# Soundness and Completeness of Implicational Logic

Asta Halkjær From    Jørgen Villadsen

March 17, 2025

## 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** ‹⊨› *50*) **where**

‹I ⊨ · n = I n› |
‹I ⊨ p → q = (I ⊨ p ⟶ I ⊨ q)›

**inductive** *Ax* (‹⊢ -› *50*) **where**

*Simp*: ‹⊢ p → q → p› |
*Tran*: ‹⊢ (p → q) → (q → r) → p → r› |
*MP*: ‹⊢ p → q ⟹ ⊢ p ⟹ ⊢ q› |
*PR*: ‹⊢ (p → q) → p ⟹ ⊢ p›

## 1.2 Soundness and Derived Formulas

**theorem** *soundness*: ‹⊢ p ⟹ I ⊨ p›
**by** (*induct p rule*: *Ax.induct*) *auto*

**lemma** *Swap*: ‹⊢ (p → q → r) → q → p → r›
**proof** −
**have** ‹⊢ q → (q → r) → r›
**using** *MP PR Simp Tran* **by** *metis*
**then show** *?thesis*
**using** *MP Tran* **by** *meson*
**qed**

**lemma** *Peirce*: ‹⊢ ((p → q) → p) → p›
**using** *MP PR Simp Swap Tran* **by** *meson*

**lemma** *Hilbert*: ‹⊢ (p → p → q) → p → q›
**using** *MP MP Tran Tran Peirce* **.**

**lemma** *Id*: ‹⊢ p → p›
**using** *MP Hilbert Simp* **.**

**lemma** *Tran′*: ‹⊢ (q → r) → (p → q) → p → r›
**using** *MP Swap Tran* **.**

**lemma** *Frege*: ‹⊢ (p → q → r) → (p → q) → p → r›
**using** *MP MP Tran MP MP Tran Swap Tran′ MP Tran′ Hilbert* **.**

**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*

## 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 (λn. 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*: $‹(\vdash p) = (\forall I.\ I \models p)›$
  **using** *soundness completeness* **by** *blast*

### 1.4 Reference

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

**end**

# 2 Formalization of ukasiewicz'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**
  $‹I \models \cdot\ n = I\ n›$ |
  $‹I \models \sim p = (\neg\ I \models p)›$ |
  $‹I \models p \to q = (I \models p \longrightarrow I \models q)›$

**inductive** *Ax* (‹⊢ -› *50*) **where**
  *01*: $‹\vdash (p \to q) \to (q \to r) \to p \to r›$ |
  *02*: $‹\vdash (\sim p \to p) \to p›$ |
  *03*: $‹\vdash p \to \sim p \to q›$ |
  *MP*: $‹\vdash p \to q \Longrightarrow \vdash p \Longrightarrow \vdash q›$

## 2.2 Soundness and Derived Formulas

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

**lemma** *04*: $‹\vdash (((q \to r) \to p \to r) \to s) \to (p \to q) \to s›$
  **using** *MP 01 01* **.**

**lemma** *05*: $‹\vdash (p \to q \to r) \to (s \to q) \to p \to s \to r›$
  **using** *MP 04 04* **.**

**lemma** *06*: $‹\vdash (p \to q) \to ((p \to r) \to s) \to (q \to r) \to s›$
  **using** *MP 04 01* **.**

**lemma** *07*: $‹\vdash (t \to (p \to r) \to s) \to (p \to q) \to t \to (q \to r) \to s›$
  **using** *MP 05 06* **.**

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

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

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

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

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

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

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

**lemma** *16*: ‹$\vdash p \to p$›
  **using** *MP 09 02* .

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

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

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

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

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

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

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

**lemma** *24*: ‹$\vdash ((p \to q) \to p) \to p$›
  **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 ⊢ 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 im-
ply-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 (λn. 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 (Neg q)
  consider ‹¬ I ⊨ q› | ‹I ⊨ q›
    by blast
  then show ?case
  proof cases
    case 1
    then have ‹lifts I s (pros (∼ q)) ⊢ q → s›
      using Neg by simp
    then have ‹lifts I s (pros (∼ q)) ⊢ (∼ q → s) → 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 ukasiewicz: Elements of Mathematical Logic (English Tr. 1963)

```
end
theory Implicational-Logic-Sequent-Calculus imports Main begin

datatype form =
  Pro nat (‹·›) |
  Imp form form (infixr ‹→› 100)

primrec semantics (infix ‹⊨› 50) where
```

```
  ‹I ⊨ ·n = I n› |
  ‹I ⊨ p → q = (I ⊨ p ⟶ I ⊨ q)›

abbreviation sc (‹⟦-⟧›) where ‹⟦I⟧ X Y ≡ (∀ p ∈ set X. I ⊨ p) ⟶ (∃ q ∈ set Y. I ⊨ q)›

inductive SC (infix ‹≫› 50) where
  Imp-L: ‹p → q # X ≫ Y› if ‹X ≫ p # Y› and ‹q # X ≫ Y› |
  Imp-R: ‹X ≫ p → q # Y› if ‹p # X ≫ q # Y› |
  Set-L: ‹X' ≫ Y› if ‹X ≫ Y› and ‹set X' = set X› |
  Set-R: ‹X ≫ Y'› if ‹X ≫ Y› and ‹set Y' = set Y› |
  Basic: ‹p # - ≫ p # -›

function mp where
  ‹mp A B (p → q # C) [] = (mp A B C [p] ∧ mp A B (q # C) [])› |
  ‹mp A B C (p → q # D) = mp A B (p # C) (q # D)› |
  ‹mp A B [] [] = (set A ∩ set B ≠ {})› |
  ‹mp A B (·n # C) [] = mp (n # A) B C []› |
  ‹mp A B C (·n # D) = mp A (n # B) C D›
  by pat-completeness simp-all

termination
  by (relation ‹measure (λ(-, -, C, D). 2 * (∑p ← C @ D. size p) + size (C @ D))›) simp-all

lemma main: ‹(∀ I. ⟦I⟧ (map · A @ C) (map · B @ D)) ⟷ mp A B C D›
  by (induct rule: mp.induct) (auto 5 2)

definition prover (‹⊢›) where ‹⊢ p ≡ mp [] [] [] [p]›

theorem prover-correct: ‹⊢ p ⟷ (∀ I. I ⊨ p)›
  unfolding prover-def by (simp flip: main)

export-code ⊢ in SML

lemma MP: ‹mp A B C D ⟹ set X ⊇ set (map · A @ C) ⟹ set Y ⊇ set (map · B @ D) ⟹ X ≫ Y›
proof (induct A B C D arbitrary: X Y rule: mp.induct[case-names Imp-L Imp-R Basic])
  case (Imp-L A B p q C)
  have
    ‹set (map · A @ C) ⊆ set X›
    ‹set (map · B) ⊆ set Y›
    using Imp-L(4,5) by auto
  moreover from this have
    ‹set (map · A @ C) ⊆ set (q # X)›
    ‹set (map · B) ⊆ set (p # Y)›
    by auto
  ultimately have ‹p → q # X ≫ Y›
```

```
    using Imp-L(1−3) SC.Imp-L by simp
  then show ?case
    using Imp-L(4) Set-L by fastforce
next
  case (Imp-R A B C p q D)
  have
    ‹set (map · A @ C) ⊆ set (p # X)›
    ‹set (map · B @ D) ⊆ set (q # Y)›
    using Imp-R(3,4) by auto
  then have ‹X ≫ p → q # Y›
    using Imp-R(1,2) SC.Imp-R by simp
  then show ?case
    using Imp-R(4) Set-R by fastforce
next
  case (Basic A B)
  obtain n where
    ‹n ∈ set A›
    ‹n ∈ set B›
    using Basic(1) by auto
  then have
    ‹set (·n # X) = set X›
    ‹set (·n # Y) = set Y›
    using Basic(2,3) by auto
  then show ?case
    using Set-L Set-R SC.Basic by metis
qed simp-all

theorem OK: ‹(∀ I. ⟦I⟧ X Y) ⟷ X ≫ Y›
  by (rule, use MP main[of ‹[]› - ‹[]› -] in simp, induct rule: SC.induct) auto

corollary ‹[] ≫ [p] ⟷ (∀ I. I ⊨ p)›
  using OK by force

proposition ‹[] ≫ [p → p]›
proof −
  from Imp-R have ?thesis if ‹[p] ≫ [p]›
    using that by force
  with Basic show ?thesis
    by force
qed

proposition ‹[] ≫ [p → (p → q) → q]›
proof −
  from Imp-R have ?thesis if ‹[p] ≫ [(p → q) → q]›
    using that by force
  with Imp-R have ?thesis if ‹[p → q, p] ≫ [q]›
    using that by force
  with Imp-L have ?thesis if ‹[p] ≫ [p, q]› and ‹[q, p] ≫ [q]›
    using that by force
```

**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** $\langle[] \gg [p \to q \to q \to p]\rangle$
**proof** −
**from** *Imp-R* **have** *?thesis* **if** $\langle[p] \gg [q \to q \to p]\rangle$
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** $\langle[q,\ p] \gg [q \to p]\rangle$
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** $\langle[q,\ q,\ p] \gg [p]\rangle$
**using** *that* **by** *force*
**with** *Set-L* **have** *?thesis* **if** $\langle[p,\ q] \gg [p]\rangle$
**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** $\langle[] \gg [(p \to q) \to p \to q]\rangle$
**proof** −
**from** *Imp-R* **have** *?thesis* **if** $\langle[p \to q] \gg [p \to q]\rangle$
**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** $\langle[] \gg [p \to p \to q \to q]\rangle$
**proof** −
**from** *Imp-R* **have** *?thesis* **if** $\langle[p] \gg [p \to q \to q]\rangle$
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** $\langle[p,\ p] \gg [q \to q]\rangle$
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** $\langle[q,\ p,\ p] \gg [q]\rangle$
**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** $\langle[] \gg [(p \to p \to q) \to p \to q]\rangle$
**proof** −
**from** *Imp-R* **have** *?thesis* **if** $\langle[p \to p \to q] \gg [p \to q]\rangle$
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** $\langle[p,\ p \to p \to q] \gg [q]\rangle$
**using** *that* **by** *force*
**with** *Set-L* **have** *?thesis* **if** $\langle[p \to p \to q,\ p] \gg [q]\rangle$
**using** *that* **by** *force*
**with** *Imp-L* **have** *?thesis* **if** $\langle[p] \gg [p,\ q]\rangle$ **and** $\langle[p \to q,\ p] \gg [q]\rangle$
**using** *that* **by** *force*
**with** *Imp-L* **have** *?thesis* **if** $\langle[p] \gg [p,\ q]\rangle$ **and** $\langle[q,\ p] \gg [q]\rangle$ **and** $\langle[p] \gg [p,\ q]\rangle$

**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** ‹[] $\gg$ [$p \to q \to p$]›
**proof** −
**from** *Imp-R* **have** *?thesis* **if** ‹[$p$] $\gg$ [$q \to p$]›
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** ‹[$q$, $p$] $\gg$ [$p$]›
**using** *that* **by** *force*
**with** *Set-L* **have** *?thesis* **if** ‹[$p$, $q$] $\gg$ [$p$]›
**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** ‹[] $\gg$ [$(p \to r) \to (r \to q) \to p \to q$]›
**proof** −
**from** *Imp-R* **have** *?thesis* **if** ‹[$p \to r$] $\gg$ [$(r \to q) \to p \to q$]›
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** ‹[$r \to q$, $p \to r$] $\gg$ [$p \to q$]›
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** ‹[$p$, $r \to q$, $p \to r$] $\gg$ [$q$]›
**using** *that* **by** *force*
**with** *Set-L* **have** *?thesis* **if** ‹[$r \to q$, $p \to r$, $p$] $\gg$ [$q$]›
**using** *that* **by** *force*
**with** *Imp-L* **have** *?thesis* **if** ‹[$p \to r$, $p$] $\gg$ [$r$, $q$]› **and** ‹[$q$, $p \to r$, $p$] $\gg$ [$q$]›
**using** *that* **by** *force*
**with** *Basic* **have** *?thesis* **if** ‹[$p \to r$, $p$] $\gg$ [$r$, $q$]›
**using** *that* **by** *force*
**with** *Imp-L* **have** *?thesis* **if** ‹[$p$] $\gg$ [$p$, $r$, $q$]› **and** ‹[$r$, $p$] $\gg$ [$r$, $q$]›
**using** *that* **by** *force*
**with** *Basic* **show** *?thesis*
**by** *force*
**qed**

**proposition** ‹[] $\gg$ [$((p \to q) \to p) \to p$]›
**proof** −
**from** *Imp-R* **have** *?thesis* **if** ‹[$(p \to q) \to p$] $\gg$ [$p$]›
**using** *that* **by** *force*
**with** *Imp-L* **have** *?thesis* **if** ‹[] $\gg$ [$p \to q$, $p$]› **and** ‹[$p$] $\gg$ [$p$]›
**using** *that* **by** *force*
**with** *Basic* **have** *?thesis* **if** ‹[] $\gg$ [$p \to q$, $p$]›
**using** *that* **by** *force*
**with** *Imp-R* **have** *?thesis* **if** ‹[$p$] $\gg$ [$q$, $p$]›
**using** *that* **by** *force*
**with** *Set-R* **have** *?thesis* **if** ‹[$p$] $\gg$ [$p$, $q$]›
**using** *that* **by** *force*

```
  with Basic show ?thesis
    by force
qed

end
theory Implicational-Logic-Natural-Deduction imports Main begin

datatype form =
  Pro nat (‹·›) |
  Imp form form (infixr ‹→› 100)

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

inductive Calculus (infix ‹⇝› 50) where
  Assm: ‹p ∈ set A ⟹ A ⇝ p› |
  ImpI: ‹p # A ⇝ q ⟹ A ⇝ p → q› |
  ImpE: ‹A ⇝ p → q ⟹ A ⇝ p ⟹ A ⇝ q› |
  ImpC: ‹p → - # A ⇝ p ⟹ A ⇝ p›

abbreviation natural-deduction (‹⊢ -› [100] 100) where ‹⊢ p ≡ [] ⇝ p›

theorem soundness: ‹A ⇝ p ⟹ ∀ p ∈ set A. I ⊨ p ⟹ I ⊨ p›
  by (induct rule: Calculus.induct) auto

lemma weaken': ‹A ⇝ p ⟹ set A = set B ⟹ B ⇝ p›
proof (induct arbitrary: B rule: Calculus.induct)
  case ImpC
  with Calculus.ImpC show ?case
    using list.set(2) by metis
qed (auto intro: Calculus.intros)

lemma weak: ‹A ⇝ p ⟹ q # A ⇝ p›
proof (induct rule: Calculus.induct)
  case ImpI
  with Calculus.ImpI show ?case
    using insert-commute list.set(2) weaken' by (metis (full-types))
next
  case ImpC
  with Calculus.ImpC show ?case
    using insert-commute list.set(2) weaken' by (metis (full-types))
qed (auto intro: Calculus.intros)

lemma add-assumptions: ‹⊢ p ⟹ A ⇝ p›
  by (induct A) (simp-all add: weak)

lemma weaken: ‹A ⇝ p ⟹ set A ⊆ set B ⟹ B ⇝ p›
proof (induct A arbitrary: p)
```

```
  case (Cons q A)
  moreover from this have ‹A ⇝ q → p› and ‹set A ⊆ set B› and ‹B ⇝ q›
    by (simp-all add: Assm ImpI)
  ultimately show ?case
    using ImpE by blast
qed (simp add: add-assumptions)

lemma deduct: ‹A ⇝ p → q ⟹ p # A ⇝ q›
  using Assm ImpE list.set-intros(1) weak by meson

lemma Peirce: ‹⊢ ((p → q) → p) → p›
  using Assm ImpC ImpI deduct list.set-intros(1) by meson

lemma Simp: ‹⊢ p → q → p›
  by (simp add: Assm ImpI)

lemma Tran: ‹⊢ (p → q) → (q → r) → p → r›
proof –
  have ‹[p, q → r, p → q] ⇝ r›
    using Assm ImpE list.set-intros(1) weak by meson
  then show ?thesis
    using ImpI by blast
qed

lemma Swap: ‹⊢ (p → q → r) → q → p → r›
proof –
  have ‹[p, q, p → q → r] ⇝ r›
    using Assm ImpE list.set-intros(1) weak by meson
  then show ?thesis
    using ImpI by blast
qed

lemma Tran′: ‹⊢ (q → r) → (p → q) → p → r›
  using ImpE Swap Tran .

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

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

lemma Imp3: ‹⊢ ((q → s) → s) → (r → s) → (q → r) → s›
  using ImpE Tran Tran′ by meson

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 simp-all

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

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

lemma lifts-weaken: ‹lifts I s l ⇝ p ⟹ set l ⊆ set l′ ⟹ lifts I s l′ ⇝ p›
proof −
  assume ‹lifts I s l ⇝ p›
  moreover assume ‹set l ⊆ set l′›
  then have ‹set ((lifts I s) l) ⊆ set ((lifts I s) l′)›
    by auto
  ultimately show ?thesis
    using weaken by blast
qed

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› and ‹¬ 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 ImpE add-assumptions 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 ImpE add-assumptions 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 ImpE add-assumptions by blast
    with 3 show ?thesis
      by simp
  qed
```

**qed** (*simp add*: *Assm*)

**lemma** *lifts-pros*: ‹$I \models p \Longrightarrow$ *lifts* $I\ p$ (*pros* $p$) $\rightsquigarrow p$›
**proof** −
  **assume** ‹$I \models p$›
  **then have** ‹*lifts* $I\ p$ (*pros* $p$) $\rightsquigarrow (p \to p) \to p$›
    **using** *lifts-pros-lift*[*of* $I\ p\ p$] **by** *simp*
  **then show** *?thesis*
    **using** *ImpC deduct* **by** *blast*
**qed**

**theorem** *completeness*: ‹$\forall I.\ I \models p \Longrightarrow\ \vdash p$›
**proof** −
  **let** *?A* = ‹$\lambda l\ I.$ *lifts* $I\ p\ l \rightsquigarrow p$›
  **let** *?B* = ‹$\lambda l.\ \forall I.$ *?A* $l\ I \wedge$ *distinct* $l$›
  **assume** ‹$\forall I.\ I \models p$›
  **moreover have** ‹*?B* $l \Longrightarrow (\bigwedge n\ l.$ *?B* $(n \# l) \Longrightarrow$ *?B* $l) \Longrightarrow$ *?B* []› **for** $l$
    **by** (*induct* $l$) *blast+*
  **moreover have** ‹*?B* $(n \# l) \Longrightarrow$ *?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** ‹$\forall m \in$ *set* $l.\ \forall t.\ (I(n := t))\ m = I\ m$›
        **by** *simp*
      **ultimately have**
        ‹$((\cdot n \to p) \to p)\ \#$ *lifts* $I\ p\ l \rightsquigarrow p$›
        ‹$(\cdot n \to p)\ \#$ *lifts* $I\ p\ l \rightsquigarrow p$›
        **by** (*simp-all cong*: *map-cong*)
      **then have** ‹*?A* $l\ I$›
        **using** *ImpE ImpI* **by** *blast*
      **moreover from** $*$ **have** ‹*distinct* $(n \# l)$›
        **by** *blast*
      **ultimately show** ‹*?A* $l\ I \wedge$ *distinct* $l$›
        **by** *simp*
    **qed**
  **qed**
  **ultimately have** ‹*?B* []›
    **using** *lifts-pros distinct-pros* **by** *blast*
  **then show** *?thesis*
    **by** *simp*
**qed**

**primrec** *chain* **where**

```
  ‹chain p [] = p› |
  ‹chain p (q # A) = q → chain p A›

lemma chain-rev: ‹B ⇝ chain p A ⟹ rev A @ B ⇝ p›
  by (induct A arbitrary: B) (simp-all add: deduct)

lemma chain-deduct: ‹⊢ chain p A ⟹ A ⇝ p›
proof (induct A)
  case (Cons q A)
  then have ‹rev (q # A) @ [] ⇝ p›
    using chain-rev by blast
  moreover have ‹set (rev (q # A) @ []) = set (q # A)›
    by simp
  ultimately show ?case
    using weaken by blast
qed simp

lemma chain-semantics: ‹I ⊨ chain p A = ((∀ p ∈ set A. I ⊨ p) ⟶ I ⊨ p)›
  by (induct A) auto

theorem main: ‹A ⇝ p = (∀ I. (∀ p ∈ set A. I ⊨ p) ⟶ I ⊨ p)›
  using chain-deduct chain-semantics completeness soundness by meson

end
```