Miscellaneous FOL Examples

November 22, 2007

Contents

1 A simple formulation of First-Order Logic

theory *First-Order-Logic* **imports** *Pure* **begin**

The subsequent theory development illustrates single-sorted intuitionistic first-order logic with equality, formulated within the Pure framework. Actually this is not an example of Isabelle/FOL, but of Isabelle/Pure.

1.1 Syntax

typedecl *i*
typedecl *o*

judgment
Trueprop :: $o \Rightarrow prop$ (- 5)

1.2 Propositional logic

axiomatization
  false :: o ($\bot$) **and**
  imp :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\longrightarrow$ *25*) **and**
  conj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\wedge$ *35*) **and**
  disj :: $o \Rightarrow o \Rightarrow o$ (**infixr** $\vee$ *30*)
where
  falseE [*elim*]: $\bot \Longrightarrow A$ **and**

  impI [*intro*]: $(A \Longrightarrow B) \Longrightarrow A \longrightarrow B$ **and**
  mp [*dest*]: $A \longrightarrow B \Longrightarrow A \Longrightarrow B$ **and**

  conjI [*intro*]: $A \Longrightarrow B \Longrightarrow A \wedge B$ **and**
  conjD1: $A \wedge B \Longrightarrow A$ **and**
  conjD2: $A \wedge B \Longrightarrow B$ **and**

  disjE [*elim*]: $A \vee B \Longrightarrow (A \Longrightarrow C) \Longrightarrow (B \Longrightarrow C) \Longrightarrow C$ **and**
  disjI1 [*intro*]: $A \Longrightarrow A \vee B$ **and**
  disjI2 [*intro*]: $B \Longrightarrow A \vee B$

theorem *conjE* [*elim*]:
  assumes $A \wedge B$
  obtains A **and** B
proof
  from $\langle A \wedge B \rangle$ **show** A **by** (*rule conjD1*)
  from $\langle A \wedge B \rangle$ **show** B **by** (*rule conjD2*)
qed

definition
  true :: o ($\top$) **where**
  $\top \equiv \bot \longrightarrow \bot$

definition
  not :: $o \Rightarrow o$ ($\neg$ - [*40*] *40*) **where**
  $\neg A \equiv A \longrightarrow \bot$

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

theorem *trueI* [*intro*]: $\top$
proof (*unfold true-def*)
  show $\bot \longrightarrow \bot$ **..**
qed

theorem *notI* [*intro*]: $(A \Longrightarrow \bot) \Longrightarrow \neg A$
proof (*unfold not-def*)
  assume $A \Longrightarrow \bot$

then show $A \longrightarrow \bot$ **..**
qed

theorem *notE* [*elim*]: $\neg\ A \Longrightarrow A \Longrightarrow B$
proof (*unfold not-def*)
assume $A \longrightarrow \bot$ **and** A
then have $\bot$ **.. then show** B **..**
qed

theorem *iffI* [*intro*]: $(A \Longrightarrow B) \Longrightarrow (B \Longrightarrow A) \Longrightarrow A \longleftrightarrow B$
proof (*unfold iff-def*)
assume $A \Longrightarrow B$ **then have** $A \longrightarrow B$ **..**
moreover assume $B \Longrightarrow A$ **then have** $B \longrightarrow A$ **..**
ultimately show $(A \longrightarrow B) \wedge (B \longrightarrow A)$ **..**
qed

theorem *iff1* [*elim*]: $A \longleftrightarrow B \Longrightarrow A \Longrightarrow B$
proof (*unfold iff-def*)
assume $(A \longrightarrow B) \wedge (B \longrightarrow A)$
then have $A \longrightarrow B$ **..**
then show $A \Longrightarrow B$ **..**
qed

theorem *iff2* [*elim*]: $A \longleftrightarrow B \Longrightarrow B \Longrightarrow A$
proof (*unfold iff-def*)
assume $(A \longrightarrow B) \wedge (B \longrightarrow A)$
then have $B \longrightarrow A$ **..**
then show $B \Longrightarrow A$ **..**
qed

1.3 Equality

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

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

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

1.4 Quantifiers

axiomatization

All :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\forall$ *10*) **and**
Ex :: $(i \Rightarrow o) \Rightarrow o$ (**binder** $\exists$ *10*)
where
allI [*intro*]: $(\bigwedge x.\ P(x)) \Longrightarrow \forall x.\ P(x)$ **and**
allD [*dest*]: $\forall x.\ P(x) \Longrightarrow P(a)$ **and**
exI [*intro*]: $P(a) \Longrightarrow \exists x.\ P(x)$ **and**
exE [*elim*]: $\exists x.\ P(x) \Longrightarrow (\bigwedge x.\ P(x) \Longrightarrow C) \Longrightarrow C$

lemma $(\exists x.\ P(f(x))) \longrightarrow (\exists y.\ P(y))$
proof
  assume $\exists x.\ P(f(x))$
  then show $\exists y.\ P(y)$
  proof
    fix x **assume** $P(f(x))$
    then show *?thesis* **..**
  qed
qed

lemma $(\exists x.\ \forall y.\ R(x, y)) \longrightarrow (\forall y.\ \exists x.\ R(x, y))$
proof
  assume $\exists x.\ \forall y.\ R(x, y)$
  then show $\forall y.\ \exists x.\ R(x, y)$
  proof
    fix x **assume** a: $\forall y.\ R(x, y)$
    show *?thesis*
    proof
      fix y **from** a **have** $R(x, y)$ **..**
      then show $\exists x.\ R(x, y)$ **..**
    qed
  qed
qed

end

2 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*
arities *nat* :: *term*

consts
Zero :: *nat* (*0*)
Suc :: *nat* => *nat*

```
  rec :: [nat, 'a, [nat, 'a] => 'a] => 'a

axioms
  induct [case-names 0 Suc, induct type: nat]:
    P(0) ==> (!!x. P(x) ==> P(Suc(x))) ==> P(n)
  Suc-inject: Suc(m) = Suc(n) ==> m = n
  Suc-neq-0: Suc(m) = 0 ==> R
  rec-0: rec(0, a, f) = a
  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
    thus False by (rule Suc-neq-0)
  qed
  fix n assume hyp: Suc(n) ≠ n
  show Suc(Suc(n)) ≠ Suc(n)
  proof
    assume Suc(Suc(n)) = Suc(n)
    hence Suc(n) = n by (rule Suc-inject)
    with hyp show False by contradiction
  qed
qed


constdefs
  add :: [nat, nat] => nat    (infixl + 60)
  m + n == rec(m, n, λx y. Suc(y))

lemma add-0 [simp]: 0 + n = n
  by (unfold add-def) (rule rec-0)

lemma add-Suc [simp]: Suc(m) + n = Suc(m + n)
  by (unfold add-def) (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 (!!n. f(Suc(n)) = Suc(f(n))) ==> f(i + j) = i + f(j)
proof −
  assume !!n. f(Suc(n)) = Suc(f(n))
```

thus *?thesis* **by** (*induct i*) *simp-all*
qed

end

3 Examples for the manual "Introduction to Isabelle"

theory *Intro*
imports *FOL*
begin

3.0.1 Some simple backward proofs

lemma *mythm*: *P|P --> P*
apply (*rule impI*)
apply (*rule disjE*)
prefer *3* **apply** (*assumption*)
prefer *2* **apply** (*assumption*)
apply *assumption*
done

lemma (*P* & *Q*) | *R --> (P | R)*
apply (*rule impI*)
apply (*erule disjE*)
apply (*drule conjunct1*)
apply (*rule disjI1*)
apply (*rule-tac* [*2*] *disjI2*)
apply *assumption+*
done

lemma (*ALL x y. P(x,y)*) *-->* (*ALL z w. P(w,z)*)
apply (*rule impI*)
apply (*rule allI*)
apply (*rule allI*)
apply (*drule spec*)
apply (*drule spec*)
apply *assumption*
done

3.0.2 Demonstration of *fast*

lemma (*EX y. ALL x. J(y,x) <-> ~J(x,x)*)
    --> ~ (ALL x. EX y. ALL z. J(z,y) <-> ~ J(z,x))
apply *fast*
done

lemma *ALL x. P(x,f(x)) <−>*
        (EX y. (ALL z. P(z,y) −−> P(z,f(x))) & P(x,y))
apply *fast*
done

3.0.3 Derivation of conjunction elimination rule

lemma
  assumes *major*: *P&Q*
    and *minor*: [| *P*; *Q* |] ==> *R*
  shows *R*
apply (*rule minor*)
apply (*rule major* [*THEN conjunct1*])
apply (*rule major* [*THEN conjunct2*])
done

3.1 Derived rules involving definitions

Derivation of negation introduction

lemma
  assumes *P* ==> *False*
  shows ~ *P*
apply (*unfold not-def*)
apply (*rule impI*)
apply (*rule prems*)
apply *assumption*
done

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

end

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

theory *Nat*
imports *FOL*
begin

typedecl *nat*
arities *nat* :: *term*

consts
  0 :: *nat*   (*0*)
  Suc :: *nat* => *nat*
  rec :: [*nat*, '*a*, [*nat*,'*a*]=>'*a*] => '*a*
  add :: [*nat*, *nat*] => *nat*   (**infixl** + *60*)

axioms
  induct: [| *P*(*0*); !!*x*. *P*(*x*) ==> *P*(*Suc*(*x*)) |] ==> *P*(*n*)
  Suc-inject: *Suc*(*m*)=*Suc*(*n*) ==> *m*=*n*
  Suc-neq-0: *Suc*(*m*)=*0*   ==> *R*
  rec-0:   *rec*(*0*,*a*,*f*) = *a*
  rec-Suc: *rec*(*Suc*(*m*), *a*, *f*) = *f*(*m*, *rec*(*m*,*a*,*f*))

defs
  add-def: *m*+*n* == *rec*(*m*, *n*, %*x* *y*. *Suc*(*y*))

4.1 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 = k* **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-0-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*: !!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

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

theory *Foundation*
imports *IFOL*
begin

lemma $A\&B$ −−> $(C$−−>$A\&C)$
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 & B
    and A ==> B ==> C
  shows C
apply (*rule prems*)
apply (*rule conjunct1*)
apply (*rule prems*)
apply (*rule conjunct2*)
apply (*rule prems*)
done

lemma
  assumes !!A. ~ ~A ==> A
  shows B | ~B
apply (*rule prems*)
apply (*rule notI*)
apply (*rule-tac* P = ~B **in** *notE*)
apply (*rule-tac* [*2*] *notI*)
apply (*rule-tac* [*2*] P = B | ~B **in** *notE*)
prefer *2* **apply** *assumption*
apply (*rule-tac* [*2*] *disjI1*)
prefer *2* **apply** *assumption*
apply (*rule notI*)
apply (*rule-tac* P = B | ~B **in** *notE*)
apply *assumption*
apply (*rule disjI2*)
apply *assumption*
done

lemma
  assumes !!A. ~ ~A ==> A
  shows B | ~B
apply (*rule prems*)
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 A | ~A
    and ~ ~A
  shows A
apply (rule disjE)
apply (rule prems)
apply assumption
apply (rule FalseE)
apply (rule-tac P = ~A in notE)
apply (rule prems)
apply assumption
done
```

5.1 Examples with quantifiers

```
lemma
  assumes ALL z. G(z)
  shows ALL z. G(z)|H(z)
apply (rule allI)
apply (rule disjI1)
apply (rule prems [THEN spec])
done

lemma ALL x. EX y. x=y
apply (rule allI)
apply (rule exI)
apply (rule refl)
done

lemma EX y. ALL x. x=y
apply (rule exI)
apply (rule allI)
apply (rule refl)?
oops
```

Parallel lifting example.

```
lemma EX u. ALL x. EX v. ALL y. EX 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 (EX z. F(z)) & B
  shows EX z. F(z) & B
apply (rule conjE)
apply (rule prems)
```

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 (*EX y. ALL x. Q*(*x*,*y*)) *−−>* (*ALL x. EX 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*
arities *list* :: (*term*) *term*
consts
  Nil :: *'a list*
  Cons :: [*'a*, *'a list*]=> *'a list* (**infixr** : *60*)
  app :: [*'a list*, *'a list*, *'a list*] => *o*
  rev :: [*'a list*, *'a list*] => *o*
axioms
  appNil: *app*(*Nil*,*ys*,*ys*)
  appCons: *app*(*xs*,*ys*,*zs*) ==> *app*(*x*:*xs*, *ys*, *x*:*zs*)
  revNil: *rev*(*Nil*,*Nil*)
  revCons: [| *rev*(*xs*,*ys*); *app*(*ys*, *x*:*Nil*, *zs*) |] ==> *rev*(*x*:*xs*, *zs*)

lemma *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

lemma *app*(*?x*, *c*:*d*:*Nil*, *a*:*b*:*c*:*d*:*Nil*)
apply (*rule appNil appCons*)+

done

lemma *app*(*?x*, *?y*, *a*:*b*:*c*:*d*:*Nil*)
apply (*rule appNil appCons*)+
back
back
back
back
done

lemmas *rules* = *appNil appCons revNil revCons*

lemma *rev*(*a*:*b*:*c*:*d*:*Nil*, *?x*)
apply (*rule rules*)+
done

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

lemma *rev*(*?x*, *a*:*b*:*c*:*Nil*)
apply (*rule rules*)+ — does not solve it directly!
back
back
done

ML ⟪
val prolog-tac = *DEPTH-FIRST* (*has-fewer-prems 1*) (*resolve-tac* (*thms rules*) *1*)
⟫

lemma *rev*(*?x*, *a*:*b*:*c*:*Nil*)
apply (*tactic prolog-tac*)
done

lemma *rev*(*a*:*?x*:*c*:*?y*:*Nil*, *d*:*?z*:*b*:*?u*)
apply (*tactic prolog-tac*)
done

lemma *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* ([*refl*, *conjI*] @ *thms rules*) *1*) ⟫)
done

lemma *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* ([*refl*, *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 conjuction 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 $Q \rightarrow P$ is provable intuitionistically, so $\neg\neg(Q \rightarrow P)$ is also provable intuitionistically. The latter is intuitionistically equivalent to $\neg\neg Q \rightarrow \neg\neg P$, 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 ~~(*P*&*Q*) <–> ~~*P* & ~~*Q*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~ ((~*P* ––> *Q*) ––> (~*P* ––> ~*Q*) ––> *P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Double-negation does NOT distribute over disjunction

lemma ~~(*P*––>*Q*) <–> (~~*P* ––> ~~*Q*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~~*P* <–> ~*P*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ~~((*P* ––> *Q* | *R*) ––> (*P*––>*Q*) | (*P*––>*R*))
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma (*P*<–>*Q*) <–> (*Q*<–>*P*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma ((*P* ––> (*Q* | (*Q*––>*R*))) ––> *R*) ––> *R*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

lemma (((*G*––>*A*) ––> *J*) ––> *D* ––> *E*) ––> (((*H*––>*B*)––>*I*)––>*C*––>*J*)
––> (*A*––>*H*) ––> *F* ––> *G* ––> (((*C*––>*B*)––>*I*)––>*D*)––>(*A*––>*C*)
––> (((*F*––>*A*)––>*B*) ––> *I*) ––> *E*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

Lemmas for the propositional double-negation translation

lemma $P \dashrightarrow$ ~~P
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

lemma ~~(~~P $-->$ P)
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

lemma ~~P & ~~(P $-->$ Q) $-->$ ~~Q
by (*tactic*⟨⟨*IntPr.fast-tac 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 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 1*⟩⟩ | −)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

7.1 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 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 1*⟩⟩)

Problem 1.1

lemma (*ALL x. EX y. ALL z.* $p(x)$ & $q(y)$ & $r(z)$) <->
(*ALL z. EX y. ALL x.* $p(x)$ & $q(y)$ & $r(z)$)
by (*tactic*⟨⟨*IntPr.best-dup-tac 1*⟩⟩) — SLOW

Problem 3.1

lemma ~ (*EX x. ALL y.* *mem*(y,x) <-> ~ *mem*(x,x))
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

Problem 4.1: hopeless!

lemma $(ALL\, x.\ p(x) --> p(h(x)) \mid p(g(x))) \ \&\ (EX\, x.\ p(x)) \ \&\ (ALL\, x.\ {\sim}p(h(x)))$
$--> (EX\ x.\ p(g(g(g(g(g(x)))))))$
oops

7.2 Intuitionistic FOL: propositional problems based on Pelletier.

1

lemma $\sim\sim((P-->Q) \ <-> \ ({\sim}Q --> {\sim}P))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

2

lemma $\sim\sim({\sim}{\sim}P \ <-> \ P)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

3

lemma ${\sim}(P-->Q) --> (Q-->P)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

4

lemma $\sim\sim(({\sim}P-->Q) \ <-> \ ({\sim}Q --> P))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

5

lemma $\sim\sim((P|Q-->P|R) --> P|(Q-->R))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

6

lemma $\sim\sim(P \mid {\sim}P)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

7

lemma $\sim\sim(P \mid {\sim}{\sim}{\sim}P)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

8. Peirce's law

lemma $\sim\sim(((P-->Q) --> P) \ --> \ P)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

9

lemma $((P|Q) \ \&\ ({\sim}P|Q) \ \&\ (P|\ {\sim}Q)) --> {\sim}\ ({\sim}P \mid {\sim}Q)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

10

lemma $(Q-->R) --> (R-->P\&Q) --> (P-->(Q|R)) --> (P<->Q)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

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

lemma $P{<}{-}{>}P$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

12. Dijkstra's law

lemma $\sim\sim(((P <-> Q) <-> R)\ \ <->\ \ (P <-> (Q <-> R)))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

lemma $((P <-> Q) <-> R)\ \ -->\ \ \sim\sim(P <-> (Q <-> R))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

13. Distributive law

lemma $P \mid (Q \ \&\ R)\ \ <-> (P \mid Q) \ \&\ (P \mid R)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

14

lemma $\sim\sim((P <-> Q) <-> ((Q \mid \sim P) \ \&\ (\sim Q|P)))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

15

lemma $\sim\sim((P --> Q) <-> (\sim P \mid Q))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

16

lemma $\sim\sim((P{-}{-}{>}Q) \mid (Q{-}{-}{>}P))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

17

lemma $\sim\sim(((P \ \&\ (Q{-}{-}{>}R)){-}{-}{>}S) <-> ((\sim P \mid Q \mid S) \ \&\ (\sim P \mid \sim R \mid S)))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

Dijkstra's "Golden Rule"

lemma $(P\&Q) <-> P <-> Q <-> (P|Q)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

7.4 ****Examples with quantifiers****

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

lemma $(EX\ x.\ P(x){-}{-}{>}Q)\ \ -->\ \ (ALL\ x.\ P(x)) --> Q$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

lemma $((ALL\ x.\ P(x)){-}{-}{>}Q) --> \sim\ (ALL\ x.\ P(x) \ \&\ \sim Q)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

lemma $((ALL\ x.\ \sim P(x)){-}{-}{>}Q)\ \ -->\ \ \sim\ (ALL\ x.\ \sim\ (P(x)|Q))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

lemma $(ALL\ x.\ P(x)) \mid Q\ \ {-}{-}{>}\ \ (ALL\ x.\ P(x) \mid Q)$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

lemma $(EX\ x.\ P\ {-}{-}{>}\ Q(x))\ {-}{-}{>}\ (P\ {-}{-}{>}\ (EX\ x.\ Q(x)))$
by $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle)$

7.6 The following are not constructively valid!

The attempt to prove them terminates quickly!

lemma $((ALL\ x.\ P(x)){-}{-}{>}Q)\ {-}{-}{>}\ (EX\ x.\ P(x){-}{-}{>}Q)$
apply $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle \mid -)$
apply $(rule\ asm\text{-}rl)$ — Checks that subgoals remain: proof failed.
oops

lemma $(P\ {-}{-}{>}\ (EX\ x.\ Q(x)))\ {-}{-}{>}\ (EX\ x.\ P{-}{-}{>}Q(x))$
apply $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle \mid -)$
apply $(rule\ asm\text{-}rl)$ — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ P(x) \mid Q)\ {-}{-}{>}\ ((ALL\ x.\ P(x)) \mid Q)$
apply $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle \mid -)$
apply $(rule\ asm\text{-}rl)$ — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ {\sim}{\sim}P(x))\ {-}{-}{>}\ {\sim}{\sim}(ALL\ x.\ P(x))$
apply $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle \mid -)$
apply $(rule\ asm\text{-}rl)$ — Checks that subgoals remain: proof failed.
oops

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

lemma $EX\ x.\ Q(x)\ {-}{-}{>}\ (ALL\ x.\ Q(x))$
apply $(tactic\langle\langle IntPr.fast\text{-}tac\ 1\rangle\rangle \mid -)$
apply $(rule\ asm\text{-}rl)$ — Checks that subgoals remain: proof failed.
oops

7.7 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 ${\sim}{\sim}(EX\ y.\ ALL\ x.\ P(y){-}{-}{>}P(x))$
oops — NOT PROVED

19

lemma ${\sim}{\sim}(EX\ x.\ ALL\ y\ z.\ (P(y){-}{-}{>}Q(z))\ {-}{-}{>}\ (P(x){-}{-}{>}Q(x)))$
oops — NOT PROVED

20

lemma $(ALL\ x\ y.\ EX\ z.\ ALL\ w.\ (P(x)\&Q(y){-}{-}{>}R(z)\&S(w)))$
$\quad {-}{-}{>}\ (EX\ x\ y.\ P(x)\ \&\ Q(y))\ {-}{-}{>}\ (EX\ z.\ R(z))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

21

lemma $(EX\ x.\ P{-}{-}{>}Q(x))\ \&\ (EX\ x.\ Q(x){-}{-}{>}P)\ {-}{-}{>}\ {\sim}{\sim}(EX\ x.\ P{<}{-}{>}Q(x))$
oops — NOT PROVED; needs quantifier duplication

22

lemma $(ALL\ x.\ P\ {<}{-}{>}\ Q(x))\ \ {-}{-}{>}\ \ (P\ {<}{-}{>}\ (ALL\ x.\ Q(x)))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

23

lemma ${\sim}{\sim}\ ((ALL\ x.\ P\ |\ Q(x))\ \ {<}{-}{>}\ \ (P\ |\ (ALL\ x.\ Q(x))))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

24

lemma ${\sim}(EX\ x.\ S(x)\&Q(x))\ \&\ (ALL\ x.\ P(x)\ {-}{-}{>}\ Q(x)|R(x))\ \&$
$\quad ({\sim}(EX\ x.\ P(x))\ {-}{-}{>}\ (EX\ x.\ Q(x)))\ \&\ (ALL\ x.\ Q(x)|R(x)\ {-}{-}{>}\ S(x))$
$\quad {-}{-}{>}\ {\sim}{\sim}(EX\ x.\ P(x)\&R(x))$

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

apply (*tactic*⟨⟨ *IntPr.safe-tac*⟩⟩)
apply (*erule impE*)
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

25

lemma $(EX\ x.\ P(x))\ \&$
$\quad (ALL\ x.\ L(x)\ {-}{-}{>}\ {\sim}\ (M(x)\ \&\ R(x)))\ \&$
$\quad (ALL\ x.\ P(x)\ {-}{-}{>}\ (M(x)\ \&\ L(x)))\ \&$
$\quad ((ALL\ x.\ P(x){-}{-}{>}Q(x))\ |\ (EX\ x.\ P(x)\&R(x)))$
$\quad {-}{-}{>}\ (EX\ x.\ Q(x)\&P(x))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

26

lemma $({\sim}{\sim}(EX\ x.\ p(x))\ {<}{-}{>}\ {\sim}{\sim}(EX\ x.\ q(x)))\ \&$
$\quad (ALL\ x.\ ALL\ y.\ p(x)\ \&\ q(y)\ {-}{-}{>}\ (r(x)\ {<}{-}{>}\ s(y)))$
$\quad {-}{-}{>}\ ((ALL\ x.\ p(x){-}{-}{>}r(x))\ {<}{-}{>}\ (ALL\ x.\ q(x){-}{-}{>}s(x)))$
oops — NOT PROVED

27

lemma $(EX\ x.\ P(x)\ \&\ {\sim}Q(x))\ \&$
$\quad (ALL\ x.\ P(x)\ {-}{-}{>}\ R(x))\ \&$
$\quad (ALL\ x.\ M(x)\ \&\ L(x)\ {-}{-}{>}\ P(x))\ \&$
$\quad ((EX\ x.\ R(x)\ \&\ {\sim}\ Q(x))\ {-}{-}{>}\ (ALL\ x.\ L(x)\ {-}{-}{>}\ {\sim}\ R(x)))$

```
         --> (ALL x. M(x) --> ~L(x))
by (tactic⟪IntPr.fast-tac 1⟫)
```

28. AMENDED

```
lemma (ALL x. P(x) --> (ALL x. Q(x))) &
       (~~(ALL x. Q(x)|R(x)) --> (EX x. Q(x)&S(x))) &
       (~~(EX x. S(x)) --> (ALL x. L(x) --> M(x)))
   --> (ALL x. P(x) & L(x) --> M(x))
by (tactic⟪IntPr.fast-tac 1⟫)
```

29. Essentially the same as Principia Mathematica *11.71

```
lemma (EX x. P(x)) & (EX y. Q(y))
   --> ((ALL x. P(x)-->R(x)) & (ALL y. Q(y)-->S(y))   <->
        (ALL x y. P(x) & Q(y) --> R(x) & S(y)))
by (tactic⟪IntPr.fast-tac 1⟫)
```

30

```
lemma (ALL x. (P(x) | Q(x)) --> ~ R(x)) &
       (ALL x. (Q(x) --> ~ S(x)) --> P(x) & R(x))
   --> (ALL x. ~~S(x))
by (tactic⟪IntPr.fast-tac 1⟫)
```

31

```
lemma ~(EX x. P(x) & (Q(x) | R(x))) &
       (EX x. L(x) & P(x)) &
       (ALL x. ~ R(x) --> M(x))
   --> (EX x. L(x) & M(x))
by (tactic⟪IntPr.fast-tac 1⟫)
```

32

```
lemma (ALL x. P(x) & (Q(x)|R(x))-->S(x)) &
       (ALL x. S(x) & R(x) --> L(x)) &
       (ALL x. M(x) --> R(x))
   --> (ALL x. P(x) & M(x) --> L(x))
by (tactic⟪IntPr.fast-tac 1⟫)
```

33

```
lemma (ALL x. ~~(P(a) & (P(x)-->P(b))-->P(c)))  <->
      (ALL x. ~~((~P(a) | P(x) | P(c)) & (~P(a) | ~P(b) | P(c))))
apply (tactic⟪IntPr.best-tac 1⟫)
done
```

36

```
lemma (ALL x. EX y. J(x,y)) &
      (ALL x. EX y. G(x,y)) &
      (ALL x y. J(x,y) | G(x,y) --> (ALL z. J(y,z) | G(y,z) --> H(x,z)))
  --> (ALL x. EX y. H(x,y))
by (tactic⟪IntPr.fast-tac 1⟫)
```

37

lemma (*ALL z. EX w. ALL x. EX y.*
~~(P(x,z)-->P(y,w)) & P(y,z) & (P(y,w) --> (EX u. Q(u,w)))) &
(ALL x z. ~P(x,z) --> (EX y. Q(y,z))) &
(~~(EX x y. Q(x,y)) --> (ALL x. R(x,x)))
--> ~~(ALL x. EX y. R(x,y))
oops — NOT PROVED

39

lemma ~ (*EX x. ALL y. F(y,x) <-> ~F(y,y)*)
by (*tactic⟪IntPr.fast-tac 1⟫*)

40. AMENDED

lemma (*EX y. ALL x. F(x,y) <-> F(x,x)) -->*
~(ALL x. EX y. ALL z. F(z,y) <-> ~ F(z,x))
by (*tactic⟪IntPr.fast-tac 1⟫*)

44

lemma (*ALL x. f(x) -->*
(EX y. g(y) & h(x,y) & (EX y. g(y) & ~ h(x,y)))) &
(EX x. j(x) & (ALL y. g(y) --> h(x,y)))
--> (EX x. j(x) & ~f(x))
by (*tactic⟪IntPr.fast-tac 1⟫*)

48

lemma (*a=b | c=d) & (a=c | b=d) --> a=d | b=c*
by (*tactic⟪IntPr.fast-tac 1⟫*)

51

lemma (*EX z w. ALL x y. P(x,y) <-> (x=z & y=w)) -->*
(EX z. ALL x. EX w. (ALL y. P(x,y) <-> y=w) <-> x=z)
by (*tactic⟪IntPr.fast-tac 1⟫*)

52

Almost the same as 51.

lemma (*EX z w. ALL x y. P(x,y) <-> (x=z & y=w)) -->*
(EX w. ALL y. EX z. (ALL x. P(x,y) <-> x=z) <-> y=w)
by (*tactic⟪IntPr.fast-tac 1⟫*)

56

lemma (*ALL x. (EX y. P(y) & x=f(y)) --> P(x)) <-> (ALL x. P(x) --> P(f(x)))*
by (*tactic⟪IntPr.fast-tac 1⟫*)

57

lemma *P(f(a,b), f(b,c)) & P(f(b,c), f(a,c)) &*

(*ALL x y z. P(x,y) & P(y,z) --> P(x,z)) --> P(f(a,b), f(a,c)*)
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

60

lemma *ALL x. P(x,f(x)) <-> (EX y. (ALL z. P(z,y) --> P(z,f(x))) & P(x,y))*
by (*tactic*⟪*IntPr.fast-tac 1*⟫)

end

8 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 1*)

lemma *P | Q --> Q | P*
by (*tactic IntPr.fast-tac 1*)

associative laws of & and |

lemma *(P & Q) & R --> P & (Q & R)*
by (*tactic IntPr.fast-tac 1*)

lemma *(P | Q) | R --> P | (Q | R)*
by (*tactic IntPr.fast-tac 1*)

distributive laws of & and |

lemma *(P & Q) | R --> (P | R) & (Q | R)*
by (*tactic IntPr.fast-tac 1*)

lemma *(P | R) & (Q | R) --> (P & Q) | R*
by (*tactic IntPr.fast-tac 1*)

lemma *(P | Q) & R --> (P & R) | (Q & R)*
by (*tactic IntPr.fast-tac 1*)

lemma *(P & R) | (Q & R) --> (P | Q) & R*
by (*tactic IntPr.fast-tac 1*)

Laws involving implication

lemma $(P-->R)$ & $(Q-->R)$ $<->$ $(P|Q$ $-->$ $R)$
  by (*tactic IntPr.fast-tac 1*)

lemma $(P$ & Q $-->$ $R)$ $<->$ $(P-->$ $(Q-->R))$
  by (*tactic IntPr.fast-tac 1*)

lemma $((P-->R)-->R)$ $-->$ $((Q-->R)-->R)$ $-->$ $(P\&Q-->R)$ $-->$
R
  by (*tactic IntPr.fast-tac 1*)

lemma $\sim(P-->R)$ $-->$ $\sim(Q-->R)$ $-->$ $\sim(P\&Q-->R)$
  by (*tactic IntPr.fast-tac 1*)

lemma $(P$ $-->$ Q & $R)$ $<->$ $(P-->Q)$ & $(P-->R)$
  by (*tactic IntPr.fast-tac 1*)

Propositions-as-types

— The combinator K
lemma P $-->$ $(Q$ $-->$ $P)$
  by (*tactic IntPr.fast-tac 1*)

— The combinator S
lemma $(P-->Q-->R)$ $-->$ $(P-->Q)$ $-->$ $(P-->R)$
  by (*tactic IntPr.fast-tac 1*)

— Converse is classical
lemma $(P-->Q)$ | $(P-->R)$ $-->$ $(P$ $-->$ Q | $R)$
  by (*tactic IntPr.fast-tac 1*)

lemma $(P-->Q)$ $-->$ $(\sim Q$ $-->$ $\sim P)$
  by (*tactic IntPr.fast-tac 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 1*)

lemma *stab-to-peirce*:
  $(((P$ $-->$ $R)$ $-->$ $R)$ $-->$ $P)$ $-->$ $(((Q$ $-->$ $R)$ $-->$ $R)$ $-->$ $Q)$
    $-->$ $((P$ $-->$ $Q)$ $-->$ $P)$ $-->$ P
  by (*tactic IntPr.fast-tac 1*)

lemma *peirce-imp1*: $(((Q$ $-->$ $R)$ $-->$ $Q)$ $-->$ $Q)$
    $-->$ $(((P$ $-->$ $Q)$ $-->$ $R)$ $-->$ P $-->$ $Q)$ $-->$ P $-->$ Q
  by (*tactic IntPr.fast-tac 1*)

lemma *peirce-imp2*: $(((P$ $-->$ $R)$ $-->$ $P)$ $-->$ $P)$ $-->$ $((P$ $-->$ Q $-->$
$R)$ $-->$ $P)$ $-->$ P
  by (*tactic IntPr.fast-tac 1*)

lemma *mints*: ((((P --> Q) --> P) --> P) --> Q) --> Q
  by (*tactic IntPr.fast-tac 1*)

lemma *mints-solovev*: (P --> (Q --> R) --> Q) --> ((P --> Q) --> R) --> R
  by (*tactic IntPr.fast-tac 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 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 1*)

end

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

theory *Quantifiers-Int*
imports *IFOL*
begin

lemma (ALL x y. P(x,y)) --> (ALL y x. P(x,y))
  by (*tactic IntPr.fast-tac 1*)

lemma (EX x y. P(x,y)) --> (EX y x. P(x,y))
  by (*tactic IntPr.fast-tac 1*)

— Converse is false
lemma (ALL x. P(x)) | (ALL x. Q(x)) --> (ALL x. P(x) | Q(x))
  by (*tactic IntPr.fast-tac 1*)

lemma (ALL x. P-->Q(x)) <-> (P--> (ALL x. Q(x)))
  by (*tactic IntPr.fast-tac 1*)

lemma (ALL x. P(x)-->Q) <-> ((EX x. P(x)) --> Q)

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

Some harder ones

lemma (*EX x. P(x) | Q(x)*) *<−>* (*EX x. P(x)*) | (*EX x. Q(x)*)
by (*tactic IntPr.fast-tac 1*)

— Converse is false
lemma (*EX x. P(x)&Q(x)*) *−−>* (*EX x. P(x)*) & (*EX x. Q(x)*)
by (*tactic IntPr.fast-tac 1*)

Basic test of quantifier reasoning

— TRUE
lemma (*EX y. ALL x. Q(x,y)*) *−−>* (*ALL x. EX y. Q(x,y)*)
by (*tactic IntPr.fast-tac 1*)

lemma (*ALL x. Q(x)*) *−−>* (*EX x. Q(x)*)
by (*tactic IntPr.fast-tac 1*)

The following should fail, as they are false!

lemma (*ALL x. EX y. Q(x,y)*) *−−>* (*EX y. ALL x. Q(x,y)*)
apply (*tactic IntPr.fast-tac 1*)*?*
oops

lemma (*EX x. Q(x)*) *−−>* (*ALL x. Q(x)*)
apply (*tactic IntPr.fast-tac 1*)*?*
oops

lemma *P(?a)* *−−>* (*ALL x. P(x)*)
apply (*tactic IntPr.fast-tac 1*)*?*
oops

lemma (*P(?a)* *−−>* (*ALL x. Q(x)*)) *−−>* (*ALL x. P(x)* *−−>* *Q(x)*)
apply (*tactic IntPr.fast-tac 1*)*?*
oops

Back to things that are provable . . .

lemma (*ALL x. P(x)−−>Q(x)*) & (*EX x. P(x)*) *−−>* (*EX x. Q(x)*)
by (*tactic IntPr.fast-tac 1*)

— An example of why exI should be delayed as long as possible
lemma (*P* *−−>* (*EX x. Q(x)*)) & *P* *−−>* (*EX x. Q(x)*)
by (*tactic IntPr.fast-tac 1*)

lemma (*ALL x. P(x)−−>Q(f(x))*) & (*ALL x. Q(x)−−>R(g(x))*) & *P(d)* *−−>* *R(?a)*
by (*tactic IntPr.fast-tac 1*)

lemma (*ALL x. Q(x)*) *−−>* (*EX x. Q(x)*)

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

Some slow ones

— Principia Mathematica *11.53
lemma (*ALL x y. P(x) −−> Q(y)*) <−> ((*EX x. P(x)*) −−> (*ALL y. Q(y)*))
by (*tactic IntPr.fast-tac 1*)

lemma (*EX x y. P(x) & Q(x,y)*) <−> (*EX x. P(x) &* (*EX y. Q(x,y)*))
by (*tactic IntPr.fast-tac 1*)

lemma (*EX y. ALL x. P(x) −−> Q(x,y)*) −−> (*ALL x. P(x) −−>* (*EX y. Q(x,y)*))
by (*tactic IntPr.fast-tac 1*)

end

10 Classical Predicate Calculus Problems

theory *Classical* **imports** *FOL* **begin**

lemma (*P −−> Q | R*) −−> (*P−−>Q*) | (*P−−>R*)
by *blast*

If and only if

lemma (*P<−>Q*) <−> (*Q<−>P*)
by *blast*

lemma ~ (*P* <−> ~*P*)
by *blast*

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.

10.1 Pelletier's examples

1

lemma (*P−−>Q*) <−> (~*Q* −−> ~*P*)
by *blast*

2

lemma ~ ~ *P* <−> *P*

by *blast*

3

lemma $\sim(P-->Q)\ -->\ (Q-->P)$
by *blast*

4

lemma $(\sim P-->Q)\ \ <->\ \ (\sim Q\ -->\ P)$
by *blast*

5

lemma $((P|Q)-->(P|R))\ -->\ (P|(Q-->R))$
by *blast*

6

lemma $P\ |\ \sim\ P$
by *blast*

7

lemma $P\ |\ \sim\ \sim\ \sim\ P$
by *blast*

8. Peirce's law

lemma $((P-->Q)\ -->\ P)\ \ -->\ \ P$
by *blast*

9

lemma $((P|Q)\ \&\ (\sim P|Q)\ \&\ (P|\ \sim Q))\ -->\ \sim\ (\sim P\ |\ \sim Q)$
by *blast*

10

lemma $(Q-->R)\ \&\ (R-->P\&Q)\ \&\ (P-->Q|R)\ -->\ (P<->Q)$
by *blast*

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

lemma $P<->P$
by *blast*

12. "Dijkstra's law"

lemma $((P\ <->\ Q)\ <->\ R)\ \ <->\ \ (P\ <->\ (Q\ <->\ R))$
by *blast*

13. Distributive law

lemma $P\ |\ (Q\ \&\ R)\ \ <->\ (P\ |\ Q)\ \&\ (P\ |\ R)$
by *blast*

14

lemma $(P <-> Q) <-> ((Q \mid {\sim}P) \& ({\sim}Q|P))$
by *blast*

15

lemma $(P --> Q) <-> ({\sim}P \mid Q)$
by *blast*

16

lemma $(P-->Q) \mid (Q-->P)$
by *blast*

17

lemma $((P \& (Q-->R))-->S) <-> (({\sim}P \mid Q \mid S) \& ({\sim}P \mid {\sim}R \mid S))$
by *blast*

10.2 Classical Logic: examples with quantifiers

lemma $(\forall x. P(x) \& Q(x)) <-> (\forall x. P(x)) \ \& \ (\forall x. Q(x))$
by *blast*

lemma $(\exists x. P-->Q(x)) \ <-> \ (P --> (\exists x. Q(x)))$
by *blast*

lemma $(\exists x. P(x)-->Q) \ <-> \ (\forall x. P(x)) --> Q$
by *blast*

lemma $(\forall x. P(x)) \mid Q \ <-> \ (\forall x. P(x) \mid Q)$
by *blast*

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

lemma ${\sim}((\exists x.{\sim}P(x)) \& ((\exists x. P(x)) \mid (\exists x. P(x) \& Q(x))) \& {\sim} (\exists x. P(x)))$
by *blast*

10.3 Problems requiring quantifier duplication

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

lemma $(\exists x. \forall y. P(x) <-> P(y)) --> ((\exists x. P(x)) <-> (\forall y. P(y)))$
by *blast*

Needs multiple instantiation of ALL.

lemma $(\forall x. P(x)-->P(f(x))) \ \& \ P(d)-->P(f(f(f(d))))$
by *blast*

Needs double instantiation of the quantifier

lemma $\exists x. P(x) --> P(a) \& P(b)$

by *blast*

lemma $\exists z.\ P(z)\ \texttt{-->}\ (\forall x.\ P(x))$
by *blast*

lemma $\exists x.\ (\exists y.\ P(y))\ \texttt{-->}\ P(x)$
by *blast*

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

lemma $\exists x\ x'.\ \forall y.\ \exists z\ z'.$
$(\sim P(y,y)\ |\ P(x,x)\ |\ \sim S(z,x))\ \&$
$(S(x,y)\ |\ \sim S(y,z)\ |\ Q(z',z'))\ \&$
$(Q(x',y)\ |\ \sim Q(y,z')\ |\ S(x',x'))$
oops

10.4 Hard examples with quantifiers

18

lemma $\exists y.\ \forall x.\ P(y)\texttt{-->}P(x)$
by *blast*

19

lemma $\exists x.\ \forall y\ z.\ (P(y)\texttt{-->}Q(z))\ \texttt{-->}\ (P(x)\texttt{-->}Q(x))$
by *blast*

20

lemma $(\forall x\ y.\ \exists z.\ \forall w.\ (P(x)\&Q(y)\texttt{-->}R(z)\&S(w)))$
$\texttt{-->}\ (\exists x\ y.\ P(x)\ \&\ Q(y))\ \texttt{-->}\ (\exists z.\ R(z))$
by *blast*

21

lemma $(\exists x.\ P\texttt{-->}Q(x))\ \&\ (\exists x.\ Q(x)\texttt{-->}P)\ \texttt{-->}\ (\exists x.\ P\texttt{<->}Q(x))$
by *blast*

22

lemma $(\forall x.\ P\ \texttt{<->}\ Q(x))\ \ \texttt{-->}\ \ (P\ \texttt{<->}\ (\forall x.\ Q(x)))$
by *blast*

23

lemma $(\forall x.\ P\ |\ Q(x))\ \ \texttt{<->}\ \ (P\ |\ (\forall x.\ Q(x)))$
by *blast*

24

lemma $\sim(\exists x.\ S(x)\&Q(x))\ \&\ (\forall x.\ P(x)\ \texttt{-->}\ Q(x)|R(x))\ \&$
$(\sim(\exists x.\ P(x))\ \texttt{-->}\ (\exists x.\ Q(x)))\ \&\ (\forall x.\ Q(x)|R(x)\ \texttt{-->}\ S(x))$
$\texttt{-->}\ (\exists x.\ P(x)\&R(x))$

```
by blast
```

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 blast
```

26

```
lemma ((∃x. p(x)) <-> (∃x. q(x))) &
      (∀x. ∀y. p(x) & q(y) --> (r(x) <-> s(y)))
  --> ((∀x. p(x)-->r(x)) <-> (∀x. q(x)-->s(x)))
by blast
```

27

```
lemma (∃x. P(x) & ~Q(x)) &
      (∀x. P(x) --> R(x)) &
      (∀x. M(x) & L(x) --> P(x)) &
      ((∃x. R(x) & ~ Q(x)) --> (∀x. L(x) --> ~ R(x)))
  --> (∀x. M(x) --> ~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 $(\forall x.\ P(x)\ \&\ (Q(x)|R(x)) {-}{-}{>} S(x))\ \&$
$(\forall x.\ S(x)\ \&\ R(x)\ {-}{-}{>}\ L(x))\ \&$
$(\forall x.\ M(x)\ {-}{-}{>}\ R(x))$
${-}{-}{>}\ (\forall x.\ P(x)\ \&\ M(x)\ {-}{-}{>}\ L(x))$
by *blast*

33

lemma $(\forall x.\ P(a)\ \&\ (P(x) {-}{-}{>} P(b)) {-}{-}{>} P(c))\ \ {<}{-}{>}$
$(\forall x.\ ({\sim}P(a)\ |\ P(x)\ |\ P(c))\ \&\ ({\sim}P(a)\ |\ {\sim}P(b)\ |\ P(c)))$
by *blast*

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

lemma $((\exists x.\ \forall y.\ p(x)\ {<}{-}{>}\ p(y))\ \ {<}{-}{>}$
$((\exists x.\ q(x))\ {<}{-}{>}\ (\forall y.\ p(y))))\quad {<}{-}{>}$
$((\exists x.\ \forall y.\ q(x)\ {<}{-}{>}\ q(y))\ \ {<}{-}{>}$
$((\exists x.\ p(x))\ {<}{-}{>}\ (\forall y.\ q(y))))$
by *blast*

35

lemma $\exists x\ y.\ P(x,y)\ {-}{-}{>}\ \ (\forall u\ v.\ P(u,v))$
by *blast*

36

lemma $(\forall x.\ \exists y.\ J(x,y))\ \&$
$(\forall x.\ \exists y.\ G(x,y))\ \&$
$(\forall x\ y.\ J(x,y)\ |\ G(x,y)\ {-}{-}{>}\ (\forall z.\ J(y,z)\ |\ G(y,z)\ {-}{-}{>}\ H(x,z)))$
${-}{-}{>}\ (\forall x.\ \exists y.\ H(x,y))$
by *blast*

37

lemma $(\forall z.\ \exists w.\ \forall x.\ \exists y.$
$(P(x,z) {-}{-}{>} P(y,w))\ \&\ P(y,z)\ \&\ (P(y,w)\ {-}{-}{>}\ (\exists u.\ Q(u,w))))\ \&$
$(\forall x\ z.\ {\sim}P(x,z)\ {-}{-}{>}\ (\exists y.\ Q(y,z)))\ \&$
$((\exists x\ y.\ Q(x,y))\ {-}{-}{>}\ (\forall x.\ R(x,x)))$
${-}{-}{>}\ (\forall x.\ \exists y.\ R(x,y))$
by *blast*

38

lemma $(\forall x.\ p(a)\ \&\ (p(x)\ {-}{-}{>}\ (\exists y.\ p(y)\ \&\ r(x,y)))\ {-}{-}{>}$
$(\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z)))\ \ {<}{-}{>}$
$(\forall x.\ ({\sim}p(a)\ |\ p(x)\ |\ (\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z)))\ \&$
$({\sim}p(a)\ |\ {\sim}(\exists y.\ p(y)\ \&\ r(x,y))\ |$
$(\exists z.\ \exists w.\ p(z)\ \&\ r(x,w)\ \&\ r(w,z))))$
by *blast*

39

lemma $\sim (\exists x.\ \forall y.\ F(y,x) <-> \sim F(y,y))$
by *blast*

40. AMENDED

lemma $(\exists y.\ \forall x.\ F(x,y) <-> F(x,x)) -->$
$\quad\quad \sim(\forall x.\ \exists y.\ \forall z.\ F(z,y) <-> \sim F(z,x))$
by *blast*

41

lemma $(\forall z.\ \exists y.\ \forall x.\ f(x,y) <-> f(x,z)\ \&\ \sim f(x,x))$
$\quad --> \sim (\exists z.\ \forall x.\ f(x,z))$
by *blast*

42

lemma $\sim (\exists y.\ \forall x.\ p(x,y) <-> \sim (\exists z.\ p(x,z)\ \&\ p(z,x)))$
by *blast*

43

lemma $(\forall x.\ \forall y.\ q(x,y) <-> (\forall z.\ p(z,x) <-> p(z,y)))$
$\quad --> (\forall x.\ \forall y.\ q(x,y) <-> q(y,x))$
by *blast*

44

lemma $(\forall x.\ f(x) --> (\exists y.\ g(y)\ \&\ h(x,y)\ \&\ (\exists y.\ g(y)\ \&\ \sim h(x,y))))\ \&$
$\quad (\exists x.\ j(x)\ \&\ (\forall y.\ g(y) --> h(x,y)))$
$\quad --> (\exists x.\ j(x)\ \&\ \sim f(x))$
by *blast*

45

lemma $(\forall x.\ f(x)\ \&\ (\forall y.\ g(y)\ \&\ h(x,y) --> j(x,y))$
$\quad\quad\quad --> (\forall y.\ g(y)\ \&\ h(x,y) --> k(y)))\ \&$
$\quad \sim (\exists y.\ l(y)\ \&\ k(y))\ \&$
$\quad (\exists x.\ f(x)\ \&\ (\forall y.\ h(x,y) --> l(y))$
$\quad\quad\quad \&\ (\forall y.\ g(y)\ \&\ h(x,y) --> j(x,y)))$
$\quad --> (\exists x.\ f(x)\ \&\ \sim (\exists y.\ g(y)\ \&\ h(x,y)))$
by *blast*

46

lemma $(\forall x.\ f(x)\ \&\ (\forall y.\ f(y)\ \&\ h(y,x) --> g(y)) --> g(x))\ \&$
$\quad ((\exists x.\ f(x)\ \&\ \sim g(x)) -->$
$\quad (\exists x.\ f(x)\ \&\ \sim g(x)\ \&\ (\forall y.\ f(y)\ \&\ \sim g(y) --> j(x,y))))\ \&$
$\quad (\forall x\ y.\ f(x)\ \&\ f(y)\ \&\ h(x,y) --> \sim j(y,x))$
$\quad --> (\forall x.\ f(x) --> g(x))$
by *blast*

10.5 Problems (mainly) involving equality or functions

48

lemma $(a=b \mid c=d)$ & $(a=c \mid b=d)$ $-->$ $a=d \mid b=c$
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 $(\exists\, x\ y{::}'a.\ \forall\, z.\ z=x \mid z=y)$ & $P(a)$ & $P(b)$ & $a\tilde{}=b$
$\quad -->$ $(\forall\, u{::}'a.\ P(u))$
apply *safe*
apply (*rule-tac* $x = a$ **in** *allE*, *assumption*)
apply (*rule-tac* $x = b$ **in** *allE*, *assumption*, *fast*)
— blast's treatment of equality can't do it
done

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

lemma $(\forall\, x.\ P(a,x) \mid (\forall\, y.\ P(x,y)))$ $-->$ $(\exists\, x.\ \forall\, y.\ P(x,y))$
by *blast*

51

lemma $(\exists\, z\ w.\ \forall\, x\ y.\ P(x,y)$ $<->$ $(x=z$ & $y=w))$ $-->$
$\quad (\exists\, z.\ \forall\, x.\ \exists\, w.\ (\forall\, y.\ P(x,y)$ $<->$ $y=w)$ $<->$ $x=z)$
by *blast*

52

Almost the same as 51.

lemma $(\exists\, z\ w.\ \forall\, x\ y.\ P(x,y)$ $<->$ $(x=z$ & $y=w))$ $-->$
$\quad (\exists\, w.\ \forall\, y.\ \exists\, z.\ (\forall\, x.\ P(x,y)$ $<->$ $x=z)$ $<->$ $y=w)$
by *blast*

55

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

lemma *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 $(\forall x.\ (\exists y.\ P(y)\ \&\ x{=}f(y))\ {-}{-}{>}\ P(x))\ {<}{-}{>}\ (\forall x.\ P(x)\ {-}{-}{>}\ P(f(x)))$
by *blast*

57

lemma $P(f(a,b),\ f(b,c))\ \&\ P(f(b,c),\ f(a,c))\ \&$
$\quad(\forall x\ y\ z.\ P(x,y)\ \&\ P(y,z)\ {-}{-}{>}\ P(x,z))\quad {-}{-}{>}\quad P(f(a,b),\ f(a,c))$
by *blast*

58 NOT PROVED AUTOMATICALLY

lemma $(\forall x\ y.\ f(x){=}g(y))\ {-}{-}{>}\ (\forall x\ y.\ f(f(x)){=}f(g(y)))$
by (*slow elim*: *subst-context*)

59

lemma $(\forall x.\ P(x)\ {<}{-}{>}\ {\sim}P(f(x)))\ {-}{-}{>}\ (\exists x.\ P(x)\ \&\ {\sim}P(f(x)))$
by *blast*

60

lemma $\forall x.\ P(x,f(x))\ {<}{-}{>}\ (\exists y.\ (\forall z.\ P(z,y)\ {-}{-}{>}\ P(z,f(x)))\ \&\ P(x,y))$
by *blast*

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

lemma $(\forall x.\ p(a)\ \&\ (p(x)\ {-}{-}{>}\ p(f(x)))\ {-}{-}{>}\ p(f(f(x))))\ \ {<}{-}{>}$
$\quad(\forall x.\ ({\sim}p(a)\ |\ p(x)\ |\ p(f(f(x))))\ \&$
$\qquad({\sim}p(a)\ |\ {\sim}p(f(x))\ |\ p(f(f(x)))))$
by *blast*

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

lemma $(\forall x.\ F(x)\ \&\ {\sim}G(x)\ {-}{-}{>}\ (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$\qquad(\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y)\ {-}{-}{>}\ K(y)))\ \&$
$\qquad(\forall x.\ K(x)\ {-}{-}{>}\ {\sim}G(x))\ \ {-}{-}{>}\ \ (\exists x.\ K(x)\ \&\ J(x))$
by *fast*

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

lemma $(\forall x.\ F(x)\ \&\ {\sim}G(x)\ {-}{-}{>}\ (\exists y.\ H(x,y)\ \&\ J(y)))\ \&$
$\quad(\exists x.\ K(x)\ \&\ F(x)\ \&\ (\forall y.\ H(x,y)\ {-}{-}{>}\ K(y)))\ \ \&$
$\quad(\forall x.\ K(x)\ {-}{-}{>}\ {\sim}G(x))\quad {-}{-}{>}\quad (\exists x.\ K(x)\ {-}{-}{>}\ {\sim}G(x))$
by *fast*

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

lemma $((\exists x.\ A(x)\ \&\ (\forall y.\ C(y)\ {-}{-}{>}\ (\forall z.\ D(x,y,z))))\ {-}{-}{>}$
$\ (\exists w.\ C(w)\ \&\ (\forall y.\ C(y)\ {-}{-}{>}\ (\forall z.\ D(w,y,z)))))$
$\ \&$

```
  (∀w. C(w) & (∀u. C(u) --> (∀v. D(w,u,v))) -->
        (∀y z.
            (C(y) &  P(y,z) --> Q(w,y,z) & OO(w,g)) &
            (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))))
  &
  (∀w. C(w) &
    (∀y z.
        (C(y) & P(y,z) --> Q(w,y,z) & OO(w,g)) &
        (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))) -->
    (∃v. C(v) &
          (∀y. ((C(y) & Q(w,y,y)) & OO(w,g) --> ~P(v,y)) &
                  ((C(y) & Q(w,y,y)) & OO(w,b) --> P(v,y) & OO(v,b)))))
  -->
  ~ (∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z))))
by (tactic⟪Blast.depth-tac (claset ()) 12 1⟫)
```

— 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 ((∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z)))) -->
   (∃w. C(w) & (∀y. C(y) --> (∀z. D(w,y,z)))))
  &
  (∀w. C(w) & (∀u. C(u) --> (∀v. D(w,u,v))) -->
        (∀y z.
            (C(y) &  P(y,z) --> Q(w,y,z) & OO(w,g)) &
            (C(y) & ~P(y,z) --> Q(w,y,z) & OO(w,b))))
  &
  ((∃w. C(w) & (∀y. (C(y) &  P(y,y) --> Q(w,y,y) & OO(w,g)) &
                    (C(y) & ~P(y,y) --> Q(w,y,y) & OO(w,b))))
   -->
   (∃v. C(v) & (∀y. (C(y) &  P(y,y) --> P(v,y) & OO(v,g)) &
                    (C(y) & ~P(y,y) --> P(v,y) & OO(v,b)))))
  -->
  ((∃v. C(v) & (∀y. (C(y) &  P(y,y) --> P(v,y) & OO(v,g)) &
                    (C(y) & ~P(y,y) --> P(v,y) & OO(v,b))))
   -->
   (∃u. C(u) & (∀y. (C(y) &  P(y,y) --> ~P(u,y)) &
                    (C(y) & ~P(y,y) --> P(u,y) & OO(u,b)))))
   -->
  ~ (∃x. A(x) & (∀y. C(y) --> (∀z. D(x,y,z))))
by blast
```

Challenge found on info-hol

```
lemma ∀x. ∃v w. ∀y z. P(x) & Q(y) --> (P(v) | R(w)) & (R(z) --> Q(v))
by blast
```

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

```
lemma (∀x. honest(x) & industrious(x) --> healthy(x)) &
      ~ (∃x. grocer(x) & healthy(x)) &
```

$(\forall x.\ industrious(x)\ \&\ grocer(x)\ {-}{-}{>}\ honest(x))\ \&$
$(\forall x.\ cyclist(x)\ {-}{-}{>}\ industrious(x))\ \&$
$(\forall x.\ {\sim}healthy(x)\ \&\ cyclist(x)\ {-}{-}{>}\ {\sim}honest(x))$
${-}{-}{>}\ (\forall x.\ grocer(x)\ {-}{-}{>}\ {\sim}cyclist(x))$
by *blast*

end

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

theory *Propositional-Cla*
imports *FOL*
begin

commutative laws of & and |

lemma $P\ \&\ Q\ {-}{-}{>}\ Q\ \&\ P$
by (*tactic IntPr.fast-tac 1*)

lemma $P \mid Q\ {-}{-}{>}\ Q \mid P$
by *fast*

associative laws of & and |

lemma $(P\ \&\ Q)\ \&\ R\ {-}{-}{>}\ P\ \&\ (Q\ \&\ R)$
by *fast*

lemma $(P \mid Q) \mid R\ {-}{-}{>}\ P \mid (Q \mid R)$
by *fast*

distributive laws of & and |

lemma $(P\ \&\ Q) \mid R\ {-}{-}{>} (P \mid R)\ \&\ (Q \mid R)$
by *fast*

lemma $(P \mid R)\ \&\ (Q \mid R)\ {-}{-}{>} (P\ \&\ Q) \mid R$
by *fast*

lemma $(P \mid Q)\ \&\ R\ {-}{-}{>} (P\ \&\ R) \mid (Q\ \&\ R)$
by *fast*

lemma $(P\ \&\ R) \mid (Q\ \&\ R)\ {-}{-}{>} (P \mid Q)\ \&\ R$
by *fast*

Laws involving implication

lemma (*P*-->*R*) & (*Q*-->*R*) <-> (*P*|*Q* --> *R*)
  by *fast*

lemma (*P* & *Q* --> *R*) <-> (*P*--> (*Q*-->*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

— The combinator K
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

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

theory *Quantifiers-Cla*
imports *FOL*
begin

lemma (ALL x y. P(x,y)) --> (ALL y x. P(x,y))
  by *fast*

lemma (EX x y. P(x,y)) --> (EX y x. P(x,y))
  by *fast*

— Converse is false
lemma (ALL x. P(x)) | (ALL x. Q(x)) --> (ALL x. P(x) | Q(x))
  by *fast*

lemma (ALL x. P-->Q(x)) <-> (P--> (ALL x. Q(x)))
  by *fast*

lemma (ALL x. P(x)-->Q) <-> ((EX x. P(x)) --> Q)

by *fast*

Some harder ones

lemma $(EX\ x.\ P(x)\ |\ Q(x))$ <–> $(EX\ x.\ P(x))\ |\ (EX\ x.\ Q(x))$
by *fast*

— Converse is false
lemma $(EX\ x.\ P(x) \& Q(x))$ ––> $(EX\ x.\ P(x))\ \&\ (EX\ x.\ Q(x))$
by *fast*

Basic test of quantifier reasoning

— TRUE
lemma $(EX\ y.\ ALL\ x.\ Q(x,y))$ ––> $(ALL\ x.\ EX\ y.\ Q(x,y))$
by *fast*

lemma $(ALL\ x.\ Q(x))$ ––> $(EX\ x.\ Q(x))$
by *fast*

The following should fail, as they are false!

lemma $(ALL\ x.\ EX\ y.\ Q(x,y))$ ––> $(EX\ y.\ ALL\ x.\ Q(x,y))$
apply *fast?*
oops

lemma $(EX\ x.\ Q(x))$ ––> $(ALL\ x.\ Q(x))$
apply *fast?*
oops

lemma $P(?a)$ ––> $(ALL\ x.\ P(x))$
apply *fast?*
oops

lemma $(P(?a)$ ––> $(ALL\ x.\ Q(x)))$ ––> $(ALL\ x.\ P(x)$ ––> $Q(x))$
apply *fast?*
oops

Back to things that are provable . . .

lemma $(ALL\ x.\ P(x)$––>$Q(x))\ \&\ (EX\ x.\ P(x))$ ––> $(EX\ x.\ Q(x))$
by *fast*

— An example of why exI should be delayed as long as possible
lemma $(P$ ––> $(EX\ x.\ Q(x)))\ \&\ P$ ––> $(EX\ x.\ Q(x))$
by *fast*

lemma $(ALL\ x.\ P(x)$––>$Q(f(x)))\ \&\ (ALL\ x.\ Q(x)$––>$R(g(x)))\ \&\ P(d)$ ––> $R(?a)$
by *fast*

lemma $(ALL\ x.\ Q(x))$ ––> $(EX\ x.\ Q(x))$

by *fast*

Some slow ones

— Principia Mathematica *11.53
lemma (*ALL x y. P(x) --> Q(y)*) <-> ((*EX x. P(x)*) --> (*ALL y. Q(y)*))
by *fast*

lemma (*EX x y. P(x) & Q(x,y)*) <-> (*EX x. P(x) & (EX y. Q(x,y)*))
by *fast*

lemma (*EX y. ALL x. P(x) --> Q(x,y)*) --> (*ALL x. P(x) --> (EX y. Q(x,y)*))
by *fast*

end

theory *Miniscope*
imports *FOL*
begin

lemmas *ccontr = FalseE [THEN classical]*

12.1 Negation Normal Form

12.1.1 de Morgan laws

lemma *demorgans*:
~(P&Q) <-> ~P | ~Q
~(P|Q) <-> ~P & ~Q
~~P <-> P
!!P. ~(ALL x. P(x)) <-> (EX x. ~P(x))
!!P. ~(EX x. P(x)) <-> (ALL x. ~P(x))
by *blast+*

lemma *nnf-simps*:
(P-->Q) <-> (~P | Q)
~(P-->Q) <-> (P & ~Q)
(P<->Q) <-> (~P | Q) & (~Q | P)
~(P<->Q) <-> (P | Q) & (~P | ~Q)
by *blast+*

12.1.2 Pushing in the existential quantifiers

lemma *ex-simps*:
  $(EX\ x.\ P) <-> P$
  $!!P\ Q.\ (EX\ x.\ P(x)\ \&\ Q) <-> (EX\ x.\ P(x))\ \&\ Q$
  $!!P\ Q.\ (EX\ x.\ P\ \&\ Q(x)) <-> P\ \&\ (EX\ x.\ Q(x))$
  $!!P\ Q.\ (EX\ x.\ P(x) \mid Q(x)) <-> (EX\ x.\ P(x)) \mid (EX\ x.\ Q(x))$
  $!!P\ Q.\ (EX\ x.\ P(x) \mid Q) <-> (EX\ x.\ P(x)) \mid Q$
  $!!P\ Q.\ (EX\ x.\ P \mid Q(x)) <-> P \mid (EX\ x.\ Q(x))$
  by *blast+*

12.1.3 Pushing in the universal quantifiers

lemma *all-simps*:
  $(ALL\ x.\ P) <-> P$
  $!!P\ Q.\ (ALL\ x.\ P(x)\ \&\ Q(x)) <-> (ALL\ x.\ P(x))\ \&\ (ALL\ x.\ Q(x))$
  $!!P\ Q.\ (ALL\ x.\ P(x)\ \&\ Q) <-> (ALL\ x.\ P(x))\ \&\ Q$
  $!!P\ Q.\ (ALL\ x.\ P\ \&\ Q(x)) <-> P\ \&\ (ALL\ x.\ Q(x))$
  $!!P\ Q.\ (ALL\ x.\ P(x) \mid Q) <-> (ALL\ x.\ P(x)) \mid Q$
  $!!P\ Q.\ (ALL\ x.\ P \mid Q(x)) <-> P \mid (ALL\ x.\ Q(x))$
  by *blast+*

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

ML ⟪
val mini-ss = simpset() addsimps (thms mini-simps);
val mini-tac = rtac (thm ccontr) THEN' asm-full-simp-tac mini-ss;
⟫

end

13 First-Order Logic: the 'if' example

theory *If* **imports** *FOL* **begin**

constdefs
  if :: $[o,o,o]=>o$
  $if(P,Q,R) == P\&Q \mid {}^{\sim}P\&R$

lemma *ifI*:
    $[| P ==> Q;\ {}^{\sim}P ==> R |] ==> if(P,Q,R)$
apply (*simp add*: *if-def*, *blast*)
done

lemma *ifE*:
   $[| if(P,Q,R);\ [| P;\ Q |] ==> S;\ [| {}^{\sim}P;\ R |] ==> S |] ==> S$
apply (*simp add*: *if-def*, *blast*)
done

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*))
by (*simp add*: *if-def*, *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*))
apply (*simp add*: *if-def*, *auto*)
— The next step will fail unless subgoals remain
apply (*tactic all-tac*)
oops

end

theory *NatClass*
imports *FOL*
begin

This is an abstract version of theory *Nat*. Instead of axiomatizing a single type *nat* we define the class of all these types (up to isomorphism).

Note: The *rec* operator had to be made *monomorphic*, because class axioms may not contain more than one type variable.

consts
  0 :: 'a (0)
  Suc :: 'a => 'a
  rec :: ['a, 'a, ['a, 'a] => 'a] => 'a

axclass
  nat < term
  induct: [| P(0); !!x. P(x) ==> P(Suc(x)) |] ==> P(n)
  Suc-inject: Suc(m) = Suc(n) ==> m = n
  Suc-neq-0:   Suc(m) = 0 ==> R
  rec-0:   rec(0, a, f) = a
  rec-Suc: rec(Suc(m), a, f) = f(m, rec(m, a, f))

definition
  add :: ['a::nat, 'a] => 'a (**infixl** + 60) **where**
  m + n = rec(m, n, %x y. Suc(y))

lemma Suc-n-not-n: Suc(k) ~= (k::'a::nat)
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 = k* **in** *induct*)
apply *simp*
apply *simp*
done

lemma *add-0-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*: !!*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

14 Example of Declaring an Oracle

theory *IffOracle*
imports *FOL*
begin

14.1 Oracle declaration

This oracle makes tautologies of the form $P <-> P <-> P <-> P$. The length is specified by an integer, which is checked to be even and positive.

```
oracle iff-oracle (int) = ⟪
  let
    fun mk-iff 1 = Var ((P, 0), @{typ o})
      | mk-iff n = FOLogic.iff $ Var ((P, 0), @{typ o}) $ mk-iff (n − 1);
  in
    fn thy => fn n =>
      if n > 0 andalso n mod 2 = 0
      then FOLogic.mk-Trueprop (mk-iff n)
      else raise Fail (iff-oracle:  ^ string-of-int n)
  end
⟫
```

14.2 Oracle as low-level rule

ML ⟪ *iff-oracle @{theory} 2* ⟫
ML ⟪ *iff-oracle @{theory} 10* ⟫
ML ⟪ *#der (Thm.rep-thm it)* ⟫

These oracle calls had better fail.

```
ML ⟪
  (iff-oracle @{theory} 5; error ?)
    handle Fail - => warning Oracle failed, as expected
⟫
```

```
ML ⟪
  (iff-oracle @{theory} 1; error ?)
    handle Fail - => warning Oracle failed, as expected
⟫
```

14.3 Oracle as proof method

```
method-setup iff = ⟪
  Method.simple-args Args.nat (fn n => fn ctxt =>
    Method.SIMPLE-METHOD
      (HEADGOAL (Tactic.rtac (iff-oracle (ProofContext.theory-of ctxt) n))
        handle Fail - => no-tac))
⟫ iff oracle
```

lemma *A <−> A*
 by *(iff 2)*

lemma *A <−> A <−> A <−> A <−> A <−> A <−> A <−> A <−> A <−> A*
 by *(iff 10)*

lemma *A <−> A <−> A <−> A <−> A*
 apply *(iff 5)?*
 oops

lemma *A*
 apply *(iff 1)?*
 oops

end