Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

September 11, 2023

Contents

1 Intuitionistic first-order logic

theory *IFOL*
imports *Pure*
abbrevs *?<* = $\exists_{\leq 1}$
begin

⟨*ML*⟩

1.1 Syntax and axiomatic basis

⟨*ML*⟩

class *term*
default-sort ‹*term*›

typedecl *o*

judgment
Trueprop :: ‹*o* ⇒ *prop*› (‹(-)› *5*)

1.1.1 Equality

axiomatization
eq :: ‹['*a*, '*a*] ⇒ *o*› (**infixl** ‹=› *50*)
where
refl: ‹$a = a$› **and**
subst: ‹$a = b \Longrightarrow P(a) \Longrightarrow P(b)$›

1.1.2 Propositional logic

axiomatization
False :: ‹*o*› **and**

$conj$ :: ‹[o, o] => o› (**infixr** ‹∧› 35) **and**
$disj$ :: ‹[o, o] => o› (**infixr** ‹∨› 30) **and**
imp :: ‹[o, o] => o› (**infixr** ‹⟶› 25)
where
$conjI$: ‹$[\![P;\ \ Q]\!] \Longrightarrow P \wedge Q$› **and**
$conjunct1$: ‹$P \wedge Q \Longrightarrow P$› **and**
$conjunct2$: ‹$P \wedge Q \Longrightarrow Q$› **and**

$disjI1$: ‹$P \Longrightarrow P \vee Q$› **and**
$disjI2$: ‹$Q \Longrightarrow P \vee Q$› **and**
$disjE$: ‹$[\![P \vee Q;\ P \Longrightarrow R;\ Q \Longrightarrow R]\!] \Longrightarrow R$› **and**

$impI$: ‹$(P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q$› **and**
mp: ‹$[\![P \longrightarrow Q;\ P]\!] \Longrightarrow Q$› **and**

$FalseE$: ‹$False \Longrightarrow P$›

1.1.3 Quantifiers

axiomatization
All :: ‹$('a \Rightarrow o) \Rightarrow o$› (**binder** ‹∀ › 10) **and**
Ex :: ‹$('a \Rightarrow o) \Rightarrow o$› (**binder** ‹∃ › 10)
where
$allI$: ‹$(\bigwedge x.\ P(x)) \Longrightarrow (\forall x.\ P(x))$› **and**
$spec$: ‹$(\forall x.\ P(x)) \Longrightarrow P(x)$› **and**
exI: ‹$P(x) \Longrightarrow (\exists x.\ P(x))$› **and**
exE: ‹$[\![\exists x.\ P(x);\ \bigwedge x.\ P(x) \Longrightarrow R]\!] \Longrightarrow R$›

1.1.4 Definitions

definition ‹$True \equiv False \longrightarrow False$›

definition Not (‹¬ -› [40] 40)
where $not\text{-}def$: ‹$\neg\ P \equiv P \longrightarrow False$›

definition iff (**infixr** ‹⟷› 25)
where ‹$P \longleftrightarrow Q \equiv (P \longrightarrow Q) \wedge (Q \longrightarrow P)$›

definition $Uniq$ :: $('a \Rightarrow o) \Rightarrow o$
where ‹$Uniq(P) \equiv (\forall x\ y.\ P(x) \longrightarrow P(y) \longrightarrow y = x)$›

definition $Ex1$ :: ‹$('a \Rightarrow o) \Rightarrow o$› (**binder** ‹∃!› 10)
where $ex1\text{-}def$: ‹$\exists !x.\ P(x) \equiv \exists x.\ P(x) \wedge (\forall y.\ P(y) \longrightarrow y = x)$›

axiomatization where — Reflection, admissible
$eq\text{-}reflection$: ‹$(x = y) \Longrightarrow (x \equiv y)$› **and**
$iff\text{-}reflection$: ‹$(P \longleftrightarrow Q) \Longrightarrow (P \equiv Q)$›

abbreviation $not\text{-}equal$ :: ‹$['a,\ 'a] \Rightarrow o$› (**infixl** ‹≠› 50)
where ‹$x \neq y \equiv \neg\ (x = y)$›

syntax *-Uniq* :: *pttrn* ⇒ *o* ⇒ *o* ((*2*$\exists_{\leq 1}$ -./ -) [*0*, *10*] *10*)
translations $\exists_{\leq 1}x.\ P \rightleftharpoons$ *CONST Uniq* (λ*x*. *P*)

⟨*ML*⟩

1.1.5 Old-style ASCII syntax

notation (*ASCII*)
not-equal (**infixl** ‹~=› *50*) **and**
Not (‹~ -› [*40*] *40*) **and**
conj (**infixr** ‹&› *35*) **and**
disj (**infixr** ‹|› *30*) **and**
All (**binder** ‹*ALL* › *10*) **and**
Ex (**binder** ‹*EX* › *10*) **and**
Ex1 (**binder** ‹*EX*! › *10*) **and**
imp (**infixr** ‹−−>› *25*) **and**
iff (**infixr** ‹<−>› *25*)

1.2 Lemmas and proof tools

lemmas *strip* = *impI allI*

lemma *TrueI*: ‹*True*›
⟨*proof*⟩

1.2.1 Sequent-style elimination rules for ∧ ⟶ and ∀

lemma *conjE*:
assumes *major*: ‹$P \wedge Q$›
and *r*: ‹$[\![P;\ Q]\!] \Longrightarrow R$›
shows ‹*R*›
⟨*proof*⟩

lemma *impE*:
assumes *major*: ‹$P \longrightarrow Q$›
and ‹*P*›
and *r*: ‹$Q \Longrightarrow R$›
shows ‹*R*›
⟨*proof*⟩

lemma *allE*:
assumes *major*: ‹$\forall x.\ P(x)$›
and *r*: ‹$P(x) \Longrightarrow R$›
shows ‹*R*›
⟨*proof*⟩

Duplicates the quantifier; for use with `eresolve_tac`.

lemma *all-dupE*:
assumes *major*: ‹$\forall x.\ P(x)$›

and r: ‹$\llbracket P(x); \forall x.\ P(x)\rrbracket \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

1.2.2 Negation rules, which translate between $\neg P$ and $P \longrightarrow False$

lemma *notI*: ‹$(P \Longrightarrow False) \Longrightarrow \neg P$›
⟨*proof*⟩

lemma *notE*: ‹$\llbracket \neg P; P\rrbracket \Longrightarrow R$›
⟨*proof*⟩

lemma *rev-notE*: ‹$\llbracket P; \neg P\rrbracket \Longrightarrow R$›
⟨*proof*⟩

This is useful with the special implication rules for each kind of P.

lemma *not-to-imp*:
assumes ‹$\neg P$›
and r: ‹$P \longrightarrow False \Longrightarrow Q$›
shows ‹Q›
⟨*proof*⟩

For substitution into an assumption P, reduce Q to $P \longrightarrow Q$, substitute into this implication, then apply *impI* to move P back into the assumptions.

lemma *rev-mp*: ‹$\llbracket P; P \longrightarrow Q\rrbracket \Longrightarrow Q$›
⟨*proof*⟩

Contrapositive of an inference rule.

lemma *contrapos*:
assumes *major*: ‹$\neg Q$›
and *minor*: ‹$P \Longrightarrow Q$›
shows ‹$\neg P$›
⟨*proof*⟩

1.2.3 Modus Ponens Tactics

Finds $P \longrightarrow Q$ and P in the assumptions, replaces implication by Q.

⟨*ML*⟩

1.3 If-and-only-if

lemma *iffI*: ‹$\llbracket P \Longrightarrow Q; Q \Longrightarrow P\rrbracket \Longrightarrow P \longleftrightarrow Q$›
⟨*proof*⟩

lemma *iffE*:
assumes *major*: ‹$P \longleftrightarrow Q$›
and r: ‹$\llbracket P \longrightarrow Q; Q \longrightarrow P\rrbracket \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

1.3.1 Destruct rules for $\longleftrightarrow$ similar to Modus Ponens

lemma *iffD1*: ‹$[\![P \longleftrightarrow Q;\ P]\!] \Longrightarrow Q$›
  ⟨*proof*⟩

lemma *iffD2*: ‹$[\![P \longleftrightarrow Q;\ Q]\!] \Longrightarrow P$›
  ⟨*proof*⟩

lemma *rev-iffD1*: ‹$[\![P;\ P \longleftrightarrow Q]\!] \Longrightarrow Q$›
  ⟨*proof*⟩

lemma *rev-iffD2*: ‹$[\![Q;\ P \longleftrightarrow Q]\!] \Longrightarrow P$›
  ⟨*proof*⟩

lemma *iff-refl*: ‹$P \longleftrightarrow P$›
  ⟨*proof*⟩

lemma *iff-sym*: ‹$Q \longleftrightarrow P \Longrightarrow P \longleftrightarrow Q$›
  ⟨*proof*⟩

lemma *iff-trans*: ‹$[\![P \longleftrightarrow Q;\ Q \longleftrightarrow R]\!] \Longrightarrow P \longleftrightarrow R$›
  ⟨*proof*⟩

1.4 Unique existence

NOTE THAT the following 2 quantifications:

- $\exists!x$ such that [$\exists!y$ such that P(x,y)] (sequential)
- $\exists!x,y$ such that P(x,y) (simultaneous)

do NOT mean the same thing. The parser treats $\exists!x\ y.P(x,y)$ as sequential.

lemma *ex1I*: ‹$P(a) \Longrightarrow (\bigwedge x.\ P(x) \Longrightarrow x = a) \Longrightarrow \exists!x.\ P(x)$›
  ⟨*proof*⟩

Sometimes easier to use: the premises have no shared variables. Safe!

lemma *ex-ex1I*: ‹$\exists x.\ P(x) \Longrightarrow (\bigwedge x\ y.\ [\![P(x);\ P(y)]\!] \Longrightarrow x = y) \Longrightarrow \exists!x.\ P(x)$›
  ⟨*proof*⟩

lemma *ex1E*: ‹$\exists!\ x.\ P(x) \Longrightarrow (\bigwedge x.\ [\![P(x);\ \forall y.\ P(y) \longrightarrow y = x]\!] \Longrightarrow R) \Longrightarrow R$›
  ⟨*proof*⟩

1.4.1 $\longleftrightarrow$ congruence rules for simplification

Use *iffE* on a premise. For *conj-cong*, *imp-cong*, *all-cong*, *ex-cong*.

⟨*ML*⟩

lemma *conj-cong*:

assumes ‹$P \longleftrightarrow P'$›
and ‹$P' \Longrightarrow Q \longleftrightarrow Q'$›
shows ‹$(P \wedge Q) \longleftrightarrow (P' \wedge Q')$›
⟨*proof*⟩

Reversed congruence rule! Used in ZF/Order.

lemma *conj-cong2*:
assumes ‹$P \longleftrightarrow P'$›
and ‹$P' \Longrightarrow Q \longleftrightarrow Q'$›
shows ‹$(Q \wedge P) \longleftrightarrow (Q' \wedge P')$›
⟨*proof*⟩

lemma *disj-cong*:
assumes ‹$P \longleftrightarrow P'$› **and** ‹$Q \longleftrightarrow Q'$›
shows ‹$(P \vee Q) \longleftrightarrow (P' \vee Q')$›
⟨*proof*⟩

lemma *imp-cong*:
assumes ‹$P \longleftrightarrow P'$›
and ‹$P' \Longrightarrow Q \longleftrightarrow Q'$›
shows ‹$(P \longrightarrow Q) \longleftrightarrow (P' \longrightarrow Q')$›
⟨*proof*⟩

lemma *iff-cong*: ‹$[\![P \longleftrightarrow P'; Q \longleftrightarrow Q']\!] \Longrightarrow (P \longleftrightarrow Q) \longleftrightarrow (P' \longleftrightarrow Q')$›
⟨*proof*⟩

lemma *not-cong*: ‹$P \longleftrightarrow P' \Longrightarrow \neg P \longleftrightarrow \neg P'$›
⟨*proof*⟩

lemma *all-cong*:
assumes ‹$\bigwedge x. P(x) \longleftrightarrow Q(x)$›
shows ‹$(\forall x. P(x)) \longleftrightarrow (\forall x. Q(x))$›
⟨*proof*⟩

lemma *ex-cong*:
assumes ‹$\bigwedge x. P(x) \longleftrightarrow Q(x)$›
shows ‹$(\exists x. P(x)) \longleftrightarrow (\exists x. Q(x))$›
⟨*proof*⟩

lemma *ex1-cong*:
assumes ‹$\bigwedge x. P(x) \longleftrightarrow Q(x)$›
shows ‹$(\exists !x. P(x)) \longleftrightarrow (\exists !x. Q(x))$›
⟨*proof*⟩

1.5 Equality rules

lemma *sym*: ‹$a = b \Longrightarrow b = a$›
⟨*proof*⟩

lemma *trans*: ‹$[\![a = b; b = c]\!] \Longrightarrow a = c$›
⟨*proof*⟩

lemma *not-sym*: ‹$b \neq a \Longrightarrow a \neq b$›
⟨*proof*⟩

Two theorems for rewriting only one instance of a definition: the first for definitions of formulae and the second for terms.

lemma *def-imp-iff*: ‹$(A \equiv B) \Longrightarrow A \longleftrightarrow B$›
⟨*proof*⟩

lemma *meta-eq-to-obj-eq*: ‹$(A \equiv B) \Longrightarrow A = B$›
⟨*proof*⟩

lemma *meta-eq-to-iff*: ‹$x \equiv y \Longrightarrow x \longleftrightarrow y$›
⟨*proof*⟩

Substitution.

lemma *ssubst*: ‹$[\![b = a; P(a)]\!] \Longrightarrow P(b)$›
⟨*proof*⟩

A special case of *ex1E* that would otherwise need quantifier expansion.

lemma *ex1-equalsE*: ‹$[\![\exists !x. P(x); P(a); P(b)]\!] \Longrightarrow a = b$›
⟨*proof*⟩

1.6 Simplifications of assumed implications

Roy Dyckhoff has proved that *conj-impE*, *disj-impE*, and *imp-impE* used with `mp_tac` (restricted to atomic formulae) is COMPLETE for intuitionistic propositional logic.

See R. Dyckhoff, Contraction-free sequent calculi for intuitionistic logic (preprint, University of St Andrews, 1991).

lemma *conj-impE*:
assumes *major*: ‹$(P \wedge Q) \longrightarrow S$›
and *r*: ‹$P \longrightarrow (Q \longrightarrow S) \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

lemma *disj-impE*:
assumes *major*: ‹$(P \vee Q) \longrightarrow S$›
and *r*: ‹$[\![P \longrightarrow S; Q \longrightarrow S]\!] \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

Simplifies the implication. Classical version is stronger. Still UNSAFE since Q must be provable – backtracking needed.

lemma *imp-impE*:

assumes *major*: ‹$(P \longrightarrow Q) \longrightarrow S$›
and *r1*: ‹$[\![P;\ Q \longrightarrow S]\!] \Longrightarrow Q$›
and *r2*: ‹$S \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

Simplifies the implication. Classical version is stronger. Still UNSAFE since P must be provable – backtracking needed.

lemma *not-impE*: ‹$\neg\ P \longrightarrow S \Longrightarrow (P \Longrightarrow \mathit{False}) \Longrightarrow (S \Longrightarrow R) \Longrightarrow R$›
⟨*proof*⟩

Simplifies the implication. UNSAFE.

lemma *iff-impE*:
assumes *major*: ‹$(P \longleftrightarrow Q) \longrightarrow S$›
and *r1*: ‹$[\![P;\ Q \longrightarrow S]\!] \Longrightarrow Q$›
and *r2*: ‹$[\![Q;\ P \longrightarrow S]\!] \Longrightarrow P$›
and *r3*: ‹$S \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

What if $(\forall\, x.\ \neg\ \neg\ P(x)) \longrightarrow \neg\ \neg\ (\forall\, x.\ P(x))$ is an assumption? UNSAFE.

lemma *all-impE*:
assumes *major*: ‹$(\forall\, x.\ P(x)) \longrightarrow S$›
and *r1*: ‹$\bigwedge x.\ P(x)$›
and *r2*: ‹$S \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

Unsafe: $\exists\, x.\ P(x)) \longrightarrow S$ is equivalent to $\forall\, x.\ P(x) \longrightarrow S$.

lemma *ex-impE*:
assumes *major*: ‹$(\exists\, x.\ P(x)) \longrightarrow S$›
and *r*: ‹$P(x) \longrightarrow S \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

Courtesy of Krzysztof Grabczewski.

lemma *disj-imp-disj*: ‹$P \lor Q \Longrightarrow (P \Longrightarrow R) \Longrightarrow (Q \Longrightarrow S) \Longrightarrow R \lor S$›
⟨*proof*⟩

⟨*ML*⟩

lemma *thin-refl*: ‹$[\![x = x;\ \mathit{PROP}\ W]\!] \Longrightarrow \mathit{PROP}\ W$› ⟨*proof*⟩

⟨*ML*⟩

1.7 Intuitionistic Reasoning

⟨*ML*⟩

lemma *impE′*:
  assumes *1*: ‹$P \longrightarrow Q$›
    and *2*: ‹$Q \Longrightarrow R$›
    and *3*: ‹$P \longrightarrow Q \Longrightarrow P$›
  shows ‹R›
⟨*proof*⟩

lemma *allE′*:
  assumes *1*: ‹$\forall x.\ P(x)$›
    and *2*: ‹$P(x) \Longrightarrow \forall x.\ P(x) \Longrightarrow Q$›
  shows ‹Q›
⟨*proof*⟩

lemma *notE′*:
  assumes *1*: ‹$\neg P$›
    and *2*: ‹$\neg P \Longrightarrow P$›
  shows ‹R›
⟨*proof*⟩

lemmas [*Pure.elim*!] = *disjE iffE FalseE conjE exE*
  and [*Pure.intro*!] = *iffI conjI impI TrueI notI allI refl*
  and [*Pure.elim 2*] = *allE notE′ impE′*
  and [*Pure.intro*] = *exI disjI2 disjI1*

⟨*ML*⟩

lemma *iff-not-sym*: ‹$\neg (Q \longleftrightarrow P) \Longrightarrow \neg (P \longleftrightarrow Q)$›
  ⟨*proof*⟩

lemmas [*sym*] = *sym iff-sym not-sym iff-not-sym*
  and [*Pure.elim?*] = *iffD1 iffD2 impE*

lemma *eq-commute*: ‹$a = b \longleftrightarrow b = a$›
  ⟨*proof*⟩

1.8 Polymorphic congruence rules

lemma *subst-context*: ‹$a = b \Longrightarrow t(a) = t(b)$›
  ⟨*proof*⟩

lemma *subst-context2*: ‹$[\![a = b;\ c = d]\!] \Longrightarrow t(a,c) = t(b,d)$›
  ⟨*proof*⟩

lemma *subst-context3*: ‹$[\![a = b;\ c = d;\ e = f]\!] \Longrightarrow t(a,c,e) = t(b,d,f)$›
  ⟨*proof*⟩

Useful with `eresolve_tac` for proving equalities from known equalities.

a = b | | c = d

lemma *box-equals*: ‹$[\![a = b;\ a = c;\ b = d]\!] \Longrightarrow c = d$›
⟨*proof*⟩

Dual of *box-equals*: for proving equalities backwards.

lemma *simp-equals*: ‹$[\![a = c;\ b = d;\ c = d]\!] \Longrightarrow a = b$›
⟨*proof*⟩

1.8.1 Congruence rules for predicate letters

lemma *pred1-cong*: ‹$a = a' \Longrightarrow P(a) \longleftrightarrow P(a')$›
⟨*proof*⟩

lemma *pred2-cong*: ‹$[\![a = a';\ b = b']\!] \Longrightarrow P(a,b) \longleftrightarrow P(a',b')$›
⟨*proof*⟩

lemma *pred3-cong*: ‹$[\![a = a';\ b = b';\ c = c']\!] \Longrightarrow P(a,b,c) \longleftrightarrow P(a',b',c')$›
⟨*proof*⟩

Special case for the equality predicate!

lemma *eq-cong*: ‹$[\![a = a';\ b = b']\!] \Longrightarrow a = b \longleftrightarrow a' = b'$›
⟨*proof*⟩

1.9 Atomizing meta-level rules

lemma *atomize-all* [*atomize*]: ‹$(\bigwedge x.\ P(x)) \equiv$ *Trueprop* $(\forall x.\ P(x))$›
⟨*proof*⟩

lemma *atomize-imp* [*atomize*]: ‹$(A \Longrightarrow B) \equiv$ *Trueprop* $(A \longrightarrow B)$›
⟨*proof*⟩

lemma *atomize-eq* [*atomize*]: ‹$(x \equiv y) \equiv$ *Trueprop* $(x = y)$›
⟨*proof*⟩

lemma *atomize-iff* [*atomize*]: ‹$(A \equiv B) \equiv$ *Trueprop* $(A \longleftrightarrow B)$›
⟨*proof*⟩

lemma *atomize-conj* [*atomize*]: ‹$(A\ \&\&\&\ B) \equiv$ *Trueprop* $(A \wedge B)$›
⟨*proof*⟩

lemmas [*symmetric*, *rulify*] = *atomize-all atomize-imp*
and [*symmetric*, *defn*] = *atomize-all atomize-imp atomize-eq atomize-iff*

1.10 Atomizing elimination rules

lemma *atomize-exL*[*atomize-elim*]: ‹$(\bigwedge x.\ P(x) \Longrightarrow Q) \equiv ((\exists x.\ P(x)) \Longrightarrow Q)$›
⟨*proof*⟩

lemma *atomize-conjL*[*atomize-elim*]: ‹$(A \Longrightarrow B \Longrightarrow C) \equiv (A \wedge B \Longrightarrow C)$›

⟨*proof*⟩

lemma *atomize-disjL*[*atomize-elim*]: ‹$((A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C) \equiv ((A \vee B \Longrightarrow C) \Longrightarrow C)$›
⟨*proof*⟩

lemma *atomize-elimL*[*atomize-elim*]: ‹$(\bigwedge B.\ (A \Longrightarrow B) \Longrightarrow B) \equiv Trueprop(A)$›
⟨*proof*⟩

1.11 Calculational rules

lemma *forw-subst*: ‹$a = b \Longrightarrow P(b) \Longrightarrow P(a)$›
⟨*proof*⟩

lemma *back-subst*: ‹$P(a) \Longrightarrow a = b \Longrightarrow P(b)$›
⟨*proof*⟩

Note that this list of rules is in reverse order of priorities.

lemmas *basic-trans-rules* [*trans*] =
forw-subst
back-subst
rev-mp
mp
trans

1.12 "Let" declarations

nonterminal *letbinds* **and** *letbind*

definition *Let* :: ‹['a::{}, 'a => 'b] ⇒ ('b::{})›
where ‹$Let(s, f) \equiv f(s)$›

syntax
-bind :: ‹[*pttrn*, '*a*] => *letbind*› (‹(*2*- =/ -)› *10*)
:: ‹*letbind* => *letbinds*› (‹-›)
-binds :: ‹[*letbind*, *letbinds*] => *letbinds*› (‹-;/ -›)
-Let :: ‹[*letbinds*, '*a*] => '*a*› (‹(*let* (-)/ *in* (-))› *10*)

translations
-Let(*-binds*(*b*, *bs*), *e*) == *-Let*(*b*, *-Let*(*bs*, *e*))
let x = *a in e* == *CONST Let*(*a*, λ*x*. *e*)

lemma *LetI*:
assumes ‹$\bigwedge x.\ x = t \Longrightarrow P(u(x))$›
shows ‹$P(let\ x = t\ in\ u(x))$›
⟨*proof*⟩

1.13 Intuitionistic simplification rules

lemma *conj-simps*:

‹$P \wedge True \longleftrightarrow P$›
‹$True \wedge P \longleftrightarrow P$›
‹$P \wedge False \longleftrightarrow False$›
‹$False \wedge P \longleftrightarrow False$›
‹$P \wedge P \longleftrightarrow P$›
‹$P \wedge P \wedge Q \longleftrightarrow P \wedge Q$›
‹$P \wedge \neg P \longleftrightarrow False$›
‹$\neg P \wedge P \longleftrightarrow False$›
‹$(P \wedge Q) \wedge R \longleftrightarrow P \wedge (Q \wedge R)$›
⟨*proof*⟩

lemma *disj-simps*:
‹$P \vee True \longleftrightarrow True$›
‹$True \vee P \longleftrightarrow True$›
‹$P \vee False \longleftrightarrow P$›
‹$False \vee P \longleftrightarrow P$›
‹$P \vee P \longleftrightarrow P$›
‹$P \vee P \vee Q \longleftrightarrow P \vee Q$›
‹$(P \vee Q) \vee R \longleftrightarrow P \vee (Q \vee R)$›
⟨*proof*⟩

lemma *not-simps*:
‹$\neg (P \vee Q) \longleftrightarrow \neg P \wedge \neg Q$›
‹$\neg False \longleftrightarrow True$›
‹$\neg True \longleftrightarrow False$›
⟨*proof*⟩

lemma *imp-simps*:
‹$(P \longrightarrow False) \longleftrightarrow \neg P$›
‹$(P \longrightarrow True) \longleftrightarrow True$›
‹$(False \longrightarrow P) \longleftrightarrow True$›
‹$(True \longrightarrow P) \longleftrightarrow P$›
‹$(P \longrightarrow P) \longleftrightarrow True$›
‹$(P \longrightarrow \neg P) \longleftrightarrow \neg P$›
⟨*proof*⟩

lemma *iff-simps*:
‹$(True \longleftrightarrow P) \longleftrightarrow P$›
‹$(P \longleftrightarrow True) \longleftrightarrow P$›
‹$(P \longleftrightarrow P) \longleftrightarrow True$›
‹$(False \longleftrightarrow P) \longleftrightarrow \neg P$›
‹$(P \longleftrightarrow False) \longleftrightarrow \neg P$›
⟨*proof*⟩

The $x = t$ versions are needed for the simplification procedures.

lemma *quant-simps*:
‹$\bigwedge P.\ (\forall x.\ P) \longleftrightarrow P$›
‹$(\forall x.\ x = t \longrightarrow P(x)) \longleftrightarrow P(t)$›
‹$(\forall x.\ t = x \longrightarrow P(x)) \longleftrightarrow P(t)$›

$\langle\bigwedge P.\ (\exists\, x.\ P) \longleftrightarrow P\rangle$
$\langle\exists\, x.\ x = t\rangle$
$\langle\exists\, x.\ t = x\rangle$
$\langle(\exists\, x.\ x = t \wedge P(x)) \longleftrightarrow P(t)\rangle$
$\langle(\exists\, x.\ t = x \wedge P(x)) \longleftrightarrow P(t)\rangle$
⟨*proof*⟩

These are NOT supplied by default!

lemma *distrib-simps*:
$\langle P \wedge (Q \vee R) \longleftrightarrow P \wedge Q \vee P \wedge R\rangle$
$\langle(Q \vee R) \wedge P \longleftrightarrow Q \wedge P \vee R \wedge P\rangle$
$\langle(P \vee Q \longrightarrow R) \longleftrightarrow (P \longrightarrow R) \wedge (Q \longrightarrow R)\rangle$
⟨*proof*⟩

lemma *subst-all*:
$\langle(\bigwedge x.\ x = a \Longrightarrow PROP\ P(x)) \equiv PROP\ P(a)\rangle$
$\langle(\bigwedge x.\ a = x \Longrightarrow PROP\ P(x)) \equiv PROP\ P(a)\rangle$
⟨*proof*⟩

1.13.1 Conversion into rewrite rules

lemma *P-iff-F*: $\langle\neg\ P \Longrightarrow (P \longleftrightarrow False)\rangle$
⟨*proof*⟩
lemma *iff-reflection-F*: $\langle\neg\ P \Longrightarrow (P \equiv False)\rangle$
⟨*proof*⟩

lemma *P-iff-T*: $\langle P \Longrightarrow (P \longleftrightarrow True)\rangle$
⟨*proof*⟩
lemma *iff-reflection-T*: $\langle P \Longrightarrow (P \equiv True)\rangle$
⟨*proof*⟩

1.13.2 More rewrite rules

lemma *conj-commute*: $\langle P \wedge Q \longleftrightarrow Q \wedge P\rangle$ ⟨*proof*⟩
lemma *conj-left-commute*: $\langle P \wedge (Q \wedge R) \longleftrightarrow Q \wedge (P \wedge R)\rangle$ ⟨*proof*⟩
lemmas *conj-comms* = *conj-commute conj-left-commute*

lemma *disj-commute*: $\langle P \vee Q \longleftrightarrow Q \vee P\rangle$ ⟨*proof*⟩
lemma *disj-left-commute*: $\langle P \vee (Q \vee R) \longleftrightarrow Q \vee (P \vee R)\rangle$ ⟨*proof*⟩
lemmas *disj-comms* = *disj-commute disj-left-commute*

lemma *conj-disj-distribL*: $\langle P \wedge (Q \vee R) \longleftrightarrow (P \wedge Q \vee P \wedge R)\rangle$ ⟨*proof*⟩
lemma *conj-disj-distribR*: $\langle(P \vee Q) \wedge R \longleftrightarrow (P \wedge R \vee Q \wedge R)\rangle$ ⟨*proof*⟩

lemma *disj-conj-distribL*: $\langle P \vee (Q \wedge R) \longleftrightarrow (P \vee Q) \wedge (P \vee R)\rangle$ ⟨*proof*⟩
lemma *disj-conj-distribR*: $\langle(P \wedge Q) \vee R \longleftrightarrow (P \vee R) \wedge (Q \vee R)\rangle$ ⟨*proof*⟩

lemma *imp-conj-distrib*: $\langle(P \longrightarrow (Q \wedge R)) \longleftrightarrow (P \longrightarrow Q) \wedge (P \longrightarrow R)\rangle$ ⟨*proof*⟩
lemma *imp-conj*: $\langle((P \wedge Q) \longrightarrow R) \longleftrightarrow (P \longrightarrow (Q \longrightarrow R))\rangle$ ⟨*proof*⟩
lemma *imp-disj*: $\langle(P \vee Q \longrightarrow R) \longleftrightarrow (P \longrightarrow R) \wedge (Q \longrightarrow R)\rangle$ ⟨*proof*⟩

lemma *de-Morgan-disj*: ‹$(\neg (P \vee Q)) \longleftrightarrow (\neg P \wedge \neg Q)$› ⟨*proof*⟩

lemma *not-ex*: ‹$(\neg (\exists x. P(x))) \longleftrightarrow (\forall x. \neg P(x))$› ⟨*proof*⟩
lemma *imp-ex*: ‹$((\exists x. P(x)) \longrightarrow Q) \longleftrightarrow (\forall x. P(x) \longrightarrow Q)$› ⟨*proof*⟩

lemma *ex-disj-distrib*: ‹$(\exists x. P(x) \vee Q(x)) \longleftrightarrow ((\exists x. P(x)) \vee (\exists x. Q(x)))$›
⟨*proof*⟩

lemma *all-conj-distrib*: ‹$(\forall x. P(x) \wedge Q(x)) \longleftrightarrow ((\forall x. P(x)) \wedge (\forall x. Q(x)))$›
⟨*proof*⟩

end

2 Classical first-order logic

theory *FOL*
imports *IFOL*
keywords *print-claset print-induct-rules* :: *diag*
begin

⟨*ML*⟩

2.1 The classical axiom

axiomatization where
classical: ‹$(\neg P \Longrightarrow P) \Longrightarrow P$›

2.2 Lemmas and proof tools

lemma *ccontr*: ‹$(\neg P \Longrightarrow False) \Longrightarrow P$›
⟨*proof*⟩

2.2.1 Classical introduction rules for ∨ and ∃

lemma *disjCI*: ‹$(\neg Q \Longrightarrow P) \Longrightarrow P \vee Q$›
⟨*proof*⟩

Introduction rule involving only ∃

lemma *ex-classical*:
assumes *r*: ‹$\neg (\exists x. P(x)) \Longrightarrow P(a)$›
shows ‹$\exists x. P(x)$›
⟨*proof*⟩

Version of above, simplifying ¬∃ to ∀¬.

lemma *exCI*:
assumes *r*: ‹$\forall x. \neg P(x) \Longrightarrow P(a)$›
shows ‹$\exists x. P(x)$›
⟨*proof*⟩

lemma *excluded-middle*: ‹$\neg P \vee P$›
⟨*proof*⟩

lemma *case-split* [*case-names True False*]:
assumes *r1*: ‹$P \Longrightarrow Q$›
and *r2*: ‹$\neg P \Longrightarrow Q$›
shows ‹Q›
⟨*proof*⟩

⟨*ML*⟩

2.3 Special elimination rules

Classical implies ($\longrightarrow$) elimination.

lemma *impCE*:
assumes *major*: ‹$P \longrightarrow Q$›
and *r1*: ‹$\neg P \Longrightarrow R$›
and *r2*: ‹$Q \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

This version of $\longrightarrow$ elimination works on Q before P. It works best for those cases in which P holds "almost everywhere". Can't install as default: would break old proofs.

lemma *impCE'*:
assumes *major*: ‹$P \longrightarrow Q$›
and *r1*: ‹$Q \Longrightarrow R$›
and *r2*: ‹$\neg P \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

Double negation law.

lemma *notnotD*: ‹$\neg \neg P \Longrightarrow P$›
⟨*proof*⟩

lemma *contrapos2*: ‹$[\![Q; \neg P \Longrightarrow \neg Q]\!] \Longrightarrow P$›
⟨*proof*⟩

2.3.1 Tactics for implication and contradiction

Classical $\longleftrightarrow$ elimination. Proof substitutes $P = Q$ in $\neg P \Longrightarrow \neg Q$ and $P \Longrightarrow Q$.

lemma *iffCE*:
assumes *major*: ‹$P \longleftrightarrow Q$›
and *r1*: ‹$[\![P; Q]\!] \Longrightarrow R$›
and *r2*: ‹$[\![\neg P; \neg Q]\!] \Longrightarrow R$›

shows ‹R›
⟨*proof*⟩

lemma *alt-ex1E*:
assumes *major*: ‹$\exists!\ x.\ P(x)$›
and *r*: ‹$\bigwedge x.\ [\![P(x);\ \ \forall y\ y'.\ P(y) \wedge P(y') \longrightarrow y = y']\!] \Longrightarrow R$›
shows ‹R›
⟨*proof*⟩

lemma *imp-elim*: ‹$P \longrightarrow Q \Longrightarrow (\neg\ R \Longrightarrow P) \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$›
⟨*proof*⟩

lemma *swap*: ‹$\neg\ P \Longrightarrow (\neg\ R \Longrightarrow P) \Longrightarrow R$›
⟨*proof*⟩

3 Classical Reasoner

⟨*ML*⟩

lemmas [*intro*!] = *refl TrueI conjI disjCI impI notI iffI*
and [*elim*!] = *conjE disjE impCE FalseE iffCE*
⟨*ML*⟩

lemmas [*intro*!] = *allI ex-ex1I*
and [*intro*] = *exI*
and [*elim*!] = *exE alt-ex1E*
and [*elim*] = *allE*
⟨*ML*⟩

lemma *ex1-functional*: ‹$[\![\exists!\ z.\ P(a,z);\ P(a,b);\ P(a,c)]\!] \Longrightarrow b = c$›
⟨*proof*⟩

Elimination of *True* from assumptions:

lemma *True-implies-equals*: ‹$(True \Longrightarrow PROP\ P) \equiv PROP\ P$›
⟨*proof*⟩

lemma *uncurry*: ‹$P \longrightarrow Q \longrightarrow R \Longrightarrow P \wedge Q \longrightarrow R$›
⟨*proof*⟩

lemma *iff-allI*: ‹$(\bigwedge x.\ P(x) \longleftrightarrow Q(x)) \Longrightarrow (\forall x.\ P(x)) \longleftrightarrow (\forall x.\ Q(x))$›
⟨*proof*⟩

lemma *iff-exI*: ‹$(\bigwedge x.\ P(x) \longleftrightarrow Q(x)) \Longrightarrow (\exists x.\ P(x)) \longleftrightarrow (\exists x.\ Q(x))$›
⟨*proof*⟩

lemma *all-comm*: ‹$(\forall x\ y.\ P(x,y)) \longleftrightarrow (\forall y\ x.\ P(x,y))$›
  ⟨*proof*⟩

lemma *ex-comm*: ‹$(\exists x\ y.\ P(x,y)) \longleftrightarrow (\exists y\ x.\ P(x,y))$›
  ⟨*proof*⟩

3.1 Classical simplification rules

Avoids duplication of subgoals after *expand-if*, when the true and false cases boil down to the same thing.

lemma *cases-simp*: ‹$(P \longrightarrow Q) \wedge (\neg P \longrightarrow Q) \longleftrightarrow Q$›
  ⟨*proof*⟩

3.1.1 Miniscoping: pushing quantifiers in

We do NOT distribute of $\forall$ over $\wedge$, or dually that of $\exists$ over $\vee$.

Baaz and Leitsch, On Skolemization and Proof Complexity (1994) show that this step can increase proof length!

Existential miniscoping.

lemma *int-ex-simps*:
  ‹$\bigwedge P\ Q.\ (\exists x.\ P(x) \wedge Q) \longleftrightarrow (\exists x.\ P(x)) \wedge Q$›
  ‹$\bigwedge P\ Q.\ (\exists x.\ P \wedge Q(x)) \longleftrightarrow P \wedge (\exists x.\ Q(x))$›
  ‹$\bigwedge P\ Q.\ (\exists x.\ P(x) \vee Q) \longleftrightarrow (\exists x.\ P(x)) \vee Q$›
  ‹$\bigwedge P\ Q.\ (\exists x.\ P \vee Q(x)) \longleftrightarrow P \vee (\exists x.\ Q(x))$›
  ⟨*proof*⟩

Classical rules.

lemma *cla-ex-simps*:
  ‹$\bigwedge P\ Q.\ (\exists x.\ P(x) \longrightarrow Q) \longleftrightarrow (\forall x.\ P(x)) \longrightarrow Q$›
  ‹$\bigwedge P\ Q.\ (\exists x.\ P \longrightarrow Q(x)) \longleftrightarrow P \longrightarrow (\exists x.\ Q(x))$›
  ⟨*proof*⟩

lemmas *ex-simps = int-ex-simps cla-ex-simps*

Universal miniscoping.

lemma *int-all-simps*:
  ‹$\bigwedge P\ Q.\ (\forall x.\ P(x) \wedge Q) \longleftrightarrow (\forall x.\ P(x)) \wedge Q$›
  ‹$\bigwedge P\ Q.\ (\forall x.\ P \wedge Q(x)) \longleftrightarrow P \wedge (\forall x.\ Q(x))$›
  ‹$\bigwedge P\ Q.\ (\forall x.\ P(x) \longrightarrow Q) \longleftrightarrow (\exists\ x.\ P(x)) \longrightarrow Q$›
  ‹$\bigwedge P\ Q.\ (\forall x.\ P \longrightarrow Q(x)) \longleftrightarrow P \longrightarrow (\forall x.\ Q(x))$›
  ⟨*proof*⟩

Classical rules.

lemma *cla-all-simps*:
  ‹$\bigwedge P\ Q.\ (\forall x.\ P(x) \vee Q) \longleftrightarrow (\forall x.\ P(x)) \vee Q$›

‹$\bigwedge P\ Q.\ (\forall x.\ P \vee Q(x)) \longleftrightarrow P \vee (\forall x.\ Q(x))$›
⟨*proof*⟩

lemmas *all-simps* = *int-all-simps cla-all-simps*

3.1.2 Named rewrite rules proved for IFOL

lemma *imp-disj1*: ‹$(P \longrightarrow Q) \vee R \longleftrightarrow (P \longrightarrow Q \vee R)$› ⟨*proof*⟩
lemma *imp-disj2*: ‹$Q \vee (P \longrightarrow R) \longleftrightarrow (P \longrightarrow Q \vee R)$› ⟨*proof*⟩

lemma *de-Morgan-conj*: ‹$(\neg (P \wedge Q)) \longleftrightarrow (\neg P \vee \neg Q)$› ⟨*proof*⟩

lemma *not-imp*: ‹$\neg (P \longrightarrow Q) \longleftrightarrow (P \wedge \neg Q)$› ⟨*proof*⟩
lemma *not-iff*: ‹$\neg (P \longleftrightarrow Q) \longleftrightarrow (P \longleftrightarrow \neg Q)$› ⟨*proof*⟩

lemma *not-all*: ‹$(\neg (\forall x.\ P(x))) \longleftrightarrow (\exists x.\ \neg P(x))$› ⟨*proof*⟩
lemma *imp-all*: ‹$((\forall x.\ P(x)) \longrightarrow Q) \longleftrightarrow (\exists x.\ P(x) \longrightarrow Q)$› ⟨*proof*⟩

lemmas *meta-simps* =
triv-forall-equality — prunes params
True-implies-equals — prune asms *True*

lemmas *IFOL-simps* =
refl [*THEN P-iff-T*] *conj-simps disj-simps not-simps*
imp-simps iff-simps quant-simps

lemma *notFalseI*: ‹$\neg$ *False*› ⟨*proof*⟩

lemma *cla-simps-misc*:
‹$\neg (P \wedge Q) \longleftrightarrow \neg P \vee \neg Q$›
‹$P \vee \neg P$›
‹$\neg P \vee P$›
‹$\neg \neg P \longleftrightarrow P$›
‹$(\neg P \longrightarrow P) \longleftrightarrow P$›
‹$(\neg P \longleftrightarrow \neg Q) \longleftrightarrow (P \longleftrightarrow Q)$› ⟨*proof*⟩

lemmas *cla-simps* =
de-Morgan-conj de-Morgan-disj imp-disj1 imp-disj2
not-imp not-all not-ex cases-simp cla-simps-misc

⟨*ML*⟩

3.2 Other simple lemmas

lemma [*simp*]: ‹$((P \longrightarrow R) \longleftrightarrow (Q \longrightarrow R)) \longleftrightarrow ((P \longleftrightarrow Q) \vee R)$›
⟨*proof*⟩

lemma [*simp*]: ‹$((P \longrightarrow Q) \longleftrightarrow (P \longrightarrow R)) \longleftrightarrow (P \longrightarrow (Q \longleftrightarrow R))$›

⟨*proof*⟩

lemma *not-disj-iff-imp*: ‹$\neg$ $P \vee Q \longleftrightarrow (P \longrightarrow Q)$›
⟨*proof*⟩

3.2.1 Monotonicity of implications

lemma *conj-mono*: ‹$[\![P1 \longrightarrow Q1;\ P2 \longrightarrow Q2]\!] \Longrightarrow (P1 \wedge P2) \longrightarrow (Q1 \wedge Q2)$›
⟨*proof*⟩

lemma *disj-mono*: ‹$[\![P1 \longrightarrow Q1;\ P2 \longrightarrow Q2]\!] \Longrightarrow (P1 \vee P2) \longrightarrow (Q1 \vee Q2)$›
⟨*proof*⟩

lemma *imp-mono*: ‹$[\![Q1 \longrightarrow P1;\ P2 \longrightarrow Q2]\!] \Longrightarrow (P1 \longrightarrow P2) \longrightarrow (Q1 \longrightarrow Q2)$›
⟨*proof*⟩

lemma *imp-refl*: ‹$P \longrightarrow P$›
⟨*proof*⟩

The quantifier monotonicity rules are also intuitionistically valid.

lemma *ex-mono*: ‹$(\bigwedge x.\ P(x) \longrightarrow Q(x)) \Longrightarrow (\exists\, x.\ P(x)) \longrightarrow (\exists\, x.\ Q(x))$›
⟨*proof*⟩

lemma *all-mono*: ‹$(\bigwedge x.\ P(x) \longrightarrow Q(x)) \Longrightarrow (\forall\, x.\ P(x)) \longrightarrow (\forall\, x.\ Q(x))$›
⟨*proof*⟩

3.3 Proof by cases and induction

Proper handling of non-atomic rule statements.

context
begin

qualified definition ‹$\mathit{induct\text{-}forall}(P) \equiv \forall\, x.\ P(x)$›
qualified definition ‹$\mathit{induct\text{-}implies}(A,\ B) \equiv A \longrightarrow B$›
qualified definition ‹$\mathit{induct\text{-}equal}(x,\ y) \equiv x = y$›
qualified definition ‹$\mathit{induct\text{-}conj}(A,\ B) \equiv A \wedge B$›

lemma *induct-forall-eq*: ‹$(\bigwedge x.\ P(x)) \equiv \mathit{Trueprop}(\mathit{induct\text{-}forall}(\lambda x.\ P(x)))$›
⟨*proof*⟩

lemma *induct-implies-eq*: ‹$(A \Longrightarrow B) \equiv \mathit{Trueprop}(\mathit{induct\text{-}implies}(A,\ B))$›
⟨*proof*⟩

lemma *induct-equal-eq*: ‹$(x \equiv y) \equiv \mathit{Trueprop}(\mathit{induct\text{-}equal}(x,\ y))$›
⟨*proof*⟩

lemma *induct-conj-eq*: ‹$(A\ \&\&\&\ B) \equiv \mathit{Trueprop}(\mathit{induct\text{-}conj}(A,\ B))$›
⟨*proof*⟩

lemmas *induct-atomize* = *induct-forall-eq induct-implies-eq induct-equal-eq induct-conj-eq*
lemmas *induct-rulify* [*symmetric*] = *induct-atomize*
lemmas *induct-rulify-fallback* =
induct-forall-def induct-implies-def induct-equal-def induct-conj-def

Method setup.

⟨*ML*⟩

declare *case-split* [*cases type*: *o*]

end

⟨*ML*⟩

hide-const (**open**) *eq*

end