Adding Types and Theory Kinds to Drasil

Jason Balaci
Under the supervision of Dr. Jacques Carette

McMaster University

Dec. 8^{th}, 2022

Table of Contents

Table of Contents

Overview

1. Background: Drasil
2. 4 Research Areas:
    1. Structuring theories
    2. Restricting mathematical expression terminology to appropriate contexts
    3. Well-typedness of mathematical expressions
    4. An extensible database for remembering everything

Table of Contents

What is Drasil? How does it work?

"Generate All The Things!"

1. Software generation suite for "well-understood" domains
2. De-duplicating and capturing knowledge across software artifacts
3. Uses a Software Requirements Specification (SRS) template to decompose scientific problems and generate software

Example
SRS to Code

Using SRS components:

1. Symbols (inputs, outputs, and everything in-between)
2. Problem description and goals
3. Assumptions
4. Abstract theories
5. Concrete theories
6. ...

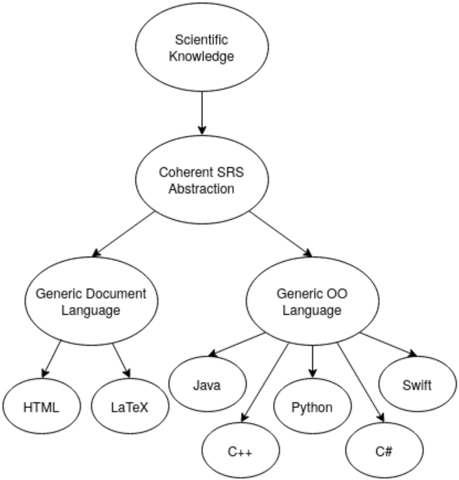


Table of Contents

How are theories used?

Refname	IM:calOfLandingDist
Label	Calculation of landing position
Input	v_{launch}, θ
Output	p_{land}
...	...
Equation	$p_{\text{land}} = \frac{2 v_{\text{launch}}{}^2 \sin(\theta) \cos(\theta)}{\mathbf{g}}$
Description	p_{land} is the landing position (m) v_{launch} is the launch speed ($\frac{\text{m}}{\text{s}}$) θ is the launch angle (rad) $\boldsymbol{g}$ is the gravitational acceleration ($\frac{\text{m}}{\text{s}^2}$)

```
...
landPosExpr :: Expr
landPosExpr = sy landPos $= 2 * square (sy
↪   launSpeed) * sin (sy launAngle) * cos (sy
↪   launAngle) / sy gravitationalAccelConst

landPosRC :: RelationConcept
landPosRC = makeRC "landPosRC" (nounPhraseSP
↪   "calculation of landing position")
↪   landPosConsNote landPosExpr
...
```

How are theories used?

```haskell
relToQD :: ExprRelat c => ChunkDB -> c -> QDefinition
relToQD sm r = convertRel sm (r ^. relat)

convertRel :: ChunkDB -> Expr -> QDefinition
convertRel d (BinaryOp Eq (C x) r) = ec (symbResolve d x) r
convertRel _ _ = error "Conversion failed"
```

```java
public static double func_p_land(double v_launch, double theta, double g_vect)
↪ {
  return 2 * Math.pow(v_launch, 2) * Math.sin(theta) * Math.cos(theta) /
  ↪ g_vect;
}
```

Decompose, classify, and encode

```haskell
data ModelKinds e where
  NewDEModel               :: DifferentialModel -> ModelKinds e
  DEModel                  :: RelationConcept   -> ModelKinds e
  EquationalConstraints    :: ConstraintSet e   -> ModelKinds e
  EquationalModel          :: QDefinition e     -> ModelKinds e
  EquationalRealm          :: MultiDefn e       -> ModelKinds e
  OthModel                 :: RelationConcept   -> ModelKinds e
```

- EquationalModel: $x := f(x, y, z, \dots)$
- EquationalConstraints: $a \wedge b \wedge c \wedge \dots$
- EquationalRealm: $x := f(x, y, z, \dots) \vee x := g(x, y, z, \dots) \vee \dots$
- DEModel & NewDEModel: $dy = f(x, y, \dots)$
- OthModel: ?

Bigger picture

- Categorization through types/constructors.
- Structured creation, interaction, and analysis.
- More opportunity for (domain-specific) interpretation.
    - Code generation!

Table of Contents

What are they used for?

In encoding...

Code (OO)

```
def theta(...):
  return math.asin(d * g
  ↪  / (v ** 2)) / 2
```

Concrete theories

Refname	IM:calOfLandingDist
Label	Calculation of landing position
Input	v_{launch}, θ
Output	p_{land}
...	...
Equation	$p_{\text{land}} = \frac{2{v_{\text{launch}}}^2 \sin(\theta)\cos(\theta)}{\mathbf{g}}$
Description	p_{land} is the landing position (m) v_{launch} is the launch speed ($\frac{m}{s}$) θ is the launch angle (rad) $\boldsymbol{g}$ is the gravitational acceleration ($\frac{m}{s^2}$)

Abstract theories

Refname	TM:acceleration
Label	Acceleration
Equation	$\mathbf{a} = \frac{d\mathbf{v}}{dt}$
Description	$\boldsymbol{a}$ is the acceleration ($\frac{m}{s^2}$) t is the time (s) $\boldsymbol{v}$ is the velocity ($\frac{m}{s}$)
Source	accelerationWiki
RefBy	GD:rectVel

Restricting terms by context

1. Split up the expression language by context.
2. "Typed-tagless" encoding to make usage seamless and interoperable.

$$\texttt{Expr} \Rightarrow \texttt{Expr} \cup \texttt{ModelExpr} \cup \texttt{CodeExpr}$$

$$\texttt{ModelExpr} \supseteq \texttt{Expr} \subseteq \texttt{CodeExpr}$$

Bigger picture

1. Restricting terms by context.
2. Stop users from entering in unexpected expressions.
3. Know when "Equational Models" are usable for (OO) code generation.

Validity of Expressions

```
data Expr where
  Lit       :: Literal -> Expr
  AssocA    :: AssocArithOper -> [Expr] -> Expr
  AssocB    :: AssocBoolOper  -> [Expr] -> Expr
  C         :: UID -> Expr
  ...
```

```
illTyped :: Expr
illTyped = int 1 $+ str "Drasil"
```

```
public static double func_ex() {
    return 1 + "Drasil";
}
```

```
error: incompatible
↪   types: String
↪   cannot be
↪   converted to
↪   double
  return 1 +
   ↪   "Drasil";
          ^
```

Typing

1. Adding *bidirectional* type-checking, reporting en masse.

$$\frac{v : s \in \Gamma}{\Gamma \vdash v \Rightarrow s}\ \textsc{Symbols}$$

$$\frac{l \text{ is a literal of type } s}{\Gamma \vdash l \Rightarrow s}\ \textsc{Literals}$$

$$\frac{\oplus \in \{+, \cdot\} \quad isNum(s) \quad \Gamma \vdash e_1 \Rightarrow s \quad \Gamma \vdash e_2 \Rightarrow s \quad \ldots \quad \Gamma \vdash e_n \Rightarrow s}{\Gamma \vdash (e_1 \oplus e_2 \oplus \ldots \oplus e_n) \Rightarrow s}\ \textsc{Assoc. Arith. Ops}$$

$$\frac{\oplus \in \{\wedge, \vee\} \quad \Gamma \vdash e_1 \Rightarrow \mathbb{B} \quad \Gamma \vdash e_2 \Rightarrow \mathbb{B} \quad \ldots \quad \Gamma \vdash e_n \Rightarrow \mathbb{B}}{\Gamma \vdash (e_1 \oplus e_2 \oplus \ldots \oplus e_n) \Rightarrow \mathbb{B}}\ \textsc{Assoc. Bool. Ops}$$

$$\frac{f : (s_1 \times s_2 \times \ldots \times s_n) \rightarrow s \in \Gamma \quad \Gamma \vdash e_1 \Rightarrow s_1 \quad \Gamma \vdash e_2 \Rightarrow s_2 \quad \ldots \quad \Gamma \vdash e_n \Rightarrow s_n}{\Gamma \vdash f(e_1, e_2, \ldots, e_n) \Rightarrow s}\ \textsc{Fun. App.}$$

$$\frac{\Gamma \vdash e \Rightarrow s}{\Gamma \vdash e \Leftarrow s}\ \textsc{Checked by Inference}$$

$$\frac{\begin{matrix} \Gamma \vdash e_1 \Rightarrow s & \Gamma \vdash c_1 \Rightarrow \mathbb{B} \\ \Gamma \vdash e_2 \Rightarrow s & \Gamma \vdash c_2 \Rightarrow \mathbb{B} \\ \vdots & \\ \Gamma \vdash e_n \Rightarrow s & \Gamma \vdash c_n \Rightarrow \mathbb{B} \end{matrix}}{\Gamma \vdash \texttt{Cases}(e_1, \ldots, e_n, c_1, \ldots, c_n) \Rightarrow s}\ \textsc{Cases}$$

$$\frac{\begin{matrix} \Gamma \vdash e_{11} \Rightarrow s & \Gamma \vdash e_{12} \Rightarrow s & \ldots & \Gamma \vdash e_{1n} \Rightarrow s \\ \Gamma \vdash e_{21} \Rightarrow s & \Gamma \vdash e_{22} \Rightarrow s & \ldots & \Gamma \vdash e_{2n} \Rightarrow s \\ \vdots & & & \\ \Gamma \vdash e_{m1} \Rightarrow s & \Gamma \vdash e_{m2} \Rightarrow s & \ldots & \Gamma \vdash e_{mn} \Rightarrow s \end{matrix}}{Matrix(e_{11}, \ldots, e_{mn}) \Rightarrow \texttt{Matrix}(m, n, s)}\ \textsc{Matrix}$$

$$\frac{isNum(s) \quad s \neq \mathbb{N} \quad \Gamma \vdash e \Rightarrow s}{\Gamma \vdash |e| \Rightarrow s}\ \textsc{Abs.}$$

$$\frac{isNum(s) \quad s \neq \mathbb{N} \quad \Gamma \vdash e \Rightarrow s}{\Gamma \vdash -e \Rightarrow s}\ \textsc{Neg.}$$

$$\frac{s \in \{\mathbb{R}, \mathbb{Z}\} \quad \Gamma \vdash p \Rightarrow s}{\Gamma \vdash \mathsf{e}^p \Rightarrow \mathbb{R}}\ \textsc{Exp.}$$

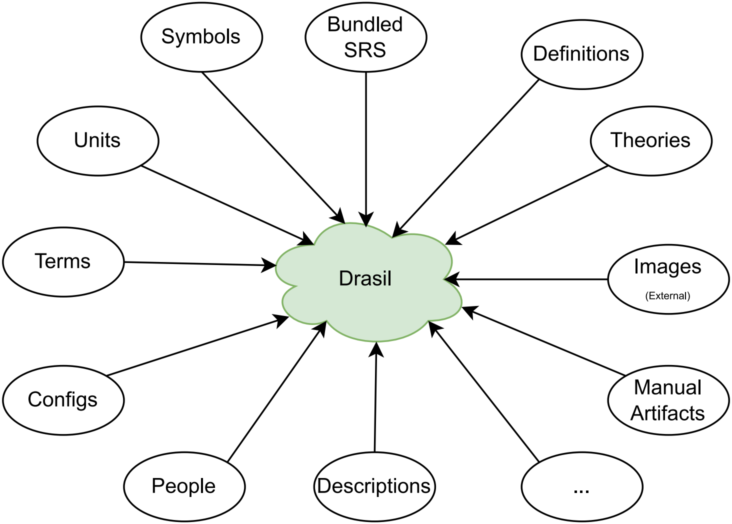
Note: not all type rules shown here.

Bigger picture

1. Found many typing issues!
2. Realized we needed more `Expr` terms.

Table of Contents

How is data stored in Drasil?



How is data stored in Drasil?

```haskell
data ChunkDB = CDB
  { symbolTable :: SymbolMap
  , termTable :: TermMap
  , defTable  :: ConceptMap
  , _unitTable :: UnitMap
  , _traceTable :: TraceMap
  , _refbyTable :: RefbyMap
  , _dataDefnTable  :: DatadefnMap
  , _insmodelTable   :: InsModelMap
  , _gendefTable   :: GendefMap
  , _theoryModelTable :: TheoryModelMap
  , _conceptinsTable :: ConceptInstanceMap
  , _sectionTable :: SectionMap
  , _labelledcontentTable :: LabelledContentMap
  } --TODO: Expand and add more databases
```

Scaling against new *kinds* of data

Mask the type information!

```haskell
{-# LANGUAGE ExistentialQuantification, ConstraintKinds #-}

type IsChunk a = (HasUID a, HasChunkRefs a, Typeable a)

data Chunk = forall a. IsChunk a => Chunk a

type ChunkDB = Map UID Chunk

unChunk :: Typeable a => Chunk -> Maybe a
unChunk (Chunk c) = cast c
```

Scaling against new *kinds* of data

Extra machinery

```haskell
type ReferredBy = [UID]

type ChunkByUID = M.Map UID (Chunk, ReferredBy)

type ChunksByTypeRep = M.Map TypeRep [Chunk]

newtype ChunkDB = ChunkDB (ChunkByUID, ChunksByTypeRep)
```

Bigger picture

1. One place!
2. Simpler chunk analysis:
   1. type usage analytics
   2. build chunk dependency tree
   3. find cyclic knowledge
3. ChunkDB manipulation
4. Usable "base" across Drasil-like projects

Table of Contents

Future Work

Regarding...

1. Theories:
   1. Explore alternative `ModelKinds` design.
   2. Explore remaining theories (e.g., structure `OthModels`).
2. Expressions:
   1. Add typing to `ModelExpr`, `CodeExpr`, and GOOL.
   2. For vectors and matrices in `Expr`, add length information, where applicable, to type-checker.
3. `ChunkDB`:
   1. Adjust existing chunk schema to merge in new `ChunkDB` implementation.
   2. Add analysis capabilities.

Table of Contents

Concluding Remarks & Takeaways

To sum up, we...

1. structured theories,
2. restricted expressions by context,
3. added type-checking to expressions,
4. and prototyped an extensible chunk database structure.