Todo list

1 Syntax

1.1 Current

An idealized version of the current syntax.

Type	τ	::=	`Integer`	$\mathbb{Z}$	Integer numbers
		$\mid$	`Real`	$\mathbb{R}$	Real numbers
		$\mid$	`String`	$String$	Text
		$\mid$	`Bool`	$\mathbb{B}$	Truth values (true/false)
		$\mid$	`Vector(`τ`, `n`)`	$[\tau]_n$	Vectors (single element t
		$\mid$	`Tuple(`$\tau_1 ... \tau_n$`)`	$\tau_1 \times \tau_2 \times ... \times \tau_n$	Alternative vectors/tuple
Literal	l	::=	`Integer[`n`]`	n	Integer number
		$\mid$	`Real[`r`]`	r	Real number
		$\mid$	`String[`s`]`	"s''	Text
		$\mid$	`Bool[`b`]`	b	Boolean value
		$\mid$	`Vector(`$l_1 ... l_n$`)`	$< l_1, ..., l_n >$	Vectors*
		$\mid$	`Tuple(`$l_1 ... l_n$`)`	$(l_1, ..., l_n)$	Tuples*
UnaryOp	$\ominus$	::=	`Not`	$\neg\_$	Logical negation
		$\mid$	`Neg`	$-\_$	Numeric negation
		$\mid$	`...`		omitted for brevity
BinaryOp	$\oplus$	::=	`Sub`	$\_ - \_$	Subtraction
		$\mid$	`Pow`	$\_^{\_}$	Powers
		$\mid$	`...`		omitted for brevity
AssocBinOp	$\otimes$	::=	`Add`	$\_ + \_$	Addition
		$\mid$	`Mul`	$\_ \times \_$	Multiplication
		$\mid$	`...`		omitted for brevity
UID	u	::=	`UID(s)`	`UID ‘‘s’’`	UIDs
Expr	e	::=	`Literal(`l`)`	l	Literal values
		$\mid$	`Vector(`$e_1 ... e_n$`)`	$< e_1, ..., e_n >$	Vectors
		$\mid$	`Var(`u`)`	u	Variable (QuantityDict C
		$\mid$	`FuncCall(`$f, e_1 ... e_n$`)`	$f(e_1 ... e_n)$	"Complete" function app
		$\mid$	`UnaryOp(`$\ominus$`,`e`)`	$\ominus\ e$	Unary operations
		$\mid$	`BinaryOp(`$\oplus$`,`e_1`,`e_2`)`	$e_1 \oplus e_2$	Binary operations
		$\mid$	`AssocOp(`$\otimes$`, `$e_1 ... e_n$`)`	$e_1 \otimes ... \otimes e_n$	Associative binary operat
		$\mid$	`Case(`$e_{1c} e_{1e} ... e_{nc} e_{ne}$`)`	$if\ e_{1c}\ then\ e_{2e}\ elif\ e_{2c}\ ...$	If-then-else-if-then-else (S
		$\mid$	`BigAsBinOp(`$\otimes, e_1, e_2$`)`	$\bigotimes_{i=e_1}^{e_2} i$	Apply a "big" op to a dis
		$\mid$	`IsInRlItrvl(`u, e_1, e_2`)`	$u \in [e_1, e_2]$	Variable in range

$*$: does not currently appear in the code at the moment, but would be needed/desired

2 Typing Rules

2.1 Literal

1. Integers:
$$\frac{i : \texttt{Integer}}{\mathit{Integer}[i] : \texttt{Literal Integer}} \tag{1}$$

2. Strings (Text):
$$\frac{s : \texttt{String}}{\mathit{Str}[s] : \texttt{Literal String}} \tag{2}$$

3. Real numbers:
$$\frac{d : \texttt{Double}}{\mathit{Dbl}[d] : \texttt{Literal Real}} \tag{3}$$

4. Whole numbered reals ($\mathbb{Z} \subset \mathbb{R}$):
$$\frac{d : \texttt{Integer}}{\mathit{ExactDbl}[d] : \texttt{Literal Real}} \tag{4}$$

5. Percentages:
$$\frac{n : \texttt{Integer} \quad d : \texttt{Integer}}{\mathit{Perc}[n, d] : \texttt{Literal Real}} \tag{5}$$

2.2 Miscellaneous

Sorts Legend		
	Numerics(T)	: any numeric type
	NumericsWithNegation(T)	: any signed numeric type

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

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.

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

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) – $f(x, y, z, a = \text{"}b\text{"})$.

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

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.

Quantities discussion – remain-

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:

$$\frac{}{Complete[] : \texttt{Completeness}} \tag{6}$$

$$\frac{}{Incomplete[] : \texttt{Completeness}} \tag{7}$$

2. AssocOp:

   (a) Numerics:

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

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

   (b) Bool:

$$\frac{}{And[] : \texttt{AssocOp Bool}} \tag{10}$$

$$\frac{}{Or[] : \texttt{AssocOp Bool}} \tag{11}$$

3. UnaryOp:

   (a) Numerics:

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

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

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

   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:

$$\frac{}{\$TRG[] : \texttt{UnaryOp Real Real}} \tag{15}$$

$$\frac{}{RtoI[] : \texttt{UnaryOp Real Integer}} \tag{16}$$

$$\frac{}{ItoR[] : \texttt{UnaryOp Integer Real}} \tag{17}$$

$$\frac{}{Floor[] : \texttt{UnaryOp Real Integer}} \tag{18}$$

$$\frac{}{Ceil[] : \texttt{UnaryOp Real Integer}} \tag{19}$$

$$\frac{}{Round[] : \texttt{UnaryOp Real Integer}} \tag{20}$$

$$\frac{}{Trunc[] : \texttt{UnaryOp Real Integer}} \tag{21}$$

(b) Vectors:

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

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

$$\frac{\texttt{x} : \tau}{Dim[] : \texttt{UnaryOp [x] Integer}} \tag{24}$$

(c) Booleans:

$$\frac{}{Not[] : \texttt{UnaryOp Bool Bool}} \tag{25}$$

4. BinaryOp:

(a) Arithmetic:

$$\frac{}{FracI[] : \texttt{BinaryOp Integer Integer Integer}} \tag{26}$$

$$\frac{}{FracR[] : \texttt{BinaryOp Real Real Real}} \tag{27}$$

modulo, remainder, etc.

(b) Bool:

$$\frac{}{Impl[] : \texttt{BinaryOp Bool Bool Bool}} \tag{28}$$

$$\frac{}{Iff[] : \texttt{BinaryOp Bool Bool Bool}} \tag{29}$$

(c) Equality:

$$\frac{\mathtt{x} : \tau}{Eq[] : \mathtt{BinaryOp\ x\ x\ Bool}} \tag{30}$$

$$\frac{\mathtt{x} : \tau}{NEq[] : \mathtt{BinaryOp\ x\ x\ Bool}} \tag{31}$$

(d) Ordering:

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

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

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

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

(e) Indexing:

$$\frac{\mathtt{x} : \tau}{Index[] : \mathtt{BinaryOp\ [x]\ Integer\ x}} \tag{36}$$

(f) Vectors:

discuss vectors in general

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

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

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

5. RTopology:

$$\frac{}{Discrete[] : \mathtt{RTopology}} \tag{40}$$

$$\frac{}{Continuous[] : \mathtt{RTopology}} \tag{41}$$

6. DomainDesc:

oddity: topology appears as a constructor arg and signature arg but can desync – can we just remove the constructor arg?

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

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

7. Inclusive:

$$\frac{}{Inc[] : \texttt{Inclusive}} \tag{44}$$

$$\frac{}{Exc[] : \texttt{Inclusive}} \tag{45}$$

8. RealInterval:

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

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

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

2.3 Expr

1. Literals:

$$\frac{\texttt{x} : \tau \quad l : \texttt{Literal x}}{Lit[l] : \texttt{Expr x}} \tag{49}$$

2. Associative Operations:

$$\frac{\texttt{x} : \tau \quad op : \texttt{AssocOp x} \quad args : \texttt{[Expr x]}}{Assoc[op, args] : \texttt{Expr x}} \tag{50}$$

3. Symbols:

$$\frac{\texttt{x} : \tau \quad u : \texttt{UID}}{C[u] : \texttt{Expr x}} \tag{51}$$

4. Function Call:

addressed in “misc” section

5. Case:

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

6. Matrices:

$$\frac{\texttt{x} : \tau \quad es : \texttt{[[Expr x]]}}{Matrix[es] : \texttt{Expr x}} \tag{53}$$

7. Unary Operations:

$$\frac{\texttt{x} : \tau \quad \texttt{y} : \tau \quad op : \texttt{UnaryOp x y} \quad e : \texttt{Expr x}}{Unary[op, e] : \texttt{Expr y}} \tag{54}$$

8. Binary Operations:

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

9. “Big” Operations:

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

10. “Is in interval” operator:

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

2.4 ModelExpr

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

2.5 CodeExpr

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