

What's in Main

Tobias Nipkow

January 30, 2011

Abstract

This document lists the main types, functions and syntax provided by theory *Main*. It is meant as a quick overview of what is available. The sophisticated class structure is only hinted at. For details see <http://isabelle.in.tum.de/dist/library/HOL/>.

1 HOL

The basic logic: $x = y$, *True*, *False*, $\neg P$, $P \wedge Q$, $P \vee Q$, $P \longrightarrow Q$, $\forall x. P$, $\exists x. P$, $\exists!x. P$, *THE* $x. P$.

undefined :: 'a

default :: 'a

Syntax

$x \neq y$ $\equiv \neg (x = y)$ (\neq)

$P \longleftrightarrow Q$ $\equiv P = Q$

if x *then* y *else* z \equiv *If* x y z

let $x = e_1$ *in* e_2 \equiv *Let* e_1 $(\lambda x. e_2)$

2 Orderings

A collection of classes defining basic orderings: preorder, partial order, linear order, dense linear order and wellorder.

op \leq :: 'a \Rightarrow 'a \Rightarrow bool (\leq)

op $<$:: 'a \Rightarrow 'a \Rightarrow bool

Least :: ('a \Rightarrow bool) \Rightarrow 'a

min :: 'a \Rightarrow 'a \Rightarrow 'a

max :: 'a \Rightarrow 'a \Rightarrow 'a

top :: 'a

bot :: $'a$
 $mono$:: $('a \Rightarrow 'b) \Rightarrow bool$
 $strict-mono$:: $('a \Rightarrow 'b) \Rightarrow bool$

Syntax

$x \geq y$ \equiv $y \leq x$ (\geq)
 $x > y$ \equiv $y < x$
 $\forall x \leq y. P$ \equiv $\forall x. x \leq y \longrightarrow P$
 $\exists x \leq y. P$ \equiv $\exists x. x \leq y \wedge P$
 Similarly for $<$, \geq and $>$
 $LEAST x. P$ \equiv $Least (\lambda x. P)$

3 Lattices

Classes semilattice, lattice, distributive lattice and complete lattice (the latter in theory *Set*).

inf :: $'a \Rightarrow 'a \Rightarrow 'a$
 sup :: $'a \Rightarrow 'a \Rightarrow 'a$
 Inf :: $'a\ set \Rightarrow 'a$
 Sup :: $'a\ set \Rightarrow 'a$

Syntax

Available by loading theory *Lattice-Syntax* in directory *Library*.

$x \sqsubseteq y$ \equiv $x \leq y$
 $x \sqsubset y$ \equiv $x < y$
 $x \sqcap y$ \equiv $inf\ x\ y$
 $x \sqcup y$ \equiv $sup\ x\ y$
 $\bigsqcap A$ \equiv $Sup\ A$
 $\bigsqcup A$ \equiv $Inf\ A$
 \top \equiv top
 \perp \equiv bot

4 Set

Sets are predicates: $'a\ set = 'a \Rightarrow bool$

$\{\}$:: $'a\ set$
 $insert$:: $'a \Rightarrow 'a\ set \Rightarrow 'a\ set$
 $Collect$:: $('a \Rightarrow bool) \Rightarrow 'a\ set$
 $op \in$:: $'a \Rightarrow 'a\ set \Rightarrow bool$ ($:$)
 $op \cup$:: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ (\cup)

$op \cap \quad :: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set} \quad (\text{Int})$
 $UNION \quad :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow 'b \text{ set}$
 $INTER \quad :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow 'b \text{ set}$
 $Union \quad :: 'a \text{ set set} \Rightarrow 'a \text{ set}$
 $Inter \quad :: 'a \text{ set set} \Rightarrow 'a \text{ set}$
 $Pow \quad :: 'a \text{ set} \Rightarrow 'a \text{ set set}$
 $UNIV \quad :: 'a \text{ set}$
 $op ' \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set}$
 $Ball \quad :: 'a \text{ set} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$
 $Bex \quad :: 'a \text{ set} \Rightarrow ('a \Rightarrow \text{bool}) \Rightarrow \text{bool}$

Syntax

$\{x_1, \dots, x_n\} \equiv insert\ x_1\ (\dots\ (insert\ x_n\ \{\})\dots)$
 $x \notin A \equiv \neg(x \in A)$
 $A \subseteq B \equiv A \leq B$
 $A \subset B \equiv A < B$
 $A \supseteq B \equiv B \leq A$
 $A \supset B \equiv B < A$
 $\{x. P\} \equiv Collect\ (\lambda x. P)$
 $\bigcup_{x \in I}. A \equiv UNION\ I\ (\lambda x. A) \quad (\text{UN})$
 $\bigcup x. A \equiv UNION\ UNIV\ (\lambda x. A)$
 $\bigcap_{x \in I}. A \equiv INTER\ I\ (\lambda x. A) \quad (\text{INT})$
 $\bigcap x. A \equiv INTER\ UNIV\ (\lambda x. A)$
 $\forall x \in A. P \equiv Ball\ A\ (\lambda x. P)$
 $\exists x \in A. P \equiv Bex\ A\ (\lambda x. P)$
 $range\ f \equiv f\ ' UNIV$

5 Fun

$id \quad :: 'a \Rightarrow 'a$
 $op \circ \quad :: ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'a) \Rightarrow 'c \Rightarrow 'b \quad (\text{o})$
 $inj\text{-on} \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow \text{bool}$
 $inj \quad :: ('a \Rightarrow 'b) \Rightarrow \text{bool}$
 $surj \quad :: ('a \Rightarrow 'b) \Rightarrow \text{bool}$
 $bij \quad :: ('a \Rightarrow 'b) \Rightarrow \text{bool}$
 $bij\text{-betw} \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow \text{bool}$
 $fun\text{-upd} \quad :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b$

Syntax

$f(x := y) \equiv fun\text{-upd}\ f\ x\ y$
 $f(x_1 := y_1, \dots, x_n := y_n) \equiv f(x_1 := y_1) \dots (x_n := y_n)$

6 Hilbert_Choice

Hilbert's selection (ε) operator: *SOME* x . P .

$inv\text{-}into :: 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$

Syntax

$inv \equiv inv\text{-}into\ UNIV$

7 Fixed Points

Theory: *Inductive*.

Least and greatest fixed points in a complete lattice $'a$:

$lfp :: ('a \Rightarrow 'a) \Rightarrow 'a$

$gfp :: ('a \Rightarrow 'a) \Rightarrow 'a$

Note that in particular sets $('a \Rightarrow bool)$ are complete lattices.

8 Sum_Type

Type constructor $+$.

$Inl :: 'a \Rightarrow 'a + 'b$

$Inr :: 'a \Rightarrow 'b + 'a$

$op\ <+> :: 'a\ set \Rightarrow 'b\ set \Rightarrow ('a + 'b)\ set$

9 Product_Type

Types *unit* and \times .

$() :: unit$

$Pair :: 'a \Rightarrow 'b \Rightarrow 'a \times 'b$

$fst :: 'a \times 'b \Rightarrow 'a$

$snd :: 'a \times 'b \Rightarrow 'b$

$split :: ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \times 'b \Rightarrow 'c$

$curry :: ('a \times 'b \Rightarrow 'c) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'c$

$Sigma :: 'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow ('a \times 'b)\ set$

Syntax

$(a, b) \equiv Pair\ a\ b$

$\lambda(x, y). t \equiv split\ (\lambda x\ y. t)$

$A \times B \equiv Sigma\ A\ (\lambda_. B)\ (<*>)$

Pairs may be nested. Nesting to the right is printed as a tuple, e.g. (a, b, c) is really $(a, (b, c))$. Pattern matching with pairs and tuples extends to all binders,

e.g. $\forall (x, y) \in A. P, \{(x, y). P\}$, etc.

10 Relation

converse :: ('a × 'b) set ⇒ ('b × 'a) set
op O :: ('a × 'b) set ⇒ ('b × 'c) set ⇒ ('a × 'c) set
op “ :: ('a × 'b) set ⇒ 'a set ⇒ 'b set
inv-image :: ('a × 'a) set ⇒ ('b ⇒ 'a) ⇒ ('b × 'b) set
Id-on :: 'a set ⇒ ('a × 'a) set
Id :: ('a × 'a) set
Domain :: ('a × 'b) set ⇒ 'a set
Range :: ('a × 'b) set ⇒ 'b set
Field :: ('a × 'a) set ⇒ 'a set
refl-on :: 'a set ⇒ ('a × 'a) set ⇒ bool
refl :: ('a × 'a) set ⇒ bool
sym :: ('a × 'a) set ⇒ bool
antisym :: ('a × 'a) set ⇒ bool
trans :: ('a × 'a) set ⇒ bool
irrefl :: ('a × 'a) set ⇒ bool
total-on :: 'a set ⇒ ('a × 'a) set ⇒ bool
total :: ('a × 'a) set ⇒ bool

Syntax

$r^{-1} \equiv \text{converse } r \quad (\hat{-1})$

11 Equiv_Relations

equiv :: 'a set ⇒ ('a × 'a) set ⇒ bool
op // :: 'a set ⇒ ('a × 'a) set ⇒ 'a set set
congruent :: ('a × 'a) set ⇒ ('a ⇒ 'b) ⇒ bool
congruent2 :: ('a × 'a) set ⇒ ('b × 'b) set ⇒ ('a ⇒ 'b ⇒ 'c) ⇒ bool

Syntax

f respects r ≡ *congruent r f*
f respects2 r ≡ *congruent2 r r f*

12 Transitive_Closure

rtranc1 :: ('a × 'a) set ⇒ ('a × 'a) set
tranc1 :: ('a × 'a) set ⇒ ('a × 'a) set
reflcl :: ('a × 'a) set ⇒ ('a × 'a) set
op ^^ :: ('a × 'a) set ⇒ nat ⇒ ('a × 'a) set

$op + \quad op - \quad uminus \quad op * \quad op ^ \quad op div \quad op mod \quad op dvd$
 $op \leq \quad op < \quad min \quad max \quad Min \quad Max$
 $abs \quad sgn$
 $nat \quad :: \quad int \Rightarrow nat$
 $of-int \quad :: \quad int \Rightarrow 'a$
 $\mathbb{Z} \quad :: \quad 'a \text{ set} \quad (\text{Ints})$

Syntax

$int \equiv of-nat$

16 Finite_Set

$finite \quad :: \quad 'a \text{ set} \Rightarrow bool$
 $card \quad :: \quad 'a \text{ set} \Rightarrow nat$
 $fold \quad :: \quad ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a \text{ set} \Rightarrow 'b$
 $fold-image \quad :: \quad ('b \Rightarrow 'b \Rightarrow 'b) \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a \text{ set} \Rightarrow 'b$
 $setsum \quad :: \quad ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b$
 $setprod \quad :: \quad ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b$

Syntax

$\sum A \quad \equiv \quad setsum (\lambda x. x) A \quad (\text{SUM})$
 $\sum_{x \in A} t \quad \equiv \quad setsum (\lambda x. t) A$
 $\sum_{x|P} t \quad \equiv \quad \sum x | P. t$
 Similarly for \prod instead of \sum (PROD)

17 Wellfounded

$wf \quad :: \quad ('a \times 'a) \text{ set} \Rightarrow bool$
 $acyclic \quad :: \quad ('a \times 'a) \text{ set} \Rightarrow bool$
 $acc \quad :: \quad ('a \times 'a) \text{ set} \Rightarrow 'a \text{ set}$
 $measure \quad :: \quad ('a \Rightarrow nat) \Rightarrow ('a \times 'a) \text{ set}$
 $op <*lex*> \quad :: \quad ('a \times 'a) \text{ set} \Rightarrow ('b \times 'b) \text{ set} \Rightarrow (('a \times 'b) \times 'a \times 'b) \text{ set}$
 $op <*mlex*> \quad :: \quad ('a \Rightarrow nat) \Rightarrow ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
 $less-than \quad :: \quad (nat \times nat) \text{ set}$
 $pred-nat \quad :: \quad (nat \times nat) \text{ set}$

18 SetInterval

$lessThan \quad :: \quad 'a \Rightarrow 'a \text{ set}$
 $atMost \quad :: \quad 'a \Rightarrow 'a \text{ set}$
 $greaterThan \quad :: \quad 'a \Rightarrow 'a \text{ set}$

$atLeast \quad \quad \quad :: 'a \Rightarrow 'a \text{ set}$
 $greaterThanLessThan :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $atLeastLessThan \quad \quad :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $greaterThanAtMost \quad \quad :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $atLeastAtMost \quad \quad \quad :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$

Syntax

$\{..<y\} \quad \quad \quad \equiv \text{lessThan } y$
 $\{..y\} \quad \quad \quad \equiv \text{atMost } y$
 $\{x<..\} \quad \quad \quad \equiv \text{greaterThan } x$
 $\{x..\} \quad \quad \quad \equiv \text{atLeast } x$
 $\{x<..<y\} \quad \quad \quad \equiv \text{greaterThanLessThan } x \ y$
 $\{x..<y\} \quad \quad \quad \equiv \text{atLeastLessThan } x \ y$
 $\{x<..y\} \quad \quad \quad \equiv \text{greaterThanAtMost } x \ y$
 $\{x..y\} \quad \quad \quad \equiv \text{atLeastAtMost } x \ y$
 $\bigcup i \leq n. A \quad \quad \equiv \bigcup i \in \{..n\}. A$
 $\bigcup i < n. A \quad \quad \equiv \bigcup i \in \{..<n\}. A$

Similarly for \bigcap instead of \bigcup

$\sum x = a..b. t \quad \equiv \text{setsum } (\lambda x. t) \{a..b\}$
 $\sum x = a..<b. t \quad \equiv \text{setsum } (\lambda x. t) \{a..<b\}$
 $\sum x \leq b. t \quad \equiv \text{setsum } (\lambda x. t) \{..b\}$
 $\sum x < b. t \quad \equiv \text{setsum } (\lambda x. t) \{..<b\}$

Similarly for \prod instead of \sum

19 Power

$op \wedge :: 'a \Rightarrow nat \Rightarrow 'a$

20 Option

datatype $'a \text{ option} = None \mid Some 'a$

$the \quad \quad \quad :: 'a \text{ option} \Rightarrow 'a$
 $Option.map :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ option} \Rightarrow 'b \text{ option}$
 $Option.set \quad \quad :: 'a \text{ option} \Rightarrow 'a \text{ set}$
 $Option.bind :: 'a \text{ option} \Rightarrow ('a \Rightarrow 'b \text{ option}) \Rightarrow 'b \text{ option}$

21 List

datatype $'a \text{ list} = [] \mid op \# 'a ('a \text{ list})$

op @ :: 'a list ⇒ 'a list ⇒ 'a list
butlast :: 'a list ⇒ 'a list
concat :: 'a list list ⇒ 'a list
distinct :: 'a list ⇒ bool
drop :: nat ⇒ 'a list ⇒ 'a list
dropWhile :: ('a ⇒ bool) ⇒ 'a list ⇒ 'a list
filter :: ('a ⇒ bool) ⇒ 'a list ⇒ 'a list
foldl :: ('a ⇒ 'b ⇒ 'a) ⇒ 'a ⇒ 'b list ⇒ 'a
foldr :: ('a ⇒ 'b ⇒ 'b) ⇒ 'a list ⇒ 'b ⇒ 'b
hd :: 'a list ⇒ 'a
last :: 'a list ⇒ 'a
length :: 'a list ⇒ nat
lenlex :: ('a × 'a) set ⇒ ('a list × 'a list) set
lex :: ('a × 'a) set ⇒ ('a list × 'a list) set
lexn :: ('a × 'a) set ⇒ nat ⇒ ('a list × 'a list) set
lexord :: ('a × 'a) set ⇒ ('a list × 'a list) set
listrel :: ('a × 'a) set ⇒ ('a list × 'a list) set
listrel1 :: ('a × 'a) set ⇒ ('a list × 'a list) set
lists :: 'a set ⇒ 'a list set
listset :: 'a set list ⇒ 'a list set
listsum :: 'a list ⇒ 'a
list-all2 :: ('a ⇒ 'b ⇒ bool) ⇒ 'a list ⇒ 'b list ⇒ bool
list-update :: 'a list ⇒ nat ⇒ 'a ⇒ 'a list
map :: ('a ⇒ 'b) ⇒ 'a list ⇒ 'b list
measures :: ('a ⇒ nat) list ⇒ ('a × 'a) set
op ! :: 'a list ⇒ nat ⇒ 'a
remdups :: 'a list ⇒ 'a list
removeAll :: 'a ⇒ 'a list ⇒ 'a list
remove1 :: 'a ⇒ 'a list ⇒ 'a list
replicate :: nat ⇒ 'a ⇒ 'a list
rev :: 'a list ⇒ 'a list
rotate :: nat ⇒ 'a list ⇒ 'a list
rotate1 :: 'a list ⇒ 'a list
set :: 'a list ⇒ 'a set
sort :: 'a list ⇒ 'a list
sorted :: 'a list ⇒ bool
splice :: 'a list ⇒ 'a list ⇒ 'a list
sublist :: 'a list ⇒ (nat ⇒ bool) ⇒ 'a list
take :: nat ⇒ 'a list ⇒ 'a list
takeWhile :: ('a ⇒ bool) ⇒ 'a list ⇒ 'a list
tl :: 'a list ⇒ 'a list

$upt :: nat \Rightarrow nat \Rightarrow nat\ list$
 $upto :: int \Rightarrow int \Rightarrow int\ list$
 $zip :: 'a\ list \Rightarrow 'b\ list \Rightarrow ('a \times 'b)\ list$

Syntax

$[x_1, \dots, x_n] \equiv x_1 \# \dots \# x_n \# []$
 $[m..<n] \equiv upt\ m\ n$
 $[i..j] \equiv upto\ i\ j$
 $[e.\ x \leftarrow xs] \equiv map\ (\lambda x.\ e)\ xs$
 $[x \leftarrow xs.\ b] \equiv filter\ (\lambda x.\ b)\ xs$
 $xs[n := x] \equiv list\text{-}update\ xs\ n\ x$
 $\sum x \leftarrow xs.\ e \equiv listsum\ (map\ (\lambda x.\ e)\ xs)$

List comprehension: $[e.\ q_1, \dots, q_n]$ where each qualifier q_i is either a generator $pat \leftarrow e$ or a guard, i.e. boolean expression.

22 Map

Maps model partial functions and are often used as finite tables. However, the domain of a map may be infinite.

$'a \mapsto 'b = 'a \Rightarrow 'b\ option$

$Map.empty :: 'a \Rightarrow 'b\ option$
 $op ++ :: ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow 'a \Rightarrow 'b\ option$
 $op \circ_m :: ('a \Rightarrow 'b\ option) \Rightarrow ('c \Rightarrow 'a\ option) \Rightarrow 'c \Rightarrow 'b\ option$
 $op |' :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b\ option$
 $dom :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set$
 $ran :: ('a \Rightarrow 'b\ option) \Rightarrow 'b\ set$
 $op \subseteq_m :: ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow bool$
 $map\text{-}of :: ('a \times 'b)\ list \Rightarrow 'a \Rightarrow 'b\ option$
 $map\text{-}upds :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow 'a \Rightarrow 'b\ option$

Syntax

$Map.empty \equiv \lambda x.\ None$
 $m(x \mapsto y) \equiv m(x := Some\ y)$
 $m(x_1 \mapsto y_1, \dots, x_n \mapsto y_n) \equiv m(x_1 \mapsto y_1) \dots (x_n \mapsto y_n)$
 $[x_1 \mapsto y_1, \dots, x_n \mapsto y_n] \equiv Map.empty(x_1 \mapsto y_1, \dots, x_n \mapsto y_n)$
 $m(xs\ [\mapsto]\ ys) \equiv map\text{-}upds\ m\ xs\ ys$