

# Isabelle/FOL — First-Order Logic

Larry Paulson and Markus Wenzel

December 17, 2025

## Contents

<b>1</b>	<b>Intuitionistic first-order logic</b>	<b>1</b>
1.1	Syntax and axiomatic basis . . . . .	1
1.1.1	Equality . . . . .	1
1.1.2	Propositional logic . . . . .	1
1.1.3	Quantifiers . . . . .	2
1.1.4	Definitions . . . . .	2
1.1.5	Old-style ASCII syntax . . . . .	3
1.2	Lemmas and proof tools . . . . .	3
1.2.1	Sequent-style elimination rules for $\wedge \longrightarrow$ and $\forall$ . . . .	3
1.2.2	Negation rules, which translate between $\neg P$ and $P \longrightarrow False$ . . . . .	4
1.2.3	Modus Ponens Tactics . . . . .	4
1.3	If-and-only-if . . . . .	4
1.3.1	Destruct rules for $\longleftrightarrow$ similar to Modus Ponens . . . .	5
1.4	Unique existence . . . . .	5
1.4.1	$\longleftrightarrow$ congruence rules for simplification . . . . .	5
1.5	Equality rules . . . . .	6
1.6	Simplifications of assumed implications . . . . .	7
1.7	Intuitionistic Reasoning . . . . .	8
1.8	Polymorphic congruence rules . . . . .	9
1.8.1	Congruence rules for predicate letters . . . . .	10
1.9	Atomizing meta-level rules . . . . .	10
1.10	Atomizing elimination rules . . . . .	10
1.11	Calculational rules . . . . .	11
1.12	“Let” declarations . . . . .	11
1.13	Intuitionistic simplification rules . . . . .	12
1.13.1	Conversion into rewrite rules . . . . .	13
1.13.2	More rewrite rules . . . . .	13

<b>2</b>	<b>Classical first-order logic</b>	<b>14</b>
2.1	The classical axiom . . . . .	14
2.2	Lemmas and proof tools . . . . .	14
2.2.1	Classical introduction rules for $\vee$ and $\exists$ . . . . .	14
2.3	Special elimination rules . . . . .	15
2.3.1	Tactics for implication and contradiction . . . . .	15
<b>3</b>	<b>Classical Reasoner</b>	<b>16</b>
3.1	Classical simplification rules . . . . .	17
3.1.1	Miniscoping: pushing quantifiers in . . . . .	17
3.1.2	Named rewrite rules proved for IFOL . . . . .	18
3.2	Other simple lemmas . . . . .	18
3.2.1	Monotonicity of implications . . . . .	19
3.3	Proof by cases and induction . . . . .	19

## 1 Intuitionistic first-order logic

```
theory IFOL
  imports Pure
  abbrevs ?< =  $\exists_{\leq 1}$ 
begin
```

$\langle ML \rangle$

### 1.1 Syntax and axiomatic basis

$\langle ML \rangle$

```
class term
default-sort  $\langle term \rangle$ 
```

```
typedecl o
```

```
judgment
  Trueprop ::  $\langle o \Rightarrow prop \rangle$  ( $\langle \langle notation=judgment \rangle - \rangle$  5)
```

#### 1.1.1 Equality

```
axiomatization
  eq ::  $\langle [ 'a, 'a ] \Rightarrow o \rangle$  (infixl  $\langle == \rangle$  50)
where
  refl:  $\langle a = a \rangle$  and
  subst:  $\langle a = b \Longrightarrow P(a) \Longrightarrow P(b) \rangle$ 
```

#### 1.1.2 Propositional logic

```
axiomatization
  False ::  $\langle o \rangle$  and
```

$conj :: \langle [o, o] \Rightarrow o \rangle$  (**infixr**  $\langle \wedge \rangle$  35) **and**  
 $disj :: \langle [o, o] \Rightarrow o \rangle$  (**infixr**  $\langle \vee \rangle$  30) **and**  
 $imp :: \langle [o, o] \Rightarrow o \rangle$  (**infixr**  $\langle \longrightarrow \rangle$  25)  
**where**  
 $conjI :: \langle [P; Q] \Longrightarrow P \wedge Q \rangle$  **and**  
 $conjunct1 :: \langle P \wedge Q \Longrightarrow P \rangle$  **and**  
 $conjunct2 :: \langle P \wedge Q \Longrightarrow Q \rangle$  **and**  
  
 $disjI1 :: \langle P \Longrightarrow P \vee Q \rangle$  **and**  
 $disjI2 :: \langle Q \Longrightarrow P \vee Q \rangle$  **and**  
 $disjE :: \langle [P \vee Q; P \Longrightarrow R; Q \Longrightarrow R] \Longrightarrow R \rangle$  **and**  
  
 $impI :: \langle (P \Longrightarrow Q) \Longrightarrow P \longrightarrow Q \rangle$  **and**  
 $mp :: \langle [P \longrightarrow Q; P] \Longrightarrow Q \rangle$  **and**  
  
 $FalseE :: \langle False \Longrightarrow P \rangle$

### 1.1.3 Quantifiers

#### axiomatization

$All :: \langle ('a \Rightarrow o) \Rightarrow o \rangle$  (**binder**  $\langle \forall \rangle$  10) **and**  
 $Ex :: \langle ('a \Rightarrow o) \Rightarrow o \rangle$  (**binder**  $\langle \exists \rangle$  10)

#### where

$allI :: \langle (\bigwedge x. P(x)) \Longrightarrow (\forall x. P(x)) \rangle$  **and**  
 $spec :: \langle (\forall x. P(x)) \Longrightarrow P(x) \rangle$  **and**  
 $exI :: \langle P(x) \Longrightarrow (\exists x. P(x)) \rangle$  **and**  
 $exE :: \langle [\exists x. P(x); \bigwedge x. P(x) \Longrightarrow R] \Longrightarrow R \rangle$

### 1.1.4 Definitions

**definition**  $\langle True \equiv False \longrightarrow False \rangle$

**definition** *Not* ( $\langle \langle \text{open-block notation} = \langle \text{prefix } \neg \rangle \neg \rangle \rangle$  [40] 40)  
**where** *not-def*:  $\langle \neg P \equiv P \longrightarrow False \rangle$

**definition** *iff* (**infixr**  $\langle \longleftrightarrow \rangle$  25)  
**where**  $\langle P \longleftrightarrow Q \equiv (P \longrightarrow Q) \wedge (Q \longrightarrow P) \rangle$

**definition** *Uniq* ::  $\langle 'a \Rightarrow o \rangle \Rightarrow o$   
**where**  $\langle Uniq(P) \equiv (\forall x y. P(x) \longrightarrow P(y) \longrightarrow y = x) \rangle$

**definition** *Ex1* ::  $\langle ('a \Rightarrow o) \Rightarrow o \rangle$  (**binder**  $\langle \exists ! \rangle$  10)  
**where** *ex1-def*:  $\langle \exists ! x. P(x) \equiv \exists x. P(x) \wedge (\forall y. P(y) \longrightarrow y = x) \rangle$

**axiomatization where** — Reflection, admissible

$eq\text{-}reflection :: \langle (x = y) \Longrightarrow (x \equiv y) \rangle$  **and**  
 $iff\text{-}reflection :: \langle (P \longleftrightarrow Q) \Longrightarrow (P \equiv Q) \rangle$

**abbreviation** *not-equal* ::  $\langle ['a, 'a] \Rightarrow o \rangle$  (**infixl**  $\langle \neq \rangle$  50)  
**where**  $\langle x \neq y \equiv \neg (x = y) \rangle$

**syntax** *-Uniq* :: *pttrn*  $\Rightarrow$  *o*  $\Rightarrow$  *o* ( $\langle \langle \text{indent}=2 \text{ notation}=\text{binder } \exists_{\leq 1} \rangle \rangle \exists_{\leq 1} \text{ -./ -} \rangle$   
 $[0, 10] \ 10$ )  
**syntax-consts** *-Uniq*  $\Rightarrow$  *Uniq*  
**translations**  $\exists_{\leq 1} x. P \Rightarrow \text{CONST } \text{Uniq } (\lambda x. P)$   
 $\langle \text{ML} \rangle$

### 1.1.5 Old-style ASCII syntax

**notation** (*ASCII*)  
*not-equal* (**infixl**  $\langle \sim = \rangle \ 50$ ) **and**  
*Not* ( $\langle \langle \text{open-block notation}=\text{prefix } \sim \rangle \rangle \sim \text{ -} \rangle \ [40] \ 40$ ) **and**  
*conj* (**infixr**  $\langle \& \rangle \ 35$ ) **and**  
*disj* (**infixr**  $\langle | \rangle \ 30$ ) **and**  
*All* (**binder**  $\langle \text{ALL} \rangle \ 10$ ) **and**  
*Ex* (**binder**  $\langle \text{EX} \rangle \ 10$ ) **and**  
*Ex1* (**binder**  $\langle \text{EX}! \rangle \ 10$ ) **and**  
*imp* (**infixr**  $\langle --> \rangle \ 25$ ) **and**  
*iff* (**infixr**  $\langle <-> \rangle \ 25$ )

## 1.2 Lemmas and proof tools

**lemmas** *strip* = *impI allI*

**lemma** *TrueI*:  $\langle \text{True} \rangle$   
 $\langle \text{proof} \rangle$

### 1.2.1 Sequent-style elimination rules for $\wedge \longrightarrow$ and $\forall$

**lemma** *conjE*:  
**assumes** *major*:  $\langle P \wedge Q \rangle$   
**and** *r*:  $\langle \llbracket P; Q \rrbracket \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *impE*:  
**assumes** *major*:  $\langle P \longrightarrow Q \rangle$   
**and**  $\langle P \rangle$   
**and** *r*:  $\langle Q \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *allE*:  
**assumes** *major*:  $\langle \forall x. P(x) \rangle$   
**and** *r*:  $\langle P(x) \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$

Duplicates the quantifier; for use with `eresolve_tac`.

**lemma** *all-dupE*:

**assumes** *major*:  $\langle \forall x. P(x) \rangle$   
**and** *r*:  $\langle \llbracket P(x); \forall x. P(x) \rrbracket \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$

### 1.2.2 Negation rules, which translate between $\neg P$ and $P \longrightarrow \text{False}$

**lemma** *notI*:  $\langle (P \Longrightarrow \text{False}) \Longrightarrow \neg P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *notE*:  $\langle \llbracket \neg P; P \rrbracket \Longrightarrow R \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *rev-notE*:  $\langle \llbracket P; \neg P \rrbracket \Longrightarrow R \rangle$   
 $\langle \text{proof} \rangle$

This is useful with the special implication rules for each kind of  $P$ .

**lemma** *not-to-imp*:  
**assumes**  $\langle \neg P \rangle$   
**and** *r*:  $\langle P \longrightarrow \text{False} \Longrightarrow Q \rangle$   
**shows**  $\langle Q \rangle$   
 $\langle \text{proof} \rangle$

For substitution into an assumption  $P$ , reduce  $Q$  to  $P \longrightarrow Q$ , substitute into this implication, then apply *impI* to move  $P$  back into the assumptions.

**lemma** *rev-mp*:  $\langle \llbracket P; P \longrightarrow Q \rrbracket \Longrightarrow Q \rangle$   
 $\langle \text{proof} \rangle$

Contrapositive of an inference rule.

**lemma** *contrapos*:  
**assumes** *major*:  $\langle \neg Q \rangle$   
**and** *minor*:  $\langle P \Longrightarrow Q \rangle$   
**shows**  $\langle \neg P \rangle$   
 $\langle \text{proof} \rangle$

### 1.2.3 Modus Ponens Tactics

Finds  $P \longrightarrow Q$  and  $P$  in the assumptions, replaces implication by  $Q$ .  
 $\langle \text{ML} \rangle$

## 1.3 If-and-only-if

**lemma** *iffI*:  $\langle \llbracket P \Longrightarrow Q; Q \Longrightarrow P \rrbracket \Longrightarrow P \longleftrightarrow Q \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *iffE*:  
**assumes** *major*:  $\langle P \longleftrightarrow Q \rangle$   
**and** *r*:  $\langle \llbracket P \longrightarrow Q; Q \longrightarrow P \rrbracket \Longrightarrow R \rangle$

shows  $\langle R \rangle$   
 $\langle proof \rangle$

### 1.3.1 Destruct rules for $\longleftrightarrow$ similar to Modus Ponens

**lemma** *iffD1*:  $\langle \llbracket P \longleftrightarrow Q; P \rrbracket \Longrightarrow Q \rangle$   
 $\langle proof \rangle$

**lemma** *iffD2*:  $\langle \llbracket P \longleftrightarrow Q; Q \rrbracket \Longrightarrow P \rangle$   
 $\langle proof \rangle$

**lemma** *rev-iffD1*:  $\langle \llbracket P; P \longleftrightarrow Q \rrbracket \Longrightarrow Q \rangle$   
 $\langle proof \rangle$

**lemma** *rev-iffD2*:  $\langle \llbracket Q; P \longleftrightarrow Q \rrbracket \Longrightarrow P \rangle$   
 $\langle proof \rangle$

**lemma** *iff-refl*:  $\langle P \longleftrightarrow P \rangle$   
 $\langle proof \rangle$

**lemma** *iff-sym*:  $\langle Q \longleftrightarrow P \Longrightarrow P \longleftrightarrow Q \rangle$   
 $\langle proof \rangle$

**lemma** *iff-trans*:  $\langle \llbracket P \longleftrightarrow Q; Q \longleftrightarrow R \rrbracket \Longrightarrow P \longleftrightarrow R \rangle$   
 $\langle proof \rangle$

## 1.4 Unique existence

NOTE THAT the following 2 quantifications:

- $\exists!x$  such that  $[\exists!y \text{ such that } P(x,y)]$  (sequential)
- $\exists!x,y$  such that  $P(x,y)$  (simultaneous)

do NOT mean the same thing. The parser treats  $\exists!x y.P(x,y)$  as sequential.

**lemma** *ex1I*:  $\langle P(a) \Longrightarrow (\bigwedge x. P(x) \Longrightarrow x = a) \Longrightarrow \exists!x. P(x) \rangle$   
 $\langle proof \rangle$

Sometimes easier to use: the premises have no shared variables. Safe!

**lemma** *ex-ex1I*:  $\langle \exists x. P(x) \Longrightarrow (\bigwedge x y. \llbracket P(x); P(y) \rrbracket \Longrightarrow x = y) \Longrightarrow \exists!x. P(x) \rangle$   
 $\langle proof \rangle$

**lemma** *ex1E*:  $\langle \exists! x. P(x) \Longrightarrow (\bigwedge x. \llbracket P(x); \forall y. P(y) \longrightarrow y = x \rrbracket \Longrightarrow R) \Longrightarrow R \rangle$   
 $\langle proof \rangle$

### 1.4.1 $\longleftrightarrow$ congruence rules for simplification

Use *iffE* on a premise. For *conj-cong*, *imp-cong*, *all-cong*, *ex-cong*.

$\langle ML \rangle$

**lemma** *conj-cong*:

**assumes**  $\langle P \longleftrightarrow P' \rangle$   
**and**  $\langle P' \Longrightarrow Q \longleftrightarrow Q' \rangle$   
**shows**  $\langle (P \wedge Q) \longleftrightarrow (P' \wedge Q') \rangle$   
 $\langle proof \rangle$

Reversed congruence rule! Used in ZF/Order.

**lemma** *conj-cong2*:

**assumes**  $\langle P \longleftrightarrow P' \rangle$   
**and**  $\langle P' \Longrightarrow Q \longleftrightarrow Q' \rangle$   
**shows**  $\langle (Q \wedge P) \longleftrightarrow (Q' \wedge P') \rangle$   
 $\langle proof \rangle$

**lemma** *disj-cong*:

**assumes**  $\langle P \longleftrightarrow P' \rangle$  **and**  $\langle Q \longleftrightarrow Q' \rangle$   
**shows**  $\langle (P \vee Q) \longleftrightarrow (P' \vee Q') \rangle$   
 $\langle proof \rangle$

**lemma** *imp-cong*:

**assumes**  $\langle P \longleftrightarrow P' \rangle$   
**and**  $\langle P' \Longrightarrow Q \longleftrightarrow Q' \rangle$   
**shows**  $\langle (P \longrightarrow Q) \longleftrightarrow (P' \longrightarrow Q') \rangle$   
 $\langle proof \rangle$

**lemma** *iff-cong*:  $\langle \llbracket P \longleftrightarrow P'; Q \longleftrightarrow Q' \rrbracket \Longrightarrow (P \longleftrightarrow Q) \longleftrightarrow (P' \longleftrightarrow Q') \rangle$   
 $\langle proof \rangle$

**lemma** *not-cong*:  $\langle P \longleftrightarrow P' \Longrightarrow \neg P \longleftrightarrow \neg P' \rangle$   
 $\langle proof \rangle$

**lemma** *all-cong*:

**assumes**  $\langle \bigwedge x. P(x) \longleftrightarrow Q(x) \rangle$   
**shows**  $\langle (\forall x. P(x)) \longleftrightarrow (\forall x. Q(x)) \rangle$   
 $\langle proof \rangle$

**lemma** *ex-cong*:

**assumes**  $\langle \bigwedge x. P(x) \longleftrightarrow Q(x) \rangle$   
**shows**  $\langle (\exists x. P(x)) \longleftrightarrow (\exists x. Q(x)) \rangle$   
 $\langle proof \rangle$

**lemma** *ex1-cong*:

**assumes**  $\langle \bigwedge x. P(x) \longleftrightarrow Q(x) \rangle$   
**shows**  $\langle (\exists! x. P(x)) \longleftrightarrow (\exists! x. Q(x)) \rangle$   
 $\langle proof \rangle$

## 1.5 Equality rules

**lemma** *sym*:  $\langle a = b \implies b = a \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *trans*:  $\langle \llbracket a = b; b = c \rrbracket \implies a = c \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *not-sym*:  $\langle b \neq a \implies a \neq b \rangle$   
 $\langle \text{proof} \rangle$

Two theorems for rewriting only one instance of a definition: the first for definitions of formulae and the second for terms.

**lemma** *def-imp-iff*:  $\langle (A \equiv B) \implies A \longleftrightarrow B \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *meta-eq-to-obj-eq*:  $\langle (A \equiv B) \implies A = B \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *meta-eq-to-iff*:  $\langle x \equiv y \implies x \longleftrightarrow y \rangle$   
 $\langle \text{proof} \rangle$

Substitution.

**lemma** *ssubst*:  $\langle \llbracket b = a; P(a) \rrbracket \implies P(b) \rangle$   
 $\langle \text{proof} \rangle$

A special case of *ex1E* that would otherwise need quantifier expansion.

**lemma** *ex1-equalsE*:  $\langle \llbracket \exists !x. P(x); P(a); P(b) \rrbracket \implies a = b \rangle$   
 $\langle \text{proof} \rangle$

## 1.6 Simplifications of assumed implications

Roy Dyckhoff has proved that *conj-impE*, *disj-impE*, and *imp-impE* used with `mp_tac` (restricted to atomic formulae) is COMPLETE for intuitionistic propositional logic.

See R. Dyckhoff, Contraction-free sequent calculi for intuitionistic logic (preprint, University of St Andrews, 1991).

**lemma** *conj-impE*:  
**assumes** *major*:  $\langle (P \wedge Q) \longrightarrow S \rangle$   
**and** *r*:  $\langle P \longrightarrow (Q \longrightarrow S) \implies R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *disj-impE*:  
**assumes** *major*:  $\langle (P \vee Q) \longrightarrow S \rangle$   
**and** *r*:  $\langle \llbracket P \longrightarrow S; Q \longrightarrow S \rrbracket \implies R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle \text{proof} \rangle$



Simplifies the implication. Classical version is stronger. Still UNSAFE since Q must be provable – backtracking needed.

**lemma** *imp-impE*:  
**assumes** *major*:  $\langle (P \longrightarrow Q) \longrightarrow S \rangle$   
**and** *r1*:  $\langle \llbracket P; Q \longrightarrow S \rrbracket \Longrightarrow Q \rangle$   
**and** *r2*:  $\langle S \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

Simplifies the implication. Classical version is stronger. Still UNSAFE since P must be provable – backtracking needed.

**lemma** *not-impE*:  $\langle \neg P \longrightarrow S \Longrightarrow (P \Longrightarrow False) \Longrightarrow (S \Longrightarrow R) \Longrightarrow R \rangle$   
 $\langle proof \rangle$

Simplifies the implication. UNSAFE.

**lemma** *iff-impE*:  
**assumes** *major*:  $\langle (P \longleftrightarrow Q) \longrightarrow S \rangle$   
**and** *r1*:  $\langle \llbracket P; Q \longrightarrow S \rrbracket \Longrightarrow Q \rangle$   
**and** *r2*:  $\langle \llbracket Q; P \longrightarrow S \rrbracket \Longrightarrow P \rangle$   
**and** *r3*:  $\langle S \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

What if  $(\forall x. \neg \neg P(x)) \longrightarrow \neg \neg (\forall x. P(x))$  is an assumption? UNSAFE.

**lemma** *all-impE*:  
**assumes** *major*:  $\langle (\forall x. P(x)) \longrightarrow S \rangle$   
**and** *r1*:  $\langle \bigwedge x. P(x) \rangle$   
**and** *r2*:  $\langle S \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

Unsafe:  $\exists x. P(x) \longrightarrow S$  is equivalent to  $\forall x. P(x) \longrightarrow S$ .

**lemma** *ex-impE*:  
**assumes** *major*:  $\langle (\exists x. P(x)) \longrightarrow S \rangle$   
**and** *r*:  $\langle P(x) \longrightarrow S \Longrightarrow R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

Courtesy of Krzysztof Grabczewski.

**lemma** *disj-imp-disj*:  $\langle P \vee Q \Longrightarrow (P \Longrightarrow R) \Longrightarrow (Q \Longrightarrow S) \Longrightarrow R \vee S \rangle$   
 $\langle proof \rangle$

$\langle ML \rangle$

**lemma** *thin-refl*:  $\langle \llbracket x = x; PROP W \rrbracket \Longrightarrow PROP W \rangle$   $\langle proof \rangle$

$\langle ML \rangle$

## 1.7 Intuitionistic Reasoning

$\langle ML \rangle$

**lemma** *impE'*:  
 assumes 1:  $\langle P \longrightarrow Q \rangle$   
 and 2:  $\langle Q \Longrightarrow R \rangle$   
 and 3:  $\langle P \longrightarrow Q \Longrightarrow P \rangle$   
 shows  $\langle R \rangle$   
 $\langle proof \rangle$

**lemma** *allE'*:  
 assumes 1:  $\langle \forall x. P(x) \rangle$   
 and 2:  $\langle P(x) \Longrightarrow \forall x. P(x) \Longrightarrow Q \rangle$   
 shows  $\langle Q \rangle$   
 $\langle proof \rangle$

**lemma** *notE'*:  
 assumes 1:  $\langle \neg P \rangle$   
 and 2:  $\langle \neg P \Longrightarrow P \rangle$   
 shows  $\langle R \rangle$   
 $\langle proof \rangle$

**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  $[Pure.intro] = exI\ disjI2\ disjI1$

$\langle ML \rangle$

**lemma** *iff-not-sym*:  $\langle \neg (Q \longleftrightarrow P) \Longrightarrow \neg (P \longleftrightarrow Q) \rangle$   
 $\langle proof \rangle$

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

**lemma** *eq-commute*:  $\langle a = b \longleftrightarrow b = a \rangle$   
 $\langle proof \rangle$

## 1.8 Polymorphic congruence rules

**lemma** *subst-context*:  $\langle a = b \Longrightarrow t(a) = t(b) \rangle$   
 $\langle proof \rangle$

**lemma** *subst-context2*:  $\langle \llbracket a = b; c = d \rrbracket \Longrightarrow t(a, c) = t(b, d) \rangle$   
 $\langle proof \rangle$

**lemma** *subst-context3*:  $\langle \llbracket a = b; c = d; e = f \rrbracket \Longrightarrow t(a, c, e) = t(b, d, f) \rangle$

$\langle \text{proof} \rangle$

Useful with `eresolve_tac` for proving equalities from known equalities.

$a = b \mid \mid c = d$

**lemma** *box-equals*:  $\langle \llbracket a = b; a = c; b = d \rrbracket \Longrightarrow c = d \rangle$

$\langle \text{proof} \rangle$

Dual of *box-equals*: for proving equalities backwards.

**lemma** *simp-equals*:  $\langle \llbracket a = c; b = d; c = d \rrbracket \Longrightarrow a = b \rangle$

$\langle \text{proof} \rangle$

### 1.8.1 Congruence rules for predicate letters

**lemma** *pred1-cong*:  $\langle a = a' \Longrightarrow P(a) \longleftrightarrow P(a') \rangle$

$\langle \text{proof} \rangle$

**lemma** *pred2-cong*:  $\langle \llbracket a = a'; b = b' \rrbracket \Longrightarrow P(a,b) \longleftrightarrow P(a',b') \rangle$

$\langle \text{proof} \rangle$

**lemma** *pred3-cong*:  $\langle \llbracket a = a'; b = b'; c = c' \rrbracket \Longrightarrow P(a,b,c) \longleftrightarrow P(a',b',c') \rangle$

$\langle \text{proof} \rangle$

Special case for the equality predicate!

**lemma** *eq-cong*:  $\langle \llbracket a = a'; b = b' \rrbracket \Longrightarrow a = b \longleftrightarrow a' = b' \rangle$

$\langle \text{proof} \rangle$

## 1.9 Atomizing meta-level rules

**lemma** *atomize-all* [*atomize*]:  $\langle (\bigwedge x. P(x)) \equiv \text{Trueprop } (\forall x. P(x)) \rangle$

$\langle \text{proof} \rangle$

**lemma** *atomize-imp* [*atomize*]:  $\langle (A \Longrightarrow B) \equiv \text{Trueprop } (A \longrightarrow B) \rangle$

$\langle \text{proof} \rangle$

**lemma** *atomize-eq* [*atomize*]:  $\langle (x \equiv y) \equiv \text{Trueprop } (x = y) \rangle$

$\langle \text{proof} \rangle$

**lemma** *atomize-iff* [*atomize*]:  $\langle (A \equiv B) \equiv \text{Trueprop } (A \longleftrightarrow B) \rangle$

$\langle \text{proof} \rangle$

**lemma** *atomize-conj* [*atomize*]:  $\langle (A \&\&\& B) \equiv \text{Trueprop } (A \wedge B) \rangle$

$\langle \text{proof} \rangle$

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

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

## 1.10 Atomizing elimination rules

**lemma** *atomize-exL* [*atomize-elim*]:  $\langle (\bigwedge x. P(x) \Longrightarrow Q) \equiv ((\exists x. P(x)) \Longrightarrow Q) \rangle$

$\langle proof \rangle$

**lemma** *atomize-conjL*[*atomize-elim*]:  $\langle (A \implies B \implies C) \equiv (A \wedge B \implies C) \rangle$   
 $\langle proof \rangle$

**lemma** *atomize-disjL*[*atomize-elim*]:  $\langle ((A \implies C) \implies (B \implies C) \implies C) \equiv ((A \vee B \implies C) \implies C) \rangle$   
 $\langle proof \rangle$

**lemma** *atomize-elimL*[*atomize-elim*]:  $\langle (\bigwedge B. (A \implies B) \implies B) \equiv Trueprop(A) \rangle$   
 $\langle proof \rangle$

## 1.11 Calculational rules

**lemma** *forw-subst*:  $\langle a = b \implies P(b) \implies P(a) \rangle$   
 $\langle proof \rangle$

**lemma** *back-subst*:  $\langle P(a) \implies a = b \implies P(b) \rangle$   
 $\langle proof \rangle$

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*

## 1.12 “Let” declarations

**nonterminal** *letbinds* and *letbind*

**definition** *Let* ::  $\langle [a::\{\}, 'a \Rightarrow 'b] \Rightarrow ('b::\{\}) \rangle$   
**where**  $\langle Let(s, f) \equiv f(s) \rangle$

**syntax**

*-bind* ::  $\langle [pttrn, 'a] \Rightarrow letbind \rangle$  ( $\langle \langle indent=2 notation=\langle infix let binding \rangle \rangle -$   
 $= / - \rangle 10$ )  
::  $\langle letbind \Rightarrow letbinds \rangle$  ( $\langle - \rangle$ )  
*-binds* ::  $\langle [letbind, letbinds] \Rightarrow letbinds \rangle$  ( $\langle -; / - \rangle$ )  
*-Let* ::  $\langle [letbinds, 'a] \Rightarrow 'a \rangle$  ( $\langle \langle notation=\langle mixfix let expression \rangle \rangle let (-) /$   
 $in (-) \rangle 10$ )

**syntax-consts**

*-Let*  $\Rightarrow$  *Let*

**translations**

*-Let(-binds(b, bs), e)* == *-Let(b, -Let(bs, e))*  
*let x = a in e* == *CONST Let(a,  $\lambda x. e$ )*

**lemma** *LetI*:

**assumes**  $\langle \bigwedge x. x = t \implies P(u(x)) \rangle$   
**shows**  $\langle P(\text{let } x = t \text{ in } u(x)) \rangle$   
 $\langle \text{proof} \rangle$

### 1.13 Intuitionistic simplification rules

**lemma** *conj-simps*:

$\langle P \wedge \text{True} \longleftrightarrow P \rangle$   
 $\langle \text{True} \wedge P \longleftrightarrow P \rangle$   
 $\langle P \wedge \text{False} \longleftrightarrow \text{False} \rangle$   
 $\langle \text{False} \wedge P \longleftrightarrow \text{False} \rangle$   
 $\langle P \wedge P \longleftrightarrow P \rangle$   
 $\langle P \wedge P \wedge Q \longleftrightarrow P \wedge Q \rangle$   
 $\langle P \wedge \neg P \longleftrightarrow \text{False} \rangle$   
 $\langle \neg P \wedge P \longleftrightarrow \text{False} \rangle$   
 $\langle (P \wedge Q) \wedge R \longleftrightarrow P \wedge (Q \wedge R) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *disj-simps*:

$\langle P \vee \text{True} \longleftrightarrow \text{True} \rangle$   
 $\langle \text{True} \vee P \longleftrightarrow \text{True} \rangle$   
 $\langle P \vee \text{False} \longleftrightarrow P \rangle$   
 $\langle \text{False} \vee P \longleftrightarrow P \rangle$   
 $\langle P \vee P \longleftrightarrow P \rangle$   
 $\langle P \vee P \vee Q \longleftrightarrow P \vee Q \rangle$   
 $\langle (P \vee Q) \vee R \longleftrightarrow P \vee (Q \vee R) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *not-simps*:

$\langle \neg (P \vee Q) \longleftrightarrow \neg P \wedge \neg Q \rangle$   
 $\langle \neg \text{False} \longleftrightarrow \text{True} \rangle$   
 $\langle \neg \text{True} \longleftrightarrow \text{False} \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *imp-simps*:

$\langle (P \longrightarrow \text{False}) \longleftrightarrow \neg P \rangle$   
 $\langle (P \longrightarrow \text{True}) \longleftrightarrow \text{True} \rangle$   
 $\langle (\text{False} \longrightarrow P) \longleftrightarrow \text{True} \rangle$   
 $\langle (\text{True} \longrightarrow P) \longleftrightarrow P \rangle$   
 $\langle (P \longrightarrow P) \longleftrightarrow \text{True} \rangle$   
 $\langle (P \longrightarrow \neg P) \longleftrightarrow \neg P \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *iff-simps*:

$\langle (\text{True} \longleftrightarrow P) \longleftrightarrow P \rangle$   
 $\langle (P \longleftrightarrow \text{True}) \longleftrightarrow P \rangle$   
 $\langle (P \longleftrightarrow P) \longleftrightarrow \text{True} \rangle$   
 $\langle (\text{False} \longleftrightarrow P) \longleftrightarrow \neg P \rangle$   
 $\langle (P \longleftrightarrow \text{False}) \longleftrightarrow \neg P \rangle$

$\langle proof \rangle$

The  $x = t$  versions are needed for the simplification procedures.

**lemma** *quant-simps*:

$\langle \bigwedge P. (\forall x. P) \longleftrightarrow P \rangle$   
 $\langle (\forall x. x = t \longrightarrow P(x)) \longleftrightarrow P(t) \rangle$   
 $\langle (\forall x. t = x \longrightarrow P(x)) \longleftrightarrow P(t) \rangle$   
 $\langle \bigwedge P. (\exists x. P) \longleftrightarrow P \rangle$   
 $\langle \exists x. x = t \rangle$   
 $\langle \exists x. t = x \rangle$   
 $\langle (\exists x. x = t \wedge P(x)) \longleftrightarrow P(t) \rangle$   
 $\langle (\exists x. t = x \wedge P(x)) \longleftrightarrow P(t) \rangle$   
 $\langle proof \rangle$

These are NOT supplied by default!

**lemma** *distrib-simps*:

$\langle P \wedge (Q \vee R) \longleftrightarrow P \wedge Q \vee P \wedge R \rangle$   
 $\langle (Q \vee R) \wedge P \longleftrightarrow Q \wedge P \vee R \wedge P \rangle$   
 $\langle (P \vee Q \longrightarrow R) \longleftrightarrow (P \longrightarrow R) \wedge (Q \longrightarrow R) \rangle$   
 $\langle proof \rangle$

**lemma** *subst-all*:

$\langle (\bigwedge x. x = a \implies PROP P(x)) \equiv PROP P(a) \rangle$   
 $\langle (\bigwedge x. a = x \implies PROP P(x)) \equiv PROP P(a) \rangle$   
 $\langle proof \rangle$

### 1.13.1 Conversion into rewrite rules

**lemma** *P-iff-F*:  $\langle \neg P \implies (P \longleftrightarrow False) \rangle$

$\langle proof \rangle$

**lemma** *iff-reflection-F*:  $\langle \neg P \implies (P \equiv False) \rangle$

$\langle proof \rangle$

**lemma** *P-iff-T*:  $\langle P \implies (P \longleftrightarrow True) \rangle$

$\langle proof \rangle$

**lemma** *iff-reflection-T*:  $\langle P \implies (P \equiv True) \rangle$

$\langle proof \rangle$

### 1.13.2 More rewrite rules

**lemma** *conj-commute*:  $\langle P \wedge Q \longleftrightarrow Q \wedge P \rangle$   $\langle proof \rangle$

**lemma** *conj-left-commute*:  $\langle P \wedge (Q \wedge R) \longleftrightarrow Q \wedge (P \wedge R) \rangle$   $\langle proof \rangle$

**lemmas** *conj-comms* = *conj-commute conj-left-commute*

**lemma** *disj-commute*:  $\langle P \vee Q \longleftrightarrow Q \vee P \rangle$   $\langle proof \rangle$

**lemma** *disj-left-commute*:  $\langle P \vee (Q \vee R) \longleftrightarrow Q \vee (P \vee R) \rangle$   $\langle proof \rangle$

**lemmas** *disj-comms* = *disj-commute disj-left-commute*

**lemma** *conj-disj-distribL*:  $\langle P \wedge (Q \vee R) \longleftrightarrow (P \wedge Q \vee P \wedge R) \rangle$   $\langle proof \rangle$

**lemma** *conj-disj-distribR*:  $\langle (P \vee Q) \wedge R \longleftrightarrow (P \wedge R \vee Q \wedge R) \rangle \langle \text{proof} \rangle$   
**lemma** *disj-conj-distribL*:  $\langle P \vee (Q \wedge R) \longleftrightarrow (P \vee Q) \wedge (P \vee R) \rangle \langle \text{proof} \rangle$   
**lemma** *disj-conj-distribR*:  $\langle (P \wedge Q) \vee R \longleftrightarrow (P \vee R) \wedge (Q \vee R) \rangle \langle \text{proof} \rangle$   
**lemma** *imp-conj-distrib*:  $\langle (P \longrightarrow (Q \wedge R)) \longleftrightarrow (P \longrightarrow Q) \wedge (P \longrightarrow R) \rangle \langle \text{proof} \rangle$   
**lemma** *imp-conj*:  $\langle ((P \wedge Q) \longrightarrow R) \longleftrightarrow (P \longrightarrow (Q \longrightarrow R)) \rangle \langle \text{proof} \rangle$   
**lemma** *imp-disj*:  $\langle (P \vee Q \longrightarrow R) \longleftrightarrow (P \longrightarrow R) \wedge (Q \longrightarrow R) \rangle \langle \text{proof} \rangle$   
**lemma** *de-Morgan-disj*:  $\langle (\neg (P \vee Q)) \longleftrightarrow (\neg P \wedge \neg Q) \rangle \langle \text{proof} \rangle$   
**lemma** *not-ex*:  $\langle (\neg (\exists x. P(x))) \longleftrightarrow (\forall x. \neg P(x)) \rangle \langle \text{proof} \rangle$   
**lemma** *imp-ex*:  $\langle ((\exists x. P(x)) \longrightarrow Q) \longleftrightarrow (\forall x. P(x) \longrightarrow Q) \rangle \langle \text{proof} \rangle$   
**lemma** *ex-disj-distrib*:  $\langle (\exists x. P(x) \vee Q(x)) \longleftrightarrow ((\exists x. P(x)) \vee (\exists x. Q(x))) \rangle \langle \text{proof} \rangle$   
**lemma** *all-conj-distrib*:  $\langle (\forall x. P(x) \wedge Q(x)) \longleftrightarrow ((\forall x. P(x)) \wedge (\forall x. Q(x))) \rangle \langle \text{proof} \rangle$   
**end**

## 2 Classical first-order logic

**theory** *FOL*  
**imports** *IFOL*  
**keywords** *print-claset print-induct-rules :: diag*  
**begin**  
  
 $\langle ML \rangle$

### 2.1 The classical axiom

**axiomatization** *where*  
*classical*:  $\langle (\neg P \implies P) \implies P \rangle$

### 2.2 Lemmas and proof tools

**lemma** *ccontr*:  $\langle (\neg P \implies \text{False}) \implies P \rangle \langle \text{proof} \rangle$

#### 2.2.1 Classical introduction rules for $\vee$ and $\exists$

**lemma** *disjCI*:  $\langle (\neg Q \implies P) \implies P \vee Q \rangle \langle \text{proof} \rangle$

Introduction rule involving only  $\exists$

**lemma** *ex-classical*:  
**assumes** *r*:  $\langle \neg (\exists x. P(x)) \implies P(a) \rangle$

**shows**  $\langle \exists x. P(x) \rangle$   
 $\langle proof \rangle$

Version of above, simplifying  $\neg \exists$  to  $\forall \neg$ .

**lemma** *exCI*:  
**assumes**  $r$ :  $\langle \forall x. \neg P(x) \implies P(a) \rangle$   
**shows**  $\langle \exists x. P(x) \rangle$   
 $\langle proof \rangle$

**lemma** *excluded-middle*:  $\langle \neg P \vee P \rangle$   
 $\langle proof \rangle$

**lemma** *case-split* [*case-names True False*]:  
**assumes**  $r1$ :  $\langle P \implies Q \rangle$   
**and**  $r2$ :  $\langle \neg P \implies Q \rangle$   
**shows**  $\langle Q \rangle$   
 $\langle proof \rangle$

$\langle ML \rangle$

## 2.3 Special elimination rules

Classical implies ( $\longrightarrow$ ) elimination.

**lemma** *impCE*:  
**assumes** *major*:  $\langle P \longrightarrow Q \rangle$   
**and**  $r1$ :  $\langle \neg P \implies R \rangle$   
**and**  $r2$ :  $\langle Q \implies R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

This version of  $\longrightarrow$  elimination works on  $Q$  before  $P$ . It works best for those cases in which  $P$  holds “almost everywhere”. Can’t install as default: would break old proofs.

**lemma** *impCE'*:  
**assumes** *major*:  $\langle P \longrightarrow Q \rangle$   
**and**  $r1$ :  $\langle Q \implies R \rangle$   
**and**  $r2$ :  $\langle \neg P \implies R \rangle$   
**shows**  $\langle R \rangle$   
 $\langle proof \rangle$

Double negation law.

**lemma** *notnotD*:  $\langle \neg \neg P \implies P \rangle$   
 $\langle proof \rangle$

**lemma** *contrapos2*:  $\langle \llbracket Q; \neg P \implies \neg Q \rrbracket \implies P \rangle$   
 $\langle proof \rangle$



### 2.3.1 Tactics for implication and contradiction

Classical  $\longleftrightarrow$  elimination. Proof substitutes  $P = Q$  in  $\neg P \implies \neg Q$  and  $P \implies Q$ .

**lemma** *iffCE*:  
 assumes *major*:  $\langle P \longleftrightarrow Q \rangle$   
 and *r1*:  $\langle \llbracket P; Q \rrbracket \implies R \rangle$   
 and *r2*:  $\langle \llbracket \neg P; \neg Q \rrbracket \implies R \rangle$   
 shows  $\langle R \rangle$   
 $\langle proof \rangle$

**lemma** *alt-ex1E*:  
 assumes *major*:  $\langle \exists! x. P(x) \rangle$   
 and *r*:  $\langle \bigwedge x. \llbracket P(x); \forall y y'. P(y) \wedge P(y') \longrightarrow y = y' \rrbracket \implies R \rangle$   
 shows  $\langle R \rangle$   
 $\langle proof \rangle$

**lemma** *imp-elim*:  $\langle P \longrightarrow Q \implies (\neg R \implies P) \implies (Q \implies R) \implies R \rangle$   
 $\langle proof \rangle$

**lemma** *swap*:  $\langle \neg P \implies (\neg R \implies P) \implies R \rangle$   
 $\langle proof \rangle$

## 3 Classical Reasoner

$\langle ML \rangle$

**lemmas**  $[intro!] = refl\ TrueI\ conjI\ disjCI\ impI\ notI\ iffI$   
 and  $[elim!] = conjE\ disjE\ impCE\ FalseE\ iffCE$   
 $\langle ML \rangle$

**lemmas**  $[intro!] = allI\ ex-ex1I$   
 and  $[intro] = exI$   
 and  $[elim!] = exE\ alt-ex1E$   
 and  $[elim] = allE$   
 $\langle ML \rangle$

**lemma** *ex1-functional*:  $\langle \llbracket \exists! z. P(a,z); P(a,b); P(a,c) \rrbracket \implies b = c \rangle$   
 $\langle proof \rangle$

Elimination of *True* from assumptions:

**lemma** *True-implies-equals*:  $\langle (True \implies PROP\ P) \equiv PROP\ P \rangle$   
 $\langle proof \rangle$

**lemma** *uncurry*:  $\langle P \longrightarrow Q \longrightarrow R \implies P \wedge Q \longrightarrow R \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *iff-allI*:  $\langle (\bigwedge x. P(x) \longleftrightarrow Q(x)) \implies (\forall x. P(x)) \longleftrightarrow (\forall x. Q(x)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *iff-exI*:  $\langle (\bigwedge x. P(x) \longleftrightarrow Q(x)) \implies (\exists x. P(x)) \longleftrightarrow (\exists x. Q(x)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *all-comm*:  $\langle (\forall x y. P(x,y)) \longleftrightarrow (\forall y x. P(x,y)) \rangle$   
 $\langle \text{proof} \rangle$

**lemma** *ex-comm*:  $\langle (\exists x y. P(x,y)) \longleftrightarrow (\exists y x. P(x,y)) \rangle$   
 $\langle \text{proof} \rangle$

### 3.1 Classical simplification rules

Avoids duplication of subgoals after *expand-if*, when the true and false cases boil down to the same thing.

**lemma** *cases-simp*:  $\langle (P \longrightarrow Q) \wedge (\neg P \longrightarrow Q) \longleftrightarrow Q \rangle$   
 $\langle \text{proof} \rangle$

#### 3.1.1 Miniscoping: pushing quantifiers in

We do NOT distribute of  $\forall$  over  $\wedge$ , or dually that of  $\exists$  over  $\vee$ .

Baaz and Leitsch, On Skolemization and Proof Complexity (1994) show that this step can increase proof length!

Existential miniscoping.

**lemma** *int-ex-simps*:  
 $\langle \bigwedge P Q. (\exists x. P(x) \wedge Q) \longleftrightarrow (\exists x. P(x)) \wedge Q \rangle$   
 $\langle \bigwedge P Q. (\exists x. P \wedge Q(x)) \longleftrightarrow P \wedge (\exists x. Q(x)) \rangle$   
 $\langle \bigwedge P Q. (\exists x. P(x) \vee Q) \longleftrightarrow (\exists x. P(x)) \vee Q \rangle$   
 $\langle \bigwedge P Q. (\exists x. P \vee Q(x)) \longleftrightarrow P \vee (\exists x. Q(x)) \rangle$   
 $\langle \text{proof} \rangle$

Classical rules.

**lemma** *cla-ex-simps*:  
 $\langle \bigwedge P Q. (\exists x. P(x) \longrightarrow Q) \longleftrightarrow (\forall x. P(x)) \longrightarrow Q \rangle$   
 $\langle \bigwedge P Q. (\exists x. P \longrightarrow Q(x)) \longleftrightarrow P \longrightarrow (\exists x. Q(x)) \rangle$   
 $\langle \text{proof} \rangle$

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

Universal miniscoping.

**lemma** *int-all-simps*:

$\langle \bigwedge P \ Q. (\forall x. P(x) \wedge Q) \longleftrightarrow (\forall x. P(x)) \wedge Q \rangle$   
 $\langle \bigwedge P \ Q. (\forall x. P \wedge Q(x)) \longleftrightarrow P \wedge (\forall x. Q(x)) \rangle$   
 $\langle \bigwedge P \ Q. (\forall x. P(x) \longrightarrow Q) \longleftrightarrow (\exists x. P(x)) \longrightarrow Q \rangle$   
 $\langle \bigwedge P \ Q. (\forall x. P \longrightarrow Q(x)) \longleftrightarrow P \longrightarrow (\forall x. Q(x)) \rangle$   
 $\langle proof \rangle$

Classical rules.

**lemma** *cla-all-simps*:

$\langle \bigwedge P \ Q. (\forall x. P(x) \vee Q) \longleftrightarrow (\forall x. P(x)) \vee Q \rangle$   
 $\langle \bigwedge P \ Q. (\forall x. P \vee Q(x)) \longleftrightarrow P \vee (\forall x. Q(x)) \rangle$   
 $\langle proof \rangle$

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

### 3.1.2 Named rewrite rules proved for IFOL

**lemma** *imp-disj1*:  $\langle (P \longrightarrow Q) \vee R \longleftrightarrow (P \longrightarrow Q \vee R) \rangle \langle proof \rangle$

**lemma** *imp-disj2*:  $\langle Q \vee (P \longrightarrow R) \longleftrightarrow (P \longrightarrow Q \vee R) \rangle \langle proof \rangle$

**lemma** *de-Morgan-conj*:  $\langle (\neg (P \wedge Q)) \longleftrightarrow (\neg P \vee \neg Q) \rangle \langle proof \rangle$

**lemma** *not-imp*:  $\langle \neg (P \longrightarrow Q) \longleftrightarrow (P \wedge \neg Q) \rangle \langle proof \rangle$

**lemma** *not-iff*:  $\langle \neg (P \longleftrightarrow Q) \longleftrightarrow (P \longleftrightarrow \neg Q) \rangle \langle proof \rangle$

**lemma** *not-all*:  $\langle (\neg (\forall x. P(x))) \longleftrightarrow (\exists x. \neg P(x)) \rangle \langle proof \rangle$

**lemma** *imp-all*:  $\langle ((\forall x. P(x)) \longrightarrow Q) \longleftrightarrow (\exists x. P(x) \longrightarrow Q) \rangle \langle proof \rangle$

**lemmas** *meta-simps =*

*triv-forall-equality* — prunes params

*True-implies-equals* — prune asms *True*

**lemmas** *IFOL-simps =*

*reft [THEN P-iff-T] conj-simps disj-simps not-simps*

*imp-simps iff-simps quant-simps*

**lemma** *notFalseI*:  $\langle \neg False \rangle \langle proof \rangle$

**lemma** *cla-simps-misc*:

$\langle \neg (P \wedge Q) \longleftrightarrow \neg P \vee \neg Q \rangle$   
 $\langle P \vee \neg P \rangle$   
 $\langle \neg P \vee P \rangle$   
 $\langle \neg \neg P \longleftrightarrow P \rangle$   
 $\langle (\neg P \longrightarrow P) \longleftrightarrow P \rangle$   
 $\langle (\neg P \longleftrightarrow \neg Q) \longleftrightarrow (P \longleftrightarrow Q) \rangle \langle proof \rangle$

**lemmas** *cla-simps =*

*de-Morgan-conj de-Morgan-disj imp-disj1 imp-disj2*

*not-imp not-all not-ex cases-simp cla-simps-misc*

$\langle ML \rangle$

### 3.2 Other simple lemmas

**lemma** *[simp]*:  $\langle ((P \longrightarrow R) \longleftrightarrow (Q \longrightarrow R)) \longleftrightarrow ((P \longleftrightarrow Q) \vee R) \rangle$   
 $\langle proof \rangle$

**lemma** *[simp]*:  $\langle ((P \longrightarrow Q) \longleftrightarrow (P \longrightarrow R)) \longleftrightarrow (P \longrightarrow (Q \longleftrightarrow R)) \rangle$   
 $\langle proof \rangle$

**lemma** *not-disj-iff-imp*:  $\langle \neg P \vee Q \longleftrightarrow (P \longrightarrow Q) \rangle$   
 $\langle proof \rangle$

#### 3.2.1 Monotonicity of implications

**lemma** *conj-mono*:  $\langle \llbracket P1 \longrightarrow Q1; P2 \longrightarrow Q2 \rrbracket \Longrightarrow (P1 \wedge P2) \longrightarrow (Q1 \wedge Q2) \rangle$   
 $\langle proof \rangle$

**lemma** *disj-mono*:  $\langle \llbracket P1 \longrightarrow Q1; P2 \longrightarrow Q2 \rrbracket \Longrightarrow (P1 \vee P2) \longrightarrow (Q1 \vee Q2) \rangle$   
 $\langle proof \rangle$

**lemma** *imp-mono*:  $\langle \llbracket Q1 \longrightarrow P1; P2 \longrightarrow Q2 \rrbracket \Longrightarrow (P1 \longrightarrow P2) \longrightarrow (Q1 \longrightarrow Q2) \rangle$   
 $\langle proof \rangle$

**lemma** *imp-refl*:  $\langle P \longrightarrow P \rangle$   
 $\langle proof \rangle$

The quantifier monotonicity rules are also intuitionistically valid.

**lemma** *ex-mono*:  $\langle (\bigwedge x. P(x) \longrightarrow Q(x)) \Longrightarrow (\exists x. P(x)) \longrightarrow (\exists x. Q(x)) \rangle$   
 $\langle proof \rangle$

**lemma** *all-mono*:  $\langle (\bigwedge x. P(x) \longrightarrow Q(x)) \Longrightarrow (\forall x. P(x)) \longrightarrow (\forall x. Q(x)) \rangle$   
 $\langle proof \rangle$

### 3.3 Proof by cases and induction

Proper handling of non-atomic rule statements.

**context**

**begin**

**qualified definition**  $\langle induct-forall(P) \equiv \forall x. P(x) \rangle$

**qualified definition**  $\langle induct-implies(A, B) \equiv A \longrightarrow B \rangle$

**qualified definition**  $\langle induct-equal(x, y) \equiv x = y \rangle$

**qualified definition**  $\langle induct-conj(A, B) \equiv A \wedge B \rangle$

**lemma** *induct-forall-eq*:  $\langle (\bigwedge x. P(x)) \equiv Trueprop(induct-forall(\lambda x. P(x))) \rangle$

$\langle proof \rangle$   
**lemma** *induct-implies-eq*:  $\langle (A \implies B) \equiv Trueprop(induct-implies(A, B)) \rangle$   
 $\langle proof \rangle$   
**lemma** *induct-equal-eq*:  $\langle (x \equiv y) \equiv Trueprop(induct-equal(x, y)) \rangle$   
 $\langle proof \rangle$   
**lemma** *induct-conj-eq*:  $\langle (A \&\& B) \equiv Trueprop(induct-conj(A, B)) \rangle$   
 $\langle proof \rangle$   
**lemmas** *induct-atomize* = *induct-forall-eq induct-implies-eq induct-equal-eq induct-conj-eq*  
**lemmas** *induct-rulify* [*symmetric*] = *induct-atomize*  
**lemmas** *induct-rulify-fallback* =  
*induct-forall-def induct-implies-def induct-equal-def induct-conj-def*  
 Method setup.  
 $\langle ML \rangle$   
**declare** *case-split* [*cases type: o*]  
**end**  
 $\langle ML \rangle$   
**hide-const** (**open**) *eq*  
**end**