

What's in Main

Tobias Nipkow

February 17, 2016

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. For infix operators and their precedences see the final section. The sophisticated class structure is only hinted at. For details see <http://isabelle.in.tum.de/library/HOL/>.

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 ($\lambda x. e_2$)

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

$bot \quad :: 'a$
 $mono \quad :: ('a \Rightarrow 'b) \Rightarrow bool$
 $strict_mono :: ('a \Rightarrow 'b) \Rightarrow bool$

Syntax

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

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*.

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

Set

$\{\} \quad :: 'a\ set$
 $insert \quad :: 'a \Rightarrow 'a\ set \Rightarrow 'a\ set$
 $Collect \quad :: ('a \Rightarrow bool) \Rightarrow 'a\ set$
 $op \in \quad :: 'a \Rightarrow 'a\ set \Rightarrow bool \quad (:)$
 $op \cup \quad :: 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \quad (Un)$
 $op \cap \quad :: 'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set \quad (Int)$
 $UNION \quad :: 'a\ set \Rightarrow ('a \Rightarrow 'b\ set) \Rightarrow 'b\ set$

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

Syntax

$\{a_1, \dots, a_n\} \equiv insert\ a_1\ (\dots\ (insert\ a_n\ \{\})\dots)$
 $a \notin A \equiv \neg(x \in A)$
 $A \subseteq B \equiv A \leq B$
 $A \subset B \equiv A < B$
 $A \supseteq B \equiv B \leq A$
 $A \supset B \equiv B < A$
 $\{x. P\} \equiv Collect\ (\lambda x. P)$
 $\{t \mid x_1 \dots x_n. P\} \equiv \{v. \exists x_1 \dots x_n. v = t \wedge P\}$
 $\bigcup_{x \in I}. A \equiv UNION\ I\ (\lambda x. A) \quad (\text{UN})$
 $\bigcup x. A \equiv UNION\ UNIV\ (\lambda x. A)$
 $\bigcap_{x \in I}. A \equiv INTER\ I\ (\lambda x. A) \quad (\text{INT})$
 $\bigcap x. A \equiv INTER\ UNIV\ (\lambda x. A)$
 $\forall x \in A. P \equiv Ball\ A\ (\lambda x. P)$
 $\exists x \in A. P \equiv Bex\ A\ (\lambda x. P)$
 $range\ f \equiv f \text{ ' } UNIV$

Fun

$id :: 'a \Rightarrow 'a$
 $op \circ :: ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'a) \Rightarrow 'c \Rightarrow 'b \quad (\circ)$
 $inj_on :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow bool$
 $inj :: ('a \Rightarrow 'b) \Rightarrow bool$
 $surj :: ('a \Rightarrow 'b) \Rightarrow bool$
 $bij :: ('a \Rightarrow 'b) \Rightarrow bool$
 $bij_betw :: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow bool$
 $fun_upd :: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b$

Syntax

$f(x := y) \equiv fun_upd\ f\ x\ y$
 $f(x_1 := y_1, \dots, x_n := y_n) \equiv f(x_1 := y_1) \dots (x_n := y_n)$

Hilbert__Choice

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

$inv_into :: 'a\ set \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$

Syntax

$inv \equiv inv_into\ UNIV$

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.

Sum__Type

Type constructor $+$.

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

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

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

Product__Type

Types *unit* and \times .

$() :: unit$

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

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

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

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

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

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

Syntax

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

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

$A \times B \equiv Sigma\ A\ (\lambda_. B)$

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.

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

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

Type synonym $'a \text{ rel} = ('a \times 'a) \text{ set}$

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*

Transitive_Closure

$rtranc1 :: ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
 $tranc1 :: ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
 $reflcl :: ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
 $acyclic :: ('a \times 'a) \text{ set} \Rightarrow \text{bool}$
 $op \sim :: ('a \times 'a) \text{ set} \Rightarrow \text{nat} \Rightarrow ('a \times 'a) \text{ set}$

Syntax

$r^* \equiv rtranc1 \ r \ (\sim^*)$
 $r^+ \equiv tranc1 \ r \ (\sim^+)$
 $r^- \equiv reflcl \ r \ (\sim^=)$

Algebra

Theories *Groups*, *Rings*, *Fields* and *Divides* define a large collection of classes describing common algebraic structures from semigroups up to fields. Everything is done in terms of overloaded operators:

$0 :: 'a$
 $1 :: 'a$
 $op + :: 'a \Rightarrow 'a \Rightarrow 'a$
 $op - :: 'a \Rightarrow 'a \Rightarrow 'a$
 $uminus :: 'a \Rightarrow 'a \quad (-)$
 $op * :: 'a \Rightarrow 'a \Rightarrow 'a$
 $inverse :: 'a \Rightarrow 'a$
 $op div :: 'a \Rightarrow 'a \Rightarrow 'a$
 $abs :: 'a \Rightarrow 'a$
 $sgn :: 'a \Rightarrow 'a$
 $op dvd :: 'a \Rightarrow 'a \Rightarrow \text{bool}$
 $op div :: 'a \Rightarrow 'a \Rightarrow 'a$
 $op mod :: 'a \Rightarrow 'a \Rightarrow 'a$

Syntax

$|x| \equiv abs \ x$

Nat

datatype $nat = 0 \mid Suc \ nat$

$op + \quad op - \quad op * \quad op \wedge \quad op div \quad op mod \quad op dvd$
 $op \leq \quad op < \quad min \quad max \quad Min \quad Max$
 $of_nat :: nat \Rightarrow 'a$
 $op \sim :: ('a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a$

Int

Type *int*

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

Syntax

$\text{int} \equiv \text{of_nat}$

Finite_Set

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

Syntax

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

Wellfounded

$\text{wf} \quad :: ('a \times 'a) \text{ set} \Rightarrow \text{bool}$
 $\text{Wellfounded.acc} :: ('a \times 'a) \text{ set} \Rightarrow 'a \text{ set}$
 $\text{measure} \quad :: ('a \Rightarrow \text{nat}) \Rightarrow ('a \times 'a) \text{ set}$
 $op < * \text{lex} * > \quad :: ('a \times 'a) \text{ set} \Rightarrow ('b \times 'b) \text{ set} \Rightarrow (('a \times 'b) \times 'a \times 'b) \text{ set}$
 $op < * \text{mlex} * > \quad :: ('a \Rightarrow \text{nat}) \Rightarrow ('a \times 'a) \text{ set} \Rightarrow ('a \times 'a) \text{ set}$
 $\text{less_than} \quad :: (\text{nat} \times \text{nat}) \text{ set}$
 $\text{pred_nat} \quad :: (\text{nat} \times \text{nat}) \text{ set}$

Set_Interval

$lessThan :: 'a \Rightarrow 'a \text{ set}$
 $atMost :: 'a \Rightarrow 'a \text{ set}$
 $greaterThan :: 'a \Rightarrow 'a \text{ set}$
 $atLeast :: 'a \Rightarrow 'a \text{ set}$
 $greaterThanLessThan :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $atLeastLessThan :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $greaterThanAtMost :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$
 $atLeastAtMost :: 'a \Rightarrow 'a \Rightarrow 'a \text{ set}$

Syntax

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

Similarly for \bigcap instead of \bigcup

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

Similarly for \prod instead of \sum

Power

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

Option

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

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

List

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

$op @ :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $butlast :: 'a\ list \Rightarrow 'a\ list$
 $concat :: 'a\ list\ list \Rightarrow 'a\ list$
 $distinct :: 'a\ list \Rightarrow bool$
 $drop :: nat \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $dropWhile :: ('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $filter :: ('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $find :: ('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ option$
 $fold :: ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow 'b \Rightarrow 'b$
 $foldr :: ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow 'b \Rightarrow 'b$
 $foldl :: ('a \Rightarrow 'b \Rightarrow 'a) \Rightarrow 'a \Rightarrow 'b\ list \Rightarrow 'a$
 $hd :: 'a\ list \Rightarrow 'a$
 $last :: 'a\ list \Rightarrow 'a$
 $length :: 'a\ list \Rightarrow nat$
 $lenlex :: ('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$
 $lex :: ('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$
 $lexn :: ('a \times 'a)\ set \Rightarrow nat \Rightarrow ('a\ list \times 'a\ list)\ set$
 $lexord :: ('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$
 $listrel :: ('a \times 'b)\ set \Rightarrow ('a\ list \times 'b\ list)\ set$
 $listrel1 :: ('a \times 'a)\ set \Rightarrow ('a\ list \times 'a\ list)\ set$
 $lists :: 'a\ set \Rightarrow 'a\ list\ set$
 $listset :: 'a\ set\ list \Rightarrow 'a\ list\ set$
 $listsum :: 'a\ list \Rightarrow 'a$
 $list_all2 :: ('a \Rightarrow 'b \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow bool$
 $list_update :: 'a\ list \Rightarrow nat \Rightarrow 'a \Rightarrow 'a\ list$
 $map :: ('a \Rightarrow 'b) \Rightarrow 'a\ list \Rightarrow 'b\ list$
 $measures :: ('a \Rightarrow nat)\ list \Rightarrow ('a \times 'a)\ set$
 $op ! :: 'a\ list \Rightarrow nat \Rightarrow 'a$
 $remdups :: 'a\ list \Rightarrow 'a\ list$
 $removeAll :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $remove1 :: 'a \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $replicate :: nat \Rightarrow 'a \Rightarrow 'a\ list$
 $rev :: 'a\ list \Rightarrow 'a\ list$
 $rotate :: nat \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $rotate1 :: 'a\ list \Rightarrow 'a\ list$
 $set :: 'a\ list \Rightarrow 'a\ set$
 $sort :: 'a\ list \Rightarrow 'a\ list$
 $sorted :: 'a\ list \Rightarrow bool$
 $splice :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $sublist :: 'a\ list \Rightarrow nat\ set \Rightarrow 'a\ list$
 $take :: nat \Rightarrow 'a\ list \Rightarrow 'a\ list$

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

Syntax

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

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.

Map

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

$Map.empty :: 'a \Rightarrow 'b\ option$
 $op ++ :: ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow 'a \Rightarrow 'b\ option$
 $op \circ_m :: ('a \Rightarrow 'b\ option) \Rightarrow ('c \Rightarrow 'a\ option) \Rightarrow 'c \Rightarrow 'b\ option$
 $op |' :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b\ option$
 $dom :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set$
 $ran :: ('a \Rightarrow 'b\ option) \Rightarrow 'b\ set$
 $op \subseteq_m :: ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow bool$
 $map_of :: ('a \times 'b)\ list \Rightarrow 'a \Rightarrow 'b\ option$
 $map_upds :: ('a \Rightarrow 'b\ option) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow 'a \Rightarrow 'b\ option$

Syntax

$Map.empty \equiv \lambda x. None$
 $m(x \mapsto y) \equiv m(x := Some\ y)$
 $m(x_1 \mapsto y_1, \dots, x_n \mapsto y_n) \equiv m(x_1 \mapsto y_1) \dots (x_n \mapsto y_n)$
 $[x_1 \mapsto y_1, \dots, x_n \mapsto y_n] \equiv Map.empty(x_1 \mapsto y_1, \dots, x_n \mapsto y_n)$
 $m(xs\ [\mapsto]\ ys) \equiv map_upds\ m\ xs\ ys$

Infix operators in Main

	Operator	precedence	associativity
Meta-logic	\implies	1	right
	\equiv	2	
Logic	\wedge	35	right
	\vee	30	right
	$\longrightarrow, \longleftrightarrow$	25	right
	$=, \neq$	50	left
Orderings	$\leq, <, \geq, >$	50	
Sets	$\subseteq, \subset, \supseteq, \supset$	50	
	\in, \notin	50	
	\cap	70	left
	\cup	65	left
Functions and Relations	\circ	55	left
	$'$	90	right
	O	75	right
	$''$	90	right
	\sim	80	right
Numbers	$+, -$	65	left
	$*, /$	70	left
	div, mod	70	left
	\wedge	80	right
	dvd	50	
Lists	$\#, @$	65	right
	$!$	100	left