) (use imply-Cons imply-head in auto)

lemma imply-MP: ‹⊢ ps ⇝ (p → q) → ps ⇝ p → ps ⇝ q›
proof (induct ps)
  case (Cons r ps)
  then have ‹⊢ (r → ps ⇝ (p → q)) → (r → ps ⇝ p) → r → ps ⇝ q›
    using MP Frege Simp by meson
  then show ?case
    by simp
qed (auto intro: Id)

lemma MP′: ‹ps ⊢ p → q ⟹ ps ⊢ p ⟹ ps ⊢ q›
  using MP imply-MP by metis

lemma imply-swap-append: ‹ps @ qs ⊢ r ⟹ qs @ ps ⊢ r›
 by (induct qs arbitrary: ps) (simp, metis MP′ imply-append imply-Cons imply-head
imply.simps(2))

lemma imply-deduct: ‹p # ps ⊢ q ⟹ ps ⊢ p → q›
  using imply-append imply-swap-append imply.simps by metis
```

lemma *add-imply* [*simp*]: ‹⊢ *p* ⟹ *ps* ⊢ *p*›
proof −
  note *MP*
  moreover have ‹⊢ *p* → *ps* ⇝ *p*›
    using *imply-head* **by** *simp*
  moreover assume ‹⊢ *p*›
  ultimately show *?thesis* **.**
qed

lemma *imply-weaken*: ‹*ps* ⊢ *p* ⟹ *set ps* ⊆ *set ps*′ ⟹ *ps*′ ⊢ *p*›
 by (*induct ps arbitrary*: *p*) (*simp*, *metis MP*′ *imply-deduct imply-mem insert-subset list*.*set*(*2*))

abbreviation ‹*lift t s p* ≡ *if t then* (*p* → *s*) → *s else p* → *s*›

abbreviation ‹*lifts I s* ≡ *map* (λ*n*. *lift* (*I n*) *s* (· *n*))›

lemma *lifts-weaken*: ‹*lifts I s l* ⊢ *p* ⟹ *set l* ⊆ *set l*′ ⟹ *lifts I s l*′ ⊢ *p*›
  using *imply-weaken* **by** (*metis* (*no-types*, *lifting*) *image-mono set-map*)

lemma *lifts-pros-lift*: ‹*lifts I s* (*pros p*) ⊢ *lift* (*I* ⊨ *p*) *s p*›
proof (*induct p*)
  case (*Imp q r*)
  consider ‹¬ *I* ⊨ *q*› | ‹*I* ⊨ *r*› | ‹*I* ⊨ *q*› ‹¬ *I* ⊨ *r*›
    by *blast*
  then show *?case*
  proof *cases*
    case *1*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ *q* → *s*›
      using *Imp*(*1*) *lifts-weaken*[**where** *l*′ = ‹*pros* (*q* → *r*)›] **by** *simp*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ ((*q* → *r*) → *s*) → *s*›
      using *Imp1 MP*′ *add-imply* **by** *blast*
    with *1* **show** *?thesis*
      by *simp*
  next
    case *2*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*r* → *s*) → *s*›
      using *Imp*(*2*) *lifts-weaken*[**where** *l*′ = ‹*pros* (*q* → *r*)›] **by** *simp*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ ((*q* → *r*) → *s*) → *s*›
      using *Imp2 MP*′ *add-imply* **by** *blast*
    with *2* **show** *?thesis*
      by *simp*
  next
    case *3*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*q* → *s*) → *s*› ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ *r* → *s*›
      using *Imp lifts-weaken*[**where** *l*′ = ‹*pros* (*q* → *r*)›] **by** *simp-all*
    then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*q* → *r*) → *s*›

```
      using Imp3 MP′ add-imply by blast
    with 3 show ?thesis
      by simp
  qed
qed (auto intro: Id Ax.intros)

lemma lifts-pros: ‹I ⊨ p ⟹ lifts I p (pros p) ⊢ p›
proof −
  assume ‹I ⊨ p›
  then have ‹lifts I p (pros p) ⊢ (p → p) → p›
    using lifts-pros-lift[of I p p] by simp
  then show ?thesis
    using Id MP′ add-imply by blast
qed

theorem completeness: ‹∀ I. I ⊨ p ⟹ ⊢ p›
proof −
  let ?A = ‹λl I. lifts I p l ⊢ p›
  let ?B = ‹λl. ∀ I. ?A l I ∧ distinct l›
  assume ‹∀ I. I ⊨ p›
  moreover have ‹?B l ⟹ (⋀n l. ?B (n # l) ⟹ ?B l) ⟹ ?B []› for l
    by (induct l) blast+
  moreover have ‹?B (n # l) ⟹ ?B l› for n l
  proof −
    assume ∗: ‹?B (n # l)›
    show ‹?B l›
    proof
      fix I
      from ∗ have ‹?A (n # l) (I(n := True))› ‹?A (n # l) (I(n := False))›
        by blast+
      moreover from ∗ have ‹∀ m ∈ set l. ∀ t. (I(n := t)) m = I m›
        by simp
      ultimately have ‹((· n → p) → p) # lifts I p l ⊢ p› ‹(· n → p) # lifts I p l
⊢ p›
        by (simp-all cong: map-cong)
      then have ‹?A l I›
        using MP′ imply-deduct by blast
      moreover from ∗ have ‹distinct (n # l)›
        by blast
      ultimately show ‹?A l I ∧ distinct l›
        by simp
    qed
  qed
  ultimately have ‹?B []›
    using lifts-pros distinct-pros by blast
  then show ?thesis
    by simp
qed
```

```
theorem main: ‹(⊢ p) = (∀ I. I ⊨ p)›
  using soundness completeness by blast
```

1.4 Reference

Wikipedia https://en.wikipedia.org/wiki/Implicational_propositional_calculus July 2022

end

2 Formalization of Łukasiewicz's Axiom System from 1924 for Classical Propositional Logic

2.1 Syntax, Semantics and Axiom System

```
theory Implicational-Logic-Appendix imports Main begin
```

```
datatype form =
  Pro nat (‹·›) |
  Neg form (‹∼›) |
  Imp form form (infixr ‹→› 55)
```

```
primrec semantics (infix ‹⊨› 50) where
  ‹I ⊨ · n = I n› |
  ‹I ⊨ ∼ p = (¬ I ⊨ p)› |
  ‹I ⊨ p → q = (I ⊨ p ⟶ I ⊨ q)›
```

```
inductive Ax (‹⊢ -› 50) where
  01: ‹⊢ (p → q) → (q → r) → p → r› |
  02: ‹⊢ (∼ p → p) → p› |
  03: ‹⊢ p → ∼ p → q› |
  MP: ‹⊢ p → q ⟹ ⊢ p ⟹ ⊢ q›
```

2.2 Soundness and Derived Formulas

```
theorem soundness: ‹⊢ p ⟹ I ⊨ p›
  by (induct p rule: Ax.induct) simp-all
```

```
lemma 04: ‹⊢ (((q → r) → p → r) → s) → (p → q) → s›
  using MP 01 01 .
```

```
lemma 05: ‹⊢ (p → q → r) → (s → q) → p → s → r›
  using MP 04 04 .
```

```
lemma 06: ‹⊢ (p → q) → ((p → r) → s) → (q → r) → s›
  using MP 04 01 .
```

```
lemma 07: ‹⊢ (t → (p → r) → s) → (p → q) → t → (q → r) → s›
  using MP 05 06 .
```

lemma *09*: $\langle\vdash ((\sim p \to q) \to r) \to p \to r\rangle$
using *MP 01 03* **.**

lemma *10*: $\langle\vdash p \to ((\sim p \to p) \to p) \to (q \to p) \to p\rangle$
using *MP 09 06* **.**

lemma *11*: $\langle\vdash (q \to (\sim p \to p) \to p) \to (\sim p \to p) \to p\rangle$
using *MP MP 10 02 02* **.**

lemma *12*: $\langle\vdash t \to (\sim p \to p) \to p\rangle$
using *MP 09 11* **.**

lemma *13*: $\langle\vdash (\sim p \to q) \to t \to (q \to p) \to p\rangle$
using *MP 07 12* **.**

lemma *14*: $\langle\vdash ((t \to (q \to p) \to p) \to r) \to (\sim p \to q) \to r\rangle$
using *MP 01 13* **.**

lemma *15*: $\langle\vdash (\sim p \to q) \to (q \to p) \to p\rangle$
using *MP 14 02* **.**

lemma *16*: $\langle\vdash p \to p\rangle$
using *MP 09 02* **.**

lemma *17*: $\langle\vdash p \to (q \to p) \to p\rangle$
using *MP 09 15* **.**

lemma *18*: $\langle\vdash q \to p \to q\rangle$
using *MP MP 05 17 03* **.**

lemma *19*: $\langle\vdash ((p \to q) \to r) \to q \to r\rangle$
using *MP 01 18* **.**

lemma *20*: $\langle\vdash p \to (p \to q) \to q\rangle$
using *MP 19 15* **.**

lemma *21*: $\langle\vdash (p \to q \to r) \to q \to p \to r\rangle$
using *MP 05 20* **.**

lemma *22*: $\langle\vdash (q \to r) \to (p \to q) \to p \to r\rangle$
using *MP 21 01* **.**

lemma *23*: $\langle\vdash ((q \to p \to r) \to s) \to (p \to q \to r) \to s\rangle$
using *MP 01 21* **.**

lemma *24*: $\langle\vdash ((p \to q) \to p) \to p\rangle$
using *MP MP 23 15 03* **.**

lemma *25*: ‹$\vdash ((p \to r) \to s) \to (p \to q) \to (q \to r) \to s$›
using *MP 21 06* **.**

lemma *26*: ‹$\vdash ((p \to q) \to r) \to (r \to p) \to p$›
using *MP 25 24* **.**

lemma *28*: ‹$\vdash (((r \to p) \to p) \to s) \to ((p \to q) \to r) \to s$›
using *MP 01 26* **.**

lemma *29*: ‹$\vdash ((p \to q) \to r) \to (p \to r) \to r$›
using *MP 28 26* **.**

lemma *31*: ‹$\vdash (p \to s) \to ((p \to q) \to r) \to (s \to r) \to r$›
using *MP 07 29* **.**

lemma *32*: ‹$\vdash ((p \to q) \to r) \to (p \to s) \to (s \to r) \to r$›
using *MP 21 31* **.**

lemma *33*: ‹$\vdash (p \to s) \to (s \to q \to p \to r) \to q \to p \to r$›
using *MP 32 18* **.**

lemma *34*: ‹$\vdash (s \to q \to p \to r) \to (p \to s) \to q \to p \to r$›
using *MP 21 33* **.**

lemma *35*: ‹$\vdash (p \to q \to r) \to (p \to q) \to p \to r$›
using *MP 34 22* **.**

lemma *36*: ‹$\vdash \sim p \to p \to q$›
using *MP 21 03* **.**

lemmas
Tran = *01* **and**
Clavius = *02* **and**
Expl = *03* **and**
Frege′ = *05* **and**
Clavius′ = *15* **and**
Id = *16* **and**
Simp = *18* **and**
Swap = *21* **and**
Tran′ = *22* **and**
Peirce = *24* **and**
Frege = *35* **and**
Expl′ = *36*

lemma *Neg1*: ‹$\vdash (q \to s) \to (\sim q \to s) \to s$›
using *MP Clavius′ Expl′ Frege′ Swap* **by** *meson*

lemma *Neg2*: ‹$\vdash ((q \to s) \to s) \to \sim q \to s$›
using *MP Tran MP Swap Expl* **.**

```
lemma Imp1: ‹⊢ (q → s) → ((q → r) → s) → s›
  using MP Peirce Tran Tran′ by meson

lemma Imp2: ‹⊢ ((r → s) → s) → ((q → r) → s) → s›
  using MP Tran MP Tran Simp .

lemma Imp3: ‹⊢ ((q → s) → s) → (r → s) → (q → r) → s›
  using MP Swap Tran by meson
```

2.3 Completeness and Main Theorem

```
primrec pros where
  ‹pros (· n) = [n]› |
  ‹pros (∼ p) = pros p› |
  ‹pros (p → q) = remdups (pros p @ pros q)›

lemma distinct-pros: ‹distinct (pros p)›
  by (induct p) simp-all

primrec imply (infixr ‹⇝› 56) where
  ‹[] ⇝ q = q› |
  ‹p # ps ⇝ q = p → ps ⇝ q›

lemma imply-append: ‹ps @ qs ⇝ r = ps ⇝ qs ⇝ r›
  by (induct ps) simp-all

abbreviation Ax-assms (infix ‹⊢› 50) where ‹ps ⊢ q ≡ ⊢ ps ⇝ q›

lemma imply-Cons: ‹ps ⊢ q ⟹ p # ps ⊢ q›
proof −
  assume ‹ps ⊢ q›
  with MP Simp have ‹⊢ p → ps ⇝ q› .
  then show ?thesis
    by simp
qed

lemma imply-head: ‹p # ps ⊢ p›
  by (induct ps) (use MP Frege Simp imply.simps in metis)+

lemma imply-mem: ‹p ∈ set ps ⟹ ps ⊢ p›
  by (induct ps) (use imply-Cons imply-head in auto)

lemma imply-MP: ‹⊢ ps ⇝ (p → q) → ps ⇝ p → ps ⇝ q›
proof (induct ps)
  case (Cons r ps)
  then have ‹⊢ (r → ps ⇝ (p → q)) → (r → ps ⇝ p) → r → ps ⇝ q›
    using MP Frege Simp by meson
  then show ?case
```

by *simp*
qed (*auto intro*: *Id*)

lemma *MP′*: ‹$ps \vdash p \to q \Longrightarrow ps \vdash p \Longrightarrow ps \vdash q$›
using *MP imply-MP* **by** *metis*

lemma *imply-swap-append*: ‹$ps @ qs \vdash r \Longrightarrow qs @ ps \vdash r$›
by (*induct qs arbitrary*: *ps*) (*simp*, *metis MP′ imply-append imply-Cons imply-head imply.simps*(*2*))

lemma *imply-deduct*: ‹$p \# ps \vdash q \Longrightarrow ps \vdash p \to q$›
using *imply-append imply-swap-append imply.simps* **by** *metis*

lemma *add-imply* [*simp*]: ‹$\vdash p \Longrightarrow ps \vdash p$›
proof −
note *MP*
moreover have ‹$\vdash p \to ps \leadsto p$›
using *imply-head* **by** *simp*
moreover assume ‹$\vdash p$›
ultimately show *?thesis* **.**
qed

lemma *imply-weaken*: ‹$ps \vdash p \Longrightarrow set\ ps \subseteq set\ ps' \Longrightarrow ps' \vdash p$›
by (*induct ps arbitrary*: *p*) (*simp*, *metis MP′ imply-deduct imply-mem insert-subset list.set*(*2*))

abbreviation ‹$lift\ t\ s\ p \equiv if\ t\ then\ (p \to s) \to s\ else\ p \to s$›

abbreviation ‹$lifts\ I\ s \equiv map\ (\lambda n.\ lift\ (I\ n)\ s\ (\cdot\ n))$›

lemma *lifts-weaken*: ‹$lifts\ I\ s\ l \vdash p \Longrightarrow set\ l \subseteq set\ l' \Longrightarrow lifts\ I\ s\ l' \vdash p$›
using *imply-weaken* **by** (*metis* (*no-types*, *lifting*) *image-mono set-map*)

lemma *lifts-pros-lift*: ‹$lifts\ I\ s\ (pros\ p) \vdash lift\ (I \models p)\ s\ p$›
proof (*induct p*)
case (*Neg q*)
consider ‹$\neg\ I \models q$› | ‹$I \models q$›
by *blast*
then show *?case*
proof *cases*
case *1*
then have ‹$lifts\ I\ s\ (pros\ (\sim\ q)) \vdash q \to s$›
using *Neg* **by** *simp*
then have ‹$lifts\ I\ s\ (pros\ (\sim\ q)) \vdash (\sim\ q \to s) \to s$›
using *MP′ Neg1 add-imply* **by** *blast*
with *1* **show** *?thesis*
by *simp*
next
case *2*

then have ‹*lifts I s* (*pros* (∼ *q*)) ⊢ (*q* → *s*) → *s*›
using *Neg* **by** *simp*
then have ‹*lifts I s* (*pros* (∼ *q*)) ⊢ ∼ *q* → *s*›
using *MP′ Neg2 add-imply* **by** *blast*
with *2* **show** *?thesis*
by *simp*
qed
next
case (*Imp q r*)
consider ‹¬ *I* ⊨ *q*› | ‹*I* ⊨ *r*› | ‹*I* ⊨ *q*› ‹¬ *I* ⊨ *r*›
by *blast*
then show *?case*
proof *cases*
case *1*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ *q* → *s*›
using *Imp*(*1*) *lifts-weaken*[**where** *l′* = ‹*pros* (*q* → *r*)›] **by** *simp*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ ((*q* → *r*) → *s*) → *s*›
using *Imp1 MP′ add-imply* **by** *blast*
with *1* **show** *?thesis*
by *simp*
next
case *2*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*r* → *s*) → *s*›
using *Imp*(*2*) *lifts-weaken*[**where** *l′* = ‹*pros* (*q* → *r*)›] **by** *simp*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ ((*q* → *r*) → *s*) → *s*›
using *Imp2 MP′ add-imply* **by** *blast*
with *2* **show** *?thesis*
by *simp*
next
case *3*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*q* → *s*) → *s*› ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ *r* → *s*›
using *Imp lifts-weaken*[**where** *l′* = ‹*pros* (*q* → *r*)›] **by** *simp-all*
then have ‹*lifts I s* (*pros* (*q* → *r*)) ⊢ (*q* → *r*) → *s*›
using *Imp3 MP′ add-imply* **by** *blast*
with *3* **show** *?thesis*
by *simp*
qed
qed (*auto intro*: *Id*)

lemma *lifts-pros*: ‹*I* ⊨ *p* ⟹ *lifts I p* (*pros p*) ⊢ *p*›
proof −
assume ‹*I* ⊨ *p*›
then have ‹*lifts I p* (*pros p*) ⊢ (*p* → *p*) → *p*›
using *lifts-pros-lift*[*of I p p*] **by** *simp*
then show *?thesis*
using *Id MP′ add-imply* **by** *blast*
qed

```
theorem completeness: ‹∀ I. I ⊨ p ⟹ ⊢ p›
proof −
  let ?A = ‹λl I. lifts I p l ⊢ p›
  let ?B = ‹λl. ∀ I. ?A l I ∧ distinct l›
  assume ‹∀ I. I ⊨ p›
  moreover have ‹?B l ⟹ (⋀n l. ?B (n # l) ⟹ ?B l) ⟹ ?B []› for l
    by (induct l) blast+
  moreover have ‹?B (n # l) ⟹ ?B l› for n l
  proof −
    assume ∗: ‹?B (n # l)›
    show ‹?B l›
    proof
      fix I
      from ∗ have ‹?A (n # l) (I(n := True))› ‹?A (n # l) (I(n := False))›
        by blast+
      moreover from ∗ have ‹∀ m ∈ set l. ∀ t. (I(n := t)) m = I m›
        by simp
      ultimately have ‹((· n → p) → p) # lifts I p l ⊢ p› ‹(· n → p) # lifts I p l
⊢ p›
        by (simp-all cong: map-cong)
      then have ‹?A l I›
        using MP′ imply-deduct by blast
      moreover from ∗ have ‹distinct (n # l)›
        by blast
      ultimately show ‹?A l I ∧ distinct l›
        by simp
    qed
  qed
  ultimately have ‹?B []›
    using lifts-pros distinct-pros by blast
  then show ?thesis
    by simp
qed

theorem main: ‹(⊢ p) = (∀ I. I ⊨ p)›
  using soundness completeness by blast
```

2.4 Reference

Numbered lemmas are from Jan Łukasiewicz: Elements of Mathematical Logic (English Tr. 1963)

end