Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

April 19, 2009

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*/*Tools*/*eqsubst.ML*
~~/*src*/*Provers*/*quantifier1.ML*

~~/src/Tools/intuitionistic.ML
~~/src/Tools/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** $\neq$ *50*)

notation (*xsymbols*)
Not ($\neg$ - [*40*] *40*) **and**
op & (**infixr** $\wedge$ *35*) **and**
op | (**infixr** $\vee$ *30*) **and**
All (**binder** $\forall$ *10*) **and**
Ex (**binder** $\exists$ *10*) **and**
Ex1 (**binder** $\exists!$ *10*) **and**
op −−> (**infixr** $\longrightarrow$ *25*) **and**
op <−> (**infixr** $\longleftrightarrow$ *25*)

notation (*HTML* **output**)
Not ($\neg$ - [*40*] *40*) **and**
op & (**infixr** $\wedge$ *35*) **and**
op | (**infixr** $\vee$ *30*) **and**
All (**binder** $\forall$ *10*) **and**
Ex (**binder** $\exists$ *10*) **and**
Ex1 (**binder** $\exists!$ *10*)

local

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

axioms

refl: $a{=}a$
subst: $a{=}b \Longrightarrow P(a) \Longrightarrow P(b)$

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: $[|\ P \longrightarrow Q;\ \ P\ |] \Longrightarrow Q$

FalseE: $False \Longrightarrow P$

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

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

axioms

eq-reflection: $(x=y) \quad \Longrightarrow (x==y)$
iff-reflection: $(P \longleftrightarrow Q) \Longrightarrow (P==Q)$

lemmas *strip* = *impI allI*

defs

True-def: $True \ == False \longrightarrow False$
not-def: $\sim P \quad == P \longrightarrow False$
iff-def: $P \longleftrightarrow Q == (P \longrightarrow Q)\ \&\ (Q \longrightarrow P)$

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

1.2 Lemmas and proof tools

lemma *TrueI*: *True*
⟨*proof*⟩

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

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

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

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

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

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

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

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

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

lemma *contrapos*:
assumes *major*: $\sim Q$
and *minor*: $P \Longrightarrow Q$
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*:
$P(a) \Longrightarrow (!!x.\ P(x)$ ==> x=$a) \Longrightarrow EX!\ x.\ P(x)$

⟨*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*⟩

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*:
~P −−> S ⟹ (P ==> *False*) ⟹ (S ==> R) ⟹ 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*:
$P|Q$ ⟹ (P==>R) ⟹ (Q==>S) ⟹ $R|S$
⟨*proof*⟩

⟨*ML*⟩

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

⟨*ML*⟩

1.3 Intuitionistic Reasoning

⟨*ML*⟩

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*: $\sim$ P
    and *2*: $\sim$ 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*: $\sim$ $(Q$ $<->$ $P)$ $==>$ $\sim$ $(P$ $<->$ $Q)$
  ⟨*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*]: (*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* | *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*

$(False \longrightarrow P) \longleftrightarrow True$
$(True \longrightarrow P) \longleftrightarrow P$
$(P \longrightarrow P) \longleftrightarrow True$
$(P \longrightarrow \sim P) \longleftrightarrow \sim P$
⟨*proof*⟩

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

lemma *quant-simps*:
$!!P.\ (ALL\ x.\ P) \longleftrightarrow P$
$(ALL\ x.\ x{=}t \longrightarrow P(x)) \longleftrightarrow P(t)$
$(ALL\ x.\ t{=}x \longrightarrow P(x)) \longleftrightarrow P(t)$
$!!P.\ (EX\ x.\ P) \longleftrightarrow P$
$EX\ x.\ x{=}t$
$EX\ x.\ t{=}x$
$(EX\ x.\ x{=}t\ \&\ P(x)) \longleftrightarrow P(t)$
$(EX\ x.\ t{=}x\ \&\ P(x)) \longleftrightarrow P(t)$
⟨*proof*⟩

lemma *distrib-simps*:
$P\ \&\ (Q \mid R) \longleftrightarrow P\&Q \mid P\&R$
$(Q \mid R)\ \&\ P \longleftrightarrow Q\&P \mid R\&P$
$(P \mid Q \longrightarrow R) \longleftrightarrow (P \longrightarrow R)\ \&\ (Q \longrightarrow R)$
⟨*proof*⟩

Conversion into rewrite rules

lemma *P-iff-F*: $\sim P \Longrightarrow (P \longleftrightarrow False)$ ⟨*proof*⟩
lemma *iff-reflection-F*: $\sim P \Longrightarrow (P == False)$ ⟨*proof*⟩

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

More rewrite rules

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

lemma *disj-commute*: $P|Q \longleftrightarrow Q|P$ ⟨*proof*⟩
lemma *disj-left-commute*: $P|(Q|R) \longleftrightarrow Q|(P|R)$ ⟨*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*: $(P \dashrightarrow (Q\&R)) <-> (P \dashrightarrow Q) \ \& \ (P \dashrightarrow R)$ ⟨*proof*⟩
lemma *imp-conj*: $((P\&Q) \dashrightarrow R) \quad <-> (P \dashrightarrow (Q \dashrightarrow R))$ ⟨*proof*⟩
lemma *imp-disj*: $(P|Q \dashrightarrow R) \quad <-> (P \dashrightarrow R) \ \& \ (Q \dashrightarrow R)$ ⟨*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*: $((EX\ x.\ P(x)) \dashrightarrow Q) <-> (ALL\ x.\ P(x) \dashrightarrow Q)$ ⟨*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: $(\sim P \Longrightarrow P) \Longrightarrow P$

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*: $\sim(\mathit{EX}\ x.\ P(x)) \Longrightarrow P(a)$
shows $\mathit{EX}\ x.\ P(x)$
⟨*proof*⟩

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

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

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

⟨*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*: $Q \Longrightarrow R$
and *r2*: $\sim P \Longrightarrow R$

shows R
⟨*proof*⟩

lemma *notnotD*: $\sim\sim P \Longrightarrow P$
⟨*proof*⟩

lemma *contrapos2*: $[\!|\ Q;\ \sim P \Longrightarrow \sim Q\ |\!] \Longrightarrow P$
⟨*proof*⟩

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

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

lemma *imp-elim*: $P \dashrightarrow Q \Longrightarrow (\sim R \Longrightarrow P) \Longrightarrow (Q \Longrightarrow R) \Longrightarrow R$
⟨*proof*⟩

lemma *swap*: $\sim P \Longrightarrow (\sim R \Longrightarrow P) \Longrightarrow R$
⟨*proof*⟩

3 Classical Reasoner

⟨*ML*⟩

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

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

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

lemma *iff-allI*: $(!!x.\ P(x) <-> Q(x)) ==> (ALL\ x.\ P(x)) <-> (ALL\ x.\ Q(x))$
⟨*proof*⟩

lemma *iff-exI*: $(!!x.\ P(x) <-> Q(x)) ==> (EX\ x.\ P(x)) <-> (EX\ x.\ Q(x))$
⟨*proof*⟩

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

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

lemma *cases-simp*: $(P\ --> Q)\ \&\ (\sim P\ --> Q) <-> Q$ ⟨*proof*⟩

lemma *int-ex-simps*:
$!!P\ Q.\ (EX\ x.\ P(x)\ \&\ Q) <-> (EX\ x.\ P(x))\ \&\ Q$
$!!P\ Q.\ (EX\ x.\ P\ \&\ Q(x)) <-> P\ \&\ (EX\ x.\ Q(x))$
$!!P\ Q.\ (EX\ x.\ P(x)\ |\ Q) <-> (EX\ x.\ P(x))\ |\ Q$
$!!P\ Q.\ (EX\ x.\ P\ |\ Q(x)) <-> P\ |\ (EX\ x.\ Q(x))$
⟨*proof*⟩

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

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

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

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

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

lemma *imp-disj1*: *(P−−>Q) | R <−> (P−−>Q | R)* ⟨*proof*⟩
lemma *imp-disj2*: *Q | (P−−>R) <−> (P−−>Q | R)* ⟨*proof*⟩

lemma *de-Morgan-conj*: *(~(P & Q)) <−> (~P | ~Q)* ⟨*proof*⟩

lemma *not-imp*: *~(P −−> Q) <−> (P & ~Q)* ⟨*proof*⟩
lemma *not-iff*: *~(P <−> Q) <−> (P <−> ~Q)* ⟨*proof*⟩

lemma *not-all*: *(~ (ALL x. P(x))) <−> (EX x.~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*: *~False* ⟨*proof*⟩

lemma *cla-simps-misc*:
~(P&Q) <−> ~P | ~Q
P | ~P
~P | P
~ ~ P <−> P
(~P −−> P) <−> P
(~P <−> ~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*⟩

3.1 Other simple lemmas

lemma [*simp*]: *((P−−>R) <−> (Q−−>R)) <−> ((P<−>Q) | R)*
⟨*proof*⟩

lemma [*simp*]: *((P−−>Q) <−> (P−−>R)) <−> (P −−> (Q<−>R))*

⟨*proof*⟩

lemma *not-disj-iff-imp*: ~*P* | *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*⟩

3.2 Proof by cases and induction

Proper handling of non-atomic rule statements.

constdefs
induct-forall **where** *induct-forall*(*P*) == ∀ *x*. *P*(*x*)
induct-implies **where** *induct-implies*(*A*, *B*) == *A* ⟶ *B*
induct-equal **where** *induct-equal*(*x*, *y*) == *x* = *y*
induct-conj **where** *induct-conj*(*A*, *B*) == *A* ∧ *B*

lemma *induct-forall-eq*: (!!*x*. *P*(*x*)) == *Trueprop*(*induct-forall*(λ*x*. *P*(*x*)))
⟨*proof*⟩

lemma *induct-implies-eq*: (*A* ==> *B*) == *Trueprop*(*induct-implies*(*A*, *B*))
⟨*proof*⟩

lemma *induct-equal-eq*: (*x* == *y*) == *Trueprop*(*induct-equal*(*x*, *y*))
⟨*proof*⟩

lemma *induct-conj-eq*: (*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