First Committee Meeting

Progress Report

Jason Balaci

McMaster University

Oct. 21^{st}, 2021

Table of Contents

Table of Contents

Who am I?

Who am I?

- I am **Jason Balaci**

Me, Camping in Killarney Prov. Park, Fall 2019

Who am I?

- I am **Jason Balaci**
- Graduate of *McMaster University*, holding...

Me, Camping in Killarney Prov. Park, Fall 2019

Who am I?

- I am **Jason Balaci**
- Graduate of *McMaster University*, holding...
  - Hons. Actuarial and Financial Mathematics (B.Sc.)

Me, Camping in Killarney Prov. Park, Fall 2019

Who am I?

- I am **Jason Balaci**
- Graduate of *McMaster University*, holding...
  - Hons. Actuarial and Financial Mathematics (B.Sc.)
  - Minor in Computer Science

Me, Camping in Killarney Prov. Park, Fall 2019

Who am I?

- I am **Jason Balaci**
- Graduate of *McMaster University*, holding...
  - Hons. Actuarial and Financial Mathematics (B.Sc.)
  - Minor in Computer Science
- Currently pursuing a thesis-based Master's of Computer Science (M.Sc) at *McMaster University*, under the supervision of **Dr. Jacques Carette**.

Me, Camping in Killarney Prov. Park, Fall 2019

- I'm required to complete[1][2]:

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

Overview of Progression Towards C.S. M.Sc.

Course-related progression

- I'm required to complete[1,2]:
    - One (1) "Software" course

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

Overview of Progression Towards C.S. M.Sc.

Course-related progression

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:

---

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

Overview of Progression Towards C.S. M.Sc.

Course-related progression

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:
  - CAS 701 "Logic & Discrete Mathematics" - Theory course, Fall 2020

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:
  - CAS 701 "Logic & Discrete Mathematics" - Theory course, Fall 2020
  - CAS 761 "Generative Programming" - Software course, Fall 2020

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:
  - CAS 701 "Logic & Discrete Mathematics" - Theory course, Fall 2020
  - CAS 761 "Generative Programming" - Software course, Fall 2020
  - CAS 763 "Certified Programming with Dependent Types" - Theory & Software course, Winter 2021

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

Overview of Progression Towards C.S. M.Sc.

Course-related progression

- I'm required to complete[1][2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:
  - CAS 701 "Logic & Discrete Mathematics" - Theory course, Fall 2020
  - CAS 761 "Generative Programming" - Software course, Fall 2020
  - CAS 763 "Certified Programming with Dependent Types" - Theory & Software course, Winter 2021
  - COMPSCI 6TB3 "Syntax-Based Tools and Compilers" - Systems course, Winter 2021

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- I'm required to complete[1,2]:
  - One (1) "Software" course
  - Either of:
    - Two (2) "Theory" courses, and one (1)"Systems" course
    - One (1) "Theory" course, and two (2) "Systems" courses
- I've completed:
  - CAS 701 "Logic & Discrete Mathematics" - Theory course, Fall 2020
  - CAS 761 "Generative Programming" - Software course, Fall 2020
  - CAS 763 "Certified Programming with Dependent Types" - Theory & Software course, Winter 2021
  - COMPSCI 6TB3 "Syntax-Based Tools and Compilers" - Systems course, Winter 2021
- Together, the courses completed satisfies the "Courses Requirement" as mentioned in the academic calendar[1] and the "Regulations for the Computer Science M.Sc. Program" document[2].

[1] https://academiccalendars.romcmaster.ca/preview_program.php?catoid=45&poid=23470&returnto=9166

[2] http://www.cas.mcmaster.ca/cas/0files/reg_master_cs_2019a.pdf

- Conducted “full-time” research for at least 1 full semester (Spring/Summer 2021), and “part-time” research during courses.

Overview of Progression Towards C.S. M.Sc.

Thesis/research-related Progression

- Conducted "full-time" research for at least 1 full semester (Spring/Summer 2021), and "part-time" research during courses.
- Continuing to research "full-time".

Overview of Progression Towards C.S. M.Sc.

Thesis/research-related Progression

- Conducted “full-time” research for at least 1 full semester (Spring/Summer 2021), and “part-time” research during courses.
- Continuing to research “full-time”.
- Attended a thesis defence to learn about what to expect from a thesis defence (and learn about their research).

Overview of Progression Towards C.S. M.Sc.

Thesis/research-related Progression

- Conducted "full-time" research for at least 1 full semester (Spring/Summer 2021), and "part-time" research during courses.
- Continuing to research "full-time".
- Attended a thesis defence to learn about what to expect from a thesis defence (and learn about their research).
- Supervisory committee is formed, and we are currently having our first supervisory committee meeting.
    - *Supervisor*: Dr. Jacques Carette
    - Dr. Spencer Smith
    - Dr. Wolfram Kahl

Table of Contents

Preface

What is Drasil?

Drasil...

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Drasil...

- is managed by Dr. Carette & Dr. Smith.

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.
    - Originally focused on scientific software (*Literate Scientific Software*).

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Preface

What is Drasil?

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.
    - Originally focused on scientific software (*Literate Scientific Software*).
    - Focus expanded...

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Preface

What is Drasil?

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.
  - Originally focused on scientific software (*Literate Scientific Software*).
  - Focus expanded...
- tries to "Generate All The Things"...

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Preface

What is Drasil?

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.
  - Originally focused on scientific software (*Literate Scientific Software*).
  - Focus expanded...
- tries to "Generate All The Things"...
  - with a focus on research software.

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Drasil...

- is managed by Dr. Carette & Dr. Smith.
- originates from the work of Dan Szymczak.
  - Originally focused on scientific software (*Literate Scientific Software*).
  - Focus expanded...
- tries to "Generate All The Things"...
  - with a focus on research software.
- has a website[1]!

Drasil's Logo
[Carette et al., 2021b][Yggdrasil - Wikipedia, 2021]

[1] https://jacquescarette.github.io/Drasil/

Drasil

"Generate All The Things!"

Drasil

"Generate All The Things!"

- An exploration in software-related artifact generation for "well understood" domains [Carette et al., 2021a] through strong knowledge capture.

Drasil

"Generate All The Things!"

- An exploration in software-related artifact generation for "well understood" domains [Carette et al., 2021a] through strong knowledge capture.
  - By unifying knowledge into a single framework with reusable composable units of knowledge, we eliminate code duplication, formally impose traceability and maintainability of knowledge (and software), and allow for easy knowledge transference.

Drasil

"Generate All The Things!"

- An exploration in software-related artifact generation for "well understood" domains [Carette et al., 2021a] through strong knowledge capture.
  - By unifying knowledge into a single framework with reusable composable units of knowledge, we eliminate code duplication, formally impose traceability and maintainability of knowledge (and software), and allow for easy knowledge transference.
  - Knowledge organization and capture is of utmost importance, as it is the pathway for interpreters and Domain-Specific Languages (DSLs) to make appropriate usage of the knowledge captured.

Drasil

"Generate All The Things!"

- An exploration in software-related artifact generation for "well understood" domains [Carette et al., 2021a] through strong knowledge capture.
  - By unifying knowledge into a single framework with reusable composable units of knowledge, we eliminate code duplication, formally impose traceability and maintainability of knowledge (and software), and allow for easy knowledge transference.
  - Knowledge organization and capture is of utmost importance, as it is the pathway for interpreters and Domain-Specific Languages (DSLs) to make appropriate usage of the knowledge captured.
  - By creating different kinds of "printers", we can use a stable knowledge-base to generate software that solves "well understood" problems.

Drasil

"Generate All The Things!"

- An exploration in software-related artifact generation for "well understood" domains [Carette et al., 2021a] through strong knowledge capture.
    - By unifying knowledge into a single framework with reusable composable units of knowledge, we eliminate code duplication, formally impose traceability and maintainability of knowledge (and software), and allow for easy knowledge transference.
    - Knowledge organization and capture is of utmost importance, as it is the pathway for interpreters and Domain-Specific Languages (DSLs) to make appropriate usage of the knowledge captured.
    - By creating different kinds of "printers", we can use a stable knowledge-base to generate software that solves "well understood" problems.
- Drasil currently focuses on building research software, generating Software Requirement Specification documents (SRS) in both LaTeX and HTML (with MathJaX), code to solve a problem, README files, Makefiles, graphs, etc.

Drasil Case Studies

[1] https://jacquescarette.github.io/Drasil/#Sec:Examples

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
  - **Single Pendulum** - Observing the motion of a single pendulum.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
  - **Single Pendulum** - Observing the motion of a single pendulum.
  - **Double Pendulum** - Observing the motion of a double pendulum.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
  - **Single Pendulum** - Observing the motion of a single pendulum.
  - **Double Pendulum** - Observing the motion of a double pendulum.
  - **Game Physics** - Modelling of an open source 2D rigid body physics library used for games.

[1] https://jacquescarette.github.io/Drasil/#Sec:Examples

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
  - **Single Pendulum** - Observing the motion of a single pendulum.
  - **Double Pendulum** - Observing the motion of a double pendulum.
  - **Game Physics** - Modelling of an open source 2D rigid body physics library used for games.
  - **Proportional Derivative Controller (PDController)** - Examining the output of a "Power Plant" (Process Variable) over time.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- Drasil currently contains a significant amount of Physics-related knowledge.
- As of writing, current case studies[1] are primarily related to physics, including:
  - **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
  - **Single Pendulum** - Observing the motion of a single pendulum.
  - **Double Pendulum** - Observing the motion of a double pendulum.
  - **Game Physics** - Modelling of an open source 2D rigid body physics library used for games.
  - **Proportional Derivative Controller (PDController)** - Examining the output of a "Power Plant" (Process Variable) over time.
  - **Solar Water Heating System (SWHS)** - Modelling of a solar water heating system with phase change material, predicting temperatures and change in heat energy of water and the PCM over time.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- *cont.d*[1]:
  - **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

- *cont.d*[1]:
  - **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.
  - **Projectile** - Determining if a launched projectile hits a target, assuming no flight collisions.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- *cont.d*[1]:
  - **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.
  - **Projectile** - Determining if a launched projectile hits a target, assuming no flight collisions.
  - **Slope Stability Analysis Program (SSP)** - Assessment of the safety of a slope (composed of rock and soil) subject to gravity, identifying the surface most likely to experience slip and an index of its relative stability (factor of safety).

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

- *cont.d*[1]:
  - **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.
  - **Projectile** - Determining if a launched projectile hits a target, assuming no flight collisions.
  - **Slope Stability Analysis Program (SSP)** - Assessment of the safety of a slope (composed of rock and soil) subject to gravity, identifying the surface most likely to experience slip and an index of its relative stability (factor of safety).
  - **Heat Transfer Coefficients between Fuel and Cladding in Fuel Rods (HGHC)** - Examining the heat transfer coefficients related to clad.

[1] `https://jacquescarette.github.io/Drasil/#Sec:Examples`

Drasil Case Studies

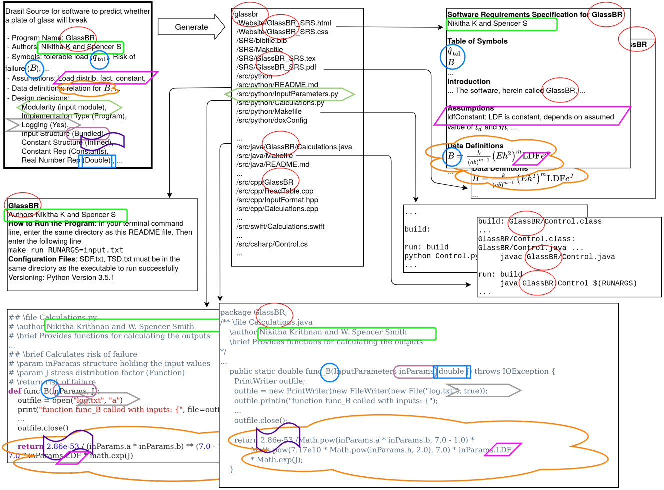
- *cont.d*[1]:
  - **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.
  - **Projectile** - Determining if a launched projectile hits a target, assuming no flight collisions.
  - **Slope Stability Analysis Program (SSP)** - Assessment of the safety of a slope (composed of rock and soil) subject to gravity, identifying the surface most likely to experience slip and an index of its relative stability (factor of safety).
  - **Heat Transfer Coefficients between Fuel and Cladding in Fuel Rods (HGHC)** - Examining the heat transfer coefficients related to clad.

The Drasil website is also generated by Drasil!

[1] https://jacquescarette.github.io/Drasil/#Sec:Examples

Taking a closer look at one of the examples: GlassBR

GlassBR Generates Code!



Knowledge flow from "knowledge-base"/source to artifacts, by Dr. Spencer Smith

Which case studies currently generate code?

Which case studies currently generate code?

- **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.

Which case studies currently generate code?

- **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
- **Proportional Derivative Controller (PDController)** - Examining the output of a "Power Plant" (Process Variable) over time.

Which case studies currently generate code?

- **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
- **Proportional Derivative Controller (PDController)** - Examining the output of a "Power Plant" (Process Variable) over time.
- **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.

Which case studies currently generate code?

- **GlassBR** - Predicting whether or not a glass slab is likely to resist a specified blast.
- **Proportional Derivative Controller (PDController)** - Examining the output of a "Power Plant" (Process Variable) over time.
- **SWHS without Phase Change Material (NoPCM)** - Modelling of a solar water heating system without phase change material, predicting temperatures and change in heat energy of water and the PCM over time.
- **Projectile** - Determining if a launched projectile hits a target, assuming no flight collisions.

Why don't all case studies generate software artifacts?

Where will I be contributing?

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?

Where will I be contributing?

After all,

[1] Terminology is currently being changed, but is not reflected in many documents yet.

After all,

- They're all covered under "well understood" domains!

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?

Where will I be contributing?

After all,

- They're all covered under "well understood" domains!
- The SRS documents are generated!

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?

Where will I be contributing?

After all,

- They're all covered under "well understood" domains!
- The SRS documents are generated!

Generating view-only data (e.g., SRS documents) is considerably easier than generating artifacts that are "evaluated" in some way or another (e.g., compilation/interpretation/static analysis). These are largely different "printers".

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?
Where will I be contributing?

After all,

- They're all covered under "well understood" domains!
- The SRS documents are generated!

Generating view-only data (e.g., SRS documents) is considerably easier than generating artifacts that are "evaluated" in some way or another (e.g., compilation/interpretation/static analysis). These are largely different "printers".

A few, notable, blocking problems:

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?

Where will I be contributing?

After all,

- They're all covered under "well understood" domains!
- The SRS documents are generated!

Generating view-only data (e.g., SRS documents) is considerably easier than generating artifacts that are "evaluated" in some way or another (e.g., compilation/interpretation/static analysis). These are largely different "printers".

A few, notable, blocking problems:

- Confidently generating usable software artifacts without strong type information places significant stress on developers, resulting in a higher likelihood of bugs in artifacts.

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Why don't all case studies generate software artifacts?

Where will I be contributing?

After all,

- They're all covered under "well understood" domains!
- The SRS documents are generated!

Generating view-only data (e.g., SRS documents) is considerably easier than generating artifacts that are "evaluated" in some way or another (e.g., compilation/interpretation/static analysis). These are largely different "printers".

A few, notable, blocking problems:

- Confidently generating usable software artifacts without strong type information places significant stress on developers, resulting in a higher likelihood of bugs in artifacts.
- Existing "theories"/"*Models"[1] don't expose enough information. They must be enriched, so that we can better interact with, and understand them.

[1] Terminology is currently being changed, but is not reflected in many documents yet.

Goal #1: Typed Expression Language

Problem Description

Goal #1: Typed Expression Language

Problem Description

- Ensure only admissible expressions are used in GOOL-supported languages, and that all expressions are coherent.

Goal #1: Typed Expression Language

Problem Description

- Ensure only admissible expressions are used in GOOL-supported languages, and that all expressions are coherent.
- We want to ease developer cognitive load when writing expressions, as they will need to ensure their expressions are coherent, or else various problems (type, syntax, etc) can occur at runtime (of generated software artifacts).

Goal #1: Typed Expression Language

What makes up a "good" solution?

Goal #1: Typed Expression Language

What makes up a "good" solution?

- Catches, within reason, all possible scenarios where an expression goes awry.

Goal #1: Typed Expression Language

What makes up a "good" solution?

- Catches, within reason, all possible scenarios where an expression goes awry.
- Allows GOOL code generator to also become typed!

Goal #1: Typed Expression Language

What makes up a "good" solution?

- Catches, within reason, all possible scenarios where an expression goes awry.
- Allows GOOL code generator to also become typed!
- Add extra functionality to existing expression languages safely, allowing for new data types to be introduced.

Goal #1: Typed Expression Language

What makes up a "good" solution?

- Catches, within reason, all possible scenarios where an expression goes awry.
- Allows GOOL code generator to also become typed!
- Add extra functionality to existing expression languages safely, allowing for new data types to be introduced.
- Adding type information to expressions shouldn't be a burden!

Goal #1: Typed Expression Language

What makes up a "good" solution?

- Catches, within reason, all possible scenarios where an expression goes awry.
- Allows GOOL code generator to also become typed!
- Add extra functionality to existing expression languages safely, allowing for new data types to be introduced.
- Adding type information to expressions shouldn't be a burden!
- Decomposing/splitting vocabularies so that we can impose restrictions on allowed terms, while not causing problems for interoperability.

Goal #1: Typed Expression Language

Current Progression

Goal #1: Typed Expression Language

Current Progression

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)
  - Expr is intended to be "directly computable" language (e.g., an advanced calculator), expected to have a *total* conversion into code.

Goal #1: Typed Expression Language

Current Progression

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)
  - Expr is intended to be "directly computable" language (e.g., an advanced calculator), expected to have a *total* conversion into code.
  - Created ModelExpr, which contains all other terms we might want to express, but won't necessarily be directly convertible into code. There are still a few operations left in Expr that need to be moved over, however.

Goal #1: Typed Expression Language

Current Progression

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)
  - Expr is intended to be "directly computable" language (e.g., an advanced calculator), expected to have a *total* conversion into code.
  - Created ModelExpr, which contains all other terms we might want to express, but won't necessarily be directly convertible into code. There are still a few operations left in Expr that need to be moved over, however.
    - Theories that rely on discussion of terms only found in ModelExpr may only have representational code generated if we have rich surrounding data (preferably also that generates said "ModelExpr"s, see goal #2).

Goal #1: Typed Expression Language

Current Progression

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)
  - Expr is intended to be "directly computable" language (e.g., an advanced calculator), expected to have a *total* conversion into code.
  - Created ModelExpr, which contains all other terms we might want to express, but won't necessarily be directly convertible into code. There are still a few operations left in Expr that need to be moved over, however.
    - Theories that rely on discussion of terms only found in ModelExpr may only have representational code generated if we have rich surrounding data (preferably also that generates said "ModelExpr"s, see goal #2).
  - CodeExpr is a clone of Expr, with a few extra functionalities for GOOL.

Goal #1: Typed Expression Language

Current Progression

- Split core mathematical expression language (Expr) into 3 variants (Expr, ModelExpr, and CodeExpr)
  - Expr is intended to be "directly computable" language (e.g., an advanced calculator), expected to have a *total* conversion into code.
  - Created ModelExpr, which contains all other terms we might want to express, but won't necessarily be directly convertible into code. There are still a few operations left in Expr that need to be moved over, however.
    - Theories that rely on discussion of terms only found in ModelExpr may only have representational code generated if we have rich surrounding data (preferably also that generates said "ModelExpr"s, see goal #2).
  - CodeExpr is a clone of Expr, with a few extra functionalities for GOOL.
  - Created a "typed tagless final" [Carette et al., 2009] smart constructor encoding for writing expressions in Expr, and/or ModelExpr.

Goal #1: Typed Expression Language

Current Progression

```haskell
offset' :: PExpr
offset' = sy landPos $- sy targPos

message :: PExpr
message = completeCase [case1, case2, case3]
  where case1 = (str "The target was hit.",        abs_ (sy offset $/ sy targPos) $< sy tol)
        case2 = (str "The projectile fell short.", sy offset $< exactDbl 0)
        case3 = (str "The projectile went long.",  sy offset $> exactDbl 0)

--
speed' :: PExpr
speed' = sy QP.iSpeed `addRe` (sy QP.constAccel `mulRe` sy time)

scalarPos' :: PExpr
scalarPos' = sy iPos `addRe` (sy QP.iSpeed `mulRe` sy time `addRe` half (sy QP.constAccel `mulRe` square (sy time)))
```

Polymorphic Expressions can become Exprs or ModelExprs

Goal #1: Typed Expression Language

Current Progression

```haskell
rectPosDerivEqn1, rectPosDerivEqn2, rectPosDerivEqn3 :: ModelExpr
rectPosDerivEqn1 = sy speed $= deriv (sy scalarPos) time
rectPosDerivEqn2 = defint (eqSymb scalarPos) (sy iPos) (sy scalarPos) (exactDbl 1) $=
                   defint (eqSymb time) (exactDbl 0) (sy time) (sy speed)
rectPosDerivEqn3 = defint (eqSymb scalarPos) (sy iPos) (sy scalarPos) (exactDbl 1) $=
                   defint (eqSymb time) (exactDbl 0) (sy time) (sy QP.iSpeed `addRe` (sy QP.constAccel `mulRe` sy time))
```

Derivation Expressions, using terms only available in ModelExpr

Goal #1: Typed Expression Language

What are the next steps?

Goal #1: Typed Expression Language

What are the next steps?

- Continue moving inadmissible terms from Expr into ModelExpr.

Goal #1: Typed Expression Language

What are the next steps?

- Continue moving inadmissible terms from Expr into ModelExpr.
- Moving literals from Expr & ModelExpr into their own small language, so that areas that want *strictly* literals can also have stronger restrictions.

Goal #1: Typed Expression Language

What are the next steps?

- Continue moving inadmissible terms from Expr into ModelExpr.
- Moving literals from Expr & ModelExpr into their own small language, so that areas that want *strictly* literals can also have stronger restrictions.
- Adjusting containers to allow for expressions with a type variable.

Goal #1: Typed Expression Language

What are the next steps?

- Continue moving inadmissible terms from Expr into ModelExpr.
- Moving literals from Expr & ModelExpr into their own small language, so that areas that want *strictly* literals can also have stronger restrictions.
- Adjusting containers to allow for expressions with a type variable.
- Adding the final type signatures, using Haskell GADT syntax.

Assume we have a general concept consisting of $y = m \cdot x + b$, and a natural description "The equation of a line, with a constant, b, and a dependent and independent variable, y and x, respectively.".

Goal #2: Theory Discrimination – "ModelKinds"

Problem Description

Assume we have a general concept consisting of $y = m \cdot x + b$, and a natural description "The equation of a line, with a constant, b, and a dependent and independent variable, y and x, respectively.".

- With basic analysis of the formation of the equation, generating code to represent a calculation of "y", given "x", should be straightforward.

Goal #2: Theory Discrimination – "ModelKinds"

Problem Description

Assume we have a general concept consisting of $y = m \cdot x + b$, and a natural description "The equation of a line, with a constant, b, and a dependent and independent variable, y and x, respectively.".

- With basic analysis of the formation of the equation, generating code to represent a calculation of "y", given "x", should be straightforward.
- But if the equation were written slightly different (e.g. $y - b = m \cdot x$), generating code based on this "Relation"/Expr is significantly more tedious because we will need to solve for "y".

Assume we have a general concept consisting of $y = m \cdot x + b$, and a natural description "The equation of a line, with a constant, b, and a dependent and independent variable, y and x, respectively.".

- With basic analysis of the formation of the equation, generating code to represent a calculation of "y", given "x", should be straightforward.
- But if the equation were written slightly different (e.g. $y - b = m \cdot x$), generating code based on this "Relation"/Expr is significantly more tedious because we will need to solve for "y".
- What if we want to change the symbols used? How do we know the "b" referred to in the equation is the same as the symbol "b" in the natural description?

Goal #2: Theory Discrimination – “ModelKinds”

Problem Description

Assume we have a general concept consisting of $y = m \cdot x + b$, and a natural description “The equation of a line, with a constant, b, and a dependent and independent variable, y and x, respectively.”.

- With basic analysis of the formation of the equation, generating code to represent a calculation of “y”, given “x”, should be straightforward.
- But if the equation were written slightly different (e.g. $y - b = m \cdot x$), generating code based on this “Relation”/Expr is significantly more tedious because we will need to solve for “y”.
- What if we want to change the symbols used? How do we know the “b” referred to in the equation is the same as the symbol “b” in the natural description?
- We’re ignoring the obvious information that this is the equation of a line, and that this is just an alternative “Definition”.

Goal #2: Theory Discrimination – "ModelKinds"

Problem Description, *cont.d*

- "RelationConcepts" were heavily used in both displaying expressions, and code generation. They are essentially "Relation"s ("Expr"s) with a natural language description of them.

Goal #2: Theory Discrimination – “ModelKinds”

Problem Description, *cont.d*

- “RelationConcepts” were heavily used in both displaying expressions, and code generation. They are essentially “Relation”s (“Expr”s) with a natural language description of them.
- “RelationConcept”s don’t contain enough information on their own to be a core component usable in general code generation.

- "RelationConcepts" were heavily used in both displaying expressions, and code generation. They are essentially "Relation"s ("Expr"s) with a natural language description of them.
- "RelationConcept"s don't contain enough information on their own to be a core component usable in general code generation.
- If the "shape" of the expressions are not uniform, then writing more "interpreters"/"views"/code generators for them required difficult pattern analysis. It's also not a total-conversion.

Goal #2: Theory Discrimination – "ModelKinds"

What makes up a "good" solution?

Goal #2: Theory Discrimination – "ModelKinds"

What makes up a "good" solution?

- A good solution involves making the "Relation"s a "view" of a more data-rich specialized container for each kind of "Theory"/"*Model".

Goal #2: Theory Discrimination – "ModelKinds"

What makes up a "good" solution?

- A good solution involves making the "Relation"s a "view" of a more data-rich specialized container for each kind of "Theory"/"*Model".
- "ModelKinds"

Goal #2: Theory Discrimination – "ModelKinds"

What makes up a "good" solution?

- A good solution involves making the "Relation"s a "view" of a more data-rich specialized container for each kind of "Theory"/"*Model".
- "ModelKinds"
- By constructing our final data views through "more steps" (e.g., with more depth), we obtain a better understanding of our "theories"/"*Model"s/"ModelKinds", allowing us to do more with them.

Goal #2: Theory Discrimination – "ModelKinds"

What makes up a "good" solution?

- A good solution involves making the "Relation"s a "view" of a more data-rich specialized container for each kind of "Theory"/"*Model".
- "ModelKinds"
- By constructing our final data views through "more steps" (e.g., with more depth), we obtain a better understanding of our "theories"/"*Model"s/"ModelKinds", allowing us to do more with them.
- We should be able to easily add extra "ModelKind" variants.

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s

Goal #2: Theory Discrimination – “ModelKinds”

Current Progression

- All “RelationConcepts” have been replaced, with one of:
  - EquationalModels: “QDefinition”s
  - EquationalRealms: “MultiDefn”s

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s
  - EquationalRealms: "MultiDefn"s
  - EquationalConstraints: "ConstraintSet"s

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s
  - EquationalRealms: "MultiDefn"s
  - EquationalConstraints: "ConstraintSet"s
  - DEModels: "RelationConcept"s

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s
  - EquationalRealms: "MultiDefn"s
  - EquationalConstraints: "ConstraintSet"s
  - DEModels: "RelationConcept"s
  - OthModels: "RelationConcept"s

Goal #2: Theory Discrimination – "ModelKinds"

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s
  - EquationalRealms: "MultiDefn"s
  - EquationalConstraints: "ConstraintSet"s
  - DEModels: "RelationConcept"s
  - OthModels: "RelationConcept"s
- Considerable number of "theories"/"*Models" have been restructured, but there are still many that are pending classification.

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

- All "RelationConcepts" have been replaced, with one of:
  - EquationalModels: "QDefinition"s
  - EquationalRealms: "MultiDefn"s
  - EquationalConstraints: "ConstraintSet"s
  - DEModels: "RelationConcept"s
  - OthModels: "RelationConcept"s
- Considerable number of "theories"/"*Models" have been restructured, but there are still many that are pending classification.
  - Most are best to be done once we have a typed expression language (so that we can better handle expressions that involve collections), and the rest are differential equation-related models (primarily Dong's domain).

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

```haskell
data QDefinition e where
  QD :: Express e => DefinedQuantityDict -> [UID] -> e -> QDefinition e
```

Current QDefinitions

```haskell
-- | 'DefiningExpr' are the data that make up a (quantity) definition, namely
--   the description, the defining (rhs) expression and the context domain(s).
--   These are meant to be 'alternate' but equivalent definitions for a single concept.
data DefiningExpr e = DefiningExpr {
  _deUid  :: UID,             -- ^ UID
  _cd     :: [UID],           -- ^ Concept domain
  _rvDesc :: Sentence,        -- ^ Defining description/statement
  _expr   :: Express e => e   -- ^ Defining expression
}
makeLenses ''DefiningExpr

-- | 'MultiDefn's are QDefinition factories, used for showing one or more ways we
--   can define a QDefinition.
data MultiDefn e = MultiDefn {
    _rUid  :: UID,                                       -- ^ UID
    _qd    :: QuantityDict,                              -- ^ Underlying quantity it defines
    _rDesc :: Sentence,                                  -- ^ Defining description/statement
    _rvs   :: Express e => NE.NonEmpty (DefiningExpr e) -- ^ All possible/omitted ways we can define the related quantity
}
makeLenses ''MultiDefn
```

Current MultiDefns

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

```haskell
-- | Models can be of different kinds:
--
--     * 'DEModel's represent differential equations as 'RelationConcept's
--     * 'EquationalConstraint's represent invariants that will hold in a system of equations.
--     * 'EquationalModel's represent quantities that are calculated via a single definition/'QDefinition'.
--     * 'EquationalRealm's represent MultiDefns; quantities that may be calculated using any one of many 'DefiningExpr's (e.g., 'x = A = ... = Z')
--     * 'FunctionalModel's represent quantity-resulting function definitions.
--     * 'OthModel's are placeholders for models. No new 'OthModel's should be created, they should be using one of the other kinds.
data ModelKinds e where
  DEModel              :: Express e => RelationConcept -> ModelKinds e
  EquationalConstraints :: Express e => ConstraintSet e -> ModelKinds e
  EquationalModel      :: Express e => QDefinition e   -> ModelKinds e
  EquationalRealm      :: Express e => MultiDefn e     -> ModelKinds e
  OthModel             :: Express e => RelationConcept -> ModelKinds e -- TODO: Remove (after having removed all instances of it).
```

Current ModelKinds

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

```haskell
-- | An instance model is a ModelKind that may have specific inputs, outputs, and output
-- constraints. It also has attributes like references, derivation, labels ('ShortName'), reference address, and notes.
data InstanceModel = IM { _mk       :: ModelKind Expr
                        , _imInputs :: Inputs
                        , _imOutput :: (Output, OutputConstraints)
                        , _rf       :: [DecRef]
                        , _deri     :: Maybe Derivation
                        ,  lb       :: ShortName
                        ,  ra       :: String
                        , _notes    :: [Sentence]
                        }
makeLenses ''InstanceModel
```

Current InstanceModel

Goal #2: Theory Discrimination – "ModelKinds"

Current Progression

```haskell
-- | A general definition is a 'ModelKind' that may have units, a derivation,
-- references (as 'DecRef's), a shortname, a reference address, and notes.
data GenDefn = GD { _mk     :: ModelKind ModelExpr
                  , gdUnit :: Maybe UnitDefn
                  , _deri   :: Maybe Derivation
                  , _rf     :: [DecRef]
                  , _sn     :: ShortName
                  , _ra     :: String -- RefAddr
                  , _notes  :: [Sentence]
                  }
makeLenses ''GenDefn
```

Current General Definitions

Goal #2: Theory Discrimination – “ModelKinds”

Current Progression

```haskell
data TheoryModel = TM
  { _mk     :: ModelKind ModelExpr
  , _vctx   :: [TheoryModel]
  , _spc    :: [SpaceDefn]
  , _quan   :: [QuantityDict]
  , _ops    :: [ConceptChunk]
  , _defq   :: [ModelQDef]
  , _invs   :: [ModelExpr]
  , _dfun   :: [ModelQDef]
  , _rf     :: [DecRef]
  ,  lb     :: ShortName
  ,  ra     :: String
  , _notes  :: [Sentence]
  }
makeLenses ''TheoryModel
```

Current TheoryModels

Goal #2: Theory Discrimination – "ModelKinds"

What are the next steps?

Goal #2: Theory Discrimination – "ModelKinds"

What are the next steps?

- Understanding what kinds of needs we have for "collections", pushing this information back into the typed expression language (once that is fully typed), and then creating model containers for these models.

Goal #2: Theory Discrimination – "ModelKinds"

What are the next steps?

- Understanding what kinds of needs we have for "collections", pushing this information back into the typed expression language (once that is fully typed), and then creating model containers for these models.
- For the differential equation-related models, we will need to build appropriate models for each, as necessary.

Goal #2: Theory Discrimination – "ModelKinds"

What are the next steps?

- Understanding what kinds of needs we have for "collections", pushing this information back into the typed expression language (once that is fully typed), and then creating model containers for these models.
- For the differential equation-related models, we will need to build appropriate models for each, as necessary.
- Cleaning up the existing implementation (removing the expression type parameter), and understanding where we want to go from here.

Goal #2: Theory Discrimination – "ModelKinds"

What are the next steps?

- Understanding what kinds of needs we have for "collections", pushing this information back into the typed expression language (once that is fully typed), and then creating model containers for these models.
- For the differential equation-related models, we will need to build appropriate models for each, as necessary.
- Cleaning up the existing implementation (removing the expression type parameter), and understanding where we want to go from here.
- A current proposal is pending further discussion in GitHub Issue #2853, potentially rebuilds ModelKinds to be extensible by using type constraints instead of GADTs.

Acknowledgements

Acknowledgements

- Both goals are as designated by Dr. Carette, Dr. Smith, and past (and present) Drasil authors.

Acknowledgements

- Both goals are as designated by Dr. Carette, Dr. Smith, and past (and present) Drasil authors.
- "ModelKinds" is based on Dr. Carette's implementation.

Acknowledgements

- Both goals are as designated by Dr. Carette, Dr. Smith, and past (and present) Drasil authors.
- "ModelKinds" is based on Dr. Carette's implementation.
- Dr. Smith created the GlassBR figure earlier shown.

Acknowledgements

- Both goals are as designated by Dr. Carette, Dr. Smith, and past (and present) Drasil authors.
- "ModelKinds" is based on Dr. Carette's implementation.
- Dr. Smith created the GlassBR figure earlier shown.
- Drasil has had significant development by past (and present) authors. Their public notes in the issue tracker, and their works (in particular, that of Brooks' thesis) have been especially helpful in learning about Drasil.

Fin.

Thank you!

Table of Contents

References I

Carette, J., Kiselyov, O., and Shan, C.-c. (2009).
Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages.
Journal of Functional Programming, 19(5):509–543.

Carette, J., Smith, S., and Balaci, J. (2021a).
Pending publication: When capturing knowledge improves productivity.

Carette, J., Smith, S., Balaci, J., Hunt, A., Wu, T.-Y., Crawford, S., Chen, D., Szymczak, D., MacLachlan, B., Scime, D., and Niazi, M. (2021b).
Drasil.

Yggdrasil - Wikipedia (2021).
Yggdrasil.