

Alphabetic List of Functions

Standard Dictionary for Path Semantics

by Sven Nilsen, 2017

A

$\text{abs} := \lambda(a) = \text{if } a < 0 \{ -a \} \text{ else } \{ a \}$

$\text{add}_A := \lambda(a : A, b : A) = a + b$

When written `a : [+ b] c` it means `a` plus `b` is equal to `c` .

$\text{add}_{\mathbb{C}} : \text{complex} \times \text{complex} \rightarrow \text{complex}$

$\text{add}_{\mathbb{N}} : \text{nat} \times \text{nat} \rightarrow \text{nat}$

$\text{add}_{\mathbb{Q}} : \text{rational} \times \text{rational} \rightarrow \text{rational}$

$\text{add}_{\mathbb{R}} : \text{real} \times \text{real} \rightarrow \text{real}$

$\text{add}_{\mathbb{Z}} : \text{int} \times \text{int} \rightarrow \text{int}$

$\text{and} := \lambda(a : \text{bool}, b : \text{bool}) = a \wedge b$

In C-like programming languages this is equivalent to `a \&\& b` .

When written `a : (\wedge b)` it means both `a` and `b` are `true` , or neither are.

$\text{acos} : \text{real} \rightarrow \text{real}$

The trigonometric inverse cosine function.

$\text{asin} : \text{real} \rightarrow \text{real}$

The trigonometric inverse sinus function.

$\text{asym} : \lambda(m : \text{matrix} \wedge [\text{dim}] [\text{eq}] \text{true}) = \forall i, j \{ m[i][j] == -m[j][i] \}$

$\text{atan} : \text{real} \rightarrow \text{real}$

The trigonometric inverse tangent function.

$\text{atan}_2 : \text{real} \times \text{real} \rightarrow \text{real}$

The trigonometric inverse tangent function with 2 arguments.

Returns the angle of a vector in radians `atan2(y, x)` .

C

`cardinality : set → nat` |
Returns the cardinality of a set.
The cardinality of infinite sets can be of higher order infinity (\aleph^N).
`cardinality(nat) = \aleph^0`
`cardinality(real) = \aleph^1`

`ceilA : real → A`
Rounds up real number to nearest integer value.
`ceilℂ : real → complex`
`ceilℕ : real → nat`
`ceilℚ : real → rational`
`ceilℝ : real → real`
`ceilℤ : real → int`

`concat : list × list → list`
Appends the second list to the first list, returning a new list.

`constructa := \() = a`
Constructs an object.

`cos : real → real`
The trigonometric cosine function.

`countA := \ (f : A → bool, t : bool) = $\sum x : \forall f \{ \text{if } f(x) == t \{ 1 \} \text{ else } \{ 0 \} \}$`
Enumerates all values that satisfies the trivial path of a function and has a truth value `t`.
One can write that `count : \ bool × bool → nat`.
`|f| <=> count(f, true)`
`|¬f| <=> count(f, false)`

`cross := \ (a : vector ∧ [vec_dim] 3, b : vector ∧ [vec_dim] 3) =`
`(y(a) · z(b) – z(a) · y(b), z(a) · x(b) – x(a) · z(b), x(a) · y(b) – y(a) · x(b))`
Returns the cross product between two vectors.
This is defined only for vectors in 3 dimensions.
When written `a : [× b] c` it means the cross product of `a` and `b` is `c`.

D

`dec := \ (a) = a – 1`

`dedup : list → list`
Removes duplicates from list, returning a new list.

`det : matrix → real`
Returns the determinant of a matrix.

`diag := \ (m : matrix ∧ [dim] [eq] true) = $\forall i, j \{ \text{if } i == j \{ \text{continue} \} \text{ else } \{ m[i][j] == 0 \} \}$`
Returns `true` if matrix is a diagonal matrix.

`dim : matrix → (nat, nat)`
Returns the dimensions of the matrix `(rows, columns)`.

`div := \ (a : A, b : A) = a / b`
When written `a : [/ b] c` it means `a` divided by `b` is equal to `c`.

`div_exactℕ := \ (a : nat ∧ [% b] 0, b : nat ∧ (¬= 0)) → nat { a / b }`

...D (continued)

$\text{dot} := \lambda(a : \text{vector} \wedge [\text{vec_dim}] \ n, b : \text{vector} \wedge [\text{vec_dim}] \ n) = \sum i \{ a[i] \cdot b[i] \}$

Returns the dot product between two vectors.

When written `a : [· b] c` it means the dot product of `a` and `b` equals `c`.

$\text{dup} : \lambda(a) = (a, a)$

$\text{dup}_n : \lambda(a) = (a, a, \dots)$

E

$\text{each_connected} := \lambda(m : \text{matrix}) = \forall i \{ \sum j \{ m[i][j] \} > 0 \}$

Used to reason about molecule structures where each atom must be connected.

$\text{el} : \text{nat} \times \text{nat} \times \text{matrix} \rightarrow \text{any}$

Returns element of matrix at row and column index.

Notice that this is row major, such that `y` becomes before `x`.

$\text{even} := \lambda(a : \text{nat}) = (a \% 2) == 0$

$\text{even} \iff \text{linear}(0, 2)$

Returns `true` if a number is even.

$\text{eq} := \lambda(a, b) = a == b$

$\text{exc} := \lambda(a : \text{bool}, b : \text{bool}) = a \wedge \neg b$

In C-like programming languages this is equivalent to `a && !b`.

$\text{exclude} : \text{set} \times \text{set} \rightarrow \text{set}$

Excludes elements from the second set from the first set.

$\text{exp}_A := \lambda(a : A) = e^a$

Returns the natural exponent of a number.

$\text{exp}_{\mathbb{R}} : \text{real} \rightarrow \text{real}$

$\text{exp}_{\mathbb{C}} := \lambda(a : \text{complex}) = \cos(\text{re}(a)) + i \cdot \sin(\text{im}(a))$

F

$\text{factorize} : \text{nat} \rightarrow \text{list}$

Returns a sorted list of prime factors of natural number.

$\text{factorial} := \lambda(x : \text{nat}) = \prod i [0, x+1) \{ i \}$

$\text{false}_N := \lambda(_, _, \dots) = \text{false}$

A function that always returns `false`.

$\text{false}_0 := \lambda() = \text{false}$

$\text{false}_1 := \lambda(_) = \text{false}$

$\text{floor}_A : \text{real} \rightarrow A$

Rounds down real number to nearest integer value.

$\text{floor}_{\mathbb{C}} : \text{real} \rightarrow \text{complex}$

$\text{floor}_{\mathbb{N}} : \text{real} \rightarrow \text{nat}$

$\text{floor}_{\mathbb{Q}} : \text{real} \rightarrow \text{rational}$

$\text{floor}_{\mathbb{R}} : \text{real} \rightarrow \text{real}$

$\text{floor}_{\mathbb{Z}} : \text{real} \rightarrow \text{int}$

$\text{fract} := \lambda(a : \text{real}) = a \% 1$

...F (continued)

$\text{fst} := \lambda(a, b) = a$

Returns the first element in a tuple.

G

$\text{ge} := \lambda(a, b) = a \geq b$

When written $\lambda a : (>= b)$ it means λa is greater than or equal to b .

$\text{gt} := \lambda(a, b) = a > b$

When written $\lambda a : (> b)$ it means λa is greater than b .

I

$\text{id}_A := \lambda(x : A) = x$

$\text{if} := A \times A \rightarrow (\text{bool} \rightarrow A)$

A higher order function used to construct boolean functions.

$\text{inc} := \lambda(a) = a + 1$

$\text{intersect} : \text{set} \times \text{set} \rightarrow \text{set}$

Returns a new set containing elements belonging to both sets.

$\text{inv} : \lambda(a) = 1 / a$

$\text{invert} \Leftrightarrow \text{mat_inv}$

$\text{im} : \text{complex} \rightarrow \text{real}$

Returns the imaginary part of a complex number.

J

$\text{join} \Leftrightarrow \text{add}$

Used to reason about circuit diagrams.

$\text{len} : \text{list} \rightarrow \text{nat}$

L

$\text{le} := \lambda(a, b) = a \leq b$

When written $\lambda a : (<= b)$ it means λa is less than or equal to b .

$\text{linear} := \lambda(a : \text{nat}, b : \text{nat} \wedge (> 0)) = \lambda(x) = \text{if } x < a \{ \text{false} \} \text{ else } \{ ((x - a) \% b) == 0 \}$

Returns true if a natural number is in a linear sequence of natural numbers.

$\text{ln} : \text{real} \rightarrow \text{real}$

Returns the natural logarithm of a number.

$\text{lt} := \lambda(a, b) = a < b$

When written $\lambda a : (< b)$ it means λa is less than b .

M

$\text{mat_add} : \text{matrix} \times \text{matrix} \rightarrow \text{matrix}$

Matrix addition.

$\text{mat_id} : \text{nat} \rightarrow \text{matrix}$

Constructs an identity matrix.

$\text{mat_inv} : \text{matrix} \rightarrow \text{matrix}$

Returns the inverse matrix.

$\text{mat_mul} : \text{matrix} \times \text{matrix} \rightarrow \text{matrix}$

Matrix multiplication, row major.

$\text{max_bounds} := \lambda (n : \text{nat}) = \lambda (m : \text{matrix}) = \forall i \{ \sum j \{ m[i][j] \} \leq n \}$

Used to reason about molecule structures where each atom has a limited number of bounds.

$\text{max} := \lambda (a : \text{list}) = \max i \{ a[i] \}$

$\text{max}_2 := \lambda (a, b) = \text{if } a > b \{ a \} \text{ else } \{ b \}$

$\text{min} := \lambda (a : \text{list}) = \min i \{ a[i] \}$

$\text{min}_2 := \lambda (a, b) = \text{if } a < b \{ a \} \text{ else } \{ b \}$

$\text{mul}_A := \lambda (a : A, b : A) = a \cdot b$

When written $a : [b] c$ it means a multiplied with b is equal to c .

$\text{mul}_{\mathbb{C}} : \text{complex} \times \text{complex} \rightarrow \text{complex}$

$\text{mul}_{\mathbb{N}} : \text{nat} \times \text{nat} \rightarrow \text{nat}$

$\text{mul}_{\mathbb{Q}} : \text{rational} \times \text{rational} \rightarrow \text{rational}$

$\text{mul}_{\mathbb{R}} : \text{real} \times \text{real} \rightarrow \text{real}$

$\text{mul}_{\mathbb{Z}} : \text{int} \times \text{int} \rightarrow \text{int}$

N

$\text{nand} := \lambda (a : \text{bool}, b : \text{bool}) = \text{not}(\text{and}(a, b))$

$\text{neg}_A := \lambda (a : A) = -a$

$\text{neg}_{\mathbb{C}} : \text{complex} \rightarrow \text{complex}$

$\text{neg}_{\mathbb{Q}} : \text{rational} \rightarrow \text{rational}$

$\text{neg}_{\mathbb{R}} : \text{real} \rightarrow \text{real}$

$\text{neg}_{\mathbb{Z}} : \text{int} \rightarrow \text{int}$

$\text{neq} \iff \text{xor}$

$\text{nexc} := \lambda (a : \text{bool}, b : \text{bool}) = \text{not}(\text{exc}(a, b))$

$\text{non_diag} := \lambda (m : \text{matrix} \wedge [\text{dim}] [\text{eq}] \text{true}) = \forall i \{ m[i][i] == 0 \}$

Returns `true` when all elements on the diagonal are zero.

$\text{nor} := \lambda (a : \text{bool}, b : \text{bool}) = \text{not}(\text{or}(a, b))$

$\text{not} := \lambda (a : \text{bool}) = \neg a$

In C-like programming languages this is written `!a`.

$\text{nrex} := \lambda (a : \text{bool}, b : \text{bool}) = \text{not}(\text{rex}(a, b))$

$\text{nxor} \iff \text{eq}$

O

$\text{odd} := \lambda(a : \text{nat}) = (a \% 2) == 1$

$\text{odd} \iff \text{linear}(1, 2)$

Returns `true` if a number is odd.

$\text{or} := \lambda(a : \text{bool}, b : \text{bool}) = a \vee b$

In C-like programming languages this is equivalent to `a || b`.

When written `a : (v b)` it means `a` or `b` are `true`.

P

$\text{pair} := \lambda(a) = \lambda(b) = (a, b)$

$\text{prime} : \text{nat} \rightarrow \text{bool}$

Returns `true` if natural number is a prime number.

$\text{pop} : \text{list} \rightarrow (\text{list}, \text{any})$

Removes an item from a list, returning a new list and the item removed.

$\text{pow}_A : A \times A \rightarrow A$

Returns the power of a number.

When written `a : [^ab] c` it means `a` powered by `b` is equal to `c`.

$\text{pow}_{\mathbb{C}} : \text{complex} \times \text{complex} \rightarrow \text{complex}$

$\text{pow}_{\mathbb{N}} : \text{nat} \times \text{nat} \rightarrow \text{nat}$

$\text{pow}_{\mathbb{Q}} : \text{rational} \times \text{rational} \rightarrow \text{rational}$

$\text{pow}_{\mathbb{R}} : \text{real} \times \text{real} \rightarrow \text{real}$

$\text{pow}_{\mathbb{Z}} : \text{int} \times \text{int} \rightarrow \text{int}$

$\text{prob} := \lambda(x : \text{real}) = x \geq 0 \wedge x \leq 1$

$\text{probl} := \lambda(x : \text{real}) = x \geq 0 \wedge x < 1$

$\text{probm} := \lambda(x : \text{real}) = x > 0 \wedge x < 1$

$\text{probr} := \lambda(x : \text{real}) = x > 0 \wedge x \leq 1$

$\text{probx} := \lambda(k : \text{real} \wedge [\text{prob}] \text{true}) = \lambda(x : \text{bool}) = \text{if } x \{ k \} \text{ else } \{ 1 - k \}$

$\text{prod} := \lambda(a : \text{list}) = \prod i \{ a[i] \}$

$\text{push} : \text{list} \times \text{any} \rightarrow \text{list}$

Pushes an item to the end of a list

R

`random : () → real`

Often not considered a function in the normal sense but with a hidden argument of an unknown natural number.

`random : nat → real`

`re := complex → real`

Returns the real part of a complex number.

`rem := \ (a, b) = a % b`

Also called “modulus binary operator”.

This is the rest value you get after integer division.

When written ``a : [% b] c`` it means ``a` modulus `b`` is equal to ``c``.

`rexc := \ (a : bool, b : bool) = b ∧ ¬a`

In C-like programming languages this is equivalent to ``b && !a``.

`roundA : real → A`

Rounds real number to nearest integer value.

`roundC : real → complex`

`roundN : real → nat`

`roundQ : real → rational`

`roundR : real → real`

`roundZ : real → int`

S

`sc := \ (sc, f) = \ (n) = f (sc (sc, f), n)`

`sc (sc) : ((A → B) × A → B) → (A → B)`

A convenient fixed point combinator that allows anonymous recursive calls, using the first parameter as a ``self`` function.

Here is an example of generating the numbers in the Fibonacci sequence:

`fib := \ (self : nat → nat, n : nat) = if n == 0 { 0 } else if n == 1 { 1 } else { self (n-1) + self (n-2) }`

`call_fib := sc (sc, fib)`

`call_fib (20)` // 6765

`sequence := \ (a : nat, b : nat ∧ (> 0)) = \ (x) = a + b · x`

Maps from natural numbers to a linear sequence of natural numbers.

`sin : real → real`

The trigonometric sinus function.

`snd := \ ((a, b)) = b`

Returns the second element of a tuple.

`sortf := list → list`

Sorts a list by function ``f``.

When ``f`` is not specified, default ascending order is used.

`sortedf := list → bool`

Returns ``true`` if list is sorted by function ``f``.

When ``f`` is not specified, default ascending order is used.

...

...S (continued)

$\text{split} := \lambda(s : \text{real}) = \lambda(x : \text{real}) = (s \cdot x, (1 - s) \cdot x)$

Used to reason about circuit diagrams.

$\text{square_len} := \lambda(a : \text{vector}) = \sum i \{ a[i] \cdot a[i] \}$

$\text{sqrt}_A : A \rightarrow A$

Takes the square root of a number.

$\text{sqrt}_{\mathbb{N}} : \text{nat} \rightarrow \text{nat}$

Defined only for square numbers.

$\text{sqrt}_{\mathbb{R}} : \text{real} \rightarrow \text{real}$

Defined only for non-negative numbers.

$\text{sqrt}_{\mathbb{C}} : \text{complex} \rightarrow \text{complex}$

Automatic conversion from real to complex number.

$\text{strict_subset} : \text{set} \times \text{set} \rightarrow \text{bool}$

Returns `true` if all elements of the first set belongs to the second set, and the two sets do not have equal cardinality.

When written `a : (\subset b)` it means `a` is a strict subset of `b`.

$\text{sub}_A := \lambda(a : A, b : A) = a - b$

When written `a : [- b] c` it means `a` minus `b` is equal to `c`.

$\text{sub}_{\mathbb{C}} : \text{complex} \times \text{complex} \rightarrow \text{complex}$

$\text{sub}_{\mathbb{N}} : \lambda(a : \text{nat} \wedge (>= b), b : \text{nat}) \rightarrow \text{nat} = \{ a - b \}$

$\text{sub}_{\mathbb{Q}} : \text{rational} \times \text{rational} \rightarrow \text{rational}$

$\text{sub}_{\mathbb{R}} : \text{real} \times \text{real} \rightarrow \text{real}$

$\text{sub}_{\mathbb{Z}} : \text{int} \times \text{int} \rightarrow \text{int}$

$\text{subset} : \text{set} \times \text{set} \rightarrow \text{bool}$

Returns `true` if all elements of the first set belongs to the second set.

When written `a : (\subseteq b)` it means `a` is a subset of `b`.

$\text{sum} := \lambda(a : \text{list}) = \sum i \{ a[i] \}$

$\text{swap} := \lambda((a, b)) = (b, a)$

$\text{sym} := \lambda(m : \text{matrix} \wedge [\text{dim}] [\text{eq}] \text{true}) = \forall i, j \{ m[i][j] == m[j][i] \}$

T

$\text{tan} : \text{real} \rightarrow \text{real}$

The trigonometric tangent function.

$\text{trace} := \lambda(m : \text{matrix}) = \sum i, i \{ m[i][i] \}$

$\text{transform} : \text{matrix} \times \text{vector} \rightarrow \text{vector}$

Transforms a vector through a matrix

$\text{transpose} : \text{matrix} \rightarrow \text{matrix}$

Returns the transposed matrix, where rows are swapped with columns.

$\text{true}_{\mathbb{N}} := \lambda(_, _, \dots) = \text{true}$

A function that always returns `true`.

$\text{true}_0 := \lambda() = \text{true}$

$\text{false}_1 := \lambda() = \text{false}$

U

$\text{union} : \text{set} \times \text{set} \rightarrow \text{set}$

Returns the union of two sets.

When written $`a : [\cup b] c`$ it means $`a`$ union $`b`$ results in $`c`$.

$\text{unit} : \text{any} \rightarrow ()$

Used to erase information about an input argument.

V

$\text{vec_dim} : \text{vector} \rightarrow \text{nat}$

Returns the number of dimensions of a vector.

X

$x : \text{vector} \rightarrow \text{real}$

Returns the x-component of a vector.

$\text{xor} := \lambda(a : \text{bool}, b : \text{bool}) = a \wedge \neg b \vee \neg a \wedge b$

In C-like programming languages this is equivalent to “ $a \ \&\& \ !b \ || \ !a \ \&\& \ b$ ”.

When written $`a : (\underline{\vee} b)`$ it means either $`a`$ or $`b`$ is $`true`$, but not both.

Y

$y : \text{vector} \rightarrow \text{real}$

Returns the y-component of a vector.

Z

$z : \text{vector} \rightarrow \text{real}$

Returns the z-component of a vector.

W

$w : \text{vector} \rightarrow \text{real}$

Returns the w-component of a vector.