

## Todo list

These might need to be replaced with variants for Reals/Integers . . . . .	3
Do we want to have the length of our vectors as a type argument? . . . .	3
define criteria for what a well-formed expression language should provide	3
Quantities discussion – remaining untyped . . . . .	3
modulo, remainder, etc. . . . .	5
discuss vectors in general . . . . .	6
oddity: topology appears as a constructor arg and signature arg but can desync – can we just remove the constructor arg? . . . . .	6
addressed in “misc” section . . . . .	7

## 1 Syntax

### 1.1 Current

An idealized version of the current syntax.

Type	$\tau$	$::=$	<b>Integer</b>	$\mathbb{Z}$	Integer numbers
			<b>Real</b>	$\mathbb{R}$	Real numbers
			<b>String</b>	<i>String</i>	Text
			<b>Bool</b>	$\mathbb{B}$	Truth values (true/false)
			<b>Vector</b> ( $\tau, n$ )	$[\tau]_n$	Vectors (single element type)
			<b>Tuple</b> ( $\tau_1 \dots \tau_n$ )	$\tau_1 \times \tau_2 \times \dots \times \tau_n$	Alternative vectors/tuples
Literal	$l$	$::=$	<b>Integer</b> [ $n$ ]	$n$	Integer number
			<b>Real</b> [ $r$ ]	$r$	Real number
			<b>String</b> [ $s$ ]	" $s$ "	Text
			<b>Bool</b> [ $b$ ]	$b$	Boolean value
			<b>Vector</b> ( $l_1 \dots l_n$ )	$\langle l_1, \dots, l_n \rangle$	Vectors
UnaryOp	$u$	$::=$	<b>Not</b>	$\neg$	Logical negation
			<b>Neg</b>	$-$	Numeric negation
			<b>...</b>		
BinaryOp	$\oplus$	$::=$	<b>Sub</b>	$-$	Subtraction
			<b>Pow</b>	$-^$	Powers
			<b>...</b>		
AssocBinOp	$\otimes$	$::=$	<b>Add</b>	$+$	Addition
			<b>Mul</b>	$\times$	Multiplication
			<b>...</b>		
Expr	$e$	$::=$	<b>Literal</b> ( $l$ )	$l$	Literal values
			<b>Vector</b> ( $e_1 \dots e_n$ )	$\langle e_1, \dots, e_n \rangle$	Vectors
			<b>UnaryOp</b> ( $u, e$ )	$u(e)$	Unary operations
			<b>BinaryOp</b> ( $\oplus, e_1, e_2$ )	$e_1 \oplus e_2$	Binary operations
			<b>AssocOp</b> ( $\otimes, e_1 \dots e_n$ )	$e_1 \otimes \dots \otimes e_n$	Associative binary operations
			<b>Case</b> ( $e_{1c} e_{1e} \dots e_{nc} e_{ne}$ )	<i>if</i> $e_{1c}$ <i>then</i> $e_{2e}$ <i>elif</i> $e_{2c}$ ...	If-then-else-if-then-else (Switch)

## 2 Typing Rules

### 2.1 Literal

1. Integers:

$$\frac{i : \text{Integer}}{\text{Integer}[i] : \text{Literal Integer}} \quad (1)$$

2. Strings (Text):

$$\frac{s : \text{String}}{\text{Str}[s] : \text{Literal String}} \quad (2)$$

3. Real numbers:

$$\frac{d : \text{Double}}{\text{Db}[d] : \text{Literal Real}} \quad (3)$$

4. Whole numbered reals ( $\mathbb{Z} \subset \mathbb{R}$ ):

$$\frac{d : \text{Integer}}{\text{ExactDbl}[d] : \text{Literal Real}} \quad (4)$$

5. Percentages:

$$\frac{n : \text{Integer} \quad d : \text{Integer}}{\text{Perc}[n, d] : \text{Literal Real}} \quad (5)$$

## 2.2 Miscellaneous

**Sorts Legend**     $\text{Numerics}(T)$  : any numeric type  
                           $\text{NumericsWithNegation}(T)$  : any signed numeric type

### Vectors

As of right now, Drasil/GOOL only supports lists and arrays as “code types”, which would be the representations used for representing “vectors” in Drasil.

For now, the below type rules define vectors with Haskell lists. We can choose to create our own type with the length of the vector as a parameter – likely going “too far into Haskell”.

### Functions

Presently, functions are defined through “QDefinitions”, where a list of UIDs used in an expression are marked as the parameters of the function. Function “calls”/applications are captured in “Expr” (the expression language) by providing a list of input expressions and a list of named inputs (expressions) –  $f(x, y, z, a = "b")$ .

A few solutions:

1. Leave expressions in general untyped in Haskell, and rely on calculating the “space” of an expression dynamically to ensure that expressions are well-formed. If runtime (drasil’s compiling-knowledge-time) type analysis is ever needed, this will prove much easier to use in general.
2. Push the typing rules into Haskell via Generalized Algebraic Data Types (GADTs). Here, a larger question appears regarding functions – how should we handle function creation, application, and typing?
  - (a) Currying and applying arguments (allowing partial function applications): This would work well if we only generated functional languages, but it might prove problematic for GOOL if expressions are left with partial function applications.

### Type Rules

1. Completeness:

$$\frac{}{\text{Complete}[] : \text{Completeness}} \quad (6)$$

$$\frac{}{\text{Incomplete}[] : \text{Completeness}} \quad (7)$$

These might need to be replaced with variants for Reals/Integers

Do we want to have the length of our vectors as a type argument?

define criteria for what a well-formed expression language should provide

Quantities discussion – remaining untyped

2. AssocOp:

(a) Numerics:

$$\frac{x : \text{Numerics}(T)}{\text{Add}[] : \text{AssocOp } x} \quad (8)$$

$$\frac{x : \text{Numerics}(T)}{\text{Mul}[] : \text{AssocOp } x} \quad (9)$$

(b) Bool:

$$\overline{\text{And}[] : \text{AssocOp } \text{Bool}} \quad (10)$$

$$\overline{\text{Or}[] : \text{AssocOp } \text{Bool}} \quad (11)$$

3. UnaryOp:

(a) Numerics:

$$\frac{x : \text{NumericsWithNegation}(T)}{\text{Neg}[] : \text{UnaryOp } x \ x} \quad (12)$$

$$\frac{x : \text{NumericsWithNegation}(T)}{\text{Abs}[] : \text{UnaryOp } x \ x} \quad (13)$$

$$\frac{x : \text{Numerics}(T)}{\text{Exp}[] : \text{UnaryOp } x \ \text{Real}} \quad (14)$$

For Log, Ln, Sin, Cos, Tan, Sec, Csc, Cot, Arcsin, Arccos, Arctan, and Sqrt, please use the following template, replacing “\$TRG” with the desired operator:

$$\overline{\text{\$TRG}[] : \text{UnaryOp } \text{Real } \text{Real}} \quad (15)$$

$$\overline{\text{RtoI}[] : \text{UnaryOp } \text{Real } \text{Integer}} \quad (16)$$

$$\overline{\text{ItoR}[] : \text{UnaryOp } \text{Integer } \text{Real}} \quad (17)$$

$$\overline{\text{Floor}[] : \text{UnaryOp } \text{Real } \text{Integer}} \quad (18)$$

$$\overline{\text{Ceil}[] : \text{UnaryOp } \text{Real } \text{Integer}} \quad (19)$$

$$\overline{\text{Round}[] : \text{UnaryOp } \text{Real } \text{Integer}} \quad (20)$$

$$\overline{\text{Trunc}[] : \text{UnaryOp } \text{Real } \text{Integer}} \quad (21)$$

(b) Vectors:

$$\frac{x : \text{NumericsWithNegation}(T)}{NegV[] : \text{UnaryOp } [x] \ [x]} \quad (22)$$

$$\frac{x : \text{Numerics}(T)}{Norm[] : \text{UnaryOp } [x] \ \text{Real}} \quad (23)$$

$$\frac{x : \tau}{Dim[] : \text{UnaryOp } [x] \ \text{Integer}} \quad (24)$$

(c) Booleans:

$$Not[] : \text{UnaryOp } \text{Bool } \text{Bool} \quad (25)$$

4. BinaryOp:

(a) Arithmetic:

$$FracI[] : \text{BinaryOp } \text{Integer } \text{Integer } \text{Integer} \quad (26)$$

$$FracR[] : \text{BinaryOp } \text{Real } \text{Real } \text{Real} \quad (27)$$

(b) Bool:

$$Impl[] : \text{BinaryOp } \text{Bool } \text{Bool } \text{Bool} \quad (28)$$

$$If f[] : \text{BinaryOp } \text{Bool } \text{Bool } \text{Bool} \quad (29)$$

(c) Equality:

$$\frac{x : \tau}{Eq[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (30)$$

$$\frac{x : \tau}{NEq[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (31)$$

(d) Ordering:

$$\frac{x : \text{Numerics}(T)}{Lt[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (32)$$

$$\frac{x : \text{Numerics}(T)}{Gt[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (33)$$

$$\frac{x : \text{Numerics}(T)}{LEq[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (34)$$

$$\frac{x : \text{Numerics}(T)}{GEq[] : \text{BinaryOp } x \ x \ \text{Bool}} \quad (35)$$

modulo,  
remainder,  
etc.

(e) Indexing:

$$\frac{x : \tau}{\text{Index}[] : \text{BinaryOp } [x] \text{ Integer } x} \quad (36)$$

(f) Vectors:

$$\frac{x : \text{Numerics}(T)}{\text{Cross}[] : \text{BinaryOp } [x] [x] [x]} \quad (37)$$

discuss  
vectors in  
general

$$\frac{x : \text{Numerics}(T)}{\text{Dot}[] : \text{BinaryOp } [x] [x] x} \quad (38)$$

$$\frac{x : \text{Numerics}(T)}{\text{Scale}[] : \text{BinaryOp } [x] x [x]} \quad (39)$$

5. RTopology:

$$\overline{\text{Discrete}[] : \text{RTopology}} \quad (40)$$

$$\overline{\text{Continuous}[] : \text{RTopology}} \quad (41)$$

6. DomainDesc:

$$\frac{top : \tau_1 \quad bot : \tau_2 \quad s : \text{Symbol} \quad rtop : \text{RTopology}}{\text{BoundedDD}[s, rtop, top, bot] : \text{DomainDesc Discrete } \tau_1 \tau_2} \quad (42)$$

$$\frac{topT : \tau \quad botT : \tau \quad s : \text{Symbol} \quad rtop : \text{RTopology}}{\text{AllDD}[s, rtop] : \text{DomainDesc Continuous topT botT}} \quad (43)$$

oddity:  
topology  
appears  
as a con-  
structor  
arg and  
signature  
arg but  
can desync  
– can we  
just re-  
move the  
construc-  
tor arg?

7. Inclusive:

$$\overline{\text{Inc}[] : \text{Inclusive}} \quad (44)$$

$$\overline{\text{Exc}[] : \text{Inclusive}} \quad (45)$$

8. RealInterval:

$$\frac{a : \tau \quad b : \tau \quad top : (\text{Inclusive}, a) \quad bot : (\text{Inclusive}, b)}{\text{Bounded}[top, bot] : \text{RealInterval } a \ b} \quad (46)$$

$$\frac{a : \tau \quad b : \tau \quad top : (\text{Inclusive}, a)}{\text{UpTo}[top] : \text{RealInterval } a \ b} \quad (47)$$

$$\frac{a : \tau \quad b : \tau \quad bot : (\text{Inclusive}, b)}{\text{UpFrom}[bot] : \text{RealInterval } a \ b} \quad (48)$$

## 2.3 Expr

1. Literals:

$$\frac{x : \tau \quad l : \text{Literal } x}{\text{Lit}[l] : \text{Expr } x} \quad (49)$$

2. Associative Operations:

$$\frac{x : \tau \quad op : \text{AssocOp } x \quad args : [\text{Expr } x]}{\text{Assoc}[op, args] : \text{Expr } x} \quad (50)$$

3. Symbols:

$$\frac{x : \tau \quad u : \text{UID}}{C[u] : \text{Expr } x} \quad (51)$$

4. Function Call:

5. Case:

$$\frac{x : \tau \quad c : \text{Completeness} \quad ces : [(\text{Expr Bool}, \text{Expr } x)]}{\text{Case}[c, ces] : \text{Expr } x} \quad (52)$$

6. Matrices:

$$\frac{x : \tau \quad es : [[\text{Expr } x]]}{\text{Matrix}[es] : \text{Expr } x} \quad (53)$$

7. Unary Operations:

$$\frac{x : \tau \quad y : \tau \quad op : \text{UnaryOp } x \quad y \quad e : \text{Expr } x}{\text{Unary}[op, e] : \text{Expr } y} \quad (54)$$

8. Binary Operations:

$$\frac{x : \tau \quad y : \tau \quad z : \tau \quad op : \text{BinaryOp } x \quad y \quad z \quad l : \text{Expr } x \quad r : \text{Expr } y}{\text{Binary}[op, l, r] : \text{Expr } z} \quad (55)$$

9. “Big” Operations:

$$\frac{x : \tau \quad op : \text{AssocOp } x \quad dom : \text{DomainDesc Discrete (Expr } x) \text{ (Expr } x)}{\text{BigOp}[op, dom] : \text{Expr } x} \quad (56)$$

10. “Is in interval” operator:

$$\frac{x : \tau \quad u : \text{UID} \quad itvl : \text{RealInterval (Expr } x) \text{ (Expr } x)}{\text{RealI}[u, itvl] : \text{Expr } x} \quad (57)$$

addressed  
in “misc”  
section

## 2.4 ModelExpr

- 1.

$$\frac{B \quad C}{A}$$

## 2.5 CodeExpr

1.

$$\frac{B \ C}{A}$$