Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

October 1, 2005

Contents

1 Intuitionistic first-order logic

theory *IFOL*
imports *Pure*
uses (*IFOL-lemmas.ML*) (*fologic.ML*) (*hypsubstdata.ML*) (*intprover.ML*)
begin

1.1 Syntax and axiomatic basis

global

classes *term*
final-consts term-class
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*)

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

syntax (*xsymbols*)
Not :: *o* => *o* (¬ - [*40*] *40*)
op & :: [*o*, *o*] => *o* (**infixr** ∧ *35*)
op | :: [*o*, *o*] => *o* (**infixr** ∨ *30*)
ALL :: [*idts*, *o*] => *o* ((*3*∀ -./ -) [*0*, *10*] *10*)
EX :: [*idts*, *o*] => *o* ((*3*∃ -./ -) [*0*, *10*] *10*)
EX! :: [*idts*, *o*] => *o* ((*3*∃!-./ -) [*0*, *10*] *10*)
-not-equal :: ['*a*, '*a*] => *o* (**infixl** ≠ *50*)
op --> :: [*o*, *o*] => *o* (**infixr** ⟶ *25*)
op <-> :: [*o*, *o*] => *o* (**infixr** ⟷ *25*)

syntax (*HTML* **output**)
Not :: *o* => *o* (¬ - [*40*] *40*)
op & :: [*o*, *o*] => *o* (**infixr** ∧ *35*)
op | :: [*o*, *o*] => *o* (**infixr** ∨ *30*)
ALL :: [*idts*, *o*] => *o* ((*3*∀ -./ -) [*0*, *10*] *10*)
EX :: [*idts*, *o*] => *o* ((*3*∃ -./ -) [*0*, *10*] *10*)
EX! :: [*idts*, *o*] => *o* ((*3*∃!-./ -) [*0*, *10*] *10*)
-not-equal :: ['*a*, '*a*] => *o* (**infixl** ≠ *50*)

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: $(P<->Q) ==> (P==Q)$

Thanks to Stephan Merz

theorem *subst*:
  assumes *eq*: $a = b$ **and** *p*: $P(a)$
  shows $P(b)$
proof $-$
  from *eq* **have** *meta*: $a \equiv b$

by (*rule eq-reflection*)
from *p* **show** *?thesis*
by (*unfold meta*)
qed

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

use *IFOL-lemmas.ML*

use *fologic.ML*
use *hypsubstdata.ML*
setup *hypsubst-setup*
use *intprover.ML*

1.3 Intuitionistic Reasoning

lemma *impE'*:
assumes *1*: *P* --> *Q*
and *2*: *Q* ==> *R*
and *3*: *P* --> *Q* ==> *P*
shows *R*
proof −
from *3* **and** *1* **have** *P* **.**
with *1* **have** *Q* **by** (*rule impE*)
with *2* **show** *R* **.**
qed

lemma *allE'*:
assumes *1*: *ALL x*. *P*(*x*)
and *2*: *P*(*x*) ==> *ALL x*. *P*(*x*) ==> *Q*
shows *Q*
proof −
from *1* **have** *P*(*x*) **by** (*rule spec*)
from *this* **and** *1* **show** *Q* **by** (*rule 2*)
qed

lemma *notE'*:
assumes *1*: ~ *P*

```
    and 2: ~ P ==> P
  shows R
proof -
  from 2 and 1 have P .
  with 1 show R by (rule notE)
qed

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

setup ⟪
  [ContextRules.addSWrapper (fn tac => hyp-subst-tac ORELSE' tac)]
⟫


lemma iff-not-sym: ~ (Q <-> P) ==> ~ (P <-> Q)
  by iprover

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


lemma eq-commute: a=b <-> b=a
apply (rule iffI)
apply (erule sym)+
done
```

1.4 Atomizing meta-level rules

```
lemma atomize-all [atomize]: (!!x. P(x)) == Trueprop (ALL x. P(x))
proof
  assume !!x. P(x)
  show ALL x. P(x) ..
next
  assume ALL x. P(x)
  thus !!x. P(x) ..
qed

lemma atomize-imp [atomize]: (A ==> B) == Trueprop (A --> B)
proof
  assume A ==> B
  thus A --> B ..
next
  assume A --> B and A
  thus B by (rule mp)
qed
```

lemma *atomize-eq* [*atomize*]: $(x == y) == \mathit{Trueprop}\ (x = y)$
proof
  assume $x == y$
  show $x = y$ **by** (*unfold prems*) (*rule refl*)
next
  assume $x = y$
  thus $x == y$ **by** (*rule eq-reflection*)
qed

lemma *atomize-conj* [*atomize*]:
  $(!!C.\ (A ==> B ==> \mathit{PROP}\ C) ==> \mathit{PROP}\ C) == \mathit{Trueprop}\ (A\ \&\ B)$
proof
  assume $!!C.\ (A ==> B ==> \mathit{PROP}\ C) ==> \mathit{PROP}\ C$
  show $A\ \&\ B$ **by** (*rule conjI*)
next
  fix C
  assume $A\ \&\ B$
  assume $A ==> B ==> \mathit{PROP}\ C$
  thus $\mathit{PROP}\ C$
  proof *this*
    show A **by** (*rule conjunct1*)
    show B **by** (*rule conjunct2*)
  qed
qed

lemmas [*symmetric*, *rulify*] = *atomize-all atomize-imp*

1.5 Calculational rules

lemma *forw-subst*: $a = b ==> P(b) ==> P(a)$
  by (*rule ssubst*)

lemma *back-subst*: $P(a) ==> a = b ==> P(b)$
  by (*rule subst*)

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 *prem*: (!!*x*. *x*=*t* ==> *P*(*u*(*x*)))
  shows *P*(*let x*=*t in u*(*x*))
apply (*unfold Let-def*)
apply (*rule refl* [*THEN prem*])
done

ML
⟪
val Let-def = *thm Let-def*;
val LetI = *thm LetI*;
⟫

end

2 Classical first-order logic

theory *FOL*
imports *IFOL*
uses (*FOL-lemmas1.ML*) (*cladata.ML*) (*blastdata.ML*) (*simpdata.ML*)
  (*eqrule-FOL-data.ML*)
  (~~/*src*/*Provers*/*eqsubst.ML*)
begin

2.1 The classical axiom

axioms
  classical: (~*P* ==> *P*) ==> *P*

2.2 Lemmas and proof tools

use *FOL-lemmas1.ML*
theorems *case-split* = *case-split-thm* [*case-names True False*, *cases type*: *o*]

use *cladata.ML*

setup *Cla.setup*
setup *cla-setup*
setup *case-setup*

use *blastdata.ML*
setup *Blast.setup*

lemma *ex1-functional*: [| *EX*! *z*. *P*(*a*,*z*); *P*(*a*,*b*); *P*(*a*,*c*) |] ==> *b* = *c*
by *blast*

ML ⟪
val ex1-functional = *thm ex1-functional*;
⟫

use *simpdata.ML*
setup *simpsetup*
setup *Simplifier.method-setup Splitter.split-modifiers*
setup *Splitter.setup*
setup *Clasimp.setup*

2.3 Lucas Dixon's eqstep tactic

use ~~/*src*/*Provers*/*eqsubst.ML*
use *eqrule-FOL-data.ML*

setup *EQSubstTac.setup*

2.4 Other simple lemmas

lemma [*simp*]: ((*P*-->*R*) <-> (*Q*-->*R*)) <-> ((*P*<->*Q*) | *R*)
by *blast*

lemma [*simp*]: ((*P*-->*Q*) <-> (*P*-->*R*)) <-> (*P* --> (*Q*<->*R*))
by *blast*

lemma *not-disj-iff-imp*: ~*P* | *Q* <-> (*P*-->*Q*)
by *blast*

lemma *conj-mono*: [| *P1*-->*Q1*; *P2*-->*Q2* |] ==> (*P1*&*P2*) --> (*Q1*&*Q2*)
by *fast*

lemma *disj-mono*: [| *P1*-->*Q1*; *P2*-->*Q2* |] ==> (*P1*|*P2*) --> (*Q1*|*Q2*)
by *fast*

lemma *imp-mono*: [| *Q1*-->*P1*; *P2*-->*Q2* |] ==> (*P1*-->*P2*)-->(*Q1*-->*Q2*)
by *fast*

lemma *imp-refl*: $P{-}{-}{>}P$
by (*rule impI*, *assumption*)

lemma *ex-mono*: (!!x. $P(x)$ --> $Q(x)$) ==> ($EX\ x$. $P(x)$) --> ($EX\ x$. $Q(x)$)
by *blast*

lemma *all-mono*: (!!x. $P(x)$ --> $Q(x)$) ==> ($ALL\ x$. $P(x)$) --> ($ALL\ x$. $Q(x)$)
by *blast*

2.5 Proof by cases and induction

Proper handling of non-atomic rule statements.

constdefs
induct-forall :: ('a => o) => o
induct-forall(P) == $\forall\, x$. $P(x)$
induct-implies :: o => o => o
induct-implies(A, B) == A --> B
induct-equal :: 'a => 'a => o
induct-equal(x, y) == $x = y$

lemma *induct-forall-eq*: (!!x. $P(x)$) == *Trueprop*(*induct-forall*(λx. $P(x)$))
by (*simp only*: *atomize-all induct-forall-def*)

lemma *induct-implies-eq*: (A ==> B) == *Trueprop*(*induct-implies*(A, B))
by (*simp only*: *atomize-imp induct-implies-def*)

lemma *induct-equal-eq*: (x == y) == *Trueprop*(*induct-equal*(x, y))
by (*simp only*: *atomize-eq induct-equal-def*)

lemma *induct-impliesI*: (A ==> B) ==> *induct-implies*(A, B)
by (*simp add*: *induct-implies-def*)

lemmas *induct-atomize* = *atomize-conj induct-forall-eq induct-implies-eq induct-equal-eq*
lemmas *induct-rulify1* [*symmetric*, *standard*] = *induct-forall-eq induct-implies-eq induct-equal-eq*
lemmas *induct-rulify2* = *induct-forall-def induct-implies-def induct-equal-def*

lemma *all-conj-eq*: ($ALL\ x$. $P(x)$) & ($ALL\ y$. $Q(y)$) == ($ALL\ x\ y$. $P(x)$ & $Q(y)$)
by *simp*

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

Method setup.

ML ⟪
structure InductMethod = *InductMethodFun*
(*struct*
val dest-concls = *FOLogic.dest-concls*;

```
    val cases-default = thm case-split;
    val local-impI = thm induct-impliesI;
    val conjI = thm conjI;
    val atomize = thms induct-atomize;
    val rulify1 = thms induct-rulify1;
    val rulify2 = thms induct-rulify2;
    val localize = [Thm.symmetric (thm induct-implies-def),
      Thm.symmetric (thm atomize-all), thm all-conj-eq];
  end);
⟫
```

setup *InductMethod.setup*

end