Pointfrip Quickinfo

2023-11-27

The following is about programming at the **function-level** with combinators

Rule

As a rule, **right-before-left** applies, but there are exceptions, e.g. in condition terms.
Parentheses must be used to change the evaluation of the terms.

Infix notation applies as in:	*a* **+** *b*
For functions you write:	*function* **°** *argument*

Data Types

[0] , [1] , [2] , ... , **[*i*]** , [_123]	are selectors that access the values of a list or a dict -- or are integer* numbers
name	is an identifier for an associated function
_123.5678e_30	is a real number
(10 ; 20 ; 30 ; 40 ; 50 ;)	is a list of real numbers
(10 a 20 b 30 c 40 d 50 e)	is a dict* with values and keys
( )	empty list / *null*
(*head infix* **..** *tail*)	data cell / *prop*
"abcdef"	is a string
true / false	are of type *bool*

*note that the constant combinator should be used.

Definition of Functions/Constants/Operators

identifier **==** *term*	assigns a term to the identifier
constname **== '** *literal*	Constants use the constant combinator
oprname **==** (…) **°** **ee**	Operators often use an ee and [0] and [1]

Combinators

'*name*	is the Constant combinator	
function1 **°** *function2*	is the Composition, **o** can also be used	(right-pipe)
fun1 **,** *fun2* **,** … **,** *funm* **,**	is the Construction of a list	
(*test* **->** *then* **;** *else*)	is the Condition combinator with an alternative	
(*test* **->*** *term*)	is a While loop	
(*function* **aa**)	is the Apply-to-All combinator	(map)
(*function* \)	is the Insertr combinator	(reduce)
function1 **ee** *function2*	evaluates the functions and creates a pair from them	
#*name*	picks the value for the name from a dict	
function **:** *argument*	is an Application -- *function*(*argument*)	
list **insl** '*func*	is the Insertl operator; **insr** for Insertr operation	
func **_s**	Single function is executed	
func1 **app** *func2*	Apply function to execute Functionals	
func1 **swee** *func2*	like **ee**, only the elements in the pair are swapped	
(*func* **aa0**) ° *list*,*x*,*y*, … ,	Mixture of **aa** and **distr**, expanded	
(*list*,*x*,*y*, … ,) **map0 '***func*	Mixture of **map** and **distr**, extended	
list **filter** '*boolfunc*	is the Filter operator	

List Processing Functions and Operators

val0 **;** *val1* **;** *val2* **;** … **;**	building lists with literal values
head ° *prop*	extracts the first value of a list
tail ° *prop*	extracts the rest of a list
infix ° *prop*	extracts the infix value of a list/dict
prop ° *hd*, *inf*, *tl*,	creates a data cell with three values
term ° *combi*	extracts the Term value from a Combine data type
arg ° *combi*	extracts the Arg value from a Combine data type
type ° *data*	supplies a name for the data type
list **at** *num*	picks the *num*-th value from the *list*
func **,** *list*	the comma adds an element before the *list*
iota ° *num*	creates a list of numbers from 1 upwards to *num*
num1 **to** *num2*	produces a list of numbers from *num1* to *num2*
reverse ° *list*	reverses a list; also works with a string
trans ° *matrix*	Transpose a list of lists (matrix)
data **distl** *list*	Distribution Left
list **distr** *data*	Distribution Right
data **make** *num*	creates a list of *num* *data*-values
list **take** *num*	returns a list of the first *num* elements
list **drop** *num*	returns the remainder list without the first *num* elements
list1 **++** *list2*	concatenates two lists into a new list
length ° *list*	returns the length of a *list*
list **count** *data*	returns the number of *data* in the *list*
list **find** *data*	returns the first position of *data* in the *list*

Numerical Functions and Operators

num1 **+** *num2*	Addition of numbers of the same type
num1 **-** *num2*	Subtraction of numbers of the same type
num1 ***** *num2*	Multiplication of numbers of the same type
num1 **/** *num2*	Division of numbers
num1 **^** *num2*	Exponentiation of numbers of the same type
num1 **idiv** *num2*	Division of integer numbers
num1 **imod** *num2*	Modulo of integer numbers
pred ° *num*	Predecessor function
succ ° *num*	Successor function
sign ° *num*	Signum function
abs ° *num*	Absolute function
neg ° *num*	Negation of the number
_ ° *num*	Negation of the number
floor ° *num*	Rounding down the number
ceil ° *num*	Round up the number
float ° *num*	converts to a real number
round ° *num*	rounds to an integer number
trunc ° *num*	Integer number with truncation of the decimal places
real **roundto** *num*	rounds to the *num*-th decimal place

exp ° *num*	Exponential function of the number
ln ° *num*	Natural logarithm of the number
lg ° *num*	Ten logarithm of the number
sq ° *num*	the square of the number
sqrt ° *num*	the square root of the number
cbrt ° *num*	the cube root of the number
pi	Function returns the number Pi
2pi	Function returns the perimeter of the unit circle
sin ° *num*	Sine function of the number in radians
cos ° *num*	Cosine function of the number in radians
tan ° *num*	Tangent function of the number in radians
arcsin ° *num*	Arcsine function
arccos ° *num*	Arccosine function
arctan ° *num*	Arc tangent function
y **arctan2** *x*	Phase (or Arg) to (x,y)
sinh ° *num*	Hyperbolic sine function
cosh ° *num*	Hyperbolic cosine function
tanh ° *num*	Hyperbolic tangent function
deg ° *num*	converts radians to degrees
rad ° *num*	converts degree to radian

Boolean Functions and Operators

data1 **=** *data2*	checks for equality
data1 **!=** *data2*	checks for inequality
data1 **<>** *data2*	checks for inequality, alternatively
data1 **<** *data2*	Compare to less than
data1 **>** *data2*	Compare to greater-than
data1 **<=** *data2*	Comparison on less than or equal
data1 **>=** *data2*	Greater-equal comparison
data1 **min** *data2*	Minimum of *data1* and *data2*
data1 **max** *data2*	Maximum of *data1* and *data2*
not ° *bool*	Boolean Not-function
bool1 **and** *bool2*	Boolean And function
bool1 **or** *bool2*	Boolean OR function
bool1 **xor** *bool2*	Boolean Exclusive-Or function
isatom ° *data*	Checks whether *data* belongs to the Atom types
isnull ° *data*	Checks whether *data* is the value ( ), i.e. null
isprop ° *data*	Checks whether *data* is a data cell / *prop*
islist ° *data*	Checks whether *data* is a list
isnum ° *data*	Tests whether *data* is a number, generic
iszero ° *data*	Tests whether *data* is the number 0, generic
ispos ° *num*	Tests whether *num* is a positive number, generic
isneg ° *num*	Tests whether *num* is a negative number, generic
isident ° *data*	Checks whether *data* is an identifier

isint ° *data*	Tests whether *data* is an integer number
isreal ° *data*	Checks whether *data* is a real number
isstring ° *data*	Checks whether *data* is a character string
iscons ° *data*	Checks whether *data* is a List data cell
isquote ° *data*	Checks whether *data* is a Quote value
isivar ° *data*	Checks whether *data* is an instance variable selector
iscombi ° *data*	Checks whether *data* is a Combine value
isact ° *data*	Checks whether *data* is an Act value
isbool ° *data*	Checks whether *data* is a boolean value
isbound ° *ident*	Checks whether the identifier is already defined
isundef ° *data*	Checks whether *data* is the value **_undef**
data **in** *list*	Checks whether *data* is included as an element in the *list*

Dict Functions and Operators

#*ident* ° *dict*	the selector picks the value from the *dict* for the *ident* key
dict **iget** *ident*	for the *ident** key, the value is picked out of the *dict*
dict **iput** *ident*, *value*,	the *value* for the *ident** key is newly created in the *dict*
dict **get** *key*	for the *key*, the value is picked out of the *dict*
dict **put** *key*, *value*,	the *value* for the *key* is newly created in the *dict*
(*ident* **:=** *func*) ° *dict*	as with **iput**, this "*variable*" assignment occurs
(*func* **<-** *x* ; *y* ; ... ;) ° *list*	*func* applies the generated Dict, as after an assign
keys ° *dict*	creates a list with all Keys from the *dict*
values ° *dict*	creates a list with all Values from the *dict*
it ° *dict*	picks the value associated with **_it** from the *dict*

String Functions and Operators

length ° *string*	specifies the length of the string
substring ° *string*, *i*, *len*,	copies a substring from *string*
string1 **&** *string2*	concatenates two strings
string1 **concat** *string2*	concatenates two strings
string **indexof** *substr*	searches the position of *substr* in the *string* from the left
trim ° *string*	cuts off the spaces left and right
triml ° *string*	cuts off the spaces on the left
trimr ° *string*	cuts off the spaces on the right
upper ° *string*	converts the string to uppercase
lower ° *string*	converts the string to lowercase
capitalize ° *string*	converts the string into a capital word

char ° *num*	produces a character according to the Unicode value
unicode ° *string*	specifies the Unicode value of the first character
parse ° *string*	parses the string with the Pointfrip-parser
value ° *string*	converts numbers, words, lists in the *string* into data
string ° *data*	converts the *data* into a print string
unpack ° *string*	breaks the *string* into a list of individual characters
string **split** *delstr*	breaks the *string* into a list of strings without *delstr*
list **join** *insstr*	connects the strings of the *list* with *insstr* in between

Matrix Functions and Operators

matrix1 **add** *matrix2*	Adds two matrices, component by component
matrix1 **sub** *matrix2*	Subtracts *matrix2* from *matrix1*
matrix1 **mul** *matrix2*	Multiplies two matrices
num **mul** *matrix* *matrix* **mul** *num*	Multiplies the matrix by a scalar value
ismat ° *data*	Checks whether *data* is a matrix, simplified form
trans ° *matrix*	Transpose the *matrix*
det ° *matrix*	calculates the Determinant of the *matrix*
inv ° *matrix*	calculates the Inverse matrix
num1 **zeromat** *num2*	creates a matrix with all zeros
idmat ° *num*	Identity matrix of size *num*
fail ° *infodata*	generates standard error message for a fail
list **IP** *list*	Inner Product according to John Backus
matrix **MM** *matrix*	Matrix multiplication according to John Backus
rnd ° *matrix*	Rounds *matrix* to five decimal places
zero ° *data* **zero** ° *matrix*	generates a Zero, depending on the type
one ° *data* **one** ° *matrix*	generates a One, depending on the type

Misc Functions and Operators

undef	generates error message for undefined function
id ° *data*	Identity function returns *data*
name ° *ident*	extracts the string of the identifier
body ° *ident*	extracts the definition value of the identifier
info ° *ident*	extracts the compiler-string of the identifier
identlist	outputs a list of all used identifiers
quote ° *data*	turns *data* into a Quote value
ident **error** *string*	outputs an error message with *ident* and *string*
‘*func1* **comp** ‘*func2*	chains the functions into a new function
int **act** *dict*	creates an Act value with the data - (Monade)
act **bind** '*func*	creates the *func* in the bind field of a new *act*
act **>>** *func*	creates the *func* in the bind field of a new *act*
fname **load**	reads the text from the file *fname* into the display
fname **save**	saves the text from the display to the file *fname*
files	outputs a list with all file names
fname **loadtext**	loads the string from the file *fname* in the "pf/" folder
fname **savetext** *string*	saves the *string* in the file *fname* in the "pf/" folder
stopvm	the calculation aborts with an error message
dump	displays all identifiers with their assignments
savedump (for test)	displays all info-strings of the identifiers
help	Link address to current help-PDF
pim ° *num*	gives a list of all prime factors of a number, example
(*test* **try** *then*;*else*)°*argum*	Checks *test* for Error -> *then*/*else* with **(***result* ; *argum* ;**)**

Notes on Loading and Saving Program Files

"*filename*" **save**	a program text is saved under the name *filename* in the "pf/" folder
"*filename*" **load**	a program text from the file *filename* from the "pf/" folder is read in with the definitions
files	outputs a list of all file names in the "pf/" folder

With **identlist** or **dump** you get an overview of the used words.

Expansion of Prelude with some Definitions

(*num* **r**) ° *list*	accesses the *num*-th element from the back of the *list*
tailr ° *list*	copy of the *list* without the last element
last ° *list*	the last element of the *list*
rotl ° *list*	rotation of the list elements to the left direction
rotr ° *list*	rotation of the list elements to the right direction
‘*expr* **times** *num*,*initakku*,	repeats *expr* *num* times with *initakku* as start argument
foldl ° ‘*expr*,*initakku*,*list*,	reduces the *list* with *expr* from the left side with *initakku* as the starting value
foldr ° ‘*expr*,*initakku*,*list*,	reduces the *list* with *expr* from the right side with *initakku* as the starting value

*note that the constant combinator should be used.