Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

November 22, 2007

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

1.1 Syntax and axiomatic basis

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: $(x{=}y) \quad ==> (x{==}y)$
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: ~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*) ==> ~P
⟨*proof*⟩

lemma *notE*: [| ~P; P |] ==> R
⟨*proof*⟩

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

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

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

lemma *contrapos*:
assumes *major*: ~Q
and *minor*: P ==> Q
shows ~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) ⟹ (!!x y. [| P(x); P(y) |] ==> x=y) ⟹ EX! x. P(x)
⟨*proof*⟩

lemma *ex1E*:
EX! x. P(x) ⟹ (!!x. [| P(x); ALL y. P(y) --> y=x |] ==> R) ⟹ 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*: $b\ \tilde{}= a ==> a\ \tilde{}= b$
⟨*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 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

1.6 "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*⟩

1.7 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*: $(\neg\ P \Longrightarrow \mathit{False}) \Longrightarrow P$
  $\langle proof \rangle$

lemma *disjCI*: $(\sim Q ==> P) ==> P|Q$
  $\langle proof \rangle$

lemma *ex-classical*:
  assumes r: $\sim(EX\ x.\ P(x)) ==> P(a)$
  shows $EX\ x.\ P(x)$
  $\langle proof \rangle$

lemma *exCI*:
  assumes r: $ALL\ x.\ \sim P(x) ==> P(a)$
  shows $EX\ x.\ P(x)$
  $\langle proof \rangle$

lemma *excluded-middle*: $\sim P\ |\ P$
  $\langle proof \rangle$

$\langle ML \rangle$

lemma *case-split-thm*:
  assumes $r1$: $P ==> Q$
    and $r2$: $\sim P ==> Q$

shows Q
⟨*proof*⟩

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

⟨*ML*⟩

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

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

lemma *notnotD*: $\sim\sim P ==> P$
⟨*proof*⟩

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

lemma *iffCE*:
assumes *major*: $P{<}{-}{>}Q$
and *r1*: $[|\ P;\ Q\ |] ==> R$
and *r2*: $[|\ \sim P; \sim Q\ |] ==> R$
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*⟩

⟨*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* --> *Q* --> *R* ==> *P* & *Q* --> *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*⟩

⟨*ML*⟩

2.3 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*⟩

2.4 Proof by cases and induction

Proper handling of non-atomic rule statements.

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

lemma *induct-forall-eq*: $(!!x.\ P(x)) ==$ *Trueprop*(*induct-forall*$(\lambda x.\ P(x))$)
⟨*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*:
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