What's in Main

Tobias Nipkow

October 25, 2022

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*.

$undefined :: {}'a$
$default \quad :: {}'a$

Syntax

$x \neq y$	$\equiv$	$\neg\,(x = y)$	(~=)
$P \longleftrightarrow Q$	$\equiv$	$P = Q$	
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.

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

Syntax

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

Lattices

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

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

Syntax

Available via **unbundle** *lattice_syntax*.

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

Set

$\{\}$ $\quad$ :: $'a\ set$

$insert$ $\quad$ :: $'a \Rightarrow 'a\ set \Rightarrow 'a\ set$

$Collect$ :: $('a \Rightarrow bool) \Rightarrow 'a\ set$

$(\in)$ $\quad$ :: $'a \Rightarrow 'a\ set \Rightarrow bool$ $\qquad\qquad$ (:)

$(\cup)$ $\quad$ :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ $\qquad$ (Un)

$(\cap)$ $\quad$ :: $'a\ set \Rightarrow 'a\ set \Rightarrow 'a\ set$ $\qquad$ (Int)

$\bigcup$ $\quad$ :: $'a\ set\ set \Rightarrow 'a\ set$

$\bigcap$ $\quad$ :: $'a\ set\ set \Rightarrow 'a\ set$

Pow $\quad$ :: $'a\ set \Rightarrow 'a\ set\ set$

$UNIV$ :: $'a\ set$

$(\grave{\ })$ $\quad$ :: $('a \Rightarrow 'b) \Rightarrow 'a\ set \Rightarrow 'b\ set$

$Ball$ $\quad$ :: $'a\ set \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$

Bex $\quad$ :: $'a\ set \Rightarrow ('a \Rightarrow bool) \Rightarrow bool$

Syntax

$\{a_1,\ldots,a_n\}$	$\equiv$	$insert\ a_1\ (\ldots\ (insert\ a_n\ \{\})\ldots)$	
$a \notin A$	$\equiv$	$\neg(x \in A)$	
$A \subseteq B$	$\equiv$	$A \leq B$	
$A \subset B$	$\equiv$	$A < B$	
$A \supseteq B$	$\equiv$	$B \leq A$	
$A \supset B$	$\equiv$	$B < A$	
$\{x.\ P\}$	$\equiv$	$Collect\ (\lambda x.\ P)$	
$\{t \mid x_1 \ldots x_n.\ P\}$	$\equiv$	$\{v.\ \exists x_1 \ldots x_n.\ v = t \wedge P\}$	
$\bigcup x{\in}I.\ A$	$\equiv$	$\bigcup ((\lambda x.\ A)\ \grave{\ }\ I)$	(UN)
$\bigcup x.\ A$	$\equiv$	$\bigcup ((\lambda x.\ A)\ \grave{\ }\ UNIV)$	
$\bigcap x{\in}I.\ A$	$\equiv$	$\bigcap ((\lambda x.\ A)\ \grave{\ }\ I)$	(INT)
$\bigcap x.\ A$	$\equiv$	$\bigcap ((\lambda x.\ A)\ \grave{\ }\ UNIV)$	
$\forall x{\in}A.\ P$	$\equiv$	$Ball\ A\ (\lambda x.\ P)$	
$\exists x{\in}A.\ P$	$\equiv$	$Bex\ A\ (\lambda x.\ P)$	
$range\ f$	$\equiv$	$f\ \grave{\ }\ UNIV$	

Fun

id	$:: \; 'a \Rightarrow 'a$	
$(\circ)$	$:: \; ('a \Rightarrow 'b) \Rightarrow ('c \Rightarrow 'a) \Rightarrow 'c \Rightarrow 'b$	$(\circ)$
inj_on	$:: \; ('a \Rightarrow 'b) \Rightarrow 'a \; set \Rightarrow bool$	
inj	$:: \; ('a \Rightarrow 'b) \Rightarrow bool$	
surj	$:: \; ('a \Rightarrow 'b) \Rightarrow bool$	
bij	$:: \; ('a \Rightarrow 'b) \Rightarrow bool$	
bij_betw	$:: \; ('a \Rightarrow 'b) \Rightarrow 'a \; set \Rightarrow 'b \; set \Rightarrow bool$	
monotone_on	$:: \; 'a \; set \Rightarrow ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
monotone	$:: \; ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow ('b \Rightarrow 'b \Rightarrow bool) \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
mono_on	$:: \; 'a \; set \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
mono	$:: \; ('a \Rightarrow 'b) \Rightarrow bool$	
strict_mono_on	$:: \; 'a \; set \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$	
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

$f(x := y)$	$\equiv$	$fun\_upd \; f \; x \; y$
$f(x_1{:=}y_1,\ldots,x_n{:=}y_n)$	$\equiv$	$f(x_1{:=}y_1)\ldots(x_n{:=}y_n)$

Hilbert_Choice

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

$inv\_into :: \; 'a \; set \Rightarrow ('a \Rightarrow 'b) \Rightarrow 'b \Rightarrow 'a$

Syntax

$inv \quad \equiv \quad inv\_into \; UNIV$

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

Product_Type

Types *unit* and ×.
()   :: *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

$(a,\ b)$ ≡   *Pair a b*
$\lambda(x,\ y).\ t$ ≡   *case_prod* $(\lambda x\ y.\ t)$
$A \times B$ ≡   *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   :: $('a \times 'b)\ set \Rightarrow ('b \times 'a)\ set$
(O)   :: $('a \times 'b)\ set \Rightarrow ('b \times 'c)\ set \Rightarrow ('a \times 'c)\ set$
(")   :: $('a \times 'b)\ set \Rightarrow 'a\ set \Rightarrow 'b\ set$
inv_image :: $('a \times 'a)\ set \Rightarrow ('b \Rightarrow 'a) \Rightarrow ('b \times 'b)\ set$
Id_on   :: $'a\ set \Rightarrow ('a \times 'a)\ set$
Id   :: $('a \times 'a)\ set$
Domain   :: $('a \times 'b)\ set \Rightarrow 'a\ set$
Range   :: $('a \times 'b)\ set \Rightarrow 'b\ set$

$$Field \quad :: \; ('a \times 'a) \; set \Rightarrow 'a \; set$$
$$refl\_on \; :: \; 'a \; set \Rightarrow ('a \times 'a) \; set \Rightarrow bool$$
$$refl \quad :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$sym \quad :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$antisym \; :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$trans \quad :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$irrefl \quad :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$total\_on \; :: \; 'a \; set \Rightarrow ('a \times 'a) \; set \Rightarrow bool$$
$$total \quad :: \; ('a \times 'a) \; set \Rightarrow bool$$

Syntax

$$r^{-1} \quad \equiv \quad converse \; r \quad (\texttt{\^{}-1})$$

Type synonym $\;'a \; rel = ('a \times 'a) \; set$

Equiv_Relations

$$equiv \quad :: \; 'a \; set \Rightarrow ('a \times 'a) \; set \Rightarrow bool$$
$$(//) \quad :: \; 'a \; set \Rightarrow ('a \times 'a) \; set \Rightarrow 'a \; set \; set$$
$$congruent \; :: \; ('a \times 'a) \; set \Rightarrow ('a \Rightarrow 'b) \Rightarrow bool$$
$$congruent2 \; :: \; ('a \times 'a) \; set \Rightarrow ('b \times 'b) \; set \Rightarrow ('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow bool$$

Syntax

$$f \; respects \; r \quad \equiv \quad congruent \; r \; f$$
$$f \; respects2 \; r \quad \equiv \quad congruent2 \; r \; r \; f$$

Transitive_Closure

$$rtrancl :: \; ('a \times 'a) \; set \Rightarrow ('a \times 'a) \; set$$
$$trancl \; :: \; ('a \times 'a) \; set \Rightarrow ('a \times 'a) \; set$$
$$reflcl \; :: \; ('a \times 'a) \; set \Rightarrow ('a \times 'a) \; set$$
$$acyclic :: \; ('a \times 'a) \; set \Rightarrow bool$$
$$(\frown) \quad :: \; ('a \times 'a) \; set \Rightarrow nat \Rightarrow ('a \times 'a) \; set$$

Syntax

$$
\begin{array}{lll}
r^* & \equiv & rtrancl\ r \quad (\verb|^*|) \\
r^+ & \equiv & trancl\ r \quad\ (\verb|^+|) \\
r^= & \equiv & reflcl\ r \quad\ (\verb|^=|)
\end{array}
$$

Algebra

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

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

Syntax

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

Nat

datatype $nat = 0 \mid Suc\ nat$

$$
\begin{array}{ccccccc}
(+) & (-) & (*) & (\frown) & (div) & (mod) & (dvd) \\
(\leq) & (<) & min & max & Min & Max
\end{array}
$$

$$
\begin{array}{ll}
of\_nat & :: nat \Rightarrow 'a \\
(\frown) & :: ('a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a
\end{array}
$$

Int

Type *int*

(+)	(−)	*uminus*	(∗)	(⌃)	(*div*)	(*mod*)	(*dvd*)
(≤)	(<)	*min*		*max*	*Min*	*Max*	
abs	*sgn*						

nat   :: *int* ⇒ *nat*
of_int :: *int* ⇒ ′*a*
ℤ   :: ′*a set*   (`Ints`)

Syntax

int   ≡   *of_nat*

Finite_Set

finite   :: ′*a set* ⇒ *bool*
card   :: ′*a set* ⇒ *nat*
Finite_Set.fold :: (′*a* ⇒ ′*b* ⇒ ′*b*) ⇒ ′*b* ⇒ ′*a set* ⇒ ′*b*

Lattices_Big

Min   :: ′*a set* ⇒ ′*a*
Max   :: ′*a set* ⇒ ′*a*
arg_min   :: (′*a* ⇒ ′*b*) ⇒ (′*a* ⇒ *bool*) ⇒ ′*a*
is_arg_min :: (′*a* ⇒ ′*b*) ⇒ (′*a* ⇒ *bool*) ⇒ ′*a* ⇒ *bool*
arg_max   :: (′*a* ⇒ ′*b*) ⇒ (′*a* ⇒ *bool*) ⇒ ′*a*
is_arg_max :: (′*a* ⇒ ′*b*) ⇒ (′*a* ⇒ *bool*) ⇒ ′*a* ⇒ *bool*

Syntax

ARG_MIN f x. P   ≡   *arg_min f* (λ*x. P*)
ARG_MAX f x. P ≡   *arg_max f* (λ*x. P*)

Groups_Big

sum :: (′*a* ⇒ ′*b*) ⇒ ′*a set* ⇒ ′*b*
prod :: (′*a* ⇒ ′*b*) ⇒ ′*a set* ⇒ ′*b*

Syntax

$$\sum A \qquad \equiv \quad sum\ (\lambda x.\ x)\ A \quad (\texttt{SUM})$$
$$\sum x{\in}A.\ t \quad \equiv \quad sum\ (\lambda x.\ t)\ A$$
$$\sum x|P.\ t \quad \equiv \quad \sum x\ |\ P.\ t$$

Similarly for $\prod$ instead of $\sum$ (`PROD`)

Wellfounded

wf $::\ ('a \times 'a)\ set \Rightarrow bool$
$Wellfounded.acc$ $::\ ('a \times 'a)\ set \Rightarrow 'a\ set$
$measure$ $::\ ('a \Rightarrow nat) \Rightarrow ('a \times 'a)\ set$
$(<*lex*>)$ $::\ ('a \times 'a)\ set \Rightarrow ('b \times 'b)\ set \Rightarrow (('a \times 'b) \times 'a \times 'b)\ set$
$(<*mlex*>)$ $::\ ('a \Rightarrow nat) \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
$less\_than$ $::\ (nat \times nat)\ set$
$pred\_nat$ $::\ (nat \times nat)\ set$

Set_Interval

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

Syntax

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

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

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

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

Power

$$(\hat{\ }) :: {'}a \Rightarrow nat \Rightarrow {'}a$$

Option

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

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

List

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

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

$$
\begin{array}{lll}
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 \\
sum\_list & :: & 'a\ list \Rightarrow\ 'a \\
prod\_list & :: & '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 \\
(!) & :: & 'a\ list \Rightarrow\ nat \Rightarrow\ 'a \\
nths & :: & 'a\ list \Rightarrow\ nat\ set \Rightarrow\ 'a\ list \\
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 \\
\end{array}
$$

$$
\begin{array}{lll}
sorted & :: & 'a\ list \Rightarrow bool \\
sorted\_wrt & :: & ('a \Rightarrow 'a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow bool \\
splice & :: & 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list \\
take & :: & nat \Rightarrow 'a\ list \Rightarrow 'a\ list \\
takeWhile & :: & ('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list \\
tl & :: & 'a\ list \Rightarrow 'a\ list \\
upt & :: & nat \Rightarrow nat \Rightarrow nat\ list \\
upto & :: & int \Rightarrow int \Rightarrow int\ list \\
zip & :: & 'a\ list \Rightarrow 'b\ list \Rightarrow ('a \times 'b)\ list \\
\end{array}
$$

Syntax

$$
\begin{array}{lll}
[x_1,\ldots,x_n] & \equiv & x_1\ \#\ \ldots\ \#\ x_n\ \#\ [] \\
[m..<n] & \equiv & upt\ m\ n \\
[i..j] & \equiv & upto\ i\ j \\
xs[n := x] & \equiv & list\_update\ xs\ n\ x \\
\sum x\leftarrow xs.\ e & \equiv & listsum\ (map\ (\lambda x.\ e)\ xs) \\
\end{array}
$$

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: $[e.\ q_1,\ \ldots,\ q_n]$ 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.

$$
\begin{array}{lll}
Map.empty & :: & 'a \Rightarrow 'b\ option \\
(++) & :: & ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow 'a \Rightarrow 'b\ option \\
(\circ_m) & :: & ('a \Rightarrow 'b\ option) \Rightarrow ('c \Rightarrow 'a\ option) \Rightarrow 'c \Rightarrow 'b\ option \\
(|\grave{}) & :: & ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b\ option \\
dom & :: & ('a \Rightarrow 'b\ option) \Rightarrow 'a\ set \\
ran & :: & ('a \Rightarrow 'b\ option) \Rightarrow 'b\ set \\
(\subseteq_m) & :: & ('a \Rightarrow 'b\ option) \Rightarrow ('a \Rightarrow 'b\ option) \Rightarrow bool \\
map\_of & :: & ('a \times 'b)\ list \Rightarrow 'a \Rightarrow 'b\ option \\
map\_upds & :: & ('a \Rightarrow 'b\ option) \Rightarrow 'a\ list \Rightarrow 'b\ list \Rightarrow 'a \Rightarrow 'b\ option \\
\end{array}
$$

Syntax

$$\begin{array}{lcl}
Map.empty & \equiv & \lambda\_.\ None \\
m(x \mapsto y) & \equiv & m(x{:=}Some\ y) \\
m(x_1{\mapsto}y_1,\ldots,x_n{\mapsto}y_n) & \equiv & m(x_1{\mapsto}y_1)\ldots(x_n{\mapsto}y_n) \\
[x_1{\mapsto}y_1,\ldots,x_n{\mapsto}y_n] & \equiv & Map.empty(x_1{\mapsto}y_1,\ldots,x_n{\mapsto}y_n) \\
m(xs\ [\mapsto]\ ys) & \equiv & map\_upds\ m\ xs\ ys
\end{array}$$

Infix operators in Main

	Operator	precedence	associativity
Meta-logic	$\Longrightarrow$	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
	$\sim\sim$	80	right
Numbers	$+, -$	65	left
	$*, /$	70	left
	$div,\ mod$	70	left
	$\widehat{}$	80	right
	dvd	50	
Lists	$\#, @$	65	right
	$!$	100	left