

Miscellaneous FOL Examples

April 19, 2009

Contents

1	A simple formulation of First-Order Logic	2
1.1	Syntax	2
1.2	Propositional logic	3
1.3	Equality	4
1.4	Quantifiers	4
2	Natural numbers	5
3	Examples for the manual “Introduction to Isabelle”	7
3.0.1	Some simple backward proofs	7
3.0.2	Demonstration of <i>fast</i>	7
3.0.3	Derivation of conjunction elimination rule	8
3.1	Derived rules involving definitions	8
4	Theory of the natural numbers: Peano’s axioms, primitive recursion	9
4.1	Proofs about the natural numbers	9
5	Intuitionistic FOL: Examples from The Foundation of a Generic Theorem Prover	12
5.1	Examples with quantifiers	13
6	First-Order Logic: PROLOG examples	15
7	Intuitionistic First-Order Logic	16
7.1	de Bruijn formulae	18
7.2	Intuitionistic FOL: propositional problems based on Pelletier.	18
7.3	11. Proved in each direction (incorrectly, says Pelletier!!)	19
7.4	****Examples with quantifiers****	20
7.5	The converse is classical in the following implications...	20
7.6	The following are not constructively valid!	20
7.7	Hard examples with quantifiers	21

8 First-Order Logic: propositional examples (intuitionistic version)	25
9 First-Order Logic: quantifier examples (intuitionistic version)	27
10 Classical Predicate Calculus Problems	29
10.1 Pelletier's examples	29
10.2 Classical Logic: examples with quantifiers	31
10.3 Problems requiring quantifier duplication	31
10.4 Hard examples with quantifiers	32
10.5 Problems (mainly) involving equality or functions	35
11 First-Order Logic: propositional examples (classical version)	38
12 First-Order Logic: quantifier examples (classical version)	41
12.1 Negation Normal Form	43
12.1.1 de Morgan laws	43
12.1.2 Pushing in the existential quantifiers	43
12.1.3 Pushing in the universal quantifiers	43
13 First-Order Logic: the 'if' example	44
14 Example of Declaring an Oracle	45
14.1 Oracle declaration	45
14.2 Oracle as low-level rule	45
14.3 Oracle as proof method	46

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

typedcl *i*
typedcl *o*

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

1.2 Propositional logic

axiomatization

false :: o (\perp) **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*]: $\perp \Longrightarrow A$ **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**
 $\top \equiv \perp \longrightarrow \perp$

definition

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

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 $\perp \longrightarrow \perp$..

qed

theorem *notI* [*intro*]: $(A \Longrightarrow \perp) \Longrightarrow \neg A$

proof (*unfold not-def*)

assume $A \Longrightarrow \perp$

then show $A \longrightarrow \perp$..
qed

theorem *notE* [*elim*]: $\neg A \Longrightarrow A \Longrightarrow B$
proof (*unfold not-def*)
 assume $A \longrightarrow \perp$ and A
 then have \perp .. 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 ⇒ o) ⇒ o (binder ∀ 10) and
  Ex :: (i ⇒ o) ⇒ o (binder ∃ 10)
where
  allI [intro]: (Λx. P x) ⇒ ∀ x. P x and
  allD [dest]: ∀ x. P x ⇒ P a and
  exI [intro]: P a ⇒ ∃ x. P x and
  exE [elim]: ∃ x. P x ⇒ (Λx. P x ⇒ C) ⇒ C

```

```

lemma (∃ x. P (f x)) → (∃ y. P y)
proof
  assume ∃ x. P (f x)
  then show ∃ y. P y
  proof
    fix x assume P (f x)
    then show ?thesis ..
  qed
qed

```

```

lemma (∃ x. ∀ y. R x y) → (∀ y. ∃ x. R x y)
proof
  assume ∃ x. ∀ y. R x y
  then show ∀ y. ∃ x. R x y
  proof
    fix x assume a: ∀ y. R x y
    show ?thesis
    proof
      fix y from a have R x y ..
      then show ∃ 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

```

```

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*: $Suc(k) \neq k$
proof (*induct k*)
 show $Suc(0) \neq 0$
proof
 assume $Suc(0) = 0$
 then show *False* **by** (*rule Suc-neq-0*)
 qed
 next
 fix *n* assume *hyp*: $Suc(n) \neq n$
 show $Suc(Suc(n)) \neq Suc(n)$
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**
 $m + n = rec(m, n, \lambda x y. Suc(y))$

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

```

3 Examples for the manual “Introduction to Isabelle”

```

theory Intro
imports FOL
begin

```

3.0.1 Some simple backward proofs

```

lemma mythm:  $P \mid P \longrightarrow P$ 
  apply (rule impI)
  apply (rule disjE)
  prefer 3 apply (assumption)
  prefer 2 apply (assumption)
  apply assumption
done

```

```

lemma  $(P \ \& \ Q) \mid R \longrightarrow (P \mid R)$ 
  apply (rule impI)
  apply (erule disjE)
  apply (drule conjunct1)
  apply (rule disjI1)
  apply (rule-tac [2] disjI2)
  apply assumption+
done

```

```

lemma  $(\text{ALL } x \ y. \ P(x,y)) \longrightarrow (\text{ALL } z \ w. \ P(w,z))$ 
  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  $(\text{EX } y. \ \text{ALL } x. \ J(y,x) \longleftrightarrow \sim J(x,x))$ 
   $\longrightarrow \sim (\text{ALL } x. \ \text{EX } y. \ \text{ALL } z. \ J(z,y) \longleftrightarrow \sim J(z,x))$ 
  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:  $\forall n. f(Suc(n)) = Suc(f(n))$ 
  shows  $f(i+j) = i+f(j)$ 
apply (rule-tac n = i in induct)
apply simp
apply (simp add: prem)
done

end

```

```

theory Nat-Class
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.

```

class nat =
  fixes Zero :: 'a (0)
  and Suc :: 'a  $\Rightarrow$  'a
  and rec :: 'a  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  'a)  $\Rightarrow$  'a

```

```

assumes induct:  $P(0) \implies (\bigwedge x. P(x) \implies P(\text{Suc}(x))) \implies P(n)$ 
and Suc-inject:  $\text{Suc}(m) = \text{Suc}(n) \implies m = n$ 
and Suc-neq-Zero:  $\text{Suc}(m) = 0 \implies R$ 
and rec-Zero:  $\text{rec}(0, a, f) = a$ 
and rec-Suc:  $\text{rec}(\text{Suc}(m), a, f) = f(m, \text{rec}(m, a, f))$ 
begin

```

definition

```

add :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a (infixl + 60) where
   $m + n = \text{rec}(m, n, \lambda x y. \text{Suc}(y))$ 

```

```

lemma Suc-n-not-n:  $\text{Suc}(k) \neq (k::'a)$ 
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
back
oops

```

```

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

```

```

lemma add-Suc [simp]:  $\text{Suc}(m) + n = \text{Suc}(m + n)$ 
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:  $m + \text{Suc}(n) = \text{Suc}(m + n)$ 
  apply (rule-tac  $n = m$  in induct)
  apply simp-all
done

lemma
  assumes prem:  $\bigwedge n. f(\text{Suc}(n)) = \text{Suc}(f(n))$ 
  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  $A \& B \longrightarrow (C \longrightarrow A \& C)$ 
  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  $A \implies B \implies C$ 
  shows  $C$ 
  apply (rule assms)
  apply (rule conjunct1)
  apply (rule assms)
  apply (rule conjunct2)
  apply (rule assms)
done

```

```

lemma
  assumes !!A.  $\sim \sim A \implies A$ 
  shows  $B \mid \sim B$ 
  apply (rule assms)
  apply (rule notI)
  apply (rule-tac  $P = \sim B$  in notE)
  apply (rule-tac [2] notI)
  apply (rule-tac [2]  $P = B \mid \sim B$  in notE)
  prefer 2 apply assumption
  apply (rule-tac [2] disjI1)
  prefer 2 apply assumption
  apply (rule notI)
  apply (rule-tac  $P = B \mid \sim B$  in notE)
  apply assumption
  apply (rule disjI2)
  apply assumption
done

```

```

lemma
  assumes !!A.  $\sim \sim A \implies A$ 
  shows  $B \mid \sim B$ 
  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  $A \mid \sim A$ 
  and  $\sim \sim A$ 
  shows  $A$ 
  apply (rule disjE)
  apply (rule assms)
  apply assumption
  apply (rule FalseE)
  apply (rule-tac  $P = \sim A$  in notE)
  apply (rule assms)
  apply assumption
done

```

5.1 Examples with quantifiers

```

lemma

```

```

    assumes  $ALL\ z.\ G(z)$ 
    shows  $ALL\ z.\ G(z) \mid H(z)$ 
  apply (rule allI)
  apply (rule disjI1)
  apply (rule assms [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 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  $(EX\ y.\ ALL\ x.\ Q(x,y)) \dashv\vdash (ALL\ x.\ EX\ y.\ Q(x,y))$ 
  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 ([@{thm refl}, @{thm 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 ([@{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 $Q \rightarrow P$ 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 intuitionistically equivalent to P .
[Andy Pitts]

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

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

Double-negation does NOT distribute over disjunction

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

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

lemma $\sim\sim((P \multimap Q \mid R) \multimap (P \multimap Q) \mid (P \multimap R))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

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

lemma $((P \multimap (Q \mid (Q \multimap R))) \multimap R) \multimap R$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

lemma $((((G \multimap A) \multimap J) \multimap D \multimap E) \multimap (((H \multimap B) \multimap I) \multimap C \multimap J) \multimap (A \multimap H) \multimap F \multimap G \multimap (((C \multimap B) \multimap I) \multimap D) \multimap (A \multimap C) \multimap ((F \multimap A) \multimap B) \multimap I) \multimap E$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

Lemmas for the propositional double-negation translation

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

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

lemma $\sim\sim P \& \sim\sim(P \multimap Q) \multimap \sim\sim Q$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

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

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

lemma $(P \& Q \multimap R) \multimap (P \multimap R) \mid (Q \multimap R)$
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 $((P \leftrightarrow Q) \multimap P \& Q \& R) \&$
 $((Q \leftrightarrow R) \multimap P \& Q \& R) \&$
 $((R \leftrightarrow P) \multimap P \& Q \& R) \multimap P \& Q \& R$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

de Bruijn formula with five predicates

lemma $((P \leftrightarrow Q) \multimap P \& Q \& R \& S \& T) \&$
 $((Q \leftrightarrow R) \multimap P \& Q \& R \& S \& T) \&$
 $((R \leftrightarrow S) \multimap P \& Q \& R \& S \& T) \&$
 $((S \leftrightarrow T) \multimap P \& Q \& R \& S \& T) \&$
 $((T \leftrightarrow P) \multimap P \& Q \& R \& S \& T) \multimap P \& Q \& R \& S \& T$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

Problem 1.1

lemma $(\text{ALL } x. \text{EX } y. \text{ALL } z. p(x) \& q(y) \& r(z)) \leftrightarrow$
 $(\text{ALL } z. \text{EX } y. \text{ALL } x. p(x) \& q(y) \& r(z))$
by (*tactic*⟨⟨*IntPr.best-dup-tac 1*⟩⟩) — SLOW

Problem 3.1

lemma $\sim (\text{EX } x. \text{ALL } y. \text{mem}(y, x) \leftrightarrow \sim \text{mem}(x, x))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

Problem 4.1: hopeless!

lemma $(\text{ALL } x. p(x) \multimap p(h(x)) \mid p(g(x))) \& (\text{EX } x. p(x)) \& (\text{ALL } x. \sim p(h(x)))$
 $\multimap (\text{EX } x. p(g(g(g(g(x))))))$
oops

7.2 Intuitionistic FOL: propositional problems based on Pelletier.

1

lemma $\sim \sim ((P \multimap Q) \leftrightarrow (\sim Q \multimap \sim P))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

2

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

3

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

4

lemma $\sim\sim((\sim P \multimap Q) \leftrightarrow (\sim Q \multimap P))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

5

lemma $\sim\sim((P|Q \multimap P|R) \multimap P|(Q \multimap R))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

6

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

7

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

8. Peirce's law

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

9

lemma $((P|Q) \& (\sim P|Q) \& (P|\sim Q)) \multimap \sim(\sim P | \sim Q)$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

10

lemma $(Q \multimap R) \multimap (R \multimap P \& Q) \multimap (P \multimap (Q|R)) \multimap (P \leftrightarrow Q)$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

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

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

12. Dijkstra's law

lemma $\sim\sim(((P \leftrightarrow Q) \leftrightarrow R) \leftrightarrow (P \leftrightarrow (Q \leftrightarrow R)))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

lemma $((P \leftrightarrow Q) \leftrightarrow R) \multimap \sim\sim(P \leftrightarrow (Q \leftrightarrow R))$

by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

13. Distributive law

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

14

lemma $\sim\sim((P \ \<-> \ Q) \ \<-> \ ((Q \mid \sim P) \ \& \ (\sim Q \mid P)))$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

15

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

16

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

17

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

Dijkstra's "Golden Rule"

lemma $(P \ \& \ Q) \ \<-> \ P \ \<-> \ Q \ \<-> \ (P \mid Q)$
by (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩)

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*⟨⟨*IntPr.fast-tac 1*⟩⟩)

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

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

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

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

7.6 The following are not constructively valid!

The attempt to prove them terminates quickly!

lemma $((ALL\ x.\ P(x)) \multimap Q) \multimap (EX\ x.\ P(x) \multimap Q)$
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩ | -)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(P \multimap (EX\ x.\ Q(x))) \multimap (EX\ x.\ P \multimap Q(x))$
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩ | -)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ P(x) \mid Q) \multimap ((ALL\ x.\ P(x)) \mid Q)$
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩ | -)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

lemma $(ALL\ x.\ \sim\sim P(x)) \multimap \sim\sim(ALL\ x.\ P(x))$
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩ | -)
apply (*rule asm-rl*) — Checks that subgoals remain: proof failed.
oops

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

lemma $EX\ x.\ Q(x) \multimap (ALL\ x.\ Q(x))$
apply (*tactic*⟨⟨*IntPr.fast-tac 1*⟩⟩ | -)
apply (*rule asm-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 $\sim\sim(EX\ y.\ ALL\ x.\ P(y) \multimap P(x))$
oops — NOT PROVED

19

lemma $\sim\sim(EX\ x.\ ALL\ y\ z.\ (P(y) \multimap Q(z)) \multimap (P(x) \multimap Q(x)))$
oops — NOT PROVED

20

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

21

lemma $(EX\ x.\ P \multimap Q(x)) \& (EX\ x.\ Q(x) \multimap P) \multimap \sim\sim(EX\ x.\ P \multimap Q(x))$
oops — NOT PROVED; needs quantifier duplication

22

lemma $(ALL\ x.\ P \leftrightarrow Q(x)) \dashv\vdash (P \leftrightarrow (ALL\ x.\ Q(x)))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

23

lemma $\sim\sim((ALL\ x.\ P \mid Q(x)) \leftrightarrow (P \mid (ALL\ x.\ Q(x))))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

24

lemma $\sim(EX\ x.\ S(x) \& Q(x)) \& (ALL\ x.\ P(x) \dashv\vdash Q(x) \mid R(x)) \&$
 $(\sim(EX\ x.\ P(x)) \dashv\vdash (EX\ x.\ Q(x))) \& (ALL\ x.\ Q(x) \mid R(x) \dashv\vdash S(x))$
 $\dashv\vdash \sim\sim(EX\ x.\ P(x) \& R(x))$

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

apply $(tactic\langle\langle IntPr.safe-tac \rangle\rangle)$
apply $(erule\ impE)$
apply $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

25

lemma $(EX\ x.\ P(x)) \&$
 $(ALL\ x.\ L(x) \dashv\vdash \sim(M(x) \& R(x))) \&$
 $(ALL\ x.\ P(x) \dashv\vdash (M(x) \& L(x))) \&$
 $((ALL\ x.\ P(x) \dashv\vdash Q(x)) \mid (EX\ x.\ P(x) \& R(x)))$
 $\dashv\vdash (EX\ x.\ Q(x) \& P(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

26

lemma $(\sim\sim(EX\ x.\ p(x)) \leftrightarrow \sim\sim(EX\ x.\ q(x))) \&$
 $(ALL\ x.\ ALL\ y.\ p(x) \& q(y) \dashv\vdash (r(x) \leftrightarrow s(y)))$
 $\dashv\vdash ((ALL\ x.\ p(x) \dashv\vdash r(x)) \leftrightarrow (ALL\ x.\ q(x) \dashv\vdash s(x)))$
oops — NOT PROVED

27

lemma $(EX\ x.\ P(x) \& \sim Q(x)) \&$
 $(ALL\ x.\ P(x) \dashv\vdash R(x)) \&$
 $(ALL\ x.\ M(x) \& L(x) \dashv\vdash P(x)) \&$
 $((EX\ x.\ R(x) \& \sim Q(x)) \dashv\vdash (ALL\ x.\ L(x) \dashv\vdash \sim R(x)))$
 $\dashv\vdash (ALL\ x.\ M(x) \dashv\vdash \sim L(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

28. AMENDED

lemma $(ALL\ x.\ P(x) \dashv\vdash (ALL\ x.\ Q(x))) \&$
 $(\sim\sim(ALL\ x.\ Q(x) \mid R(x)) \dashv\vdash (EX\ x.\ Q(x) \& S(x))) \&$
 $(\sim\sim(EX\ x.\ S(x)) \dashv\vdash (ALL\ x.\ L(x) \dashv\vdash M(x)))$
 $\dashv\vdash (ALL\ x.\ P(x) \& L(x) \dashv\vdash M(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

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 $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

30

lemma $(ALL\ x.\ (P(x) \mid Q(x)) --> \sim R(x)) \ \&$
 $(ALL\ x.\ (Q(x) --> \sim S(x)) --> P(x) \ \&\ R(x))$
 $--> (ALL\ x.\ \sim\sim S(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

31

lemma $\sim(EX\ x.\ P(x) \ \&\ (Q(x) \mid R(x))) \ \&$
 $(EX\ x.\ L(x) \ \&\ P(x)) \ \&$
 $(ALL\ x.\ \sim R(x) --> M(x))$
 $--> (EX\ x.\ L(x) \ \&\ M(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

32

lemma $(ALL\ x.\ P(x) \ \&\ (Q(x) \mid R(x)) --> S(x)) \ \&$
 $(ALL\ x.\ S(x) \ \&\ R(x) --> L(x)) \ \&$
 $(ALL\ x.\ M(x) --> R(x))$
 $--> (ALL\ x.\ P(x) \ \&\ M(x) --> L(x))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

33

lemma $(ALL\ x.\ \sim\sim(P(a) \ \&\ (P(x) --> P(b)) --> P(c))) \ <->$
 $(ALL\ x.\ \sim\sim((\sim P(a) \mid P(x) \mid P(c)) \ \&\ (\sim P(a) \mid \sim P(b) \mid P(c))))$
apply $(tactic\langle\langle IntPr.best-tac\ 1 \rangle\rangle)$
done

36

lemma $(ALL\ x.\ EX\ y.\ J(x,y)) \ \&$
 $(ALL\ x.\ EX\ y.\ G(x,y)) \ \&$
 $(ALL\ x\ y.\ J(x,y) \mid G(x,y) --> (ALL\ z.\ J(y,z) \mid G(y,z) --> H(x,z)))$
 $--> (ALL\ x.\ EX\ y.\ H(x,y))$
by $(tactic\langle\langle IntPr.fast-tac\ 1 \rangle\rangle)$

37

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

39

lemma $\sim (EX\ x.\ ALL\ y.\ F(y,x) <-> \sim F(y,y))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

40. AMENDED

lemma $(EX\ y.\ ALL\ x.\ F(x,y) <-> F(x,x)) \dashv\dashv$
 $\sim (ALL\ x.\ EX\ y.\ ALL\ z.\ F(z,y) <-> \sim F(z,x))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

44

lemma $(ALL\ x.\ f(x) \dashv\dashv$
 $(EX\ y.\ g(y) \ \&\ h(x,y) \ \&\ (EX\ y.\ g(y) \ \&\ \sim h(x,y)))) \ \&$
 $(EX\ x.\ j(x) \ \&\ (ALL\ y.\ g(y) \dashv\dashv h(x,y)))$
 $\dashv\dashv (EX\ x.\ j(x) \ \&\ \sim f(x))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

48

lemma $(a=b \mid c=d) \ \&\ (a=c \mid b=d) \dashv\dashv a=d \mid b=c$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

51

lemma $(EX\ z\ w.\ ALL\ x\ y.\ P(x,y) <-> (x=z \ \&\ y=w)) \dashv\dashv$
 $(EX\ z.\ ALL\ x.\ EX\ w.\ (ALL\ y.\ P(x,y) <-> y=w) <-> x=z)$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

52

Almost the same as 51.

lemma $(EX\ z\ w.\ ALL\ x\ y.\ P(x,y) <-> (x=z \ \&\ y=w)) \dashv\dashv$
 $(EX\ w.\ ALL\ y.\ EX\ z.\ (ALL\ x.\ P(x,y) <-> x=z) <-> y=w)$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

56

lemma $(ALL\ x.\ (EX\ y.\ P(y) \ \&\ x=f(y)) \dashv\dashv P(x)) <-> (ALL\ x.\ P(x) \dashv\dashv$
 $P(f(x)))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

57

lemma $P(f(a,b), f(b,c)) \ \&\ P(f(b,c), f(a,c)) \ \&$
 $(ALL\ x\ y\ z.\ P(x,y) \ \&\ P(y,z) \dashv\dashv P(x,z)) \dashv\dashv P(f(a,b), f(a,c))$
by (*tactic*⟨⟨*IntPr.fast-tac* 1⟩⟩)

60

lemma $ALL\ x.\ P(x,f(x)) <-> (EX\ y.\ (ALL\ z.\ P(z,y) \dashv\dashv P(z,f(x))) \ \&\ P(x,y))$
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  $P \& Q \dashv\vdash Q \& P$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $P | Q \dashv\vdash Q | P$ 
  by (tactic IntPr.fast-tac 1)
```

associative laws of $\&$ and $|$

```
lemma  $(P \& Q) \& R \dashv\vdash P \& (Q \& R)$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $(P | Q) | R \dashv\vdash P | (Q | R)$ 
  by (tactic IntPr.fast-tac 1)
```

distributive laws of $\&$ and $|$

```
lemma  $(P \& Q) | R \dashv\vdash (P | R) \& (Q | R)$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $(P | R) \& (Q | R) \dashv\vdash (P \& Q) | R$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $(P | Q) \& R \dashv\vdash (P \& R) | (Q \& R)$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $(P \& R) | (Q \& R) \dashv\vdash (P | Q) \& R$ 
  by (tactic IntPr.fast-tac 1)
```

Laws involving implication

```
lemma  $(P \dashv\vdash R) \& (Q \dashv\vdash R) \dashv\vdash (P | Q \dashv\vdash R)$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $(P \& Q \dashv\vdash R) \dashv\vdash (P \dashv\vdash (Q \dashv\vdash R))$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $((P \dashv\vdash R) \dashv\vdash R) \dashv\vdash ((Q \dashv\vdash R) \dashv\vdash R) \dashv\vdash (P \& Q \dashv\vdash R) \dashv\vdash R$ 
  by (tactic IntPr.fast-tac 1)
```

```
lemma  $\sim(P \dashv\vdash R) \dashv\vdash \sim(Q \dashv\vdash R) \dashv\vdash \sim(P \& Q \dashv\vdash R)$ 
  by (tactic IntPr.fast-tac 1)
```

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

Propositions-as-types

— The combinator K

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

— The combinator S

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

— Converse is classical

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

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

Schwichtenberg’s examples (via T. Nipkow)

lemma *stab-imp*: $((Q \multimap R) \multimap R) \multimap Q \multimap ((P \multimap Q) \multimap R) \multimap R \multimap P \multimap Q$
by (*tactic IntPr.fast-tac 1*)

lemma *stab-to-peirce*:

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

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

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

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

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

lemma *tatsuta*: $((P7 \multimap P1) \multimap P10) \multimap P4 \multimap P5$
 $\multimap ((P8 \multimap P2) \multimap P9) \multimap P3 \multimap P10$
 $\multimap (P1 \multimap P8) \multimap P6 \multimap P7$
 $\multimap ((P3 \multimap P2) \multimap P9) \multimap P4$
 $\multimap (P1 \multimap P3) \multimap ((P6 \multimap P1) \multimap P2) \multimap P9 \multimap P5$

```

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 ($ALL\ x.\ Q(x) \dashrightarrow (EX\ x.\ Q(x))$)
by (*tactic IntPr.fast-tac 1*)

The following should fail, as they are false!

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

lemma ($EX\ x.\ Q(x) \dashrightarrow (ALL\ x.\ Q(x))$)
apply (*tactic IntPr.fast-tac 1*)?
oops

lemma $P(?a) \dashrightarrow (ALL\ x.\ P(x))$
apply (*tactic IntPr.fast-tac 1*)?
oops

lemma ($P(?a) \dashrightarrow (ALL\ x.\ Q(x)) \dashrightarrow (ALL\ x.\ P(x) \dashrightarrow Q(x))$)
apply (*tactic IntPr.fast-tac 1*)?
oops

Back to things that are provable ...

lemma ($ALL\ x.\ P(x) \dashrightarrow Q(x) \ \& \ (EX\ x.\ P(x)) \dashrightarrow (EX\ x.\ Q(x))$)
by (*tactic IntPr.fast-tac 1*)

— An example of why exI should be delayed as long as possible

lemma ($P \dashrightarrow (EX\ x.\ Q(x)) \ \& \ P \dashrightarrow (EX\ x.\ Q(x))$)
by (*tactic IntPr.fast-tac 1*)

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

lemma ($ALL\ x.\ Q(x) \dashrightarrow (EX\ x.\ Q(x))$)
by (*tactic IntPr.fast-tac 1*)

Some slow ones

— Principia Mathematica *11.53

lemma ($ALL\ x\ y.\ P(x) \dashrightarrow Q(y) \Leftrightarrow ((EX\ x.\ P(x)) \dashrightarrow (ALL\ y.\ Q(y)))$)
by (*tactic IntPr.fast-tac 1*)

lemma ($EX\ x\ y.\ P(x) \ \& \ Q(x,y) \Leftrightarrow (EX\ x.\ P(x) \ \& \ (EX\ y.\ Q(x,y)))$)
by (*tactic IntPr.fast-tac 1*)

lemma ($EX\ y.\ ALL\ x.\ P(x) \dashrightarrow Q(x,y) \dashrightarrow (ALL\ x.\ P(x) \dashrightarrow (EX\ y.\ Q(x,y)))$)

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

end

10 Classical Predicate Calculus Problems

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

lemma $(P \multimap Q \mid R) \multimap (P \multimap Q) \mid (P \multimap R)$
by *blast*

If and only if

lemma $(P \leftrightarrow Q) \leftrightarrow (Q \leftrightarrow P)$
by *blast*

lemma $\sim (P \leftrightarrow \sim P)$
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 $(P \multimap Q) \leftrightarrow (\sim Q \multimap \sim P)$
by *blast*

2

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

3

lemma $\sim(P \multimap Q) \multimap (Q \multimap P)$
by *blast*

4

lemma $(\sim P \multimap Q) \leftrightarrow (\sim Q \multimap P)$
by *blast*

5

lemma $((P \mid Q) \multimap (P \mid R)) \multimap (P \mid (Q \multimap R))$
by *blast*

6

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

7

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

8. Peirce's law

lemma $((P \multimap Q) \multimap P) \multimap P$
by *blast*

9

lemma $((P \mid Q) \& (\sim P \mid Q) \& (P \mid \sim Q)) \multimap \sim (\sim P \mid \sim Q)$
by *blast*

10

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

14

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

15

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

16

lemma $(P \multimap Q) \mid (Q \multimap P)$
by *blast*

17

lemma $((P \& (Q \multimap R)) \multimap S) \<-> ((\sim P \mid Q \mid S) \& (\sim P \mid \sim R \mid S))$
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 $(\exists x. P \longrightarrow Q(x)) \ <-> (P \longrightarrow (\exists x. Q(x)))$
by *blast*

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

Needs multiple instantiation of ALL.

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

Needs double instantiation of the quantifier

lemma $\exists x. P(x) \longrightarrow P(a) \ \& \ P(b)$
by *blast*

lemma $\exists z. P(z) \longrightarrow (\forall x. P(x))$
by *blast*

lemma $\exists x. (\exists y. P(y)) \longrightarrow P(x)$
by *blast*

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

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

oops

10.4 Hard examples with quantifiers

18

lemma $\exists y. \forall x. P(y) \dashv\dashv P(x)$
by *blast*

19

lemma $\exists x. \forall y z. (P(y) \dashv\dashv Q(z)) \dashv\dashv (P(x) \dashv\dashv Q(x))$
by *blast*

20

lemma $(\forall x y. \exists z. \forall w. (P(x) \& Q(y) \dashv\dashv R(z) \& S(w)))$
 $\dashv\dashv (\exists x y. P(x) \& Q(y)) \dashv\dashv (\exists z. R(z))$
by *blast*

21

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

22

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

23

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

24

lemma $\sim(\exists x. S(x) \& Q(x)) \& (\forall x. P(x) \dashv\dashv Q(x) \mid R(x)) \&$
 $(\sim(\exists x. P(x)) \dashv\dashv (\exists x. Q(x))) \& (\forall x. Q(x) \mid R(x) \dashv\dashv S(x))$
 $\dashv\dashv (\exists x. P(x) \& R(x))$
by *blast*

25

lemma $(\exists x. P(x)) \&$
 $(\forall x. L(x) \dashv\dashv \sim (M(x) \& R(x))) \&$
 $(\forall x. P(x) \dashv\dashv (M(x) \& L(x))) \&$
 $((\forall x. P(x) \dashv\dashv Q(x)) \mid (\exists x. P(x) \& R(x)))$
 $\dashv\dashv (\exists x. Q(x) \& P(x))$
by *blast*

26

lemma $((\exists x. p(x)) \<-> (\exists x. q(x))) \&$
 $(\forall x. \forall y. p(x) \& q(y) \dashv\dashv (r(x) \<-> s(y)))$
 $\dashv\dashv ((\forall x. p(x) \dashv\dashv r(x)) \<-> (\forall x. q(x) \dashv\dashv s(x)))$
by *blast*

27

lemma $(\exists x. P(x) \ \& \ \sim Q(x)) \ \& \$
 $(\forall x. P(x) \ \longrightarrow \ R(x)) \ \& \$
 $(\forall x. M(x) \ \& \ L(x) \ \longrightarrow \ P(x)) \ \& \$
 $((\exists x. R(x) \ \& \ \sim Q(x)) \ \longrightarrow \ (\forall x. L(x) \ \longrightarrow \ \sim R(x)))$
 $\longrightarrow (\forall x. M(x) \ \longrightarrow \ \sim L(x))$
by *blast*

28. AMENDED

lemma $(\forall x. P(x) \ \longrightarrow \ (\forall x. Q(x))) \ \& \$
 $((\forall x. Q(x) | R(x)) \ \longrightarrow \ (\exists x. Q(x) \ \& \ S(x))) \ \& \$
 $((\exists x. S(x)) \ \longrightarrow \ (\forall x. L(x) \ \longrightarrow \ M(x)))$
 $\longrightarrow (\forall x. P(x) \ \& \ L(x) \ \longrightarrow \ M(x))$
by *blast*

29. Essentially the same as Principia Mathematica *11.71

lemma $(\exists x. P(x)) \ \& \ (\exists y. Q(y))$
 $\longrightarrow ((\forall x. P(x) \ \longrightarrow \ R(x)) \ \& \ (\forall y. Q(y) \ \longrightarrow \ S(y))) \ \longleftrightarrow$
 $(\forall x \ y. P(x) \ \& \ Q(y) \ \longrightarrow \ R(x) \ \& \ S(y))$
by *blast*

30

lemma $(\forall x. P(x) \ | \ Q(x) \ \longrightarrow \ \sim R(x)) \ \& \$
 $(\forall x. (Q(x) \ \longrightarrow \ \sim S(x)) \ \longrightarrow \ P(x) \ \& \ R(x))$
 $\longrightarrow (\forall x. S(x))$
by *blast*

31

lemma $\sim(\exists x. P(x) \ \& \ (Q(x) \ | \ R(x))) \ \& \$
 $(\exists x. L(x) \ \& \ P(x)) \ \& \$
 $(\forall x. \sim R(x) \ \longrightarrow \ M(x))$
 $\longrightarrow (\exists x. L(x) \ \& \ M(x))$
by *blast*

32

lemma $(\forall x. P(x) \ \& \ (Q(x) | R(x)) \ \longrightarrow \ S(x)) \ \& \$
 $(\forall x. S(x) \ \& \ R(x) \ \longrightarrow \ L(x)) \ \& \$
 $(\forall x. M(x) \ \longrightarrow \ R(x))$
 $\longrightarrow (\forall x. P(x) \ \& \ M(x) \ \longrightarrow \ L(x))$
by *blast*

33

lemma $(\forall x. P(a) \ \& \ (P(x) \ \longrightarrow \ P(b)) \ \longrightarrow \ P(c)) \ \longleftrightarrow$
 $(\forall x. (\sim P(a) \ | \ P(x) \ | \ P(c)) \ \& \ (\sim P(a) \ | \ \sim P(b) \ | \ P(c)))$
by *blast*

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

lemma $((\exists x. \forall y. p(x) \leftrightarrow p(y)) \leftrightarrow$
 $((\exists x. q(x) \leftrightarrow (\forall y. p(y)))) \leftrightarrow$
 $((\exists x. \forall y. q(x) \leftrightarrow q(y)) \leftrightarrow$
 $((\exists x. p(x) \leftrightarrow (\forall y. q(y))))$
by *blast*

35

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

36

lemma $(\forall x. \exists y. J(x,y)) \&$
 $(\forall x. \exists y. G(x,y)) \&$
 $(\forall x y. J(x,y) \mid G(x,y) \longrightarrow (\forall z. J(y,z) \mid G(y,z) \longrightarrow H(x,z))) \&$
 $\longrightarrow (\forall x. \exists y. H(x,y))$
by *blast*

37

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

38

lemma $(\forall x. p(a) \& (p(x) \longrightarrow (\exists y. p(y) \& r(x,y))) \longrightarrow$
 $(\exists z. \exists w. p(z) \& r(x,w) \& r(w,z))) \leftrightarrow$
 $(\forall x. (\sim p(a) \mid p(x) \mid (\exists z. \exists w. p(z) \& r(x,w) \& r(w,z))) \&$
 $(\sim p(a) \mid \sim (\exists y. p(y) \& r(x,y)) \mid$
 $(\exists z. \exists w. p(z) \& r(x,w) \& r(w,z))))$
by *blast*

39

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

40. AMENDED

lemma $(\exists y. \forall x. F(x,y) \leftrightarrow F(x,x)) \longrightarrow$
 $\sim (\forall x. \exists y. \forall z. F(z,y) \leftrightarrow \sim F(z,x))$
by *blast*

41

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

44

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

45

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

46

lemma $(\forall x. f(x) \& (\forall y. f(y) \& h(y,x) ---> g(y)) ---> g(x)) \&$
 $((\exists x. f(x) \& \sim g(x)) --->$
 $(\exists x. f(x) \& \sim g(x) \& (\forall y. f(y) \& \sim g(y) ---> j(x,y)))) \&$
 $(\forall x y. f(x) \& f(y) \& h(x,y) ---> \sim j(y,x))$
 $---> (\forall x. f(x) ---> g(x))$
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 $(\exists x y::'a. \forall z. z=x \mid z=y) \& P(a) \& P(b) \& a\sim=b$
 $---> (\forall u::'a. P(u))$

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 $(\forall x. P(a,x) \mid (\forall y. P(x,y))) \dashv\vdash (\exists x. \forall y. P(x,y))$
by *blast*

51

lemma $(\exists z w. \forall x y. P(x,y) \leftrightarrow (x=z \ \& \ y=w)) \dashv\vdash$
 $(\exists z. \forall x. \exists w. (\forall y. P(x,y) \leftrightarrow y=w) \leftrightarrow x=z)$
by *blast*

52

Almost the same as 51.

lemma $(\exists z w. \forall x y. P(x,y) \leftrightarrow (x=z \ \& \ y=w)) \dashv\vdash$
 $(\exists w. \forall y. \exists z. (\forall x. P(x,y) \leftrightarrow x=z) \leftrightarrow y=w)$
by *blast*

55

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

lemma $\text{lives}(\text{agatha}) \ \& \ \text{lives}(\text{butler}) \ \& \ \text{lives}(\text{charles}) \ \&$
 $(\text{killed}(\text{agatha}, \text{agatha}) \mid \text{killed}(\text{butler}, \text{agatha}) \mid \text{killed}(\text{charles}, \text{agatha})) \ \&$
 $(\forall x y. \text{killed}(x,y) \dashv\vdash \text{hates}(x,y) \ \& \ \sim \text{richer}(x,y)) \ \&$
 $(\forall x. \text{hates}(\text{agatha}, x) \dashv\vdash \sim \text{hates}(\text{charles}, x)) \ \&$
 $(\text{hates}(\text{agatha}, \text{agatha}) \ \& \ \text{hates}(\text{agatha}, \text{charles})) \ \&$
 $(\forall x. \text{lives}(x) \ \& \ \sim \text{richer}(x, \text{agatha}) \dashv\vdash \text{hates}(\text{butler}, x)) \ \&$
 $(\forall x. \text{hates}(\text{agatha}, x) \dashv\vdash \text{hates}(\text{butler}, x)) \ \&$
 $(\forall x. \sim \text{hates}(x, \text{agatha}) \mid \sim \text{hates}(x, \text{butler}) \mid \sim \text{hates}(x, \text{charles})) \dashv\vdash$
 $\text{killed}(\text{?who}, \text{agatha})$
by *fast* — MUCH faster than blast

56

lemma $(\forall x. (\exists y. P(y) \ \& \ x=f(y)) \dashv\vdash P(x)) \leftrightarrow (\forall x. P(x) \dashv\vdash P(f(x)))$
by *blast*

57

lemma $P(f(a,b), f(b,c)) \ \& \ P(f(b,c), f(a,c)) \ \&$
 $(\forall x y z. P(x,y) \ \& \ P(y,z) \dashv\vdash P(x,z)) \dashv\vdash P(f(a,b), f(a,c))$
by *blast*

58 NOT PROVED AUTOMATICALLY

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

59

lemma $(\forall x. P(x) \leftrightarrow \sim P(f(x))) \dashv\vdash (\exists x. P(x) \ \& \ \sim P(f(x)))$
by *blast*

60

lemma $\forall x. P(x, f(x)) \leftrightarrow (\exists y. (\forall z. P(z, y) \rightarrow P(z, f(x))) \& P(x, y))$
by *blast*

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

lemma $(\forall x. p(a) \& (p(x) \rightarrow p(f(x))) \rightarrow p(f(f(x)))) \leftrightarrow$
 $(\forall x. (\sim p(a) \mid p(x) \mid p(f(f(x)))) \&$
 $(\sim p(a) \mid \sim p(f(x)) \mid p(f(f(x))))$
by *blast*

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

lemma $(\forall x. F(x) \& \sim G(x) \rightarrow (\exists y. H(x, y) \& J(y))) \&$
 $(\exists x. K(x) \& F(x) \& (\forall y. H(x, y) \rightarrow K(y))) \&$
 $(\forall x. K(x) \rightarrow \sim G(x)) \rightarrow (\exists x. K(x) \& J(x))$
by *fast*

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

lemma $(\forall x. F(x) \& \sim G(x) \rightarrow (\exists y. H(x, y) \& J(y))) \&$
 $(\exists x. K(x) \& F(x) \& (\forall y. H(x, y) \rightarrow K(y))) \&$
 $(\forall x. K(x) \rightarrow \sim G(x)) \rightarrow (\exists x. K(x) \rightarrow \sim G(x))$
by *fast*

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

lemma $((\exists x. A(x) \& (\forall y. C(y) \rightarrow (\forall z. D(x, y, z)))) \rightarrow$
 $(\exists w. C(w) \& (\forall y. C(y) \rightarrow (\forall z. D(w, y, z))))$
 $\&$
 $(\forall w. C(w) \& (\forall u. C(u) \rightarrow (\forall v. D(w, u, v))) \rightarrow$
 $(\forall y z.$
 $(C(y) \& P(y, z) \rightarrow Q(w, y, z) \& OO(w, g)) \&$
 $(C(y) \& \sim P(y, z) \rightarrow Q(w, y, z) \& OO(w, b))))$
 $\&$
 $(\forall w. C(w) \&$
 $(\forall y z.$
 $(C(y) \& P(y, z) \rightarrow Q(w, y, z) \& OO(w, g)) \&$
 $(C(y) \& \sim P(y, z) \rightarrow Q(w, y, z) \& OO(w, b))) \rightarrow$
 $(\exists v. C(v) \&$
 $(\forall y. ((C(y) \& Q(w, y, y)) \& OO(w, g) \rightarrow \sim P(v, y)) \&$
 $((C(y) \& Q(w, y, y)) \& OO(w, b) \rightarrow P(v, y) \& OO(v, b))))$
 \rightarrow
 $\sim (\exists x. A(x) \& (\forall y. C(y) \rightarrow (\forall z. D(x, y, z))))$
by (*tactic* $\langle\langle Blast.depth-tac \ @\{claset\} \ 12 \ 1 \rangle\rangle$)
— 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 $((\exists x. A(x) \ \& \ (\forall y. C(y) \longrightarrow (\forall z. D(x,y,z)))) \longrightarrow$
 $(\exists w. C(w) \ \& \ (\forall y. C(y) \longrightarrow (\forall z. D(w,y,z))))$
 $\ \&$
 $(\forall w. C(w) \ \& \ (\forall u. C(u) \longrightarrow (\forall v. D(w,u,v))) \longrightarrow$
 $(\forall y \ z.$
 $(C(y) \ \& \ P(y,z) \longrightarrow Q(w,y,z) \ \& \ OO(w,g)) \ \&$
 $(C(y) \ \& \ \sim P(y,z) \longrightarrow Q(w,y,z) \ \& \ OO(w,b))))$
 $\ \&$
 $((\exists w. C(w) \ \& \ (\forall y. (C(y) \ \& \ P(y,y) \longrightarrow Q(w,y,y) \ \& \ OO(w,g)) \ \&$
 $(C(y) \ \& \ \sim P(y,y) \longrightarrow Q(w,y,y) \ \& \ OO(w,b))))$
 \longrightarrow
 $(\exists v. C(v) \ \& \ (\forall y. (C(y) \ \& \ P(y,y) \longrightarrow P(v,y) \ \& \ OO(v,g)) \ \&$
 $(C(y) \ \& \ \sim P(y,y) \longrightarrow P(v,y) \ \& \ OO(v,b))))$
 \longrightarrow
 $((\exists v. C(v) \ \& \ (\forall y. (C(y) \ \& \ P(y,y) \longrightarrow P(v,y) \ \& \ OO(v,g)) \ \&$
 $(C(y) \ \& \ \sim P(y,y) \longrightarrow P(v,y) \ \& \ OO(v,b))))$
 \longrightarrow
 $(\exists u. C(u) \ \& \ (\forall y. (C(y) \ \& \ P(y,y) \longrightarrow \sim P(u,y)) \ \&$
 $(C(y) \ \& \ \sim P(y,y) \longrightarrow P(u,y) \ \& \ OO(u,b))))$
 \longrightarrow
 $\sim (\exists x. A(x) \ \& \ (\forall y. C(y) \longrightarrow (\forall z. D(x,y,z))))$

by *blast*

Challenge found on info-hol

lemma $\forall x. \exists v \ w. \forall y \ z. P(x) \ \& \ Q(y) \longrightarrow (P(v) \mid R(w)) \ \& \ (R(z) \longrightarrow Q(v))$
by *blast*

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

lemma $(\forall x. \text{honest}(x) \ \& \ \text{industrious}(x) \longrightarrow \text{healthy}(x)) \ \&$
 $\sim (\exists x. \text{grocer}(x) \ \& \ \text{healthy}(x)) \ \&$
 $(\forall x. \text{industrious}(x) \ \& \ \text{grocer}(x) \longrightarrow \text{honest}(x)) \ \&$
 $(\forall x. \text{cyclist}(x) \longrightarrow \text{industrious}(x)) \ \&$
 $(\forall x. \sim \text{healthy}(x) \ \& \ \text{cyclist}(x) \longrightarrow \sim \text{honest}(x))$
 $\longrightarrow (\forall x. \text{grocer}(x) \longrightarrow \sim \text{cyclist}(x))$

by *blast*

end

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

theory *Propositional-Cla*

imports *FOL*
begin

commutative laws of $\&$ and $|$

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

lemma $P | Q \dashv\dashv Q | P$
by *fast*

associative laws of $\&$ and $|$

lemma $(P \& Q) \& R \dashv\dashv P \& (Q \& R)$
by *fast*

lemma $(P | Q) | R \dashv\dashv P | (Q | R)$
by *fast*

distributive laws of $\&$ and $|$

lemma $(P \& Q) | R \dashv\dashv (P | R) \& (Q | R)$
by *fast*

lemma $(P | R) \& (Q | R) \dashv\dashv (P \& Q) | R$
by *fast*

lemma $(P | Q) \& R \dashv\dashv (P \& R) | (Q \& R)$
by *fast*

lemma $(P \& R) | (Q \& R) \dashv\dashv (P | Q) \& R$
by *fast*

Laws involving implication

lemma $(P \dashv\dashv R) \& (Q \dashv\dashv R) \dashv\dashv (P | Q \dashv\dashv R)$
by *fast*

lemma $(P \& Q \dashv\dashv R) \dashv\dashv (P \dashv\dashv (Q \dashv\dashv R))$
by *fast*

lemma $((P \dashv\dashv R) \dashv\dashv R) \dashv\dashv ((Q \dashv\dashv R) \dashv\dashv R) \dashv\dashv (P \& Q \dashv\dashv R) \dashv\dashv R$
by *fast*

lemma $\sim(P \dashv\dashv R) \dashv\dashv \sim(Q \dashv\dashv R) \dashv\dashv \sim(P \& Q \dashv\dashv R)$
by *fast*

lemma $(P \dashv\dashv Q \& R) \dashv\dashv (P \dashv\dashv Q) \& (P \dashv\dashv R)$
by *fast*

Propositions-as-types

— The combinator K

lemma $P \multimap (Q \multimap P)$
by *fast*

— The combinator S

lemma $(P \multimap Q \multimap R) \multimap (P \multimap Q) \multimap (P \multimap R)$
by *fast*

— Converse is classical

lemma $(P \multimap Q) \mid (P \multimap R) \multimap (P \multimap Q \mid R)$
by *fast*

lemma $(P \multimap Q) \multimap (\sim Q \multimap \sim P)$
by *fast*

Schwichtenberg's examples (via T. Nipkow)

lemma *stab-imp*: $((Q \multimap R) \multimap R) \multimap Q \multimap ((P \multimap Q) \multimap R) \multimap P \multimap Q$
by *fast*

lemma *stab-to-peirce*:

$((P \multimap R) \multimap R) \multimap P \multimap ((Q \multimap R) \multimap R) \multimap Q$
 $\multimap ((P \multimap Q) \multimap P) \multimap P$

by *fast*

lemma *peirce-imp1*: $((Q \multimap R) \multimap Q) \multimap Q$
 $\multimap ((P \multimap Q) \multimap R) \multimap P \multimap Q \multimap P \multimap Q$
by *fast*

lemma *peirce-imp2*: $((P \multimap R) \multimap P) \multimap P \multimap ((P \multimap Q \multimap R) \multimap P) \multimap P$
by *fast*

lemma *mints*: $((P \multimap Q) \multimap P) \multimap P \multimap Q \multimap Q$
by *fast*

lemma *mints-solovev*: $(P \multimap (Q \multimap R) \multimap Q) \multimap ((P \multimap Q) \multimap R) \multimap R$
by *fast*

lemma *tatsuta*: $((P7 \multimap P1) \multimap P10) \multimap P4 \multimap P5$
 $\multimap ((P8 \multimap P2) \multimap P9) \multimap P3 \multimap P10$
 $\multimap (P1 \multimap P8) \multimap P6 \multimap P7$
 $\multimap ((P3 \multimap P2) \multimap P9) \multimap P4$
 $\multimap (P1 \multimap P3) \multimap ((P6 \multimap P1) \multimap P2) \multimap P9 \multimap P5$
by *fast*

lemma *tatsuta1*: $((P8 \multimap P2) \multimap P9) \multimap P3 \multimap P10$
 $\multimap ((P3 \multimap P2) \multimap P9) \multimap P4$


```

--> (((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:
   $\sim(P \& Q) \leftrightarrow \sim P \mid \sim Q$ 
   $\sim(P \mid Q) \leftrightarrow \sim P \& \sim Q$ 
   $\sim\sim P \leftrightarrow P$ 
   $!!P. \sim(ALL\ x. P(x)) \leftrightarrow (EX\ x. \sim P(x))$ 
   $!!P. \sim(EX\ x. P(x)) \leftrightarrow (ALL\ x. \sim P(x))$ 
by blast+

```

```

lemma nnf-simps:
   $(P \dashv\dashv Q) \leftrightarrow (\sim P \mid Q)$ 
   $\sim(P \dashv\dashv Q) \leftrightarrow (P \& \sim Q)$ 
   $(P \leftrightarrow Q) \leftrightarrow (\sim P \mid Q) \& (\sim Q \mid P)$ 
   $\sim(P \leftrightarrow Q) \leftrightarrow (P \mid Q) \& (\sim P \mid \sim Q)$ 
by blast+

```

12.1.2 Pushing in the existential quantifiers

```

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

```

12.1.3 Pushing in the universal quantifiers

```

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

```

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

ML $\langle\langle$
val mini-ss = $@\{simpset\}$ *addsimps* $@\{thms\}$ *mini-simps*;
val mini-tac = *rtac* $@\{thm\}$ *ccontr* *THEN'* *asm-full-simp-tac mini-ss*;
 $\rangle\rangle$
end

13 First-Order Logic: the 'if' example

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

constdefs

if :: $[o, o, o] \Rightarrow o$
if (*P*, *Q*, *R*) == *P* & *Q* | \sim *P* & *R*

lemma *ifI*:

$[[P \Rightarrow Q; \sim P \Rightarrow R]] \Rightarrow \text{if}(P, Q, R)$

apply (*simp add: if-def, blast*)

done

lemma *ifE*:

$[[\text{if}(P, Q, R); [[P; Q]] \Rightarrow S; [[\sim P; R]] \Rightarrow S]] \Rightarrow S$

apply (*simp add: if-def, blast*)

done

lemma *if-commute*: $\text{if}(P, \text{if}(Q, A, B), \text{if}(Q, C, D)) \Leftrightarrow \text{if}(Q, \text{if}(P, A, C), \text{if}(P, B, D))$

apply (*rule ifI*)

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*: $\text{if}(P, \text{if}(Q, A, B), \text{if}(Q, C, D)) \Leftrightarrow \text{if}(Q, \text{if}(P, A, C), \text{if}(P, B, D))$

by *blast*

lemma $\text{if}(\text{if}(P, Q, R), A, B) \Leftrightarrow \text{if}(P, \text{if}(Q, A, B), \text{if}(R, A, B))$

by *blast*

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

```
lemma if (if (P, Q, R), A, B)  $\leftrightarrow$  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)  $\leftrightarrow$  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)  $\leftrightarrow$  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
```

14 Example of Declaring an Oracle

```
theory Iff-Oracle
imports FOL
begin
```

14.1 Oracle declaration

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

```
oracle iff-oracle = ⟨⟨
  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, n) =>
      if n > 0 andalso n mod 2 = 0
      then Thm.ctrm-of thy (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 ⟨⟨ Thm.proof-body-of (iff-oracle (@{theory}, 10)) ⟩⟩
```

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 = <<
  Scan.lift OuterParse.nat >> (fn n => fn ctxt =>
    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
```