

What's in Main

Tobias Nipkow

May 25, 2015

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

$x \neq y$ $\equiv \neg (x = y)$ ($\sim =$)
 $P \longleftrightarrow Q$ $\equiv P = Q$
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

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

Syntax

$x \geq y \quad \quad \equiv \quad y \leq x \quad \quad \quad (>=)$
 $x > y \quad \quad \equiv \quad y < x$
 $\forall x \leq y. P \quad \equiv \quad \forall x. x \leq y \longrightarrow P$
 $\exists x \leq y. P \quad \equiv \quad \exists x. x \leq y \wedge P$
 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*).

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

Syntax

Available by loading theory *Lattice_Syntax* in directory *Library*.

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

Set

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

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

Syntax

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

Fun

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

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 .

$() :: unit$

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

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

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

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

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

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

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.

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

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

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

$f \text{ respects } r \equiv \text{congruent } r \ f$
 $f \text{ respects2 } r \equiv \text{congruent2 } r \ r \ f$

Transitive_Closure

$rtrancl :: ('a \times 'a) set \Rightarrow ('a \times 'a) set$
 $trancl :: ('a \times 'a) set \Rightarrow ('a \times 'a) set$
 $reflcl :: ('a \times 'a) set \Rightarrow ('a \times 'a) set$
 $acyclic :: ('a \times 'a) set \Rightarrow bool$
 $op \hat{\hat{}} :: ('a \times 'a) set \Rightarrow nat \Rightarrow ('a \times 'a) set$

Syntax

$r^* \equiv rtrancl\ r \quad (\hat{*})$
 $r^+ \equiv trancl\ r \quad (\hat{+})$
 $r^- \equiv reflcl\ r \quad (\hat{=})$

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:

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

Syntax

$|x| \equiv abs\ x$

Nat

datatype $nat = 0 \mid Suc\ nat$

$op + \quad op - \quad op * \quad op \hat{\quad} \quad op\ div \quad op\ mod \quad op\ dvd$
 $op \leq \quad op < \quad min \quad max \quad Min \quad Max$

$of_nat :: nat \Rightarrow 'a$
 $op \wedge \wedge :: ('a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a \Rightarrow 'a$

Int

Type *int*

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

Syntax

$int \equiv of_nat$

Finite_Set

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

Syntax

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

Wellfounded

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

Set _Interval

lessThan :: 'a ⇒ 'a set
atMost :: 'a ⇒ 'a set
greaterThan :: 'a ⇒ 'a set
atLeast :: 'a ⇒ 'a set
greaterThanLessThan :: 'a ⇒ 'a ⇒ 'a set
atLeastLessThan :: 'a ⇒ 'a ⇒ 'a set
greaterThanAtMost :: 'a ⇒ 'a ⇒ 'a set
atLeastAtMost :: 'a ⇒ 'a ⇒ 'a set

Syntax

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

Similarly for \bigcap instead of \bigcup

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

Similarly for \prod instead of \sum

Power

op ^ :: 'a ⇒ nat ⇒ 'a

Option

datatype 'a option = None | Some 'a

the :: 'a option ⇒ 'a
map_option :: ('a ⇒ 'b) ⇒ 'a option ⇒ 'b option
set_option :: 'a option ⇒ 'a set
Option.bind :: 'a option ⇒ ('a ⇒ 'b option) ⇒ 'b option

List

datatype 'a list = [] | op # 'a ('a 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
List.find :: ('a ⇒ bool) ⇒ 'a list ⇒ 'a option
fold :: ('a ⇒ 'b ⇒ 'b) ⇒ 'a list ⇒ 'b ⇒ 'b
foldr :: ('a ⇒ 'b ⇒ 'b) ⇒ 'a list ⇒ 'b ⇒ 'b
foldl :: ('a ⇒ 'b ⇒ 'a) ⇒ 'a ⇒ 'b list ⇒ 'a
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 × 'b) set ⇒ ('a list × 'b 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

$splice$ $:: 'a\ list \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $sublist$ $:: 'a\ list \Rightarrow nat\ set \Rightarrow 'a\ list$
 $take$ $:: nat \Rightarrow 'a\ list \Rightarrow 'a\ list$
 $takeWhile$ $:: ('a \Rightarrow bool) \Rightarrow 'a\ list \Rightarrow 'a\ list$
 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
	$\wedge\wedge$	80	right
Numbers	$+, -$	65	left
	$*, /$	70	left
	div, mod	70	left
	\wedge	80	right
	dvd	50	
Lists	$\#, @$	65	right
	$!$	100	left