Miscellaneous FOL Examples

October 24, 2026

Contents

1 Natural numbers **1**

2 Examples for the manual "Introduction to Isabelle" **2**
2.0.1 Some simple backward proofs 2
2.0.2 Demonstration of *fast* 3
2.0.3 Derivation of conjunction elimination rule 3
2.1 Derived rules involving definitions 3

3 Theory of the natural numbers: Peano's axioms, primitive recursion **4**
3.1 Proofs about the natural numbers 5

4 Theory of the natural numbers: Peano's axioms, primitive recursion **6**

5 Intuitionistic FOL: Examples from The Foundation of a Generic Theorem Prover **7**
5.1 Examples with quantifiers 9

6 First-Order Logic: PROLOG examples **10**

7 Intuitionistic First-Order Logic **12**
7.1 Lemmas for the propositional double-negation translation . . 13
7.2 de Bruijn formulae 13
7.3 Intuitionistic FOL: propositional problems based on Pelletier. 14
7.4 11. Proved in each direction (incorrectly, says Pelletier!!) . . . 15

8 Examples with quantifiers **16**
8.1 The converse is classical in the following implications 16
8.2 The following are not constructively valid! 16
8.3 Hard examples with quantifiers 17

9 First-Order Logic: propositional examples (intuitionistic version) 21

10 First-Order Logic: quantifier examples (intuitionistic version) 23

11 Classical Predicate Calculus Problems 25
11.0.1 If and only if . . . 25
11.1 Pelletier's examples . . . 25
11.2 Classical Logic: examples with quantifiers . . . 27
11.3 Problems requiring quantifier duplication . . . 27
11.4 Hard examples with quantifiers . . . 28
11.5 Problems (mainly) involving equality or functions . . . 32

12 First-Order Logic: propositional examples (classical version) 35

13 First-Order Logic: quantifier examples (classical version) 38
13.1 Negation Normal Form . . . 40
13.1.1 de Morgan laws . . . 40
13.1.2 Pushing in the existential quantifiers . . . 40
13.1.3 Pushing in the universal quantifiers . . . 40

14 First-Order Logic: the 'if' example 41

1 Natural numbers

theory *Natural-Numbers*
imports *FOL*
begin

Theory of the natural numbers: Peano's axioms, primitive recursion. (Modernized version of Larry Paulson's theory "Nat".)

typedecl *nat*
instance *nat* :: ‹*term*› **..**

axiomatization
Zero :: ‹*nat*› (‹*0*›) **and**
Suc :: ‹*nat* => *nat*› **and**
rec :: ‹[*nat*, *'a*, [*nat*, *'a*] => *'a*] => *'a*›
where
induct [*case-names 0 Suc*, *induct type*: *nat*]:
‹*P*(*0*) ==> (!!*x*. *P*(*x*) ==> *P*(*Suc*(*x*))) ==> *P*(*n*)› **and**
Suc-inject: ‹*Suc*(*m*) = *Suc*(*n*) ==> *m* = *n*› **and**
Suc-neq-0: ‹*Suc*(*m*) = *0* ==> *R*› **and**
rec-0: ‹*rec*(*0*, *a*, *f*) = *a*› **and**

rec-Suc: ‹*rec*(*Suc*(*m*), *a*, *f*) = *f*(*m*, *rec*(*m*, *a*, *f*))›

lemma *Suc-n-not-n*: ‹*Suc*(*k*) ≠ *k*›
proof (*induct* ‹*k*›)
  show ‹*Suc*(*0*) ≠ *0*›
  proof
    assume ‹*Suc*(*0*) = *0*›
    then show ‹*False*› **by** (*rule Suc-neq-0*)
  qed
next
  fix *n* **assume** *hyp*: ‹*Suc*(*n*) ≠ *n*›
  show ‹*Suc*(*Suc*(*n*)) ≠ *Suc*(*n*)›
  proof
    assume ‹*Suc*(*Suc*(*n*)) = *Suc*(*n*)›
    then have ‹*Suc*(*n*) = *n*› **by** (*rule Suc-inject*)
    with *hyp* **show** ‹*False*› **by** *contradiction*
  qed
qed

definition *add* :: ‹*nat* => *nat* => *nat*› (**infixl** ‹+› *60*)
  where ‹*m* + *n* = *rec*(*m*, *n*, λ*x y*. *Suc*(*y*))›

lemma *add-0* [*simp*]: ‹*0* + *n* = *n*›
  unfolding *add-def* **by** (*rule rec-0*)

lemma *add-Suc* [*simp*]: ‹*Suc*(*m*) + *n* = *Suc*(*m* + *n*)›
  unfolding *add-def* **by** (*rule rec-Suc*)

lemma *add-assoc*: ‹(*k* + *m*) + *n* = *k* + (*m* + *n*)›
  by (*induct* ‹*k*›) *simp-all*

lemma *add-0-right*: ‹*m* + *0* = *m*›
  by (*induct* ‹*m*›) *simp-all*

lemma *add-Suc-right*: ‹*m* + *Suc*(*n*) = *Suc*(*m* + *n*)›
  by (*induct* ‹*m*›) *simp-all*

lemma
  assumes ‹!!*n*. *f*(*Suc*(*n*)) = *Suc*(*f*(*n*))›
  shows ‹*f*(*i* + *j*) = *i* + *f*(*j*)›
  using *assms* **by** (*induct* ‹*i*›) *simp-all*

end

2 Examples for the manual "Introduction to Isabelle"

theory *Intro*
imports *FOL*
begin

2.0.1 Some simple backward proofs

lemma *mythm*: ‹$P \vee P \longrightarrow P$›
apply (*rule impI*)
apply (*rule disjE*)
prefer *3* **apply** (*assumption*)
prefer *2* **apply** (*assumption*)
apply *assumption*
done

lemma ‹$(P \wedge Q) \vee R \longrightarrow (P \vee R)$›
apply (*rule impI*)
apply (*erule disjE*)
apply (*drule conjunct1*)
apply (*rule disjI1*)
apply (*rule-tac* [*2*] *disjI2*)
apply *assumption*+
done

Correct version, delaying use of *spec* until last.

lemma ‹$(\forall x\ y.\ P(x,y)) \longrightarrow (\forall z\ w.\ P(w,z))$›
apply (*rule impI*)
apply (*rule allI*)
apply (*rule allI*)
apply (*drule spec*)
apply (*drule spec*)
apply *assumption*
done

2.0.2 Demonstration of *fast*

lemma ‹$(\exists y.\ \forall x.\ J(y,x) \longleftrightarrow \neg\ J(x,x)) \longrightarrow \neg\ (\forall x.\ \exists y.\ \forall z.\ J(z,y) \longleftrightarrow \neg\ J(z,x))$›
apply *fast*
done

lemma ‹$\forall x.\ P(x,f(x)) \longleftrightarrow (\exists y.\ (\forall z.\ P(z,y) \longrightarrow P(z,f(x))) \wedge P(x,y))$›
apply *fast*
done

2.0.3 Derivation of conjunction elimination rule

lemma

assumes *major*: ‹$P \wedge Q$›
and *minor*: ‹$[\![P; Q]\!] \Longrightarrow R$›
shows ‹R›
apply (*rule minor*)
apply (*rule major* [*THEN conjunct1*])
apply (*rule major* [*THEN conjunct2*])
done

2.1 Derived rules involving definitions

Derivation of negation introduction

lemma
assumes ‹$P \Longrightarrow False$›
shows ‹$\neg P$›
apply (*unfold not-def*)
apply (*rule impI*)
apply (*rule assms*)
apply *assumption*
done

lemma
assumes *major*: ‹$\neg P$›
and *minor*: ‹P›
shows ‹R›
apply (*rule FalseE*)
apply (*rule mp*)
apply (*rule major* [*unfolded not-def*])
apply (*rule minor*)
done

Alternative proof of the result above

lemma
assumes *major*: ‹$\neg P$›
and *minor*: ‹P›
shows ‹R›
apply (*rule minor* [*THEN major* [*unfolded not-def*, *THEN mp*, *THEN FalseE*]])
done

end

3 Theory of the natural numbers: Peano's axioms, primitive recursion

theory *Nat*
imports *FOL*
begin

typedecl *nat*
instance *nat* :: ‹*term*› **..**

axiomatization
  Zero :: ‹*nat*› (‹*0*›) **and**
  Suc :: ‹*nat* ⇒ *nat*› **and**
  rec :: ‹[*nat*, *'a*, [*nat*, *'a*] ⇒ *'a*] ⇒ *'a*›
where
  induct: ‹⟦*P*(*0*); ⋀*x*. *P*(*x*) ⟹ *P*(*Suc*(*x*))⟧ ⟹ *P*(*n*)› **and**
  Suc-inject: ‹*Suc*(*m*)=*Suc*(*n*) ⟹ *m*=*n*› **and**
  Suc-neq-0: ‹*Suc*(*m*)=*0* ⟹ *R*› **and**
  rec-0: ‹*rec*(*0*,*a*,*f*) = *a*› **and**
  rec-Suc: ‹*rec*(*Suc*(*m*), *a*, *f*) = *f*(*m*, *rec*(*m*,*a*,*f*))›

definition *add* :: ‹[*nat*, *nat*] ⇒ *nat*› (**infixl** ‹+› *60*)
  where ‹*m* + *n* ≡ *rec*(*m*, *n*, λ*x* *y*. *Suc*(*y*))›

3.1 Proofs about the natural numbers

lemma *Suc-n-not-n*: ‹*Suc*(*k*) ≠ *k*›
apply (*rule-tac* *n* = ‹*k*› **in** *induct*)
apply (*rule notI*)
apply (*erule Suc-neq-0*)
apply (*rule notI*)
apply (*erule notE*)
apply (*erule Suc-inject*)
done

lemma ‹(*k*+*m*)+*n* = *k*+(*m*+*n*)›
apply (*rule induct*)
back
back
back
back
back
back
oops

lemma *add-0* [*simp*]: ‹*0*+*n* = *n*›
apply (*unfold add-def*)
apply (*rule rec-0*)
done

lemma *add-Suc* [*simp*]: ‹*Suc*(*m*)+*n* = *Suc*(*m*+*n*)›
apply (*unfold add-def*)
apply (*rule rec-Suc*)
done

lemma *add-assoc*: ‹(*k*+*m*)+*n* = *k*+(*m*+*n*)›

apply (*rule-tac* $n = \langle k \rangle$ **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-0-right*: $\langle m+0 = m \rangle$
apply (*rule-tac* $n = \langle m \rangle$ **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-Suc-right*: $\langle m+Suc(n) = Suc(m+n) \rangle$
apply (*rule-tac* $n = \langle m \rangle$ **in** *induct*)
apply *simp-all*
done

lemma
  assumes *prem*: $\langle \bigwedge n.\ f(Suc(n)) = Suc(f(n)) \rangle$
  shows $\langle f(i+j) = i+f(j) \rangle$
apply (*rule-tac* $n = \langle i \rangle$ **in** *induct*)
apply *simp*
apply (*simp add*: *prem*)
done

end

4 Theory of the natural numbers: Peano's axioms, primitive recursion

theory *Nat-Class*
  imports *FOL*
begin

This is an abstract version of `Nat.thy`. Instead of axiomatizing a single type *nat*, it defines the class of all these types (up to isomorphism).

Note: The *rec* operator has been made *monomorphic*, because class axioms cannot contain more than one type variable.

class *nat* =
  fixes *Zero* :: $\langle 'a \rangle$ $(\langle 0 \rangle)$
    and *Suc* :: $\langle 'a \Rightarrow 'a \rangle$
    and *rec* :: $\langle 'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a \rangle$
  assumes *induct*: $\langle P(0) \Longrightarrow (\bigwedge x.\ P(x) \Longrightarrow P(Suc(x))) \Longrightarrow P(n) \rangle$
    and *Suc-inject*: $\langle Suc(m) = Suc(n) \Longrightarrow m = n \rangle$
    and *Suc-neq-Zero*: $\langle Suc(m) = 0 \Longrightarrow R \rangle$
    and *rec-Zero*: $\langle rec(0,\ a,\ f) = a \rangle$
    and *rec-Suc*: $\langle rec(Suc(m),\ a,\ f) = f(m,\ rec(m,\ a,\ f)) \rangle$
begin

definition *add* :: ‹'*a* ⇒ '*a* ⇒ '*a*› (**infixl** ‹+› *60*)
where ‹*m* + *n* = *rec*(*m*, *n*, λ*x y*. *Suc*(*y*))›

lemma *Suc-n-not-n*: ‹*Suc*(*k*) ≠ (*k*::'*a*)›
apply (*rule-tac n* = ‹*k*› **in** *induct*)
apply (*rule notI*)
apply (*erule Suc-neq-Zero*)
apply (*rule notI*)
apply (*erule notE*)
apply (*erule Suc-inject*)
done

lemma ‹(*k* + *m*) + *n* = *k* + (*m* + *n*)›
apply (*rule induct*)
back
back
back
back
back
oops

lemma *add-Zero* [*simp*]: ‹*0* + *n* = *n*›
apply (*unfold add-def*)
apply (*rule rec-Zero*)
done

lemma *add-Suc* [*simp*]: ‹*Suc*(*m*) + *n* = *Suc*(*m* + *n*)›
apply (*unfold add-def*)
apply (*rule rec-Suc*)
done

lemma *add-assoc*: ‹(*k* + *m*) + *n* = *k* + (*m* + *n*)›
apply (*rule-tac n* = ‹*k*› **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-Zero-right*: ‹*m* + *0* = *m*›
apply (*rule-tac n* = ‹*m*› **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-Suc-right*: ‹*m* + *Suc*(*n*) = *Suc*(*m* + *n*)›
apply (*rule-tac n* = ‹*m*› **in** *induct*)
apply *simp-all*
done

lemma

assumes *prem*: ‹$\bigwedge n.\ f(Suc(n)) = Suc(f(n))$›
shows ‹$f(i + j) = i + f(j)$›
apply (*rule-tac* $n = ‹i›$ **in** *induct*)
apply *simp*
apply (*simp add*: *prem*)
done

end

end

5 Intuitionistic FOL: Examples from The Foundation of a Generic Theorem Prover

theory *Foundation*
imports *IFOL*
begin

lemma ‹$A \wedge B \longrightarrow (C \longrightarrow A \wedge C)$›
apply (*rule impI*)
apply (*rule impI*)
apply (*rule conjI*)
prefer *2* **apply** *assumption*
apply (*rule conjunct1*)
apply *assumption*
done

A form of conj-elimination

lemma
assumes ‹$A \wedge B$›
and ‹$A \Longrightarrow B \Longrightarrow C$›
shows ‹C›
apply (*rule assms*)
apply (*rule conjunct1*)
apply (*rule assms*)
apply (*rule conjunct2*)
apply (*rule assms*)
done

lemma
assumes ‹$\bigwedge A.\ \neg\ \neg\ A \Longrightarrow A$›
shows ‹$B \vee \neg\ B$›
apply (*rule assms*)
apply (*rule notI*)
apply (*rule-tac* $P = ‹\neg\ B›$ **in** *notE*)
apply (*rule-tac* [*2*] *notI*)
apply (*rule-tac* [*2*] $P = ‹B \vee \neg\ B›$ **in** *notE*)
prefer *2* **apply** *assumption*

apply (*rule-tac* [*2*] *disjI1*)
prefer *2* **apply** *assumption*
apply (*rule notI*)
apply (*rule-tac* $P = \langle B \vee \neg\ B \rangle$ **in** *notE*)
apply *assumption*
apply (*rule disjI2*)
apply *assumption*
done

lemma
  assumes $\langle \bigwedge A.\ \neg\ \neg\ A \Longrightarrow A \rangle$
  shows $\langle B \vee \neg\ B \rangle$
apply (*rule assms*)
apply (*rule notI*)
apply (*rule notE*)
apply (*rule-tac* [*2*] *notI*)
apply (*erule-tac* [*2*] *notE*)
apply (*erule-tac* [*2*] *disjI1*)
apply (*rule notI*)
apply (*erule notE*)
apply (*erule disjI2*)
done

lemma
  assumes $\langle A \vee \neg\ A \rangle$
    and $\langle \neg\ \neg\ A \rangle$
  shows $\langle A \rangle$
apply (*rule disjE*)
apply (*rule assms*)
apply *assumption*
apply (*rule FalseE*)
apply (*rule-tac* $P = \langle \neg\ A \rangle$ **in** *notE*)
apply (*rule assms*)
apply *assumption*
done

5.1 Examples with quantifiers

lemma
  assumes $\langle \forall\, z.\ G(z) \rangle$
  shows $\langle \forall\, z.\ G(z) \vee H(z) \rangle$
apply (*rule allI*)
apply (*rule disjI1*)
apply (*rule assms* [*THEN spec*])
done

lemma $\langle \forall\, x.\ \exists\, y.\ x = y \rangle$
apply (*rule allI*)

apply (*rule exI*)
apply (*rule refl*)
done

lemma ‹$\exists\, y.\ \forall\, x.\ x = y$›
apply (*rule exI*)
apply (*rule allI*)
apply (*rule refl*)*?*
oops

Parallel lifting example.

lemma ‹$\exists\, u.\ \forall\, x.\ \exists\, v.\ \forall\, y.\ \exists\, w.\ P(u,x,v,y,w)$›
apply (*rule exI allI*)
apply (*rule exI allI*)
apply (*rule exI allI*)
apply (*rule exI allI*)
apply (*rule exI allI*)
oops

lemma
  assumes ‹$(\exists\, z.\ F(z)) \wedge B$›
  shows ‹$\exists\, z.\ F(z) \wedge B$›
apply (*rule conjE*)
apply (*rule assms*)
apply (*rule exE*)
apply *assumption*
apply (*rule exI*)
apply (*rule conjI*)
apply *assumption*
apply *assumption*
done

A bigger demonstration of quantifiers – not in the paper.

lemma ‹$(\exists\, y.\ \forall\, x.\ Q(x,y)) \longrightarrow (\forall\, x.\ \exists\, y.\ Q(x,y))$›
apply (*rule impI*)
apply (*rule allI*)
apply (*rule exE*, *assumption*)
apply (*rule exI*)
apply (*rule allE*, *assumption*)
apply *assumption*
done

end

6 First-Order Logic: PROLOG examples

theory *Prolog*
imports *FOL*

begin

typedecl *'a list*
instance *list* :: (‹*term*›) ‹*term*› **..**

axiomatization
  Nil :: ‹*'a list*› **and**
  Cons :: ‹['*a*, *'a list*]=> *'a list*› (**infixr** ‹:› *60*) **and**
  app :: ‹['*a list*, *'a list*, *'a list*] => *o*› **and**
  rev :: ‹['*a list*, *'a list*] => *o*›
where
  appNil: ‹*app*(*Nil*,*ys*,*ys*)› **and**
  appCons: ‹*app*(*xs*,*ys*,*zs*) ==> *app*(*x*:*xs*, *ys*, *x*:*zs*)› **and**
  revNil: ‹*rev*(*Nil*,*Nil*)› **and**
  revCons: ‹[| *rev*(*xs*,*ys*); *app*(*ys*, *x*:*Nil*, *zs*) |] ==> *rev*(*x*:*xs*, *zs*)›

schematic-goal ‹*app*(*a*:*b*:*c*:*Nil*, *d*:*e*:*Nil*, *?x*)›
apply (*rule appNil appCons*)
apply (*rule appNil appCons*)
apply (*rule appNil appCons*)
apply (*rule appNil appCons*)
done

schematic-goal ‹*app*(*?x*, *c*:*d*:*Nil*, *a*:*b*:*c*:*d*:*Nil*)›
apply (*rule appNil appCons*)+
done

schematic-goal ‹*app*(*?x*, *?y*, *a*:*b*:*c*:*d*:*Nil*)›
apply (*rule appNil appCons*)+
back
back
back
back
done

lemmas *rules* = *appNil appCons revNil revCons*

schematic-goal ‹*rev*(*a*:*b*:*c*:*d*:*Nil*, *?x*)›
apply (*rule rules*)+
done

schematic-goal ‹*rev*(*a*:*b*:*c*:*d*:*e*:*f*:*g*:*h*:*i*:*j*:*k*:*l*:*m*:*n*:*Nil*, *?w*)›
apply (*rule rules*)+
done

schematic-goal ‹*rev*(*?x*, *a*:*b*:*c*:*Nil*)›

apply (*rule rules*)+ — does not solve it directly!
back
back
done

ML ‹
fun prolog-tac ctxt =
DEPTH-FIRST Thm.no-prems (*resolve-tac ctxt* @{*thms rules*} *1*)
›

schematic-goal ‹*rev*(*?x*, *a*:*b*:*c*:*Nil*)›
apply (*tactic* ‹*prolog-tac* ***context***›)
done

schematic-goal ‹*rev*(*a*:*?x*:*c*:*?y*:*Nil*, *d*:*?z*:*b*:*?u*)›
apply (*tactic* ‹*prolog-tac* ***context***›)
done

schematic-goal ‹*rev*(*a*:*b*:*c*:*d*:*e*:*f*:*g*:*h*:*i*:*j*:*k*:*l*:*m*:*n*:*o*:*p*:*Nil*, *?w*)›
apply (*tactic* ‹
DEPTH-SOLVE (*resolve-tac* ***context*** ([@{*thm refl*}, @{*thm conjI*}] @ @{*thms rules*}) *1*)›)
done

schematic-goal ‹*a*:*b*:*c*:*d*:*e*:*f*:*g*:*h*:*i*:*j*:*k*:*l*:*m*:*n*:*o*:*p*:*Nil* = *?x* ∧ *app*(*?x*, *?x*, *?y*) ∧ *rev*(*?y*, *?w*)›
apply (*tactic* ‹
DEPTH-SOLVE (*resolve-tac* ***context*** ([@{*thm refl*}, @{*thm conjI*}] @ @{*thms rules*}) *1*)›)
done

end

7 Intuitionistic First-Order Logic

theory *Intuitionistic*
imports *IFOL*
begin

Metatheorem (for *propositional* formulae): P is classically provable iff $\neg\neg P$ is intuitionistically provable. Therefore $\neg P$ is classically provable iff it is intuitionistically provable.

Proof: Let Q be the conjunction of the propositions $A \vee \neg A$, one for each atom A in P. Now $\neg\neg Q$ is intuitionistically provable because $\neg\neg(A \vee \neg A)$ is and because double-negation distributes over conjunction. If P is provable classically, then clearly $Q \to P$ is provable intuitionistically, so $\neg\neg(Q \to P)$

is also provable intuitionistically. The latter is intuitionistically equivalent to $\neg\neg Q \to \neg\neg P$, hence to $\neg\neg P$, since $\neg\neg Q$ is intuitionistically provable. Finally, if P is a negation then $\neg\neg P$ is intuitionstically equivalent to P. [Andy Pitts]

lemma ‹$\neg\ \neg\ (P \wedge Q) \longleftrightarrow \neg\ \neg\ P \wedge \neg\ \neg\ Q$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$\neg\ \neg\ ((\neg\ P \longrightarrow Q) \longrightarrow (\neg\ P \longrightarrow \neg\ Q) \longrightarrow P)$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

Double-negation does NOT distribute over disjunction.

lemma ‹$\neg\ \neg\ (P \longrightarrow Q) \longleftrightarrow (\neg\ \neg\ P \longrightarrow \neg\ \neg\ Q)$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$\neg\ \neg\ \neg\ P \longleftrightarrow \neg\ P$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$\neg\ \neg\ ((P \longrightarrow Q \vee R) \longrightarrow (P \longrightarrow Q) \vee (P \longrightarrow R))$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$(P \longleftrightarrow Q) \longleftrightarrow (Q \longleftrightarrow P)$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$((P \longrightarrow (Q \vee (Q \longrightarrow R))) \longrightarrow R) \longrightarrow R$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma
‹$(((G \longrightarrow A) \longrightarrow J) \longrightarrow D \longrightarrow E) \longrightarrow (((H \longrightarrow B) \longrightarrow I) \longrightarrow C \longrightarrow J)$
$\longrightarrow (A \longrightarrow H) \longrightarrow F \longrightarrow G \longrightarrow (((C \longrightarrow B) \longrightarrow I) \longrightarrow D) \longrightarrow (A \longrightarrow C)$
$\longrightarrow (((F \longrightarrow A) \longrightarrow B) \longrightarrow I) \longrightarrow E$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

Admissibility of the excluded middle for negated formulae

lemma ‹$(P \vee \neg P \longrightarrow \neg Q) \longrightarrow \neg Q$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

The same in a more general form, no ex falso quodlibet

lemma ‹$(P \vee (P \longrightarrow R) \longrightarrow Q \longrightarrow R) \longrightarrow Q \longrightarrow R$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

7.1 Lemmas for the propositional double-negation translation

lemma ‹$P \longrightarrow \neg\ \neg\ P$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$\neg\ \neg\ (\neg\ \neg\ P \longrightarrow P)$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹¬ ¬ P ∧ ¬ ¬ (P ⟶ Q) ⟶ ¬ ¬ Q›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

The following are classically but not constructively valid. The attempt to prove them terminates quickly!

lemma ‹((P ⟶ Q) ⟶ P) ⟶ P›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma ‹(P ∧ Q ⟶ R) ⟶ (P ⟶ R) ∨ (Q ⟶ R)›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

7.2 de Bruijn formulae

de Bruijn formula with three predicates

lemma
‹((P ⟷ Q) ⟶ P ∧ Q ∧ R) ∧
(Q ⟷ R) ⟶ P ∧ Q ∧ R) ∧
(R ⟷ P) ⟶ P ∧ Q ∧ R) ⟶ P ∧ Q ∧ R›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

de Bruijn formula with five predicates

lemma
‹((P ⟷ Q) ⟶ P ∧ Q ∧ R ∧ S ∧ T) ∧
(Q ⟷ R) ⟶ P ∧ Q ∧ R ∧ S ∧ T) ∧
(R ⟷ S) ⟶ P ∧ Q ∧ R ∧ S ∧ T) ∧
(S ⟷ T) ⟶ P ∧ Q ∧ R ∧ S ∧ T) ∧
(T ⟷ P) ⟶ P ∧ Q ∧ R ∧ S ∧ T) ⟶ P ∧ Q ∧ R ∧ S ∧ T›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

Problems from of Sahlin, Franzen and Haridi, An Intuitionistic Predicate Logic Theorem Prover. J. Logic and Comp. 2 (5), October 1992, 619-656.

Problem 1.1

lemma
‹(∀ x. ∃ y. ∀ z. $p(x)$ ∧ $q(y)$ ∧ $r(z)$) ⟷
(∀ z. ∃ y. ∀ x. $p(x)$ ∧ $q(y)$ ∧ $r(z)$)›
by (*tactic* ‹*IntPr.best-dup-tac* ***context*** *1*›) — SLOW

Problem 3.1

lemma ‹¬ (∃ x. ∀ y. $mem(y,x)$ ⟷ ¬ $mem(x,x)$)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

Problem 4.1: hopeless!

lemma
‹(∀ *x*. *p*(*x*) ⟶ *p*(*h*(*x*)) ∨ *p*(*g*(*x*))) ∧ (∃ *x*. *p*(*x*)) ∧ (∀ *x*. ¬ *p*(*h*(*x*)))
⟶ (∃ *x*. *p*(*g*(*g*(*g*(*g*(*x*)))))))›
oops

7.3 Intuitionistic FOL: propositional problems based on Pelletier.

¬¬1

lemma ‹¬ ¬ ((*P* ⟶ *Q*) ⟷ (¬ *Q* ⟶ ¬ *P*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬2

lemma ‹¬ ¬ (¬ ¬ *P* ⟷ *P*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

3

lemma ‹¬ (*P* ⟶ *Q*) ⟶ (*Q* ⟶ *P*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬4

lemma ‹¬ ¬ ((¬ *P* ⟶ *Q*) ⟷ (¬ *Q* ⟶ *P*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬5

lemma ‹¬ ¬ ((*P* ∨ *Q* ⟶ *P* ∨ *R*) ⟶ *P* ∨ (*Q* ⟶ *R*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬6

lemma ‹¬ ¬ (*P* ∨ ¬ *P*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬7

lemma ‹¬ ¬ (*P* ∨ ¬ ¬ ¬ *P*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬8. Peirce's law

lemma ‹¬ ¬ (((*P* ⟶ *Q*) ⟶ *P*) ⟶ *P*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

9

lemma ‹((*P* ∨ *Q*) ∧ (¬ *P* ∨ *Q*) ∧ (*P* ∨ ¬ *Q*)) ⟶ ¬ (¬ *P* ∨ ¬ *Q*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

10

lemma ‹(*Q* ⟶ *R*) ⟶ (*R* ⟶ *P* ∧ *Q*) ⟶ (*P* ⟶ (*Q* ∨ *R*)) ⟶ (*P* ⟷ *Q*)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

7.4 11. Proved in each direction (incorrectly, says Pelletier!!)

lemma ‹$P \longleftrightarrow P$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬12. Dijkstra's law

lemma ‹$\neg\ \neg\ (((P \longleftrightarrow Q) \longleftrightarrow R) \longleftrightarrow (P \longleftrightarrow (Q \longleftrightarrow R)))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$((P \longleftrightarrow Q) \longleftrightarrow R) \longrightarrow \neg\ \neg\ (P \longleftrightarrow (Q \longleftrightarrow R))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

13. Distributive law

lemma ‹$P \vee (Q \wedge R) \longleftrightarrow (P \vee Q) \wedge (P \vee R)$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬14

lemma ‹$\neg\ \neg\ ((P \longleftrightarrow Q) \longleftrightarrow ((Q \vee \neg\ P) \wedge (\neg\ Q \vee P)))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬15

lemma ‹$\neg\ \neg\ ((P \longrightarrow Q) \longleftrightarrow (\neg\ P \vee Q))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬16

lemma ‹$\neg\ \neg\ ((P \longrightarrow Q) \vee (Q \longrightarrow P))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬17

lemma ‹$\neg\ \neg\ (((P \wedge (Q \longrightarrow R)) \longrightarrow S) \longleftrightarrow ((\neg\ P \vee Q \vee S) \wedge (\neg\ P \vee \neg\ R \vee S)))$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

Dijkstra's "Golden Rule"

lemma ‹$(P \wedge Q) \longleftrightarrow P \longleftrightarrow Q \longleftrightarrow (P \vee Q)$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

8 Examples with quantifiers

8.1 The converse is classical in the following implications ...

lemma ‹$(\exists x.\ P(x) \longrightarrow Q) \longrightarrow (\forall x.\ P(x)) \longrightarrow Q$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$((\forall x.\ P(x)) \longrightarrow Q) \longrightarrow \neg\ (\forall x.\ P(x) \wedge \neg\ Q)$›
  by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$((\forall x.\ \neg\ P(x)) \longrightarrow Q) \longrightarrow \neg\ (\forall x.\ \neg\ (P(x) \vee Q))$›

by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$(\forall x.\ P(x)) \vee Q \longrightarrow (\forall x.\ P(x) \vee Q)$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

lemma ‹$(\exists x.\ P \longrightarrow Q(x)) \longrightarrow (P \longrightarrow (\exists x.\ Q(x)))$›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

8.2 The following are not constructively valid!

The attempt to prove them terminates quickly!

lemma ‹$((\forall x.\ P(x)) \longrightarrow Q) \longrightarrow (\exists x.\ P(x) \longrightarrow Q)$›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma ‹$(P \longrightarrow (\exists x.\ Q(x))) \longrightarrow (\exists x.\ P \longrightarrow Q(x))$›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma ‹$(\forall x.\ P(x) \vee Q) \longrightarrow ((\forall x.\ P(x)) \vee Q)$›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma ‹$(\forall x.\ \neg\ \neg\ P(x)) \longrightarrow \neg\ \neg\ (\forall x.\ P(x))$›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

Classically but not intuitionistically valid. Proved by a bug in 1986!

lemma ‹$\exists x.\ Q(x) \longrightarrow (\forall x.\ Q(x))$›
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)*?*
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

8.3 Hard examples with quantifiers

The ones that have not been proved are not known to be valid! Some will require quantifier duplication – not currently available.

¬¬18

lemma ‹$\neg\ \neg\ (\exists y.\ \forall x.\ P(y) \longrightarrow P(x))$›
oops — NOT PROVED

¬¬19

lemma ‹$\neg\ \neg\ (\exists x.\ \forall y\ z.\ (P(y) \longrightarrow Q(z)) \longrightarrow (P(x) \longrightarrow Q(x)))$›

oops — NOT PROVED

20

lemma
‹(∀ x y. ∃ z. ∀ w. (P(x) ∧ Q(y) ⟶ R(z) ∧ S(w)))
⟶ (∃ x y. P(x) ∧ Q(y)) ⟶ (∃ z. R(z))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

21

lemma ‹(∃ x. P ⟶ Q(x)) ∧ (∃ x. Q(x) ⟶ P) ⟶ ¬ ¬ (∃ x. P ⟷ Q(x))›
oops — NOT PROVED; needs quantifier duplication

22

lemma ‹(∀ x. P ⟷ Q(x)) ⟶ (P ⟷ (∀ x. Q(x)))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬23

lemma ‹¬ ¬ ((∀ x. P ∨ Q(x)) ⟷ (P ∨ (∀ x. Q(x))))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

24

lemma
‹¬ (∃ x. S(x) ∧ Q(x)) ∧ (∀ x. P(x) ⟶ Q(x) ∨ R(x)) ∧
(¬ (∃ x. P(x)) ⟶ (∃ x. Q(x))) ∧ (∀ x. Q(x) ∨ R(x) ⟶ S(x))
⟶ ¬ ¬ (∃ x. P(x) ∧ R(x))›

Not clear why *fast-tac*, *best-tac*, *ASTAR* and *ITER-DEEPEN* all take forever.

apply (*tactic* ‹*IntPr.safe-tac* ***context***›)
apply (*erule impE*)
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)
apply (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)
done

25

lemma
‹(∃ x. P(x)) ∧
(∀ x. L(x) ⟶ ¬ (M(x) ∧ R(x))) ∧
(∀ x. P(x) ⟶ (M(x) ∧ L(x))) ∧
((∀ x. P(x) ⟶ Q(x)) ∨ (∃ x. P(x) ∧ R(x)))
⟶ (∃ x. Q(x) ∧ P(x))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬26

lemma
‹(¬ ¬ (∃ x. p(x)) ⟷ ¬ ¬ (∃ x. q(x))) ∧
(∀ x. ∀ y. p(x) ∧ q(y) ⟶ (r(x) ⟷ s(y)))

$\longrightarrow ((\forall x.\ p(x) \longrightarrow r(x)) \longleftrightarrow (\forall x.\ q(x) \longrightarrow s(x)))\rangle$
oops — NOT PROVED

27

lemma
$\langle(\exists x.\ P(x) \wedge \neg\ Q(x)) \wedge$
$(\forall x.\ P(x) \longrightarrow R(x)) \wedge$
$(\forall x.\ M(x) \wedge L(x) \longrightarrow P(x)) \wedge$
$((\exists x.\ R(x) \wedge \neg\ Q(x)) \longrightarrow (\forall x.\ L(x) \longrightarrow \neg\ R(x)))$
$\longrightarrow (\forall x.\ M(x) \longrightarrow \neg\ L(x))\rangle$
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬28. AMENDED

lemma
$\langle(\forall x.\ P(x) \longrightarrow (\forall x.\ Q(x))) \wedge$
$(\neg\ \neg\ (\forall x.\ Q(x) \vee R(x)) \longrightarrow (\exists x.\ Q(x) \wedge S(x))) \wedge$
$(\neg\ \neg\ (\exists x.\ S(x)) \longrightarrow (\forall x.\ L(x) \longrightarrow M(x)))$
$\longrightarrow (\forall x.\ P(x) \wedge L(x) \longrightarrow M(x))\rangle$
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

29. Essentially the same as Principia Mathematica *11.71

lemma
$\langle(\exists x.\ P(x)) \wedge (\exists y.\ Q(y))$
$\longrightarrow ((\forall x.\ P(x) \longrightarrow R(x)) \wedge (\forall y.\ Q(y) \longrightarrow S(y)) \longleftrightarrow$
$(\forall x\ y.\ P(x) \wedge Q(y) \longrightarrow R(x) \wedge S(y)))\rangle$
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬30

lemma
$\langle(\forall x.\ (P(x) \vee\ Q(x)) \longrightarrow \neg\ R(x)) \wedge$
$(\forall x.\ (Q(x) \longrightarrow \neg\ S(x)) \longrightarrow P(x) \wedge R(x))$
$\longrightarrow (\forall x.\ \neg\ \neg\ S(x))\rangle$
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

31

lemma
$\langle\neg\ (\exists x.\ P(x) \wedge (Q(x) \vee R(x))) \wedge$
$(\exists x.\ L(x) \wedge P(x)) \wedge$
$(\forall x.\ \neg\ R(x) \longrightarrow M(x))$
$\longrightarrow (\exists x.\ L(x) \wedge M(x))\rangle$
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

32

lemma
$\langle(\forall x.\ P(x) \wedge (Q(x) \vee R(x)) \longrightarrow S(x)) \wedge$
$(\forall x.\ S(x) \wedge R(x) \longrightarrow L(x)) \wedge$
$(\forall x.\ M(x) \longrightarrow R(x))$
$\longrightarrow (\forall x.\ P(x) \wedge M(x) \longrightarrow L(x))\rangle$

by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

¬¬33

lemma
‹(∀ *x*. ¬ ¬ (*P*(*a*) ∧ (*P*(*x*) ⟶ *P*(*b*)) ⟶ *P*(*c*))) ⟷
(∀ *x*. ¬ ¬ ((¬ *P*(*a*) ∨ *P*(*x*) ∨ *P*(*c*)) ∧ (¬ *P*(*a*) ∨ ¬ *P*(*b*) ∨ *P*(*c*))))›
apply (*tactic* ‹*IntPr.best-tac* ***context*** *1*›)
done

36

lemma
‹(∀ *x*. ∃ *y*. *J*(*x*,*y*)) ∧
(∀ *x*. ∃ *y*. *G*(*x*,*y*)) ∧
(∀ *x y*. *J*(*x*,*y*) ∨ *G*(*x*,*y*) ⟶ (∀ *z*. *J*(*y*,*z*) ∨ *G*(*y*,*z*) ⟶ *H*(*x*,*z*)))
⟶ (∀ *x*. ∃ *y*. *H*(*x*,*y*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

37

lemma
‹(∀ *z*. ∃ *w*. ∀ *x*. ∃ *y*.
¬ ¬ (*P*(*x*,*z*) ⟶ *P*(*y*,*w*)) ∧ *P*(*y*,*z*) ∧ (*P*(*y*,*w*) ⟶ (∃ *u*. *Q*(*u*,*w*)))) ∧
(∀ *x z*. ¬ *P*(*x*,*z*) ⟶ (∃ *y*. *Q*(*y*,*z*))) ∧
(¬ ¬ (∃ *x y*. *Q*(*x*,*y*)) ⟶ (∀ *x*. *R*(*x*,*x*)))
⟶ ¬ ¬ (∀ *x*. ∃ *y*. *R*(*x*,*y*))›
oops — NOT PROVED

39

lemma ‹¬ (∃ *x*. ∀ *y*. *F*(*y*,*x*) ⟷ ¬ *F*(*y*,*y*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

40. AMENDED

lemma
‹(∃ *y*. ∀ *x*. *F*(*x*,*y*) ⟷ *F*(*x*,*x*)) ⟶
¬ (∀ *x*. ∃ *y*. ∀ *z*. *F*(*z*,*y*) ⟷ ¬ *F*(*z*,*x*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

44

lemma
‹(∀ *x*. *f*(*x*) ⟶
(∃ *y*. *g*(*y*) ∧ *h*(*x*,*y*) ∧ (∃ *y*. *g*(*y*) ∧ ¬ *h*(*x*,*y*)))) ∧
(∃ *x*. *j*(*x*) ∧ (∀ *y*. *g*(*y*) ⟶ *h*(*x*,*y*)))
⟶ (∃ *x*. *j*(*x*) ∧ ¬ *f*(*x*))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

48

lemma ‹(*a* = *b* ∨ *c* = *d*) ∧ (*a* = *c* ∨ *b* = *d*) ⟶ *a* = *d* ∨ *b* = *c*›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

51

lemma
‹(∃ z w. ∀ x y. P(x,y) ⟷ (x = z ∧ y = w)) ⟶
(∃ z. ∀ x. ∃ w. (∀ y. P(x,y) ⟷ y = w) ⟷ x = z)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

52

Almost the same as 51.

lemma
‹(∃ z w. ∀ x y. P(x,y) ⟷ (x = z ∧ y = w)) ⟶
(∃ w. ∀ y. ∃ z. (∀ x. P(x,y) ⟷ x = z) ⟷ y = w)›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

56

lemma ‹(∀ x. (∃ y. P(y) ∧ x = f(y)) ⟶ P(x)) ⟷ (∀ x. P(x) ⟶ P(f(x)))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

57

lemma
‹P(f(a,b), f(b,c)) ∧ P(f(b,c), f(a,c)) ∧
(∀ x y z. P(x,y) ∧ P(y,z) ⟶ P(x,z)) ⟶ P(f(a,b), f(a,c))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

60

lemma ‹∀ x. P(x,f(x)) ⟷ (∃ y. (∀ z. P(z,y) ⟶ P(z,f(x))) ∧ P(x,y))›
by (*tactic* ‹*IntPr.fast-tac* ***context*** *1*›)

end

9 First-Order Logic: propositional examples (intuitionistic version)

theory *Propositional-Int*
imports *IFOL*
begin

commutative laws of ∧ and ∨

lemma ‹P ∧ Q ⟶ Q ∧ P›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹P ∨ Q ⟶ Q ∨ P›
by (*tactic IntPr.fast-tac* ***context*** *1*)

associative laws of ∧ and ∨

lemma ‹(P ∧ Q) ∧ R ⟶ P ∧ (Q ∧ R)›

by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \vee Q) \vee R \longrightarrow P \vee (Q \vee R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

distributive laws of $\wedge$ and $\vee$

lemma ‹$(P \wedge Q) \vee R \longrightarrow (P \vee R) \wedge (Q \vee R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \vee R) \wedge (Q \vee R) \longrightarrow (P \wedge Q) \vee R$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \vee Q) \wedge R \longrightarrow (P \wedge R) \vee (Q \wedge R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \wedge R) \vee (Q \wedge R) \longrightarrow (P \vee Q) \wedge R$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

Laws involving implication

lemma ‹$(P \longrightarrow R) \wedge (Q \longrightarrow R) \longleftrightarrow (P \vee Q \longrightarrow R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \wedge Q \longrightarrow R) \longleftrightarrow (P \longrightarrow (Q \longrightarrow R))$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$((P \longrightarrow R) \longrightarrow R) \longrightarrow ((Q \longrightarrow R) \longrightarrow R) \longrightarrow (P \wedge Q \longrightarrow R) \longrightarrow R$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$\neg\ (P \longrightarrow R) \longrightarrow \neg\ (Q \longrightarrow R) \longrightarrow \neg\ (P \wedge Q \longrightarrow R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \longrightarrow Q \wedge R) \longleftrightarrow (P \longrightarrow Q) \wedge (P \longrightarrow R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

Propositions-as-types

lemma ‹$P \longrightarrow (Q \longrightarrow P)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

— The combinator S
lemma ‹$(P \longrightarrow Q \longrightarrow R) \longrightarrow (P \longrightarrow Q) \longrightarrow (P \longrightarrow R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

— Converse is classical
lemma ‹$(P \longrightarrow Q) \vee (P \longrightarrow R) \longrightarrow (P \longrightarrow Q \vee R)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(P \longrightarrow Q) \longrightarrow (\neg\ Q \longrightarrow \neg\ P)$›
by (*tactic IntPr.fast-tac* ***context*** *1*)

Schwichtenberg's examples (via T. Nipkow)

lemma *stab-imp*: ‹(((*Q* ⟶ *R*) ⟶ *R*) ⟶ *Q*) ⟶ (((*P* ⟶ *Q*) ⟶ *R*) ⟶ *R*) ⟶ *P* ⟶ *Q*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *stab-to-peirce*:
‹(((*P* ⟶ *R*) ⟶ *R*) ⟶ *P*) ⟶ (((*Q* ⟶ *R*) ⟶ *R*) ⟶ *Q*)
⟶ ((*P* ⟶ *Q*) ⟶ *P*) ⟶ *P*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *peirce-imp1*:
‹(((*Q* ⟶ *R*) ⟶ *Q*) ⟶ *Q*)
⟶ (((*P* ⟶ *Q*) ⟶ *R*) ⟶ *P* ⟶ *Q*) ⟶ *P* ⟶ *Q*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *peirce-imp2*: ‹(((*P* ⟶ *R*) ⟶ *P*) ⟶ *P*) ⟶ ((*P* ⟶ *Q* ⟶ *R*) ⟶ *P*) ⟶ *P*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *mints*: ‹((((*P* ⟶ *Q*) ⟶ *P*) ⟶ *P*) ⟶ *Q*) ⟶ *Q*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *mints-solovev*: ‹(*P* ⟶ (*Q* ⟶ *R*) ⟶ *Q*) ⟶ ((*P* ⟶ *Q*) ⟶ *R*) ⟶ *R*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *tatsuta*:
‹(((*P7* ⟶ *P1*) ⟶ *P10*) ⟶ *P4* ⟶ *P5*)
⟶ (((*P8* ⟶ *P2*) ⟶ *P9*) ⟶ *P3* ⟶ *P10*)
⟶ (*P1* ⟶ *P8*) ⟶ *P6* ⟶ *P7*
⟶ (((*P3* ⟶ *P2*) ⟶ *P9*) ⟶ *P4*)
⟶ (*P1* ⟶ *P3*) ⟶ (((*P6* ⟶ *P1*) ⟶ *P2*) ⟶ *P9*) ⟶ *P5*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma *tatsuta1*:
‹(((*P8* ⟶ *P2*) ⟶ *P9*) ⟶ *P3* ⟶ *P10*)
⟶ (((*P3* ⟶ *P2*) ⟶ *P9*) ⟶ *P4*)
⟶ (((*P6* ⟶ *P1*) ⟶ *P2*) ⟶ *P9*)
⟶ (((*P7* ⟶ *P1*) ⟶ *P10*) ⟶ *P4* ⟶ *P5*)
⟶ (*P1* ⟶ *P3*) ⟶ (*P1* ⟶ *P8*) ⟶ *P6* ⟶ *P7* ⟶ *P5*›
by (*tactic IntPr.fast-tac* ***context*** *1*)

end

10 First-Order Logic: quantifier examples (intuitionistic version)

theory *Quantifiers-Int*
imports *IFOL*

begin

lemma ‹$(\forall x\ y.\ P(x,y)) \longrightarrow (\forall y\ x.\ P(x,y))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(\exists x\ y.\ P(x,y)) \longrightarrow (\exists y\ x.\ P(x,y))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

— Converse is false
lemma ‹$(\forall x.\ P(x)) \vee (\forall x.\ Q(x)) \longrightarrow (\forall x.\ P(x) \vee Q(x))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(\forall x.\ P \longrightarrow Q(x)) \longleftrightarrow (P \longrightarrow (\forall x.\ Q(x)))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(\forall x.\ P(x) \longrightarrow Q) \longleftrightarrow ((\exists x.\ P(x)) \longrightarrow Q)$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

Some harder ones

lemma ‹$(\exists x.\ P(x) \vee Q(x)) \longleftrightarrow (\exists x.\ P(x)) \vee (\exists x.\ Q(x))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

— Converse is false
lemma ‹$(\exists x.\ P(x) \wedge Q(x)) \longrightarrow (\exists x.\ P(x)) \wedge (\exists x.\ Q(x))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

Basic test of quantifier reasoning

lemma ‹$(\exists y.\ \forall x.\ Q(x,y)) \longrightarrow (\forall x.\ \exists y.\ Q(x,y))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$(\forall x.\ Q(x)) \longrightarrow (\exists x.\ Q(x))$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

The following should fail, as they are false!

lemma ‹$(\forall x.\ \exists y.\ Q(x,y)) \longrightarrow (\exists y.\ \forall x.\ Q(x,y))$›
  apply (*tactic IntPr.fast-tac* ***context*** *1*)*?*
  oops

lemma ‹$(\exists x.\ Q(x)) \longrightarrow (\forall x.\ Q(x))$›
  apply (*tactic IntPr.fast-tac* ***context*** *1*)*?*
  oops

schematic-goal ‹$P(?a) \longrightarrow (\forall x.\ P(x))$›
  apply (*tactic IntPr.fast-tac* ***context*** *1*)*?*
  oops

schematic-goal ‹$(P(?a) \longrightarrow (\forall x.\ Q(x))) \longrightarrow (\forall x.\ P(x) \longrightarrow Q(x))$›

apply (*tactic IntPr.fast-tac* ***context*** *1*)*?*
oops

Back to things that are provable ...

lemma ‹(∀*x*. *P*(*x*) ⟶ *Q*(*x*)) ∧ (∃*x*. *P*(*x*)) ⟶ (∃*x*. *Q*(*x*))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

— An example of why exI should be delayed as long as possible
lemma ‹(*P* ⟶ (∃*x*. *Q*(*x*))) ∧ *P* ⟶ (∃*x*. *Q*(*x*))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

schematic-goal ‹(∀*x*. *P*(*x*) ⟶ *Q*(*f*(*x*))) ∧ (∀*x*. *Q*(*x*) ⟶ *R*(*g*(*x*))) ∧ *P*(*d*) ⟶ *R*(*?a*)›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹(∀*x*. *Q*(*x*)) ⟶ (∃*x*. *Q*(*x*))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

Some slow ones

lemma ‹(∀*x y*. *P*(*x*) ⟶ *Q*(*y*)) ⟷ ((∃*x*. *P*(*x*)) ⟶ (∀*y*. *Q*(*y*)))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹(∃*x y*. *P*(*x*) ∧ *Q*(*x*,*y*)) ⟷ (∃*x*. *P*(*x*) ∧ (∃*y*. *Q*(*x*,*y*)))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹(∃*y*. ∀*x*. *P*(*x*) ⟶ *Q*(*x*,*y*)) ⟶ (∀*x*. *P*(*x*) ⟶ (∃*y*. *Q*(*x*,*y*)))›
by (*tactic IntPr.fast-tac* ***context*** *1*)

end

11 Classical Predicate Calculus Problems

theory *Classical*
imports *FOL*
begin

lemma ‹(*P* ⟶ *Q* ∨ *R*) ⟶ (*P* ⟶ *Q*) ∨ (*P* ⟶ *R*)›
by *blast*

11.0.1 If and only if

lemma ‹(*P* ⟷ *Q*) ⟷ (*Q* ⟷ *P*)›
by *blast*

lemma ‹¬ (*P* ⟷ ¬ *P*)›
by *blast*

11.1 Pelletier's examples

Sample problems from

- F. J. Pelletier, Seventy-Five Problems for Testing Automatic Theorem Provers, J. Automated Reasoning 2 (1986), 191-216. Errata, JAR 4 (1988), 236-236.

The hardest problems – judging by experience with several theorem provers, including matrix ones – are 34 and 43.

1

lemma ‹$(P \longrightarrow Q) \longleftrightarrow (\neg\ Q \longrightarrow \neg\ P)$›
by *blast*

2

lemma ‹$\neg\ \neg\ P \longleftrightarrow P$›
by *blast*

3

lemma ‹$\neg\ (P \longrightarrow Q) \longrightarrow (Q \longrightarrow P)$›
by *blast*

4

lemma ‹$(\neg\ P \longrightarrow Q) \longleftrightarrow (\neg\ Q \longrightarrow P)$›
by *blast*

5

lemma ‹$((P \vee Q) \longrightarrow (P \vee R)) \longrightarrow (P \vee (Q \longrightarrow R))$›
by *blast*

6

lemma ‹$P \vee \neg\ P$›
by *blast*

7

lemma ‹$P \vee \neg\ \neg\ \neg\ P$›
by *blast*

8. Peirce's law

lemma ‹$((P \longrightarrow Q) \longrightarrow P) \longrightarrow P$›
by *blast*

9

lemma ‹$((P \vee Q) \wedge (\neg\ P \vee Q) \wedge (P \vee \neg\ Q)) \longrightarrow \neg\ (\neg\ P \vee \neg\ Q)$›
by *blast*

10

lemma ‹$(Q \longrightarrow R) \wedge (R \longrightarrow P \wedge Q) \wedge (P \longrightarrow Q \vee R) \longrightarrow (P \longleftrightarrow Q)$›
by *blast*

11. Proved in each direction (incorrectly, says Pelletier!!)

lemma ‹$P \longleftrightarrow P$›
by *blast*

12. "Dijkstra's law"

lemma ‹$((P \longleftrightarrow Q) \longleftrightarrow R) \longleftrightarrow (P \longleftrightarrow (Q \longleftrightarrow R))$›
by *blast*

13. Distributive law

lemma ‹$P \vee (Q \wedge R) \longleftrightarrow (P \vee Q) \wedge (P \vee R)$›
by *blast*

14

lemma ‹$(P \longleftrightarrow Q) \longleftrightarrow ((Q \vee \neg P) \wedge (\neg Q \vee P))$›
by *blast*

15

lemma ‹$(P \longrightarrow Q) \longleftrightarrow (\neg P \vee Q)$›
by *blast*

16

lemma ‹$(P \longrightarrow Q) \vee (Q \longrightarrow P)$›
by *blast*

17

lemma ‹$((P \wedge (Q \longrightarrow R)) \longrightarrow S) \longleftrightarrow ((\neg P \vee Q \vee S) \wedge (\neg P \vee \neg R \vee S))$›
by *blast*

11.2 Classical Logic: examples with quantifiers

lemma ‹$(\forall x.\ P(x) \wedge Q(x)) \longleftrightarrow (\forall x.\ P(x)) \wedge (\forall x.\ Q(x))$›
by *blast*

lemma ‹$(\exists x.\ P \longrightarrow Q(x)) \longleftrightarrow (P \longrightarrow (\exists x.\ Q(x)))$›
by *blast*

lemma ‹$(\exists x.\ P(x) \longrightarrow Q) \longleftrightarrow (\forall x.\ P(x)) \longrightarrow Q$›
by *blast*

lemma ‹$(\forall x.\ P(x)) \vee Q \longleftrightarrow (\forall x.\ P(x) \vee Q)$›
by *blast*

Discussed in Avron, Gentzen-Type Systems, Resolution and Tableaux, JAR 10 (265-281), 1993. Proof is trivial!

lemma ‹¬ ((∃ x. ¬ P(x)) ∧ ((∃ x. P(x)) ∨ (∃ x. P(x) ∧ Q(x))) ∧ ¬ (∃ x. P(x)))›
by *blast*

11.3 Problems requiring quantifier duplication

Theorem B of Peter Andrews, Theorem Proving via General Matings, JACM 28 (1981).

lemma ‹(∃ x. ∀ y. P(x) ⟷ P(y)) ⟶ ((∃ x. P(x)) ⟷ (∀ y. P(y)))›
by *blast*

Needs multiple instantiation of ALL.

lemma ‹(∀ x. P(x) ⟶ P(f(x))) ∧ P(d) ⟶ P(f(f(f(d))))›
by *blast*

Needs double instantiation of the quantifier

lemma ‹∃ x. P(x) ⟶ P(a) ∧ P(b)›
by *blast*

lemma ‹∃ z. P(z) ⟶ (∀ x. P(x))›
by *blast*

lemma ‹∃ x. (∃ y. P(y)) ⟶ P(x)›
by *blast*

V. Lifschitz, What Is the Inverse Method?, JAR 5 (1989), 1–23. NOT PROVED.

lemma
‹∃ x x'. ∀ y. ∃ z z'.
(¬ P(y,y) ∨ P(x,x) ∨ ¬ S(z,x)) ∧
(S(x,y) ∨ ¬ S(y,z) ∨ Q(z',z')) ∧
(Q(x',y) ∨ ¬ Q(y,z') ∨ S(x',x'))›
oops

11.4 Hard examples with quantifiers

18

lemma ‹∃ y. ∀ x. P(y) ⟶ P(x)›
by *blast*

19

lemma ‹∃ x. ∀ y z. (P(y) ⟶ Q(z)) ⟶ (P(x) ⟶ Q(x))›
by *blast*

20

lemma ‹(∀ x y. ∃ z. ∀ w. (P(x) ∧ Q(y) ⟶ R(z) ∧ S(w)))
⟶ (∃ x y. P(x) ∧ Q(y)) ⟶ (∃ z. R(z))›

by *blast*

21

lemma ‹$(\exists\, x.\ P \longrightarrow Q(x)) \wedge (\exists\, x.\ Q(x) \longrightarrow P) \longrightarrow (\exists\, x.\ P \longleftrightarrow Q(x))$›
by *blast*

22

lemma ‹$(\forall\, x.\ P \longleftrightarrow Q(x)) \longrightarrow (P \longleftrightarrow (\forall\, x.\ Q(x)))$›
by *blast*

23

lemma ‹$(\forall\, x.\ P \vee Q(x)) \longleftrightarrow (P \vee (\forall\, x.\ Q(x)))$›
by *blast*

24

lemma
‹$\neg\ (\exists\, x.\ S(x) \wedge Q(x)) \wedge (\forall\, x.\ P(x) \longrightarrow Q(x) \vee R(x)) \wedge$
$(\neg\ (\exists\, x.\ P(x)) \longrightarrow (\exists\, x.\ Q(x))) \wedge (\forall\, x.\ Q(x) \vee R(x) \longrightarrow S(x))$
$\longrightarrow (\exists\, x.\ P(x) \wedge R(x))$›
by *blast*

25

lemma
‹$(\exists\, x.\ P(x)) \wedge$
$(\forall\, x.\ L(x) \longrightarrow \neg\ (M(x) \wedge R(x))) \wedge$
$(\forall\, x.\ P(x) \longrightarrow (M(x) \wedge L(x))) \wedge$
$((\forall\, x.\ P(x) \longrightarrow Q(x)) \vee (\exists\, x.\ P(x) \wedge R(x)))$
$\longrightarrow (\exists\, x.\ Q(x) \wedge P(x))$›
by *blast*

26

lemma
‹$((\exists\, x.\ p(x)) \longleftrightarrow (\exists\, x.\ q(x))) \wedge$
$(\forall\, x.\ \forall\, y.\ p(x) \wedge q(y) \longrightarrow (r(x) \longleftrightarrow s(y)))$
$\longrightarrow ((\forall\, x.\ p(x) \longrightarrow r(x)) \longleftrightarrow (\forall\, x.\ q(x) \longrightarrow s(x)))$›
by *blast*

27

lemma
‹$(\exists\, x.\ P(x) \wedge \neg\ Q(x)) \wedge$
$(\forall\, x.\ P(x) \longrightarrow R(x)) \wedge$
$(\forall\, x.\ M(x) \wedge L(x) \longrightarrow P(x)) \wedge$
$((\exists\, x.\ R(x) \wedge \neg\ Q(x)) \longrightarrow (\forall\, x.\ L(x) \longrightarrow \neg\ R(x)))$
$\longrightarrow (\forall\, x.\ M(x) \longrightarrow \neg\ L(x))$›
by *blast*

28. AMENDED

lemma
‹(∀ x. P(x) ⟶ (∀ x. Q(x))) ∧
(∀ x. Q(x) ∨ R(x)) ⟶ (∃ x. Q(x) ∧ S(x))) ∧
((∃ x. S(x)) ⟶ (∀ x. L(x) ⟶ M(x)))
⟶ (∀ x. P(x) ∧ L(x) ⟶ M(x))›
by *blast*

29. Essentially the same as Principia Mathematica *11.71

lemma
‹(∃ x. P(x)) ∧ (∃ y. Q(y))
⟶ ((∀ x. P(x) ⟶ R(x)) ∧ (∀ y. Q(y) ⟶ S(y)) ⟷
(∀ x y. P(x) ∧ Q(y) ⟶ R(x) ∧ S(y)))›
by *blast*

30

lemma
‹(∀ x. P(x) ∨ Q(x) ⟶ ¬ R(x)) ∧
(∀ x. (Q(x) ⟶ ¬ S(x)) ⟶ P(x) ∧ R(x))
⟶ (∀ x. S(x))›
by *blast*

31

lemma
‹¬ (∃ x. P(x) ∧ (Q(x) ∨ R(x))) ∧
(∃ x. L(x) ∧ P(x)) ∧
(∀ x. ¬ R(x) ⟶ M(x))
⟶ (∃ x. L(x) ∧ M(x))›
by *blast*

32

lemma
‹(∀ x. P(x) ∧ (Q(x) ∨ R(x)) ⟶ S(x)) ∧
(∀ x. S(x) ∧ R(x) ⟶ L(x)) ∧
(∀ x. M(x) ⟶ R(x))
⟶ (∀ x. P(x) ∧ M(x) ⟶ L(x))›
by *blast*

33

lemma
‹(∀ x. P(a) ∧ (P(x) ⟶ P(b)) ⟶ P(c)) ⟷
(∀ x. (¬ P(a) ∨ P(x) ∨ P(c)) ∧ (¬ P(a) ∨ ¬ P(b) ∨ P(c)))›
by *blast*

34. AMENDED (TWICE!!). Andrews's challenge.

lemma
‹((∃ x. ∀ y. p(x) ⟷ p(y)) ⟷ ((∃ x. q(x)) ⟷ (∀ y. p(y)))) ⟷
((∃ x. ∀ y. q(x) ⟷ q(y)) ⟷ ((∃ x. p(x)) ⟷ (∀ y. q(y))))›
by *blast*

35

lemma ‹$\exists\, x\ y.\ P(x,y) \longrightarrow (\forall\, u\ v.\ P(u,v))$›
by *blast*

36

lemma
‹$(\forall\, x.\ \exists\, y.\ J(x,y)) \wedge$
$(\forall\, x.\ \exists\, y.\ G(x,y)) \wedge$
$(\forall\, x\ y.\ J(x,y) \vee G(x,y) \longrightarrow (\forall\, z.\ J(y,z) \vee G(y,z) \longrightarrow H(x,z)))$
$\longrightarrow (\forall\, x.\ \exists\, y.\ H(x,y))$›
by *blast*

37

lemma
‹$(\forall\, z.\ \exists\, w.\ \forall\, x.\ \exists\, y.$
$(P(x,z) \longrightarrow P(y,w)) \wedge P(y,z) \wedge (P(y,w) \longrightarrow (\exists\, u.\ Q(u,w)))) \wedge$
$(\forall\, x\ z.\ \neg\ P(x,z) \longrightarrow (\exists\, y.\ Q(y,z))) \wedge$
$((\exists\, x\ y.\ Q(x,y)) \longrightarrow (\forall\, x.\ R(x,x)))$
$\longrightarrow (\forall\, x.\ \exists\, y.\ R(x,y))$›
by *blast*

38

lemma
‹$(\forall\, x.\ p(a) \wedge (p(x) \longrightarrow (\exists\, y.\ p(y) \wedge r(x,y))) \longrightarrow$
$(\exists\, z.\ \exists\, w.\ p(z) \wedge r(x,w) \wedge r(w,z))\ \longleftrightarrow$
$(\forall\, x.\ (\neg\ p(a) \vee p(x) \vee (\exists\, z.\ \exists\, w.\ p(z) \wedge r(x,w) \wedge r(w,z))) \wedge$
$(\neg\ p(a) \vee \neg\ (\exists\, y.\ p(y) \wedge r(x,y)) \vee$
$(\exists\, z.\ \exists\, w.\ p(z) \wedge r(x,w) \wedge r(w,z))))$›
by *blast*

39

lemma ‹$\neg\ (\exists\, x.\ \forall\, y.\ F(y,x) \longleftrightarrow \neg\ F(y,y))$›
by *blast*

40. AMENDED

lemma
‹$(\exists\, y.\ \forall\, x.\ F(x,y) \longleftrightarrow F(x,x)) \longrightarrow$
$\neg\ (\forall\, x.\ \exists\, y.\ \forall\, z.\ F(z,y) \longleftrightarrow \neg\ F(z,x))$›
by *blast*

41

lemma
‹$(\forall\, z.\ \exists\, y.\ \forall\, x.\ f(x,y) \longleftrightarrow f(x,z) \wedge \neg\ f(x,x))$
$\longrightarrow \neg\ (\exists\, z.\ \forall\, x.\ f(x,z))$›
by *blast*

42

lemma ‹¬ (∃ y. ∀ x. p(x,y) ⟷ ¬ (∃ z. p(x,z) ∧ p(z,x)))›
by *blast*

43

lemma
‹(∀ x. ∀ y. q(x,y) ⟷ (∀ z. p(z,x) ⟷ p(z,y)))
⟶ (∀ x. ∀ y. q(x,y) ⟷ q(y,x))›
by *blast*

Other proofs: Can use *auto*, which cheats by using rewriting! *Deepen-tac* alone requires 253 secs. Or *by* (*mini-tac 1 THEN Deepen-tac 5 1*).

44

lemma
‹(∀ x. f(x) ⟶ (∃ y. g(y) ∧ h(x,y) ∧ (∃ y. g(y) ∧ ¬ h(x,y)))) ∧
(∃ x. j(x) ∧ (∀ y. g(y) ⟶ h(x,y)))
⟶ (∃ x. j(x) ∧ ¬ f(x))›
by *blast*

45

lemma
‹(∀ x. f(x) ∧ (∀ y. g(y) ∧ h(x,y) ⟶ j(x,y))
⟶ (∀ y. g(y) ∧ h(x,y) ⟶ k(y))) ∧
¬ (∃ y. l(y) ∧ k(y)) ∧
(∃ x. f(x) ∧ (∀ y. h(x,y) ⟶ l(y)) ∧ (∀ y. g(y) ∧ h(x,y) ⟶ j(x,y)))
⟶ (∃ x. f(x) ∧ ¬ (∃ y. g(y) ∧ h(x,y)))›
by *blast*

46

lemma
‹(∀ x. f(x) ∧ (∀ y. f(y) ∧ h(y,x) ⟶ g(y)) ⟶ g(x)) ∧
((∃ x. f(x) ∧ ¬ g(x)) ⟶
(∃ x. f(x) ∧ ¬ g(x) ∧ (∀ y. f(y) ∧ ¬ g(y) ⟶ j(x,y)))) ∧
(∀ x y. f(x) ∧ f(y) ∧ h(x,y) ⟶ ¬ j(y,x))
⟶ (∀ x. f(x) ⟶ g(x))›
by *blast*

11.5 Problems (mainly) involving equality or functions

48

lemma ‹(a = b ∨ c = d) ∧ (a = c ∨ b = d) ⟶ a = d ∨ b = c›
by *blast*

49. NOT PROVED AUTOMATICALLY. Hard because it involves substitution for Vars; the type constraint ensures that x,y,z have the same type as a,b,u.

lemma

‹(∃ x y::'a. ∀ z. z = x ∨ z = y) ∧ P(a) ∧ P(b) ∧ a ≠ b ⟶ (∀ u::'a. P(u))›
apply *safe*
apply (*rule-tac* x = ‹a› **in** *allE*, *assumption*)
apply (*rule-tac* x = ‹b› **in** *allE*, *assumption*)
apply *fast* — blast's treatment of equality can't do it
done

50. (What has this to do with equality?)

lemma ‹(∀ x. P(a,x) ∨ (∀ y. P(x,y))) ⟶ (∃ x. ∀ y. P(x,y))›
by *blast*

51

lemma
‹(∃ z w. ∀ x y. P(x,y) ⟷ (x = z ∧ y = w)) ⟶
(∃ z. ∀ x. ∃ w. (∀ y. P(x,y) ⟷ y=w) ⟷ x = z)›
by *blast*

52

Almost the same as 51.

lemma
‹(∃ z w. ∀ x y. P(x,y) ⟷ (x = z ∧ y = w)) ⟶
(∃ w. ∀ y. ∃ z. (∀ x. P(x,y) ⟷ x = z) ⟷ y = w)›
by *blast*

55

Non-equational version, from Manthey and Bry, CADE-9 (Springer, 1988).
fast DISCOVERS who killed Agatha.

schematic-goal
‹*lives*(*agatha*) ∧ *lives*(*butler*) ∧ *lives*(*charles*) ∧
(*killed*(*agatha*,*agatha*) ∨ *killed*(*butler*,*agatha*) ∨ *killed*(*charles*,*agatha*)) ∧
(∀ x y. *killed*(x,y) ⟶ *hates*(x,y) ∧ ¬ *richer*(x,y)) ∧
(∀ x. *hates*(*agatha*,x) ⟶ ¬ *hates*(*charles*,x)) ∧
(*hates*(*agatha*,*agatha*) ∧ *hates*(*agatha*,*charles*)) ∧
(∀ x. *lives*(x) ∧ ¬ *richer*(x,*agatha*) ⟶ *hates*(*butler*,x)) ∧
(∀ x. *hates*(*agatha*,x) ⟶ *hates*(*butler*,x)) ∧
(∀ x. ¬ *hates*(x,*agatha*) ∨ ¬ *hates*(x,*butler*) ∨ ¬ *hates*(x,*charles*)) ⟶
killed(*?who*,*agatha*)›
by *fast* — MUCH faster than blast

56

lemma ‹(∀ x. (∃ y. P(y) ∧ x = f(y)) ⟶ P(x)) ⟷ (∀ x. P(x) ⟶ P(f(x)))›
by *blast*

57

lemma
‹*P*(*f*(*a*,*b*), *f*(*b*,*c*)) ∧ *P*(*f*(*b*,*c*), *f*(*a*,*c*)) ∧

$(\forall\, x\ y\ z.\ P(x,y) \wedge P(y,z) \longrightarrow P(x,z)) \longrightarrow P(f(a,b),\ f(a,c))\rangle$
by *blast*

58 NOT PROVED AUTOMATICALLY

lemma $\langle(\forall\, x\ y.\ f(x) = g(y)) \longrightarrow (\forall\, x\ y.\ f(f(x)) = f(g(y)))\rangle$
by (*slow elim*: *subst-context*)

59

lemma $\langle(\forall\, x.\ P(x) \longleftrightarrow \neg\ P(f(x))) \longrightarrow (\exists\, x.\ P(x) \wedge \neg\ P(f(x)))\rangle$
by *blast*

60

lemma $\langle\forall\, x.\ P(x,f(x)) \longleftrightarrow (\exists\, y.\ (\forall\, z.\ P(z,y) \longrightarrow P(z,f(x))) \wedge P(x,y))\rangle$
by *blast*

62 as corrected in JAR 18 (1997), page 135

lemma
$\langle(\forall\, x.\ p(a) \wedge (p(x) \longrightarrow p(f(x))) \longrightarrow p(f(f(x)))) \longleftrightarrow$
$(\forall\, x.\ (\neg\ p(a) \vee p(x) \vee p(f(f(x)))) \wedge$
$(\neg\ p(a) \vee \neg\ p(f(x)) \vee p(f(f(x)))))\rangle$
by *blast*

From Davis, Obvious Logical Inferences, IJCAI-81, 530-531 fast indeed copes!

lemma
$\langle(\forall\, x.\ F(x) \wedge \neg\ G(x) \longrightarrow (\exists\, y.\ H(x,y) \wedge J(y))) \wedge$
$(\exists\, x.\ K(x) \wedge F(x) \wedge (\forall\, y.\ H(x,y) \longrightarrow K(y))) \wedge$
$(\forall\, x.\ K(x) \longrightarrow \neg\ G(x)) \longrightarrow (\exists\, x.\ K(x) \wedge J(x))\rangle$
by *fast*

From Rudnicki, Obvious Inferences, JAR 3 (1987), 383-393. It does seem obvious!

lemma
$\langle(\forall\, x.\ F(x) \wedge \neg\ G(x) \longrightarrow (\exists\, y.\ H(x,y) \wedge J(y))) \wedge$
$(\exists\, x.\ K(x) \wedge F(x) \wedge (\forall\, y.\ H(x,y) \longrightarrow K(y))) \wedge$
$(\forall\, x.\ K(x) \longrightarrow \neg\ G(x)) \longrightarrow (\exists\, x.\ K(x) \longrightarrow \neg\ G(x))\rangle$
by *fast*

Halting problem: Formulation of Li Dafa (AAR Newsletter 27, Oct 1994.) author U. Egly.

lemma
$\langle((\exists\, x.\ A(x) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(x,y,z)))) \longrightarrow$
$(\exists\, w.\ C(w) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(w,y,z)))))$
$\wedge$
$(\forall\, w.\ C(w) \wedge (\forall\, u.\ C(u) \longrightarrow (\forall\, v.\ D(w,u,v))) \longrightarrow$
$(\forall\, y\ z.$
$(C(y) \wedge P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,g)) \wedge$
$(C(y) \wedge \neg\ P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,b))))$

$\wedge$
$(\forall\, w.\ C(w) \wedge$
$(\forall\, y\ z.$
$(C(y) \wedge P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,g)) \wedge$
$(C(y) \wedge \neg\ P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,b))) \longrightarrow$
$(\exists\, v.\ C(v) \wedge$
$(\forall\, y.\ ((C(y) \wedge Q(w,y,y)) \wedge OO(w,g) \longrightarrow \neg\ P(v,y)) \wedge$
$((C(y) \wedge Q(w,y,y)) \wedge OO(w,b) \longrightarrow P(v,y) \wedge OO(v,b)))))$
$\longrightarrow \neg\ (\exists\, x.\ A(x) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(x,y,z))))\rangle$
by (*blast 12*)
— Needed because the search for depths below 12 is very slow.

Halting problem II: credited to M. Bruschi by Li Dafa in JAR 18(1), p. 105.

lemma
$\langle((\exists\, x.\ A(x) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(x,y,z)))) \longrightarrow$
$(\exists\, w.\ C(w) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(w,y,z)))))$
$\wedge$
$(\forall\, w.\ C(w) \wedge (\forall\, u.\ C(u) \longrightarrow (\forall\, v.\ D(w,u,v))) \longrightarrow$
$(\forall\, y\ z.$
$(C(y) \wedge P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,g)) \wedge$
$(C(y) \wedge \neg\ P(y,z) \longrightarrow Q(w,y,z) \wedge OO(w,b))))$
$\wedge$
$((\exists\, w.\ C(w) \wedge (\forall\, y.\ (C(y) \wedge P(y,y) \longrightarrow Q(w,y,y) \wedge OO(w,g)) \wedge$
$(C(y) \wedge \neg\ P(y,y) \longrightarrow Q(w,y,y) \wedge OO(w,b))))$
$\longrightarrow$
$(\exists\, v.\ C(v) \wedge (\forall\, y.\ (C(y) \wedge P(y,y) \longrightarrow P(v,y) \wedge OO(v,g)) \wedge$
$(C(y) \wedge \neg\ P(y,y) \longrightarrow P(v,y) \wedge OO(v,b)))))$
$\longrightarrow$
$((\exists\, v.\ C(v) \wedge (\forall\, y.\ (C(y) \wedge P(y,y) \longrightarrow P(v,y) \wedge OO(v,g)) \wedge$
$(C(y) \wedge \neg\ P(y,y) \longrightarrow P(v,y) \wedge OO(v,b))))$
$\longrightarrow$
$(\exists\, u.\ C(u) \wedge (\forall\, y.\ (C(y) \wedge P(y,y) \longrightarrow \neg\ P(u,y)) \wedge$
$(C(y) \wedge \neg\ P(y,y) \longrightarrow P(u,y) \wedge OO(u,b)))))$
$\longrightarrow \neg\ (\exists\, x.\ A(x) \wedge (\forall\, y.\ C(y) \longrightarrow (\forall\, z.\ D(x,y,z))))\rangle$
by *blast*

Challenge found on info-hol.

lemma $\langle\forall\, x.\ \exists\, v\ w.\ \forall\, y\ z.\ P(x) \wedge Q(y) \longrightarrow (P(v) \vee R(w)) \wedge (R(z) \longrightarrow Q(v))\rangle$
by *blast*

Attributed to Lewis Carroll by S. G. Pulman. The first or last assumption can be deleted.

lemma
$\langle(\forall\, x.\ honest(x) \wedge industrious(x) \longrightarrow healthy(x)) \wedge$
$\neg\ (\exists\, x.\ grocer(x) \wedge healthy(x)) \wedge$
$(\forall\, x.\ industrious(x) \wedge grocer(x) \longrightarrow honest(x)) \wedge$
$(\forall\, x.\ cyclist(x) \longrightarrow industrious(x)) \wedge$
$(\forall\, x.\ \neg\ healthy(x) \wedge cyclist(x) \longrightarrow \neg\ honest(x))$
$\longrightarrow (\forall\, x.\ grocer(x) \longrightarrow \neg\ cyclist(x))\rangle$

by *blast*

end

12 First-Order Logic: propositional examples (classical version)

theory *Propositional-Cla*
imports *FOL*
begin

commutative laws of $\wedge$ and $\vee$

lemma ‹$P \wedge Q \longrightarrow Q \wedge P$›
  by (*tactic IntPr.fast-tac* ***context*** *1*)

lemma ‹$P \vee Q \longrightarrow Q \vee P$›
  by *fast*

associative laws of $\wedge$ and $\vee$

lemma ‹$(P \wedge Q) \wedge R \longrightarrow P \wedge (Q \wedge R)$›
  by *fast*

lemma ‹$(P \vee Q) \vee R \longrightarrow P \vee (Q \vee R)$›
  by *fast*

distributive laws of $\wedge$ and $\vee$

lemma ‹$(P \wedge Q) \vee R \longrightarrow (P \vee R) \wedge (Q \vee R)$›
  by *fast*

lemma ‹$(P \vee R) \wedge (Q \vee R) \longrightarrow (P \wedge Q) \vee R$›
  by *fast*

lemma ‹$(P \vee Q) \wedge R \longrightarrow (P \wedge R) \vee (Q \wedge R)$›
  by *fast*

lemma ‹$(P \wedge R) \vee (Q \wedge R) \longrightarrow (P \vee Q) \wedge R$›
  by *fast*

Laws involving implication

lemma ‹$(P \longrightarrow R) \wedge (Q \longrightarrow R) \longleftrightarrow (P \vee Q \longrightarrow R)$›
  by *fast*

lemma ‹$(P \wedge Q \longrightarrow R) \longleftrightarrow (P \longrightarrow (Q \longrightarrow R))$›
  by *fast*

lemma ‹((P ⟶ R) ⟶ R) ⟶ ((Q ⟶ R) ⟶ R) ⟶ (P ∧ Q ⟶ R) ⟶ R›
by *fast*

lemma ‹¬ (P ⟶ R) ⟶ ¬ (Q ⟶ R) ⟶ ¬ (P ∧ Q ⟶ R)›
by *fast*

lemma ‹(P ⟶ Q ∧ R) ⟷ (P ⟶ Q) ∧ (P ⟶ R)›
by *fast*

Propositions-as-types

lemma ‹P ⟶ (Q ⟶ P)›
by *fast*

— The combinator S
lemma ‹(P ⟶ Q ⟶ R) ⟶ (P ⟶ Q) ⟶ (P ⟶ R)›
by *fast*

— Converse is classical
lemma ‹(P ⟶ Q) ∨ (P ⟶ R) ⟶ (P ⟶ Q ∨ R)›
by *fast*

lemma ‹(P ⟶ Q) ⟶ (¬ Q ⟶ ¬ P)›
by *fast*

Schwichtenberg's examples (via T. Nipkow)

lemma *stab-imp*: ‹(((Q ⟶ R) ⟶ R) ⟶ Q) ⟶ (((P ⟶ Q) ⟶ R) ⟶ R)
⟶ P ⟶ Q›
by *fast*

lemma *stab-to-peirce*:
‹(((P ⟶ R) ⟶ R) ⟶ P) ⟶ (((Q ⟶ R) ⟶ R) ⟶ Q)
⟶ ((P ⟶ Q) ⟶ P) ⟶ P›
by *fast*

lemma *peirce-imp1*:
‹(((Q ⟶ R) ⟶ Q) ⟶ Q)
⟶ (((P ⟶ Q) ⟶ R) ⟶ P ⟶ Q) ⟶ P ⟶ Q›
by *fast*

lemma *peirce-imp2*: ‹(((P ⟶ R) ⟶ P) ⟶ P) ⟶ ((P ⟶ Q ⟶ R) ⟶ P)
⟶ P›
by *fast*

lemma *mints*: ‹((((P ⟶ Q) ⟶ P) ⟶ P) ⟶ Q) ⟶ Q›
by *fast*

lemma *mints-solovev*: ‹(P ⟶ (Q ⟶ R) ⟶ Q) ⟶ ((P ⟶ Q) ⟶ R) ⟶ R›

by *fast*

lemma *tatsuta*:
‹(((*P7* ⟶ *P1*) ⟶ *P10*) ⟶ *P4* ⟶ *P5*)
⟶ (((*P8* ⟶ *P2*) ⟶ *P9*) ⟶ *P3* ⟶ *P10*)
⟶ (*P1* ⟶ *P8*) ⟶ *P6* ⟶ *P7*
⟶ (((*P3* ⟶ *P2*) ⟶ *P9*) ⟶ *P4*)
⟶ (*P1* ⟶ *P3*) ⟶ (((*P6* ⟶ *P1*) ⟶ *P2*) ⟶ *P9*) ⟶ *P5*›
by *fast*

lemma *tatsuta1*:
‹(((*P8* ⟶ *P2*) ⟶ *P9*) ⟶ *P3* ⟶ *P10*)
⟶ (((*P3* ⟶ *P2*) ⟶ *P9*) ⟶ *P4*)
⟶ (((*P6* ⟶ *P1*) ⟶ *P2*) ⟶ *P9*)
⟶ (((*P7* ⟶ *P1*) ⟶ *P10*) ⟶ *P4* ⟶ *P5*)
⟶ (*P1* ⟶ *P3*) ⟶ (*P1* ⟶ *P8*) ⟶ *P6* ⟶ *P7* ⟶ *P5*›
by *fast*

end

13 First-Order Logic: quantifier examples (classical version)

theory *Quantifiers-Cla*
imports *FOL*
begin

lemma ‹(∀*x y*. *P*(*x*,*y*)) ⟶ (∀*y x*. *P*(*x*,*y*))›
by *fast*

lemma ‹(∃*x y*. *P*(*x*,*y*)) ⟶ (∃*y x*. *P*(*x*,*y*))›
by *fast*

Converse is false.

lemma ‹(∀*x*. *P*(*x*)) ∨ (∀*x*. *Q*(*x*)) ⟶ (∀*x*. *P*(*x*) ∨ *Q*(*x*))›
by *fast*

lemma ‹(∀*x*. *P* ⟶ *Q*(*x*)) ⟷ (*P* ⟶ (∀*x*. *Q*(*x*)))›
by *fast*

lemma ‹(∀*x*. *P*(*x*) ⟶ *Q*) ⟷ ((∃*x*. *P*(*x*)) ⟶ *Q*)›
by *fast*

Some harder ones.

lemma ‹(∃*x*. *P*(*x*) ∨ *Q*(*x*)) ⟷ (∃*x*. *P*(*x*)) ∨ (∃*x*. *Q*(*x*))›
by *fast*

— Converse is false.
lemma ‹$(\exists x.\ P(x) \wedge Q(x)) \longrightarrow (\exists x.\ P(x)) \wedge (\exists x.\ Q(x))$›
by *fast*

Basic test of quantifier reasoning.

lemma ‹$(\exists y.\ \forall x.\ Q(x,y)) \longrightarrow (\forall x.\ \exists y.\ Q(x,y))$›
by *fast*

lemma ‹$(\forall x.\ Q(x)) \longrightarrow (\exists x.\ Q(x))$›
by *fast*

The following should fail, as they are false!

lemma ‹$(\forall x.\ \exists y.\ Q(x,y)) \longrightarrow (\exists y.\ \forall x.\ Q(x,y))$›
apply *fast?*
oops

lemma ‹$(\exists x.\ Q(x)) \longrightarrow (\forall x.\ Q(x))$›
apply *fast?*
oops

schematic-goal ‹$P(?a) \longrightarrow (\forall x.\ P(x))$›
apply *fast?*
oops

schematic-goal ‹$(P(?a) \longrightarrow (\forall x.\ Q(x))) \longrightarrow (\forall x.\ P(x) \longrightarrow Q(x))$›
apply *fast?*
oops

Back to things that are provable ...

lemma ‹$(\forall x.\ P(x) \longrightarrow Q(x)) \wedge (\exists x.\ P(x)) \longrightarrow (\exists x.\ Q(x))$›
by *fast*

An example of why *exI* should be delayed as long as possible.

lemma ‹$(P \longrightarrow (\exists x.\ Q(x))) \wedge P \longrightarrow (\exists x.\ Q(x))$›
by *fast*

schematic-goal ‹$(\forall x.\ P(x) \longrightarrow Q(f(x))) \wedge (\forall x.\ Q(x) \longrightarrow R(g(x))) \wedge P(d) \longrightarrow R(?a)$›
by *fast*

lemma ‹$(\forall x.\ Q(x)) \longrightarrow (\exists x.\ Q(x))$›
by *fast*

Some slow ones

Principia Mathematica *11.53

lemma ‹$(\forall x\ y.\ P(x) \longrightarrow Q(y)) \longleftrightarrow ((\exists x.\ P(x)) \longrightarrow (\forall y.\ Q(y)))$›
by *fast*

lemma ‹$(\exists\, x\ y.\ P(x) \wedge Q(x,y)) \longleftrightarrow (\exists\, x.\ P(x) \wedge (\exists\, y.\ Q(x,y)))$›
by *fast*

lemma ‹$(\exists\, y.\ \forall\, x.\ P(x) \longrightarrow Q(x,y)) \longrightarrow (\forall\, x.\ P(x) \longrightarrow (\exists\, y.\ Q(x,y)))$›
by *fast*

end

theory *Miniscope*
imports *FOL*
begin

lemmas *ccontr* = *FalseE* [*THEN classical*]

13.1 Negation Normal Form

13.1.1 de Morgan laws

lemma *demorgans1*:
‹$\neg\ (P \wedge Q) \longleftrightarrow \neg\ P \vee \neg\ Q$›
‹$\neg\ (P \vee Q) \longleftrightarrow \neg\ P \wedge \neg\ Q$›
‹$\neg\ \neg\ P \longleftrightarrow P$›
by *blast+*

lemma *demorgans2*:
‹$\bigwedge P.\ \neg\ (\forall\, x.\ P(x)) \longleftrightarrow (\exists\, x.\ \neg\ P(x))$›
‹$\bigwedge P.\ \neg\ (\exists\, x.\ P(x)) \longleftrightarrow (\forall\, x.\ \neg\ P(x))$›
by *blast+*

lemmas *demorgans* = *demorgans1 demorgans2*

lemma *nnf-simps*:
‹$(P \longrightarrow Q) \longleftrightarrow (\neg\ P \vee Q)$›
‹$\neg\ (P \longrightarrow Q) \longleftrightarrow (P \wedge \neg\ Q)$›
‹$(P \longleftrightarrow Q) \longleftrightarrow (\neg\ P \vee Q) \wedge (\neg\ Q \vee P)$›
‹$\neg\ (P \longleftrightarrow Q) \longleftrightarrow (P \vee Q) \wedge (\neg\ P \vee \neg\ Q)$›
by *blast+*

13.1.2 Pushing in the existential quantifiers

lemma *ex-simps*:
‹$(\exists\, x.\ P) \longleftrightarrow P$›
‹$\bigwedge P\ Q.\ (\exists\, x.\ P(x) \wedge Q) \longleftrightarrow (\exists\, x.\ P(x)) \wedge Q$›
‹$\bigwedge P\ Q.\ (\exists\, x.\ P \wedge Q(x)) \longleftrightarrow P \wedge (\exists\, x.\ Q(x))$›

‹⋀P Q. (∃x. P(x) ∨ Q(x)) ⟷ (∃x. P(x)) ∨ (∃x. Q(x))›
‹⋀P Q. (∃x. P(x) ∨ Q) ⟷ (∃x. P(x)) ∨ Q›
‹⋀P Q. (∃x. P ∨ Q(x)) ⟷ P ∨ (∃x. Q(x))›
by *blast+*

13.1.3 Pushing in the universal quantifiers

lemma *all-simps*:
‹(∀x. P) ⟷ P›
‹⋀P Q. (∀x. P(x) ∧ Q(x)) ⟷ (∀x. P(x)) ∧ (∀x. Q(x))›
‹⋀P Q. (∀x. P(x) ∧ Q) ⟷ (∀x. P(x)) ∧ Q›
‹⋀P Q. (∀x. P ∧ Q(x)) ⟷ P ∧ (∀x. Q(x))›
‹⋀P Q. (∀x. P(x) ∨ Q) ⟷ (∀x. P(x)) ∨ Q›
‹⋀P Q. (∀x. P ∨ Q(x)) ⟷ P ∨ (∀x. Q(x))›
by *blast+*

lemmas *mini-simps = demorgans nnf-simps ex-simps all-simps*

ML ‹
val mini-ss = Simplifier.simpset-of (***context*** *|> Simplifier.add-simps @{thms mini-simps});*
fun mini-tac ctxt =
resolve-tac ctxt @{thms ccontr} THEN' asm-full-simp-tac (*put-simpset mini-ss ctxt*);
›

end

14 First-Order Logic: the 'if' example

theory *If*
imports *FOL*
begin

definition *if* :: ‹[o,o,o]=>o›
where ‹*if*(P,Q,R) ≡ P ∧ Q ∨ ¬ P ∧ R›

lemma *ifI*: ‹⟦P ⟹ Q; ¬ P ⟹ R⟧ ⟹ *if*(P,Q,R)›
unfolding *if-def* **by** *blast*

lemma *ifE*: ‹⟦*if*(P,Q,R); ⟦P; Q⟧ ⟹ S; ⟦¬ P; R⟧ ⟹ S⟧ ⟹ S›
unfolding *if-def* **by** *blast*

lemma *if-commute*: ‹*if*(P, *if*(Q,A,B), *if*(Q,C,D)) ⟷ *if*(Q, *if*(P,A,C), *if*(P,B,D))›
apply (*rule iffI*)
apply (*erule ifE*)
apply (*erule ifE*)
apply (*rule ifI*)
apply (*rule ifI*)
oops

Trying again from the beginning in order to use *blast*

declare *ifI* [*intro*!]
declare *ifE* [*elim*!]

lemma *if-commute*: ‹*if*(*P*, *if*(*Q*,*A*,*B*), *if*(*Q*,*C*,*D*)) ⟷ *if*(*Q*, *if*(*P*,*A*,*C*), *if*(*P*,*B*,*D*))›
  by *blast*

lemma ‹*if*(*if*(*P*,*Q*,*R*), *A*, *B*) ⟷ *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*A*,*B*))›
  by *blast*

Trying again from the beginning in order to prove from the definitions

lemma ‹*if*(*if*(*P*,*Q*,*R*), *A*, *B*) ⟷ *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*A*,*B*))›
  unfolding *if-def* **by** *blast*

An invalid formula. High-level rules permit a simpler diagnosis.

lemma ‹*if*(*if*(*P*,*Q*,*R*), *A*, *B*) ⟷ *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*B*,*A*))›
  apply *auto*
    — The next step will fail unless subgoals remain
  apply (*tactic all-tac*)
  oops

Trying again from the beginning in order to prove from the definitions.

lemma ‹*if*(*if*(*P*,*Q*,*R*), *A*, *B*) ⟷ *if*(*P*, *if*(*Q*,*A*,*B*), *if*(*R*,*B*,*A*))›
  unfolding *if-def*
  apply *auto*
    — The next step will fail unless subgoals remain
  apply (*tactic all-tac*)
  oops

end