# Alphabetic List of Functions

## Standard Dictionary for Path Semantics

by Sven Nilsen, 2017

## A

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

$add_A$ := \(a : A, b : A) = a + b

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

$add_{\mathbb{C}}$ : complex × complex → complex

$add_{\mathbb{N}}$ : nat × nat → nat

$add_{\mathbb{Q}}$ : rational × rational → rational

$add_{\mathbb{R}}$ : real × real → real

$add_{\mathbb{Z}}$ : int × int → int

and := \(a : bool, b : bool) = a ∧ b

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

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

acos : real → real

*The trigonometric inverse cosine function.*

asin : real → real

*The trigonometric inverse sinus function.*

asym : \(m : matrix ∧ [dim] [eq] true) = ∀ i, j { m[i][j] == -m[j][i] }

atan : real → real

*The trigonometric inverse tangent function.*

$atan_2$ : real × real → 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 ($\mathfrak{N}^N$).*
    cardinality(nat) = $\mathfrak{N}^0$
    cardinality(real) = $\mathfrak{N}^1$
$ceil_A$ : real → A
    *Rounds up real number to nearest integer value.*
    $ceil_{\mathbb{C}}$ : real → complex
    $ceil_{\mathbb{N}}$ : real → nat
    $ceil_{\mathbb{Q}}$ : real → rational
    $ceil_{\mathbb{R}}$ : real → real
    $ceil_{\mathbb{Z}}$ : real → int
concat : list × list → list
    *Appends the second list to the first list, returning a new list.*
$construct_a$ := \() = a
    *Constructs an object.*
cos : real → real
    *The trigonometric cosine function.*
$count_A$ := \(f : A → bool, t : bool) = ∑ x : ∀f { if f(x) == t { 1 } 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) = ∀ i, j { if i == j { continue } 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`.*
$div\_exact_{\mathbb{N}}$ := \(a : nat ∧ [% b] 0, b : nat ∧ (¬= 0)) → nat { a / b }

# ...D (continued)

dot := \(a : vector ∧ [vec_dim] n, b : vector ∧ [vec_dim] n) = ∑ i { a[i] · 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`.*
dup : \(a) = (a, a)
$dup_n$ : \(a) = (a, a, ...)

# E

each_connected := \(m : matrix) = ∀ i { ∑ j { m[i][j] } > 0 }
    *Used to reason about molecule structures where each atom must be connected.*
el : nat × nat × matrix → any
    *Returns element of matrix at row and column index.*
    *Notice that this is row major, such that `y` becomes before `x`.*
even := \(a : nat) = (a % 2) == 0
    even <=> linear(0, 2)
    *Returns `true` if a number is even.*
eq := \(a, b) = a == b
exc := \(a : bool, b : bool) = a ∧ ¬b
    *In C-like programming languages this is equivalent to `a && !b`.*
exclude : set × set → set
    *Excludes elements from the second set from the first set.*
$exp_A$ := \(a : A) = $e^a$
    *Returns the natural exponent of a number.*
    $exp_{\mathbb{R}}$ : real → real
    $exp_{\mathbb{C}}$ := \(a : complex) = cos(re(a)) + **i** · sin(im(a))

# F

factorize : nat → list
    Returns a sorted list of prime factors of natural number.
factorial := \(x : nat) = ∏ i [0, x+1) { i }
$false_N$ := \(_, _, …) = false
    *A function that always returns `false`.*
    $false_0$ := \() = false
    $false_1$ := \(_) = false
$floor_A$ : real → A
    *Rounds down real number to nearest integer value.*
    $floor_{\mathbb{C}}$ : real → complex
    $floor_{\mathbb{N}}$ : real → nat
    $floor_{\mathbb{Q}}$ : real → rational
    $floor_{\mathbb{R}}$ : real → real
    $floor_{\mathbb{Z}}$ : real → int
fract := \(a : real) = a % 1
...

# ...F (continued)

fst := \((a, b)) = a
    *Returns the first element in a tuple.*

# G

ge := \(a, b) = a >= b
    *When written `a : (>= b)` it means `a` is greater than or equal to `b`.*
gt := \(a, b) = a > b
    *When written `a : (> b)` it means `a` is greater than `b`.*

# I

$id_A$ := \(x : A) = x
if := A × A → (bool → A)
    *A higher order function used to construct boolean functions.*
inc := \(a) = a + 1
intersect : set × set → set
    *Returns a new set containing elements belonging to both sets.*
inv : \(a) = 1 / a
invert <=> mat_inv
im : complex → real
    *Returns the imaginary part of a complex number.*

# J

join <=> add
    *Used to reason about circuit diagrams.*
len : list → nat

# L

le := \(a, b) = a <= b
    *When written `a : (<= b)` it means `a` is less than or equal to `b`.*
$line_A$ := \(a : A, b : A) = \(t : real) = t * (b – a) + a
    *Can be used with any type that supports these operations, often higher dimensions*
linear := \(a : nat, b : nat ∧ (> 0)) = \(x) = if x < a { false } else { ((x – a) % b) == 0 }
    *Returns `true` if a natural number is in a linear sequence of natural numbers.*
ln : real → real
    *Returns the natural logarithm of a number.*
lt := \(a, b) = a < b
    *When written `a : (< b)` it means `a` is less than `b`.*

# M

mat_add : matrix × matrix → matrix
    *Matrix addition.*
mat_id : nat → matrix
    *Constructs an identity matrix.*
mat_inv : matrix → matrix
    *Returns the inverse matrix.*
mat_mul : matrix × matrix → matrix
    *Matrix multiplication, row major.*
max_bounds := \(n : nat) = \(m : matrix) = ∀ i { ∑ j { m[i][j] } <= n }
    *Used to reason about molecule structures where each atom has a limited number of bounds.*
max := \(a : list) = max i { a[i] }
$max_2$ := \(a, b) = if a > b { a } else { b }
min := \(a : list) = min i { a[i] }
$min_2$ := \(a, b) = if a < b { a } else { b }
$mul_A$ := \(a : A, b : A) = a · b
    *When written `a : [· b] c` it means `a` multiplied with `b` is equal to `c`.*
    $mul_ℂ$ : complex × complex → complex
    $mul_ℕ$ : nat × nat → nat
    $mul_ℚ$ : rational × rational → rational
    $mul_ℝ$ : real × real → real
    $mul_ℤ$ : int × int → int

# N

nand := \(a : bool, b : bool) = not(and(a, b))
$neg_A$ := \(a : A) = -a
    $neg_ℂ$ : complex → complex
    $neg_ℚ$ : rational → rational
    $neg_ℝ$ : real → real
    $neg_ℤ$ : int → int
neq <=> xor
nexc := \(a : bool, b : bool) = not(exc(a, b))
non_diag := \(m : matrix ∧ [dim] [eq] true) = ∀ i { m[i][i] == 0 }
    *Returns `true` when all elements on the diagonal are zero.*
nor := \(a : bool, b : bool) = not(or(a, b))
not := \(a : bool) = ¬a
    *In C-like programming languages this is written `!a`.*
nrexc := \(a : bool, b : bool) = not(rexc(a, b))
nxor <=> eq

# O

odd := \(a : nat) = (a % 2) == 1
    odd <=> linear(1, 2)
    *Returns `true` if a number is odd.*
or := \(a : bool, b : bool) = a ∨ b
    *In C-like programming languages this is equivalent to `a || b`.*
    *When written `a : (∨ b)` it means `a` or `b` are `true`.*

# P

pair := \(a) = \(b) = (a, b)
prime : nat → bool
    *Returns `true` if natural number is a prime number.*
pop : list → (list, any)
    *Removes an item from a list, returning a new list and the item removed.*
$pow_A$ : A × A → A
    *Returns the power of a number.*
    *When written `a : [^ b] c` it means `a` powered by `b` is equal to `c`.*
    $pow_{\mathbb{C}}$ : complex × complex → complex
    $pow_{\mathbb{N}}$ : nat × nat → nat
    $pow_{\mathbb{Q}}$ : rational × rational → rational
    $pow_{\mathbb{R}}$ : real × real → real
    $pow_{\mathbb{Z}}$ : int × int → int
prob := \(x : real) = x >= 0 ∧ x <= 1
probl := \(x : real) = x >= 0 ∧ x < 1
probm := \(x : real) = x > 0 ∧ x < 1
probr := \(x : real) = x > 0 ∧ x <= 1
probx := \(k : real ∧ [prob] true) = \(x : bool) = if x { k } else { 1 – k }
prod := \(a : list) = ∏ i { a[i] }
push : list × any → 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`.*
$round_A$ : real → A
*Rounds real number to nearest integer value.*
$round_{\mathbb{C}}$ : real → complex
$round_{\mathbb{N}}$ : real → nat
$round_{\mathbb{Q}}$ : real → rational
$round_{\mathbb{R}}$ : real → real
$round_{\mathbb{Z}}$ : 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.*
$sign_A$ := \(a : A) = if a > 0 {1} else if a < 0 {-1} else {0}
$sign_{\mathbb{R}}$ : real → real
$sign_{\mathbb{Z}}$ : int → int
sin : real → real
*The trigonometric sinus function.*
snd := \((a, b)) = b
*Returns the second element of a tuple.*
$sort_f$ := list → list
*Sorts a list by function `f`.*
*When `f` is not specified, default ascending order is used.*
...

# ...S (continued)

$sorted_f$ := list → bool
    *Returns `true` if list is sorted by function `f`.*
    *When `f` is not specified, default ascending order is used.*
split := \(s : real) = \(x : real) = (s · x, (1 – s) · x)
    *Used to reason about circuit diagrams.*
square_len := \(a : vector) = ∑ i { a[i] · a[i] }
$sqrt_A$ : A → A
    *Takes the square root of a number.*
    $sqrt_{\mathbb{N}}$ : nat → nat
        *Defined only for square numbers.*
    $sqrt_{\mathbb{R}}$ : real → real
        *Defined only for non-negative numbers.*
    $sqrt_{\mathbb{C}}$ : complex → complex
        *Automatic conversion from real to complex number.*
strict_subset : set × set → 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 : (⊂ b)` it means `a` is a strict subset of `b`.*
$sub_A$ := \(a : A, b : A) = a – b
    *When written `a : [- b] c` it means `a` minus `b` is equal to `c`.*
    $sub_{\mathbb{C}}$ : complex × complex → complex
    $sub_{\mathbb{N}}$ : \(a : nat ∧ (>= b), b : nat) → nat = { a – b }
    $sub_{\mathbb{Q}}$ : rational × rational → rational
    $sub_{\mathbb{R}}$ : real × real → real
    $sub_{\mathbb{Z}}$ : int × int → int
subset : set × set → bool
    *Returns `true` if all elements of the first set belongs to the second set.*
    *When written `a : (⊆ b)` it means `a` is a subset of `b`.*
sum := \(a : list) = ∑ i { a[i] }
swap := \((a, b)) = (b, a)
sym := \(m : matrix ∧ [dim] [eq] true) = ∀ i, j { m[i][j] == m[j][i] }

# T

tan : real → real
    *The trigonometric tangent function.*
trace := \(m : matrix) = ∑ i, i { m[i][i] }
transform : matrix × vector → vector
    *Transforms a vector through a matrix*
transpose : matrix → matrix
    *Returns the transposed matrix, where rows are swapped with columns.*
...

# ...T (continued)

$true_N$ := \(_, _, …) = true
*A function that always returns `true`.*
$true_0$ := \() = true
$false_1$ := \() = false

# U

union : set × set → set
*Returns the union of two sets.*
*When written `a : [∪ b] c` it means `a` union `b` results in `c`.*
unit : any → ()
*Used to erase information about an input argument.*

# V

vec_dim : vector → nat
*Returns the number of dimensions of a vector.*

# X

x : vector → real
*Returns the x-component of a vector.*
xor := \(a : bool, b : bool) = a ∧ ¬b ∨ ¬a ∧ b
*In C-like programming languages this is equivalent to "a && !b || !a && b".*
*When written `a : (⊻ b)` it means either `a` or `b` is `true`, but not both.*

# Y

y : vector → real
*Returns the y-component of a vector.*

# Z

z : vector → real
*Returns the z-component of a vector.*

# W

w : vector → real
*Returns the w-component of a vector.*