

Notable Examples in Isabelle/Pure

December 17, 2025

1 A simple formulation of First-Order Logic

The subsequent theory development illustrates single-sorted intuitionistic first-order logic with equality, formulated within the Pure framework.

```
theory First_Order_Logic
  imports Pure
begin
```

1.1 Abstract syntax

```
typedcl i
typedcl o
```

```
judgment Trueprop :: o  $\Rightarrow$  prop ( $\langle \_ \rangle$  5)
```

1.2 Propositional logic

```
axiomatization false :: o ( $\langle \bot \rangle$ )
  where falseE [elim]:  $\bot \Longrightarrow A$ 
```

```
axiomatization imp :: o  $\Rightarrow$  o  $\Rightarrow$  o (infixr  $\langle \longrightarrow \rangle$  25)
  where impI [intro]:  $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ 
    and mp [dest]:  $A \longrightarrow B \Longrightarrow A \Longrightarrow B$ 
```

```
axiomatization conj :: o  $\Rightarrow$  o  $\Rightarrow$  o (infixr  $\langle \wedge \rangle$  35)
  where conjI [intro]:  $A \Longrightarrow B \Longrightarrow A \wedge B$ 
    and conjD1:  $A \wedge B \Longrightarrow A$ 
    and conjD2:  $A \wedge B \Longrightarrow B$ 
```

```
theorem conjE [elim]:
  assumes  $A \wedge B$ 
  obtains  $A$  and  $B$ 
proof
  from  $\langle A \wedge B \rangle$  show  $A$ 
```

```

    by (rule conjD1)
  from  $\langle A \wedge B \rangle$  show  $B$ 
    by (rule conjD2)
qed

```

```

axiomatization disj ::  $o \Rightarrow o \Rightarrow o$  (infixr  $\langle \vee \rangle$  30)
  where disjE [elim]:  $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$ 
    and disjI1 [intro]:  $A \Longrightarrow A \vee B$ 
    and disjI2 [intro]:  $B \Longrightarrow A \vee B$ 

```

```

definition true ::  $o$  ( $\langle \top \rangle$ )
  where  $\top \equiv \perp \longrightarrow \perp$ 

```

```

theorem trueI [intro]:  $\top$ 
  unfolding true_def ..

```

```

definition not ::  $o \Rightarrow o$  ( $\langle \neg \_ \rangle$  [40] 40)
  where  $\neg A \equiv A \longrightarrow \perp$ 

```

```

theorem notI [intro]:  $(A \Longrightarrow \perp) \Longrightarrow \neg A$ 
  unfolding not_def ..

```

```

theorem notE [elim]:  $\neg A \Longrightarrow A \Longrightarrow B$ 
  unfolding not_def

```

```

proof -
  assume  $A \longrightarrow \perp$  and  $A$ 
  then have  $\perp$  ..
  then show  $B$  ..
qed

```

```

definition iff ::  $o \Rightarrow o \Rightarrow o$  (infixr  $\langle \longleftrightarrow \rangle$  25)
  where  $A \longleftrightarrow B \equiv (A \longrightarrow B) \wedge (B \longrightarrow A)$ 

```

```

theorem iffI [intro]:
  assumes  $A \Longrightarrow B$ 
    and  $B \Longrightarrow A$ 
  shows  $A \longleftrightarrow B$ 
  unfolding iff_def
proof
  from  $\langle A \Longrightarrow B \rangle$  show  $A \longrightarrow B$  ..
  from  $\langle B \Longrightarrow A \rangle$  show  $B \longrightarrow A$  ..
qed

```

```

theorem iff1 [elim]:
  assumes  $A \longleftrightarrow B$  and  $A$ 

```

shows B
proof –
 from $\langle A \longleftrightarrow B \rangle$ have $(A \longrightarrow B) \wedge (B \longrightarrow A)$
 unfolding *iff_def* .
 then have $A \longrightarrow B$..
 from *this* and $\langle A \rangle$ show B ..
qed

theorem *iff2* [*elim*]:
 assumes $A \longleftrightarrow B$ and B
 shows A
proof –
 from $\langle A \longleftrightarrow B \rangle$ have $(A \longrightarrow B) \wedge (B \longrightarrow A)$
 unfolding *iff_def* .
 then have $B \longrightarrow A$..
 from *this* and $\langle B \rangle$ show A ..
qed

1.3 Equality

axiomatization *equal* :: $i \Rightarrow i \Rightarrow o$ (**infixl** $\langle \Rightarrow \rangle$ 50)
 where *refl* [*intro*]: $x = x$
 and *subst*: $x = y \Longrightarrow P\ x \Longrightarrow P\ y$

theorem *trans* [*trans*]: $x = y \Longrightarrow y = z \Longrightarrow x = z$
 by (*rule subst*)

theorem *sym* [*sym*]: $x = y \Longrightarrow y = x$
proof –
 assume $x = y$
 from *this* and *refl* show $y = x$
 by (*rule subst*)
qed

1.4 Quantifiers

axiomatization *All* :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\langle \forall \rangle$ 10)
 where *allI* [*intro*]: $(\bigwedge x. P\ x) \Longrightarrow \forall x. P\ x$
 and *allD* [*dest*]: $\forall x. P\ x \Longrightarrow P\ a$

axiomatization *Ex* :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\langle \exists \rangle$ 10)
 where *exI* [*intro*]: $P\ a \Longrightarrow \exists x. P\ x$
 and *exE* [*elim*]: $\exists x. P\ x \Longrightarrow (\bigwedge x. P\ x \Longrightarrow C) \Longrightarrow C$

lemma $(\exists x. P\ (f\ x)) \longrightarrow (\exists y. P\ y)$
proof
 assume $\exists x. P\ (f\ x)$
 then obtain x where $P\ (f\ x)$..
 then show $\exists y. P\ y$..

```

qed

lemma ( $\exists x. \forall y. R\ x\ y$ )  $\longrightarrow$  ( $\forall y. \exists x. R\ x\ y$ )
proof
  assume  $\exists x. \forall y. R\ x\ y$ 
  then obtain  $x$  where  $\forall y. R\ x\ y$  ..
  show  $\forall y. \exists x. R\ x\ y$ 
  proof
    fix  $y$ 
    from  $\langle \forall y. R\ x\ y \rangle$  have  $R\ x\ y$  ..
    then show  $\exists x. R\ x\ y$  ..
  qed
qed

end

```

2 Foundations of HOL

```

theory Higher_Order_Logic
  imports Pure
begin

```

The following theory development illustrates the foundations of Higher-Order Logic. The “HOL” logic that is given here resembles [2] and its predecessor [1], but the order of axiomatizations and defined connectives has been adapted to modern presentations of λ -calculus and Constructive Type Theory. Thus it fits nicely to the underlying Natural Deduction framework of Isabelle/Pure and Isabelle/Isar.

3 HOL syntax within Pure

```

class type
default_sort type

typedcl o
instance o :: type ..
instance fun :: (type, type) type ..

judgment Trueprop ::  $o \Rightarrow prop$  ( $\langle \_ \rangle$  5)

```

4 Minimal logic (axiomatization)

```

axiomatization imp ::  $o \Rightarrow o \Rightarrow o$  (infixr  $\langle \longrightarrow \rangle$  25)
  where impI [intro]:  $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ 
    and impE [dest, trans]:  $A \longrightarrow B \Longrightarrow A \Longrightarrow B$ 

axiomatization All ::  $(\iota a \Rightarrow o) \Rightarrow o$  (binder  $\langle \forall \rangle$  10)

```

where *allI* [*intro*]: $(\bigwedge x. P\ x) \implies \forall x. P\ x$
and *allE* [*dest*]: $\forall x. P\ x \implies P\ a$
lemma *atomize_imp* [*atomize*]: $(A \implies B) \equiv \text{Trueprop } (A \longrightarrow B)$
by *standard* (*fact impI*, *fact impE*)
lemma *atomize_all* [*atomize*]: $(\bigwedge x. P\ x) \equiv \text{Trueprop } (\forall x. P\ x)$
by *standard* (*fact allI*, *fact allE*)

4.0.1 Derived connectives

definition *False* :: *o*
where *False* $\equiv \forall A. A$

lemma *FalseE* [*elim*]:
assumes *False*
shows *A*
proof –
from $\langle \text{False} \rangle$ **have** $\forall A. A$ **by** (*simp only: False_def*)
then show *A* ..
qed

definition *True* :: *o*
where *True* $\equiv \text{False} \longrightarrow \text{False}$

lemma *TrueI* [*intro*]: *True*
unfolding *True_def* ..

definition *not* :: *o* \Rightarrow *o* (\neg [40] 40)
where *not* $\equiv \lambda A. A \longrightarrow \text{False}$

lemma *notI* [*intro*]:
assumes $A \implies \text{False}$
shows $\neg A$
using *assms* **unfolding** *not_def* ..

lemma *notE* [*elim*]:
assumes $\neg A$ **and** *A*
shows *B*
proof –
from $\langle \neg A \rangle$ **have** $A \longrightarrow \text{False}$ **by** (*simp only: not_def*)
from *this* **and** $\langle A \rangle$ **have** *False* ..
then show *B* ..
qed

lemma *notE'*: $A \implies \neg A \implies B$
by (*rule notE*)

lemmas *contradiction* = *notE notE'* — proof by contradiction in any order

definition *conj* :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\langle \wedge \rangle$ 35)
 where $A \wedge B \equiv \forall C. (A \longrightarrow B \longrightarrow C) \longrightarrow C$

lemma *conjI* [*intro*]:
 assumes *A* and *B*
 shows $A \wedge B$
 unfolding *conj_def*
proof
 fix *C*
 show $(A \longrightarrow B \longrightarrow C) \longrightarrow C$
proof
 assume $A \longrightarrow B \longrightarrow C$
 also note $\langle A \rangle$
 also note $\langle B \rangle$
 finally show *C* .
 qed
 qed

lemma *conjE* [*elim*]:
 assumes $A \wedge B$
 obtains *A* and *B*
proof
 from $\langle A \wedge B \rangle$ have *: $(A \longrightarrow B \longrightarrow C) \longrightarrow C$ for *C*
 unfolding *conj_def* ..
 show *A*
proof –
 note * [*of A*]
 also have $A \longrightarrow B \longrightarrow A$
proof
 assume *A*
 then show $B \longrightarrow A$..
 qed
 finally show ?*thesis* .
 qed
 show *B*
proof –
 note * [*of B*]
 also have $A \longrightarrow B \longrightarrow B$
proof
 show $B \longrightarrow B$..
 qed
 finally show ?*thesis* .
 qed
 qed

```

definition disj ::  $o \Rightarrow o \Rightarrow o$  (infixr  $\langle \vee \rangle$  30)
  where  $A \vee B \equiv \forall C. (A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow C$ 

lemma disjI1 [intro]:
  assumes  $A$ 
  shows  $A \vee B$ 
  unfolding disj_def
proof
  fix  $C$ 
  show  $(A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow C$ 
  proof
    assume  $A \longrightarrow C$ 
    from this and  $\langle A \rangle$  have  $C$  ..
    then show  $(B \longrightarrow C) \longrightarrow C$  ..
  qed
qed

lemma disjI2 [intro]:
  assumes  $B$ 
  shows  $A \vee B$ 
  unfolding disj_def
proof
  fix  $C$ 
  show  $(A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow C$ 
  proof
    show  $(B \longrightarrow C) \longrightarrow C$ 
    proof
      assume  $B \longrightarrow C$ 
      from this and  $\langle B \rangle$  show  $C$  ..
    qed
  qed
qed

lemma disjE [elim]:
  assumes  $A \vee B$ 
  obtains  $(a) A \mid (b) B$ 
proof –
  from  $\langle A \vee B \rangle$  have  $(A \longrightarrow thesis) \longrightarrow (B \longrightarrow thesis) \longrightarrow thesis$ 
  unfolding disj_def ..
  also have  $A \longrightarrow thesis$ 
  proof
    assume  $A$ 
    then show thesis by (rule a)
  qed
  also have  $B \longrightarrow thesis$ 
  proof
    assume  $B$ 
    then show thesis by (rule b)

```

```

qed
finally show thesis .
qed

```

```

definition Ex :: ('a  $\Rightarrow$  o)  $\Rightarrow$  o (binder  $\langle \exists \rangle$  10)
  where  $\exists x. P\ x \equiv \forall C. (\forall x. P\ x \longrightarrow C) \longrightarrow C$ 

```

```

lemma exI [intro]:  $P\ a \Longrightarrow \exists x. P\ x$ 
  unfolding Ex_def
proof
  fix C
  assume  $P\ a$ 
  show  $(\forall x. P\ x \longrightarrow C) \longrightarrow C$ 
  proof
    assume  $\forall x. P\ x \longrightarrow C$ 
    then have  $P\ a \longrightarrow C$  ..
    from this and  $\langle P\ a \rangle$  show  $C$  ..
  qed
qed

```

```

lemma exE [elim]:
  assumes  $\exists x. P\ x$ 
  obtains (that)  $x$  where  $P\ x$ 
proof -
  from  $\langle \exists x. P\ x \rangle$  have  $(\forall x. P\ x \longrightarrow \textit{thesis}) \longrightarrow \textit{thesis}$ 
  unfolding Ex_def ..
  also have  $\forall x. P\ x \longrightarrow \textit{thesis}$ 
  proof
    fix x
    show  $P\ x \longrightarrow \textit{thesis}$ 
    proof
      assume  $P\ x$ 
      then show thesis by (rule that)
    qed
  qed
  finally show thesis .
qed

```

4.0.2 Extensional equality

```

axiomatization equal :: 'a  $\Rightarrow$  'a  $\Rightarrow$  o (infixl  $\langle \Rightarrow \rangle$  50)
  where refl [intro]:  $x = x$ 
    and subst:  $x = y \Longrightarrow P\ x \Longrightarrow P\ y$ 

```

```

abbreviation not_equal :: 'a  $\Rightarrow$  'a  $\Rightarrow$  o (infixl  $\langle \neq \rangle$  50)
  where  $x \neq y \equiv \neg (x = y)$ 

```

```

abbreviation iff :: o  $\Rightarrow$  o  $\Rightarrow$  o (infixr  $\langle \longleftrightarrow \rangle$  25)

```



```

where  $A \longleftrightarrow B \equiv A = B$ 

axiomatization
  where  $ext$  [intro]:  $(\bigwedge x. f\ x = g\ x) \implies f = g$ 
    and  $iff$  [intro]:  $(A \implies B) \implies (B \implies A) \implies A \longleftrightarrow B$ 
    for  $f\ g :: 'a \Rightarrow 'b$ 

lemma  $sym$  [sym]:  $y = x$  if  $x = y$ 
  using that by (rule subst) (rule refl)

lemma [trans]:  $x = y \implies P\ y \implies P\ x$ 
  by (rule subst) (rule sym)

lemma [trans]:  $P\ x \implies x = y \implies P\ y$ 
  by (rule subst)

lemma  $arg\_cong$ :  $f\ x = f\ y$  if  $x = y$ 
  using that by (rule subst) (rule refl)

lemma  $fun\_cong$ :  $f\ x = g\ x$  if  $f = g$ 
  using that by (rule subst) (rule refl)

lemma  $trans$  [trans]:  $x = y \implies y = z \implies x = z$ 
  by (rule subst)

lemma  $iff1$  [elim]:  $A \longleftrightarrow B \implies A \implies B$ 
  by (rule subst)

lemma  $iff2$  [elim]:  $A \longleftrightarrow B \implies B \implies A$ 
  by (rule subst) (rule sym)

```

4.1 Cantor's Theorem

Cantor's Theorem states that there is no surjection from a set to its powerset. The subsequent formulation uses elementary λ -calculus and predicate logic, with standard introduction and elimination rules.

```

lemma  $iff\_contradiction$ :
  assumes *:  $\neg A \longleftrightarrow A$ 
  shows  $C$ 
proof (rule notE)
  show  $\neg A$ 
  proof
    assume  $A$ 
    with * have  $\neg A$  ..
    from this and  $\langle A \rangle$  show  $False$  ..
  qed
  with * show  $A$  ..
qed

```

```

theorem Cantor:  $\neg (\exists f :: 'a \Rightarrow 'a \Rightarrow o. \forall A. \exists x. A = f\ x)$ 
proof
  assume  $\exists f :: 'a \Rightarrow 'a \Rightarrow o. \forall A. \exists x. A = f\ x$ 
  then obtain  $f :: 'a \Rightarrow 'a \Rightarrow o$  where  $*$ :  $\forall A. \exists x. A = f\ x$  ..
  let  $?D = \lambda x. \neg f\ x\ x$ 
  from  $*$  have  $\exists x. ?D = f\ x$  ..
  then obtain  $a$  where  $?D = f\ a$  ..
  then have  $?D\ a \longleftrightarrow f\ a\ a$  using refl by (rule subst)
  then have  $\neg f\ a\ a \longleftrightarrow f\ a\ a$  .
  then show False by (rule iff_contradiction)
qed

```

4.2 Characterization of Classical Logic

The subsequent rules of classical reasoning are all equivalent.

```

locale classical =
  assumes classical:  $(\neg A \Longrightarrow A) \Longrightarrow A$ 
  — predicate definition and hypothetical context
begin

```

```

lemma classical_contradiction:
  assumes  $\neg A \Longrightarrow False$ 
  shows  $A$ 
proof (rule classical)
  assume  $\neg A$ 
  then have False by (rule assms)
  then show  $A$  ..
qed

```

```

lemma double_negation:
  assumes  $\neg \neg A$ 
  shows  $A$ 
proof (rule classical_contradiction)
  assume  $\neg A$ 
  with  $\langle \neg \neg A \rangle$  show False by (rule contradiction)
qed

```

```

lemma tertium_non_datur:  $A \vee \neg A$ 
proof (rule double_negation)
  show  $\neg \neg (A \vee \neg A)$ 
  proof
    assume  $\neg (A \vee \neg A)$ 
    have  $\neg A$ 
    proof
      assume  $A$  then have  $A \vee \neg A$  ..
      with  $\langle \neg (A \vee \neg A) \rangle$  show False by (rule contradiction)
    qed
    then have  $A \vee \neg A$  ..
    with  $\langle \neg (A \vee \neg A) \rangle$  show False by (rule contradiction)
  qed

```

```

    qed
  qed

lemma classical_cases:
  obtains  $A \mid \neg A$ 
  using tertium_non_datur
proof
  assume  $A$ 
  then show thesis ..
next
  assume  $\neg A$ 
  then show thesis ..
qed

end

lemma classical_if_cases: classical
  if cases:  $\bigwedge A C. (A \implies C) \implies (\neg A \implies C) \implies C$ 
proof
  fix  $A$ 
  assume *:  $\neg A \implies A$ 
  show  $A$ 
  proof (rule cases)
    assume  $A$ 
    then show  $A$  .
  next
    assume  $\neg A$ 
    then show  $A$  by (rule *)
  qed
qed

```

5 Peirce's Law

Peirce's Law is another characterization of classical reasoning. Its statement only requires implication.

```

theorem (in classical) Peirce's_Law:  $((A \longrightarrow B) \longrightarrow A) \longrightarrow A$ 
proof
  assume *:  $(A \longrightarrow B) \longrightarrow A$ 
  show  $A$ 
  proof (rule classical)
    assume  $\neg A$ 
    have  $A \longrightarrow B$ 
    proof
      assume  $A$ 
      with  $\langle \neg A \rangle$  show  $B$  by (rule contradiction)
    qed
    with * show  $A$  ..
  qed
qed

```

qed

6 Hilbert's choice operator (axiomatization)

axiomatization $Eps :: ('a \Rightarrow o) \Rightarrow 'a$
where $someI: P\ x \Longrightarrow P\ (Eps\ P)$

syntax $_Eps :: pttrn \Rightarrow o \Rightarrow 'a$ ($\langle \langle indent=3\ notation=\langle binder\ SOME \rangle \rangle SOME$
 $_./\ _ \rangle [0, 10] 10$)

syntax_consts $_Eps \equiv Eps$

translations $SOME\ x.\ P \equiv CONST\ Eps\ (\lambda x.\ P)$

It follows a derivation of the classical law of tertium-non-datur by means of Hilbert's choice operator (due to Berghofer, Beeson, Harrison, based on a proof by Diaconescu).

theorem *Diaconescu*: $A \vee \neg A$

proof –

let $?P = \lambda x.\ (A \wedge x) \vee \neg x$

let $?Q = \lambda x.\ (A \wedge \neg x) \vee x$

have $a: ?P\ (Eps\ ?P)$

proof (*rule someI*)

have $\neg\ False\ ..$

then show $?P\ False\ ..$

qed

have $b: ?Q\ (Eps\ ?Q)$

proof (*rule someI*)

have $True\ ..$

then show $?Q\ True\ ..$

qed

from a **show** $?thesis$

proof

assume $A \wedge Eps\ ?P$

then have $A\ ..$

then show $?thesis\ ..$

next

assume $\neg\ Eps\ ?P$

from b **show** $?thesis$

proof

assume $A \wedge \neg\ Eps\ ?Q$

then have $A\ ..$

then show $?thesis\ ..$

next

assume $Eps\ ?Q$

have $neg: ?P \neq ?Q$

proof

```

    assume ?P = ?Q
    then have Eps ?P  $\longleftrightarrow$  Eps ?Q by (rule arg_cong)
    also note  $\langle \text{Eps ?Q} \rangle$ 
    finally have Eps ?P .
    with  $\langle \neg \text{Eps ?P} \rangle$  show False by (rule contradiction)
  qed
  have  $\neg A$ 
  proof
    assume A
    have ?P = ?Q
    proof (rule ext)
      show ?P x  $\longleftrightarrow$  ?Q x for x
      proof
        assume ?P x
        then show ?Q x
        proof
          assume  $\neg x$ 
          with  $\langle A \rangle$  have  $A \wedge \neg x$  ..
          then show ?thesis ..
        next
          assume  $A \wedge x$ 
          then have x ..
          then show ?thesis ..
        qed
      qed
    next
      assume ?Q x
      then show ?P x
      proof
        assume  $A \wedge \neg x$ 
        then have  $\neg x$  ..
        then show ?thesis ..
      next
        assume x
        with  $\langle A \rangle$  have  $A \wedge x$  ..
        then show ?thesis ..
      qed
    qed
  qed
  with neq show False by (rule contradiction)
  qed
  then show ?thesis ..
  qed
  qed
  qed

```

This means, the hypothetical predicate *classical* always holds unconditionally (with all consequences).

```

interpretation classical
proof (rule classical_if_cases)

```

```

fix  $A\ C$ 
assume *:  $A \implies C$ 
  and **:  $\neg A \implies C$ 
from Diaconescu [of A] show  $C$ 
proof
  assume  $A$ 
  then show  $C$  by (rule *)
next
  assume  $\neg A$ 
  then show  $C$  by (rule **)
qed
qed

thm classical
  classical_contradiction
  double_negation
  tertium_non_datur
  classical_cases
  Peirce's_Law

end

```

References

- [1] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
- [2] M. J. C. Gordon. HOL: A machine oriented formulation of higher order logic. Technical Report 68, University of Cambridge Computer Laboratory, 1985.