

What's in Main

Tobias Nipkow

August 19, 2026

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 <https://isabelle.in.tum.de/library/HOL/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

$x \neq y$	$\equiv$	$\neg (x = y)$	$(\sim=)$
$P \longleftrightarrow Q$	$\equiv$	$P = Q$	
<i>if</i> x <i>then</i> y <i>else</i> z	$\equiv$	<i>If</i> x y z	
<i>let</i> $x = e_1$ <i>in</i> e_2	$\equiv$	<i>Let</i> e_1 $(\lambda x. e_2)$	

Orderings

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

$(\leq) \quad :: 'a \Rightarrow 'a \Rightarrow bool \quad (<=)$
 $(<) \quad :: 'a \Rightarrow 'a \Rightarrow bool$
 $Least \quad :: ('a \Rightarrow bool) \Rightarrow 'a$
 $Greatest \quad :: ('a \Rightarrow bool) \Rightarrow 'a$
 $min \quad :: 'a \Rightarrow 'a \Rightarrow 'a$
 $max \quad :: 'a \Rightarrow 'a \Rightarrow 'a$
 $top \quad :: 'a$
 $bot \quad :: 'a$

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 $>$
 $LEAST x. P \quad \equiv \quad Least (\lambda x. P)$
 $GREATEST x. P \quad \equiv \quad Greatest (\lambda x. P)$

Lattices

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

$inf \quad :: 'a \Rightarrow 'a \Rightarrow 'a$
 $sup \quad :: 'a \Rightarrow 'a \Rightarrow 'a$
 $Inf \quad :: 'a \text{ set} \Rightarrow 'a$
 $Sup \quad :: 'a \text{ set} \Rightarrow 'a$

Syntax

Available via **unbundle** *lattice_syntax*.

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

Set

$\{\}$	$:: 'a \text{ set}$	
$insert$	$:: 'a \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$	
$Collect$	$:: ('a \Rightarrow bool) \Rightarrow 'a \text{ set}$	
$(\in)$	$:: 'a \Rightarrow 'a \text{ set} \Rightarrow bool$	$(:)$
$(\cup)$	$:: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$	(Un)
$(\cap)$	$:: 'a \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'a \text{ set}$	(Int)
$\bigcup$	$:: 'a \text{ set set} \Rightarrow 'a \text{ set}$	
$\bigcap$	$:: 'a \text{ set set} \Rightarrow 'a \text{ set}$	
Pow	$:: 'a \text{ set} \Rightarrow 'a \text{ set set}$	
$UNIV$	$:: 'a \text{ set}$	
$(')$	$:: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set}$	
$Ball$	$:: 'a \text{ set} \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$	
Bex	$:: 'a \text{ set} \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$	

Syntax

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

Fun

id	$:: 'a \Rightarrow 'a$	
$(\circ)$	$:: ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'a) \Rightarrow 'c \Rightarrow 'b$	(o)
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	$:: ('a \Rightarrow 'b) \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow bool$	
$monotone_on$	$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
$monotone$	$:: ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
$mono_on$	$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
$mono$	$:: ('a \Rightarrow 'b) \Rightarrow bool$	
$strict_mono_on$	$:: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
$strict_mono$	$:: ('a \Rightarrow 'b) \Rightarrow bool$	
$antimono$	$:: ('a \Rightarrow 'b) \Rightarrow bool$	
fun_upd	$:: ('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b \Rightarrow 'a \Rightarrow 'b$	

Syntax

$$\begin{aligned}
 f(x := y) &\equiv fun_upd\ f\ x\ y \\
 f(x_1 := y_1, \dots, x_n := y_n) &\equiv f(x_1 := y_1) \dots (x_n := y_n)
 \end{aligned}$$

Hilbert__Choice

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

$$inv_into :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$$

Syntax

$$inv \equiv inv_into\ UNIV$$

Fixed Points

Theory: *HOL.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 $+$.

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

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

$(<+>) :: 'a \text{ set} \Rightarrow 'b \text{ set} \Rightarrow ('a + 'b) \text{ set}$

Product__Type

Types *unit* and $\times$.

$() \quad :: unit$

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

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

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

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

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

$Sigma \quad :: 'a \text{ set} \Rightarrow ('a \Rightarrow 'b \text{ set}) \Rightarrow ('a \times 'b) \text{ set}$

Syntax

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

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

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

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

$converse \quad :: ('a \times 'b) \text{ set} \Rightarrow ('b \times 'a) \text{ set}$

$(O) \quad :: ('a \times 'b) \text{ set} \Rightarrow ('b \times 'c) \text{ set} \Rightarrow ('a \times 'c) \text{ set}$

$(\text{“}) \quad :: ('a \times 'b) \text{ set} \Rightarrow 'a \text{ set} \Rightarrow 'b \text{ set}$

$inv_image :: ('a \times 'a) \text{ set} \Rightarrow ('b \Rightarrow 'a) \Rightarrow ('b \times 'b) \text{ set}$

$Id_on \quad :: 'a \text{ set} \Rightarrow ('a \times 'a) \text{ set}$

$Id \quad :: ('a \times 'a) \text{ set}$

$Domain \quad :: ('a \times 'b) \text{ set} \Rightarrow 'a \text{ set}$

$Range \quad :: ('a \times 'b) \text{ set} \Rightarrow 'b \text{ set}$

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 rel = ('a × 'a) set

Equiv_Relations

equiv :: 'a set ⇒ ('a × 'a) set ⇒ bool
(//) :: '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

rtrancl :: ('a × 'a) set ⇒ ('a × 'a) set
trancl :: ('a × 'a) set ⇒ ('a × 'a) set
reflcl :: ('a × 'a) set ⇒ ('a × 'a) set
acyclic :: ('a × 'a) set ⇒ bool
($\sim$) :: ('a × 'a) set ⇒ nat ⇒ ('a × 'a) set

Syntax

$$\begin{aligned} r^* &\equiv rtranc\ r \quad (\wedge^*) \\ r^+ &\equiv tranc\ r \quad (\wedge^+) \\ r^- &\equiv reflcl\ r \quad (\wedge^=) \end{aligned}$$

Algebra

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

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

Syntax

$$|x| \equiv abs\ x$$

Nat

datatype *nat* = 0 | *Suc nat*

$$\begin{aligned} (+) \quad (-) \quad (*) \quad (\neg) \quad (div) \quad (mod) \quad (dvd) \\ (\leq) \quad (<) \quad min \quad max \quad Min \quad Max \\ of_nat &:: nat \Rightarrow 'a \\ (\wedge) &:: ('a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a \end{aligned}$$

Int

Type *int*

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

Syntax

int $\equiv$ *of_nat*

Finite_Set

finite $:: 'a\ set \Rightarrow bool$
card $:: 'a\ set \Rightarrow nat$
Finite_Set.fold $:: ('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a\ set \Rightarrow 'b$

Lattices_Big

Min $:: 'a\ set \Rightarrow 'a$
Max $:: 'a\ set \Rightarrow 'a$
arg_min $:: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a$
is_arg_min $:: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$
arg_max $:: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a$
is_arg_max $:: ('a \Rightarrow 'b) \Rightarrow ('a \Rightarrow bool) \Rightarrow 'a \Rightarrow bool$

Syntax

ARG_MIN *f x. P* $\equiv$ *arg_min* *f* $(\lambda x. P)$
ARG_MAX *f x. P* $\equiv$ *arg_max* *f* $(\lambda x. P)$

Groups_Big

sum $:: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b$
prod $:: ('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b$

Syntax

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

Wellfounded

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

Set_Interval

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

Syntax

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

Similarly for $\bigcap$ instead of $\bigcup$

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

Similarly for $\prod$ instead of $\sum$

Power

$(\frown) :: 'a \Rightarrow nat \Rightarrow 'a$

Option

datatype $'a\ option = None \mid Some\ 'a$

$the :: 'a\ option \Rightarrow 'a$

$map_option :: ('a \Rightarrow 'b) \Rightarrow 'a\ option \Rightarrow 'b\ option$

$set_option :: 'a\ option \Rightarrow 'a\ set$

$Option.bind :: 'a\ option \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow 'b\ option$

List

datatype $'a\ list = [] \mid (\#)\ 'a\ ('a\ list)$

$(@) :: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$

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
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
(!) :: 'a list $\Rightarrow$ nat $\Rightarrow$ 'a
nths :: 'a list $\Rightarrow$ nat set $\Rightarrow$ 'a list
prod_list :: 'a list $\Rightarrow$ '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
shuffles :: 'a list $\Rightarrow$ 'a list $\Rightarrow$ 'a list set
sort :: 'a list $\Rightarrow$ 'a list
sorted :: 'a list $\Rightarrow$ bool

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

Syntax

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

Filter input syntax $[pat \leftarrow e. b]$, where pat is a tuple pattern, which stands for $filter\ (\lambda pat. b)\ e$.

List comprehension input syntax: $[e. q_1, \dots, q_n]$ 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.

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

Syntax

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

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