Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

October 1, 2005

Contents

1 Intuitionistic first-order logic **1**
1.1 Syntax and axiomatic basis 1
1.2 Lemmas and proof tools 4
1.3 Intuitionistic Reasoning 4
1.4 Atomizing meta-level rules 5
1.5 Calculational rules 5
1.6 "Let" declarations 5

2 Classical first-order logic **6**
2.1 The classical axiom 6
2.2 Lemmas and proof tools 6
2.3 Lucas Dixon's eqstep tactic 6
2.4 Other simple lemmas 7
2.5 Proof by cases and induction 7

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 :: $o \Rightarrow prop$ (-) 5

consts

True :: o

False :: o

op = :: $['a, 'a] \Rightarrow o$ (**infixl** = 50)

Not :: $o \Rightarrow o$ ($\sim$ - [40] 40)

op & :: $[o, o] \Rightarrow o$ (**infixr** & 35)

op | :: $[o, o] \Rightarrow o$ (**infixr** | 30)

op −−> :: $[o, o] \Rightarrow o$ (**infixr** −−> 25)

op <−> :: $[o, o] \Rightarrow o$ (**infixr** <−> 25)

All :: $('a \Rightarrow o) \Rightarrow o$ (**binder** *ALL* 10)

Ex :: $('a \Rightarrow o) \Rightarrow o$ (**binder** *EX* 10)

Ex1 :: $('a \Rightarrow o) \Rightarrow o$ (**binder** *EX*! 10)

syntax

-not-equal :: $['a, 'a] \Rightarrow o$ (**infixl** $\sim$= 50)

translations

$x \sim= y$ == $\sim (x = y)$

syntax (*xsymbols*)

Not :: $o \Rightarrow o$ ($\neg$ - [40] 40)

op & :: $[o, o] \Rightarrow o$ (**infixr** $\wedge$ 35)

op | :: $[o, o] \Rightarrow o$ (**infixr** $\vee$ 30)

ALL :: $[idts, o] \Rightarrow o$ ((3$\forall$ -./ -) [0, 10] 10)

EX :: $[idts, o] \Rightarrow o$ ((3$\exists$ -./ -) [0, 10] 10)

EX! :: $[idts, o] \Rightarrow o$ ((3$\exists$!-./ -) [0, 10] 10)

-not-equal :: $['a, 'a] \Rightarrow o$ (**infixl** $\neq$ 50)

op −−> :: $[o, o] \Rightarrow o$ (**infixr** $\longrightarrow$ 25)

op <−> :: $[o, o] \Rightarrow o$ (**infixr** $\longleftrightarrow$ 25)

syntax (*HTML* **output**)

Not :: $o \Rightarrow o$ ($\neg$ - [40] 40)

op & :: $[o, o] \Rightarrow o$ (**infixr** $\wedge$ 35)

op | :: $[o, o] \Rightarrow o$ (**infixr** $\vee$ 30)

ALL :: $[idts, o] \Rightarrow o$ ((3$\forall$ -./ -) [0, 10] 10)

EX :: $[idts, o] \Rightarrow o$ ((3$\exists$ -./ -) [0, 10] 10)

EX! :: $[idts, o] \Rightarrow o$ ((3$\exists$!-./ -) [0, 10] 10)

-not-equal :: $['a, 'a] \Rightarrow o$ (**infixl** $\neq$ 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*⟩

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

⟨*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*]: $(!!x.\ P(x)) == \mathit{Trueprop}\ (\mathit{ALL}\ x.\ P(x))$
⟨*proof*⟩

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

lemma *atomize-eq* [*atomize*]: $(x == y) == \mathit{Trueprop}\ (x = y)$
⟨*proof*⟩

lemma *atomize-conj* [*atomize*]:
$(!!C.\ (A ==> B ==> \mathit{PROP}\ C) ==> \mathit{PROP}\ C) == \mathit{Trueprop}\ (A \&\ B)$
⟨*proof*⟩

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

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

syntax
-bind :: $[pttrn, 'a] => \mathit{letbind}$ (*2- =/ -) 10*)
:: *letbind* => *letbinds* (-)
-binds :: $[\mathit{letbind}, \mathit{letbinds}] => \mathit{letbinds}$ (-;/ -)

-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*))
⟨*proof*⟩

⟨*ML*⟩

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

⟨*ML*⟩
theorems *case-split* = *case-split-thm* [*case-names True False*, *cases type*: *o*]

⟨*ML*⟩

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

⟨*ML*⟩

2.3 Lucas Dixon's eqstep tactic

⟨*ML*⟩

2.4 Other simple lemmas

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

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

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

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

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

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

lemma *imp-refl*: $P--{>}P$
⟨*proof*⟩

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

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

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$(A, B) == A \dashrightarrow B$
induct-equal :: $'a => 'a => o$
induct-equal$(x, y) == x = y$

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*: $(x == y) == \mathit{Trueprop}(\mathit{induct\text{-}equal}(x, y))$
⟨*proof*⟩

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

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*: $(ALL\ x.\ P(x))\ \&\ (ALL\ y.\ Q(y)) == (ALL\ x\ y.\ P(x)\ \&\ Q(y))$
⟨*proof*⟩

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

Method setup.

⟨*ML*⟩

end