Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

June 8, 2008

Contents

1 Intuitionistic first-order logic

theory *IFOL*
imports *Pure*
uses
~~/*src*/*Provers*/*splitter.ML*
~~/*src*/*Provers*/*hypsubst.ML*
~~/*src*/*Tools*/*IsaPlanner*/*zipper.ML*
~~/*src*/*Tools*/*IsaPlanner*/*isand.ML*
~~/*src*/*Tools*/*IsaPlanner*/*rw-tools.ML*
~~/*src*/*Tools*/*IsaPlanner*/*rw-inst.ML*
~~/*src*/*Provers*/*eqsubst.ML*
~~/*src*/*Provers*/*quantifier1.ML*
~~/*src*/*Provers*/*project-rule.ML*
~~/*src*/*Tools*/*atomize-elim.ML*

(*fologic.ML*)
(*hypsubstdata.ML*)
(*intprover.ML*)
begin

1.1 Syntax and axiomatic basis

⟨*ML*⟩

global

classes *term*
defaultsort *term*

typedecl *o*

judgment
Trueprop :: *o* => *prop* ((-) *5*)

consts
True :: *o*
False :: *o*

op = :: ['*a*, '*a*] => *o* (**infixl** = *50*)

Not :: *o* => *o* (~ - [*40*] *40*)
op & :: [*o*, *o*] => *o* (**infixr** & *35*)
op | :: [*o*, *o*] => *o* (**infixr** | *30*)
op −−> :: [*o*, *o*] => *o* (**infixr** −−> *25*)
op <−> :: [*o*, *o*] => *o* (**infixr** <−> *25*)

All :: ('*a* => *o*) => *o* (**binder** *ALL* *10*)
Ex :: ('*a* => *o*) => *o* (**binder** *EX* *10*)
Ex1 :: ('*a* => *o*) => *o* (**binder** *EX*! *10*)

abbreviation
not-equal :: ['*a*, '*a*] => *o* (**infixl** ~= *50*) **where**
x ~= *y* == ~ (*x* = *y*)

notation (*xsymbols*)
not-equal (**infixl** ≠ *50*)

notation (*HTML* **output**)
not-equal (**infixl** ≠ *50*)

notation (*xsymbols*)
Not (¬ - [*40*] *40*) **and**
op & (**infixr** ∧ *35*) **and**
op | (**infixr** ∨ *30*) **and**
All (**binder** ∀ *10*) **and**
Ex (**binder** ∃ *10*) **and**
Ex1 (**binder** ∃! *10*) **and**
op −−> (**infixr** ⟶ *25*) **and**
op <−> (**infixr** ⟷ *25*)

notation (*HTML* **output**)
Not (¬ - [*40*] *40*) **and**
op & (**infixr** ∧ *35*) **and**
op | (**infixr** ∨ *30*) **and**
All (**binder** ∀ *10*) **and**
Ex (**binder** ∃ *10*) **and**
Ex1 (**binder** ∃! *10*)

local

finalconsts
False All Ex
op =
op &
op |
op −−>

axioms

refl: $a=a$

conjI: $[\!|\ P;\ \ Q\ |\!] ==> P\&Q$
conjunct1: $P\&Q ==> P$
conjunct2: $P\&Q ==> Q$

disjI1: $P ==> P|Q$
disjI2: $Q ==> P|Q$
disjE: $[\!|\ P|Q;\ \ P ==> R;\ \ Q ==> R\ |\!] ==> R$

impI: $(P ==> Q) ==> P-->Q$
mp: $[\!|\ P-->Q;\ \ P\ |\!] ==> Q$

FalseE: $False ==> P$

allI: $(!!x.\ P(x)) ==> (ALL\ x.\ P(x))$
spec: $(ALL\ x.\ P(x)) ==> P(x)$

exI: $P(x) ==> (EX\ x.\ P(x))$
exE: $[|\ EX\ x.\ P(x);\ !!x.\ P(x) ==> R\ |] ==> R$

eq-reflection: $(x=y) ==> (x==y)$
iff-reflection: $(P<->Q) ==> (P==Q)$

lemmas *strip* = *impI allI*

Thanks to Stephan Merz

theorem *subst*:
assumes *eq*: $a = b$ **and** *p*: $P(a)$
shows $P(b)$
⟨*proof*⟩

defs

True-def: $True == False-->False$
not-def: $\sim P == P-->False$
iff-def: $P<->Q == (P-->Q)\ \&\ (Q-->P)$

ex1-def: $Ex1(P) == EX\ x.\ P(x)\ \&\ (ALL\ y.\ P(y) --> y=x)$

1.2 Lemmas and proof tools

lemma *TrueI*: *True*
⟨*proof*⟩

lemma *conjE*:
assumes *major*: $P\ \&\ Q$
and *r*: $[|\ P;\ Q\ |] ==> R$
shows R
⟨*proof*⟩

lemma *impE*:

assumes *major*: P $--> Q$
and P
and r: Q $==> R$
shows R
⟨*proof*⟩

lemma *allE*:
assumes *major*: $ALL\ x.\ P(x)$
and r: $P(x)$ $==> R$
shows R
⟨*proof*⟩

lemma *all-dupE*:
assumes *major*: $ALL\ x.\ P(x)$
and r: $[|\ P(x);\ ALL\ x.\ P(x)\ |]$ $==> R$
shows R
⟨*proof*⟩

lemma *notI*: $(P ==> False) ==> \sim P$
⟨*proof*⟩

lemma *notE*: $[|\ \sim P;\ \ P\ |] ==> R$
⟨*proof*⟩

lemma *rev-notE*: $[|\ P;\ \sim P\ |] ==> R$
⟨*proof*⟩

lemma *not-to-imp*:
assumes $\sim P$
and r: P $--> False ==> Q$
shows Q
⟨*proof*⟩

lemma *rev-mp*: $[|\ P;\ \ P$ $--> Q\ |] ==> Q$
⟨*proof*⟩

lemma *contrapos*:
assumes *major*: $\sim Q$
and *minor*: $P ==> Q$
shows $\sim P$
⟨*proof*⟩

⟨*ML*⟩

lemma *iffI*: [| *P* ==> *Q*; *Q* ==> *P* |] ==> *P*<->*Q*
⟨*proof*⟩

lemma *iffE*:
assumes *major*: *P* <-> *Q*
and *r*: *P*-->*Q* ==> *Q*-->*P* ==> *R*
shows *R*
⟨*proof*⟩

lemma *iffD1*: [| *P* <-> *Q*; *P* |] ==> *Q*
⟨*proof*⟩

lemma *iffD2*: [| *P* <-> *Q*; *Q* |] ==> *P*
⟨*proof*⟩

lemma *rev-iffD1*: [| *P*; *P* <-> *Q* |] ==> *Q*
⟨*proof*⟩

lemma *rev-iffD2*: [| *Q*; *P* <-> *Q* |] ==> *P*
⟨*proof*⟩

lemma *iff-refl*: *P* <-> *P*
⟨*proof*⟩

lemma *iff-sym*: *Q* <-> *P* ==> *P* <-> *Q*
⟨*proof*⟩

lemma *iff-trans*: [| *P* <-> *Q*; *Q*<-> *R* |] ==> *P* <-> *R*
⟨*proof*⟩

lemma *ex1I*:
P(*a*) ⟹ (!!*x*. *P*(*x*) ==> *x*=*a*) ⟹ *EX*! *x*. *P*(*x*)
⟨*proof*⟩

lemma *ex-ex1I*:
$EX\ x.\ P(x) \Longrightarrow (!!x\ y.\ [|\ P(x);\ P(y)\ |] ==> x{=}y) \Longrightarrow EX!\ x.\ P(x)$
⟨*proof*⟩

lemma *ex1E*:
$EX!\ x.\ P(x) \Longrightarrow (!!x.\ [|\ P(x);\ \ ALL\ y.\ P(y)\ --> y{=}x\ |] ==> R) \Longrightarrow R$
⟨*proof*⟩

⟨*ML*⟩

lemma *conj-cong*:
assumes $P <-> P'$
and $P' ==> Q <-> Q'$
shows $(P\&Q) <-> (P'\&Q')$
⟨*proof*⟩

lemma *conj-cong2*:
assumes $P <-> P'$
and $P' ==> Q <-> Q'$
shows $(Q\&P) <-> (Q'\&P')$
⟨*proof*⟩

lemma *disj-cong*:
assumes $P <-> P'$ **and** $Q <-> Q'$
shows $(P|Q) <-> (P'|Q')$
⟨*proof*⟩

lemma *imp-cong*:
assumes $P <-> P'$
and $P' ==> Q <-> Q'$
shows $(P-->Q) <-> (P'-->Q')$
⟨*proof*⟩

lemma *iff-cong*: $[|\ P <-> P';\ Q <-> Q'\ |] ==> (P<->Q) <-> (P'<->Q')$
⟨*proof*⟩

lemma *not-cong*: $P <-> P' ==> \sim P <-> \sim P'$
⟨*proof*⟩

lemma *all-cong*:
assumes $!!x.\ P(x) <-> Q(x)$
shows $(ALL\ x.\ P(x)) <-> (ALL\ x.\ Q(x))$

⟨*proof*⟩

lemma *ex-cong*:
assumes !!x. $P(x)$ <-> $Q(x)$
shows (EX x. $P(x)$) <-> (EX x. $Q(x)$)
⟨*proof*⟩

lemma *ex1-cong*:
assumes !!x. $P(x)$ <-> $Q(x)$
shows ($EX!$ x. $P(x)$) <-> ($EX!$ x. $Q(x)$)
⟨*proof*⟩

lemma *sym*: $a=b$ ==> $b=a$
⟨*proof*⟩

lemma *trans*: [| $a=b$; $b=c$ |] ==> $a=c$
⟨*proof*⟩

lemma *not-sym*: $b \sim= a$ ==> $a \sim= b$
⟨*proof*⟩

lemma *def-imp-iff*: (A == B) ==> A <-> B
⟨*proof*⟩

lemma *meta-eq-to-obj-eq*: (A == B) ==> $A = B$
⟨*proof*⟩

lemma *meta-eq-to-iff*: x==y ==> x<->y
⟨*proof*⟩

lemma *ssubst*: [| $b = a$; $P(a)$ |] ==> $P(b)$
⟨*proof*⟩

lemma *ex1-equalsE*:
[| $EX!$ x. $P(x)$; $P(a)$; $P(b)$ |] ==> $a=b$
⟨*proof*⟩

lemma *subst-context*: [| $a=b$ |] ==> $t(a)=t(b)$
⟨*proof*⟩

lemma *subst-context2*: $[|\ a=b;\ \ c=d\ |]\ ==>\ \ t(a,c)=t(b,d)$
⟨*proof*⟩

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

lemma *box-equals*: $[|\ a=b;\ \ a=c;\ \ b=d\ |] ==> c=d$
⟨*proof*⟩

lemma *simp-equals*: $[|\ a=c;\ \ b=d;\ \ c=d\ |] ==> a=b$
⟨*proof*⟩

lemma *pred1-cong*: $a=a' ==> P(a) <-> P(a')$
⟨*proof*⟩

lemma *pred2-cong*: $[|\ a=a';\ \ b=b'\ |] ==> P(a,b) <-> P(a',b')$
⟨*proof*⟩

lemma *pred3-cong*: $[|\ a=a';\ \ b=b';\ \ c=c'\ |] ==> P(a,b,c) <-> P(a',b',c')$
⟨*proof*⟩

⟨*ML*⟩

lemma *eq-cong*: $[|\ a = a';\ b = b'\ |] ==> a = b <-> a' = b'$
⟨*proof*⟩

lemma *conj-impE*:
assumes *major*: $(P\&Q)-->S$
and *r*: $P-->(Q-->S) ==> R$
shows R
⟨*proof*⟩

lemma *disj-impE*:
assumes *major*: $(P|Q)-->S$
and *r*: $[|\ P-->S;\ Q-->S\ |] ==> R$
shows R
⟨*proof*⟩

lemma *imp-impE*:
assumes *major*: $(P--{>}Q)--{>}S$
and *r1*: $[|\ P;\ Q--{>}S\ |] =={>}\ Q$
and *r2*: $S =={>}\ R$
shows R
⟨*proof*⟩

lemma *not-impE*:
$^{\sim}P\ --{>}\ S \Longrightarrow (P =={>}\ False) \Longrightarrow (S =={>}\ R) \Longrightarrow R$
⟨*proof*⟩

lemma *iff-impE*:
assumes *major*: $(P{<}{-}{>}Q)--{>}S$
and *r1*: $[|\ P;\ Q--{>}S\ |] =={>}\ Q$
and *r2*: $[|\ Q;\ P--{>}S\ |] =={>}\ P$
and *r3*: $S =={>}\ R$
shows R
⟨*proof*⟩

lemma *all-impE*:
assumes *major*: $(ALL\ x.\ P(x))--{>}S$
and *r1*: $!!x.\ P(x)$
and *r2*: $S =={>}\ R$
shows R
⟨*proof*⟩

lemma *ex-impE*:
assumes *major*: $(EX\ x.\ P(x))--{>}S$
and *r*: $P(x)--{>}S =={>}\ R$
shows R
⟨*proof*⟩

lemma *disj-imp-disj*:
$P|Q \Longrightarrow (P{=}{=}{>}R) \Longrightarrow (Q{=}{=}{>}S) \Longrightarrow R|S$
⟨*proof*⟩

⟨*ML*⟩

lemma *thin-refl*: $!!X.\ [|x{=}x;\ PROP\ W|] =={>}\ PROP\ W$ ⟨*proof*⟩

⟨*ML*⟩

1.3 Intuitionistic Reasoning

lemma *impE'*:
  assumes *1*: P $-->$ Q
    and *2*: Q $==>$ R
    and *3*: P $-->$ Q $==>$ P
  shows R
⟨*proof*⟩

lemma *allE'*:
  assumes *1*: *ALL x*. $P(x)$
    and *2*: $P(x)$ $==>$ *ALL x*. $P(x)$ $==>$ Q
  shows Q
⟨*proof*⟩

lemma *notE'*:
  assumes *1*: ~ P
    and *2*: ~ P $==>$ P
  shows R
⟨*proof*⟩

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

⟨*ML*⟩

lemma *iff-not-sym*: ~ (Q <-> P) $==>$ ~ (P <-> Q)
  ⟨*proof*⟩

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

lemma *eq-commute*: a=b <-> b=a
⟨*proof*⟩

1.4 Atomizing meta-level rules

lemma *atomize-all* [*atomize*]: (!!x. $P(x)$) == *Trueprop* (*ALL x*. $P(x)$)
⟨*proof*⟩

lemma *atomize-imp* [*atomize*]: (A $==>$ B) == *Trueprop* (A $-->$ B)
⟨*proof*⟩

lemma *atomize-eq* [*atomize*]: (x == y) == *Trueprop* (x = y)
⟨*proof*⟩

lemma *atomize-iff* [*atomize*]: $(A == B) == Trueprop\ (A <-> B)$
⟨*proof*⟩

lemma *atomize-conj* [*atomize*]:
includes *meta-conjunction-syntax*
shows $(A \&\& B) == Trueprop\ (A \ \&\ B)$
⟨*proof*⟩

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

1.5 Atomizing elimination rules

⟨*ML*⟩

lemma *atomize-exL*[*atomize-elim*]: $(!!x.\ P(x) ==> Q) == ((EX\ x.\ P(x)) ==> Q)$
⟨*proof*⟩

lemma *atomize-conjL*[*atomize-elim*]: $(A ==> B ==> C) == (A \ \&\ B ==> C)$
⟨*proof*⟩

lemma *atomize-disjL*[*atomize-elim*]: $((A ==> C) ==> (B ==> C) ==> C) == ((A \mid B ==> C) ==> C)$
⟨*proof*⟩

lemma *atomize-elimL*[*atomize-elim*]: $(!!B.\ (A ==> B) ==> B) == Trueprop(A)$
⟨*proof*⟩

1.6 Calculational rules

lemma *forw-subst*: $a = b ==> P(b) ==> P(a)$
⟨*proof*⟩

lemma *back-subst*: $P(a) ==> a = b ==> P(b)$
⟨*proof*⟩

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

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

1.7 "Let" declarations

nonterminals *letbinds letbind*

constdefs

Let :: [*'a*::{}, *'a* => *'b*] => (*'b*::{})
Let(*s*, *f*) == *f*(*s*)

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

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

lemma *LetI*:
assumes !!*x*. *x*=*t* ==> *P*(*u*(*x*))
shows *P*(*let x*=*t in u*(*x*))
⟨*proof*⟩

1.8 Intuitionistic simplification rules

lemma *conj-simps*:
P & *True* <-> *P*
True & *P* <-> *P*
P & *False* <-> *False*
False & *P* <-> *False*
P & *P* <-> *P*
P & *P* & *Q* <-> *P* & *Q*
P & ~*P* <-> *False*
~*P* & *P* <-> *False*
(*P* & *Q*) & *R* <-> *P* & (*Q* & *R*)
⟨*proof*⟩

lemma *disj-simps*:
P | *True* <-> *True*
True | *P* <-> *True*
P | *False* <-> *P*
False | *P* <-> *P*
P | *P* <-> *P*
P | *P* | *Q* <-> *P* | *Q*
(*P* | *Q*) | *R* <-> *P* | (*Q* | *R*)
⟨*proof*⟩

lemma *not-simps*:
~(*P*|*Q*) <-> ~*P* & ~*Q*
~ *False* <-> *True*
~ *True* <-> *False*
⟨*proof*⟩

lemma *imp-simps*:
(*P* −−> *False*) <−> ~*P*
(*P* −−> *True*) <−> *True*
(*False* −−> *P*) <−> *True*
(*True* −−> *P*) <−> *P*
(*P* −−> *P*) <−> *True*
(*P* −−> ~*P*) <−> ~*P*
⟨*proof*⟩

lemma *iff-simps*:
(*True* <−> *P*) <−> *P*
(*P* <−> *True*) <−> *P*
(*P* <−> *P*) <−> *True*
(*False* <−> *P*) <−> ~*P*
(*P* <−> *False*) <−> ~*P*
⟨*proof*⟩

lemma *quant-simps*:
!!*P*. (*ALL x*. *P*) <−> *P*
(*ALL x*. *x*=*t* −−> *P*(*x*)) <−> *P*(*t*)
(*ALL x*. *t*=*x* −−> *P*(*x*)) <−> *P*(*t*)
!!*P*. (*EX x*. *P*) <−> *P*
EX x. *x*=*t*
EX x. *t*=*x*
(*EX x*. *x*=*t* & *P*(*x*)) <−> *P*(*t*)
(*EX x*. *t*=*x* & *P*(*x*)) <−> *P*(*t*)
⟨*proof*⟩

lemma *distrib-simps*:
P & (*Q* | *R*) <−> *P*&*Q* | *P*&*R*
(*Q* | *R*) & *P* <−> *Q*&*P* | *R*&*P*
(*P* | *Q* −−> *R*) <−> (*P* −−> *R*) & (*Q* −−> *R*)
⟨*proof*⟩

Conversion into rewrite rules

lemma *P-iff-F*: ~*P* ==> (*P* <−> *False*) ⟨*proof*⟩
lemma *iff-reflection-F*: ~*P* ==> (*P* == *False*) ⟨*proof*⟩

lemma *P-iff-T*: *P* ==> (*P* <−> *True*) ⟨*proof*⟩
lemma *iff-reflection-T*: *P* ==> (*P* == *True*) ⟨*proof*⟩

More rewrite rules

lemma *conj-commute*: *P*&*Q* <−> *Q*&*P* ⟨*proof*⟩
lemma *conj-left-commute*: *P*&(*Q*&*R*) <−> *Q*&(*P*&*R*) ⟨*proof*⟩
lemmas *conj-comms* = *conj-commute conj-left-commute*

lemma *disj-commute*: *P*|*Q* <−> *Q*|*P* ⟨*proof*⟩

lemma *disj-left-commute*: $P|(Q|R) <-> Q|(P|R)$ ⟨*proof*⟩
lemmas *disj-comms* = *disj-commute disj-left-commute*

lemma *conj-disj-distribL*: $P\&(Q|R) <-> (P\&Q \mid P\&R)$ ⟨*proof*⟩
lemma *conj-disj-distribR*: $(P|Q)\&R <-> (P\&R \mid Q\&R)$ ⟨*proof*⟩

lemma *disj-conj-distribL*: $P|(Q\&R) <-> (P|Q) \,\&\, (P|R)$ ⟨*proof*⟩
lemma *disj-conj-distribR*: $(P\&Q)|R <-> (P|R) \,\&\, (Q|R)$ ⟨*proof*⟩

lemma *imp-conj-distrib*: $(P --> (Q\&R)) <-> (P-->Q) \,\&\, (P-->R)$ ⟨*proof*⟩
lemma *imp-conj*: $((P\&Q)-->R) \;\; <-> (P --> (Q --> R))$ ⟨*proof*⟩
lemma *imp-disj*: $(P|Q --> R) \;\; <-> (P-->R) \,\&\, (Q-->R)$ ⟨*proof*⟩

lemma *de-Morgan-disj*: $(\sim(P \mid Q)) <-> (\sim P \,\&\, \sim Q)$ ⟨*proof*⟩

lemma *not-ex*: $(\sim (EX\ x.\ P(x))) <-> (ALL\ x.\sim P(x))$ ⟨*proof*⟩
lemma *imp-ex*: $((EX\ x.\ P(x)) --> Q) <-> (ALL\ x.\ P(x) --> Q)$ ⟨*proof*⟩

lemma *ex-disj-distrib*:
$(EX\ x.\ P(x) \mid Q(x)) <-> ((EX\ x.\ P(x)) \mid (EX\ x.\ Q(x)))$ ⟨*proof*⟩

lemma *all-conj-distrib*:
$(ALL\ x.\ P(x) \,\&\, Q(x)) <-> ((ALL\ x.\ P(x)) \,\&\, (ALL\ x.\ Q(x)))$ ⟨*proof*⟩

1.9 Legacy ML bindings

⟨*ML*⟩

end

2 Classical first-order logic

theory *FOL*
imports *IFOL*
uses
~~/*src*/*Provers*/*classical.ML*
~~/*src*/*Provers*/*blast.ML*
~~/*src*/*Provers*/*clasimp.ML*
~~/*src*/*Tools*/*induct.ML*
(*cladata.ML*)
(*blastdata.ML*)
(*simpdata.ML*)
begin

2.1 The classical axiom

axioms
classical: $(\sim P ==> P) ==> P$

2.2 Lemmas and proof tools

lemma *ccontr*: $(\neg\ P \Longrightarrow \mathit{False}) \Longrightarrow P$
⟨*proof*⟩

lemma *disjCI*: $(\sim Q \Longrightarrow P) \Longrightarrow P|Q$
⟨*proof*⟩

lemma *ex-classical*:
assumes *r*: $\sim(EX\ x.\ P(x)) \Longrightarrow P(a)$
shows $EX\ x.\ P(x)$
⟨*proof*⟩

lemma *exCI*:
assumes *r*: $ALL\ x.\ \sim P(x) \Longrightarrow P(a)$
shows $EX\ x.\ P(x)$
⟨*proof*⟩

lemma *excluded-middle*: $\sim P \mid P$
⟨*proof*⟩

⟨*ML*⟩

lemma *case-split-thm*:
assumes *r1*: $P \Longrightarrow Q$
and *r2*: $\sim P \Longrightarrow Q$
shows Q
⟨*proof*⟩

lemmas *case-split* = *case-split-thm* [*case-names True False*]

⟨*ML*⟩

lemma *impCE*:
assumes *major*: $P--{>}Q$
and *r1*: $\sim P \Longrightarrow R$
and *r2*: $Q \Longrightarrow R$
shows R
⟨*proof*⟩

lemma *impCE'*:
assumes *major*: $P-->Q$
and *r1*: Q ==> R
and *r2*: ~P ==> R
shows R
⟨*proof*⟩

lemma *notnotD*: ~~P ==> P
⟨*proof*⟩

lemma *contrapos2*: [| Q; ~ P ==> ~ Q |] ==> P
⟨*proof*⟩

lemma *iffCE*:
assumes *major*: $P<->Q$
and *r1*: [| P; Q |] ==> R
and *r2*: [| ~P; ~Q |] ==> R
shows R
⟨*proof*⟩

lemma *alt-ex1E*:
assumes *major*: *EX*! x. $P(x)$
and r: !!x. [| $P(x)$; *ALL* y y'. $P(y)$ & $P(y')$ --> y=y' |] ==> R
shows R
⟨*proof*⟩

lemma *imp-elim*: P --> Q ==> (~ R ==> P) ==> (Q ==> R) ==> R
⟨*proof*⟩

lemma *swap*: ~ P ==> (~ R ==> P) ==> R
⟨*proof*⟩

⟨*ML*⟩

lemma *ex1-functional*: [| *EX*! z. $P(a,z)$; $P(a,b)$; $P(a,c)$ |] ==> b = c
⟨*proof*⟩

lemma *True-implies-equals*: (*True* ==> *PROP P*) == *PROP P*
⟨*proof*⟩

lemma *uncurry*: $P \dashrightarrow Q \dashrightarrow R \Longrightarrow P \;\&\; Q \dashrightarrow R$
⟨*proof*⟩

lemma *iff-allI*: $(!!x.\ P(x) \leftrightarrow Q(x)) \Longrightarrow (ALL\ x.\ P(x)) \leftrightarrow (ALL\ x.\ Q(x))$
⟨*proof*⟩

lemma *iff-exI*: $(!!x.\ P(x) \leftrightarrow Q(x)) \Longrightarrow (EX\ x.\ P(x)) \leftrightarrow (EX\ x.\ Q(x))$
⟨*proof*⟩

lemma *all-comm*: $(ALL\ x\ y.\ P(x,y)) \leftrightarrow (ALL\ y\ x.\ P(x,y))$ ⟨*proof*⟩

lemma *ex-comm*: $(EX\ x\ y.\ P(x,y)) \leftrightarrow (EX\ y\ x.\ P(x,y))$ ⟨*proof*⟩

lemma *cases-simp*: $(P \dashrightarrow Q) \;\&\; (\sim P \dashrightarrow Q) \leftrightarrow Q$ ⟨*proof*⟩

lemma *int-ex-simps*:
$!!P\ Q.\ (EX\ x.\ P(x) \;\&\; Q) \leftrightarrow (EX\ x.\ P(x)) \;\&\; Q$
$!!P\ Q.\ (EX\ x.\ P \;\&\; Q(x)) \leftrightarrow P \;\&\; (EX\ x.\ Q(x))$
$!!P\ Q.\ (EX\ x.\ P(x) \mid Q) \leftrightarrow (EX\ x.\ P(x)) \mid Q$
$!!P\ Q.\ (EX\ x.\ P \mid Q(x)) \leftrightarrow P \mid (EX\ x.\ Q(x))$
⟨*proof*⟩

lemma *cla-ex-simps*:
$!!P\ Q.\ (EX\ x.\ P(x) \dashrightarrow Q) \leftrightarrow (ALL\ x.\ P(x)) \dashrightarrow Q$
$!!P\ Q.\ (EX\ x.\ P \dashrightarrow Q(x)) \leftrightarrow P \dashrightarrow (EX\ x.\ Q(x))$
⟨*proof*⟩

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

lemma *int-all-simps*:
$!!P\ Q.\ (ALL\ x.\ P(x) \;\&\; Q) \leftrightarrow (ALL\ x.\ P(x)) \;\&\; Q$
$!!P\ Q.\ (ALL\ x.\ P \;\&\; Q(x)) \leftrightarrow P \;\&\; (ALL\ x.\ Q(x))$
$!!P\ Q.\ (ALL\ x.\ P(x) \dashrightarrow Q) \leftrightarrow (EX\ x.\ P(x)) \dashrightarrow Q$
$!!P\ Q.\ (ALL\ x.\ P \dashrightarrow Q(x)) \leftrightarrow P \dashrightarrow (ALL\ x.\ Q(x))$
⟨*proof*⟩

lemma *cla-all-simps*:
!!$P\ Q.\ (ALL\ x.\ P(x) \mid Q) <-> (ALL\ x.\ P(x)) \mid Q$
!!$P\ Q.\ (ALL\ x.\ P \mid Q(x)) <-> P \mid (ALL\ x.\ Q(x))$
⟨*proof*⟩

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

lemma *imp-disj1*: $(P-->Q) \mid R <-> (P-->Q \mid R)$ ⟨*proof*⟩
lemma *imp-disj2*: $Q \mid (P-->R) <-> (P-->Q \mid R)$ ⟨*proof*⟩

lemma *de-Morgan-conj*: $(\sim(P\ \&\ Q)) <-> (\sim P \mid \sim Q)$ ⟨*proof*⟩

lemma *not-imp*: $\sim(P\ -->\ Q) <-> (P\ \&\ \sim Q)$ ⟨*proof*⟩
lemma *not-iff*: $\sim(P <-> Q) <-> (P <-> \sim Q)$ ⟨*proof*⟩

lemma *not-all*: $(\sim\ (ALL\ x.\ P(x))) <-> (EX\ x.\sim P(x))$ ⟨*proof*⟩
lemma *imp-all*: $((ALL\ x.\ P(x))\ -->\ Q) <-> (EX\ x.\ P(x)\ -->\ Q)$ ⟨*proof*⟩

lemmas *meta-simps* =
triv-forall-equality
True-implies-equals

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

lemma *notFalseI*: $\sim False$ ⟨*proof*⟩

lemma *cla-simps-misc*:
$\sim(P\&Q) <-> \sim P \mid \sim Q$
$P \mid \sim P$
$\sim P \mid P$
$\sim\ \sim\ P <-> P$
$(\sim P\ -->\ P) <-> P$
$(\sim P <-> \sim Q) <-> (P<->Q)$ ⟨*proof*⟩

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

⟨*ML*⟩

2.3 Other simple lemmas

lemma [*simp*]: $((P--{>}R) <{-}> (Q--{>}R)) <{-}> ((P<{-}>Q) \mid R)$
⟨*proof*⟩

lemma [*simp*]: $((P--{>}Q) <{-}> (P--{>}R)) <{-}> (P --{>} (Q<{-}>R))$
⟨*proof*⟩

lemma *not-disj-iff-imp*: $\sim P \mid Q <{-}> (P--{>}Q)$
⟨*proof*⟩

lemma *conj-mono*: $[\!|\ P1--{>}Q1;\ P2--{>}Q2\ |\!] =={>} (P1 \& P2) --{>} (Q1 \& Q2)$
⟨*proof*⟩

lemma *disj-mono*: $[\!|\ P1--{>}Q1;\ P2--{>}Q2\ |\!] =={>} (P1 | P2) --{>} (Q1 | Q2)$
⟨*proof*⟩

lemma *imp-mono*: $[\!|\ Q1--{>}P1;\ P2--{>}Q2\ |\!] =={>} (P1--{>}P2)--{>}(Q1--{>}Q2)$
⟨*proof*⟩

lemma *imp-refl*: $P--{>}P$
⟨*proof*⟩

lemma *ex-mono*: $(!!x.\ P(x) --{>} Q(x)) =={>} (EX\ x.\ P(x)) --{>} (EX\ x.\ Q(x))$
⟨*proof*⟩

lemma *all-mono*: $(!!x.\ P(x) --{>} Q(x)) =={>} (ALL\ x.\ P(x)) --{>} (ALL\ x.\ Q(x))$
⟨*proof*⟩

2.4 Proof by cases and induction

Proper handling of non-atomic rule statements.

constdefs
induct-forall **where** $induct\text{-}forall(P) == \forall x.\ P(x)$
induct-implies **where** $induct\text{-}implies(A,\ B) == A \longrightarrow B$
induct-equal **where** $induct\text{-}equal(x,\ y) == x = y$
induct-conj **where** $induct\text{-}conj(A,\ B) == A \wedge B$

lemma *induct-forall-eq*: $(!!x.\ P(x)) == Trueprop(induct\text{-}forall(\lambda x.\ P(x)))$
⟨*proof*⟩

lemma *induct-implies-eq*: $(A =={>} B) == Trueprop(induct\text{-}implies(A,\ B))$
⟨*proof*⟩

lemma *induct-equal-eq*: $(x == y) == Trueprop(induct\text{-}equal(x,\ y))$

⟨*proof*⟩

lemma *induct-conj-eq*:
includes *meta-conjunction-syntax*
shows (*A* &&& *B*) == *Trueprop*(*induct-conj*(*A*, *B*))
⟨*proof*⟩

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

hide *const induct-forall induct-implies induct-equal induct-conj*

Method setup.

⟨*ML*⟩
declare *case-split* [*cases type*: *o*]

end