CAS 703 Term Project: Validated General-Purpose Calculators

Hassan Zaker Jason Balaci

McMaster University
`{zakerzah, balacij}@mcmaster.ca`
April 8, 2023

Contents

1 Introduction

For our term project in CAS 703 [1], we decided to build a language for describing calculation schemes. The descriptions should be mostly understandable to anyone who has worked with Excel [2] or has used any kind of calculation software. We aim to validate the coherence[1] of the calculator descriptions. Additionally, through generative techniques, we hope to decrease the barrier to entry (as much as we can) of basic software development of calculator programs by defining a transformation of the calculator descriptions to various programming languages[2].

1.1 Objective

We aim to:

1. design a metamodel for describing calculator programs (Section 2),
2. build a concrete syntax for the metamodel, and an Integrated Development Environment (IDE) for said concrete syntax (Section 3),
3. design a set of rules that define "coherence" rules of the metamodel and audit instances of the metamodel for coherence (Section 4), and
4. define a transformer that converts the calculator description into programs and corresponding documentation (Section 5).

1.2 Tooling

We will use the tooling shown in 703, namely: Eclipse Epsilon [3] and the languages it contains, and Xtext [4].

[1]"Coherence" defined by an unambiguous set of constraints and rules.
[2]Notably, Java programs.

2 Modelling

To define our metamodel for our desired Domain-Specific Language (DSL), we first need to iron out the *requirements* of that language.

2.1 Requirements

When we (the authors) think of "calculators," we think of 3 main components: (i) a set of input symbols, (ii) a set of output symbols, and (iii) a means of deriving the output symbols from the input symbols. (i) and (ii) are really just "symbols" with designation of whether they are user-provided or calculated. (iii) is a (commonly secret) algorithm that translates (i) to (ii) and which may rely on intermediate symbols irrelevant to (i) and (ii). Thus, we think of a calculator as being a structure of (a) symbols of different kinds (inputs, intermediates, and outputs) and (b) an algorithm that calculates some symbols from an initial subset of provided symbols.

Depending on your preferred school of thought, algorithms can be thought of differently. We choose to think of them as being similar to imperative-style programming languages, where there is a memory of symbols to values, and a sequence of steps/statements that are executed sequentially against the memory.

The most important kind of "step" our algorithm must have is "symbol assignment," where some symbol is assigned the value of some expression. The expression may contain other symbols, but it should be restricted to only using symbols that have been assigned values (either as input to the calculator or by an earlier step). Logging is another important kind of step that algorithms might need, they're typically used for providing user-feedback for debugging, general information, warnings, errors, etc. Logging, similar to assignment steps, are provided an expression, for which are evaluated, but the difference is that the logging step displays the value to the user-feedback mechanism of the calculator rather than assigning the value to a symbol. For now, we choose to think of calculators as only having these 2 kinds of steps.

The steps rely on a single mathematical expression language that describes how values can be altered in a predictable way. We choose to restrict our expression language to the common first-order definite-valued mathematical language that most undergraduates in a first-year university mathematics class are familiar with.

Similar to how we expect our expression language to provide predictable results, we also expect our calculators to be predictable and reliably produce accurate results. To gain confidence in accuracy and predictability, we can test our calculators against well-known outputs for well-known inputs. In other words, we can create test cases to gain confidence that our calculator is reliable (which we want to reasonably assure ourselves).

Thus, together, we define our calculator as having 3 main aspects: (a) a set of symbols, (b) an algorithm that somehow calculates the output symbols from the inputs, and (c) a set of test cases to gain confidence that (b) is reliable. To model our calculator DSL, we chose to use Emfatic because we found it to be faster to work with. In particular, we found it to be more readable and easier to modify compared to Ecore.

2.2 Emfatic

3 Integrated Development Environment

4 Model Validation

5 Model Management Operations

6 Reflection and Concluding Thoughts

References

[1] Richard Paige. *CAS 703 - Software Design*. A course at McMaster University during the Winter 2023 semester. 2023 (cit. on p. 2).

[2] Microsoft Corporation. *Microsoft Excel*. 2023. URL: https://www.microsoft.com/en-ca/microsoft-365/excel (cit. on p. 2).

[3] Eclipse Foundation, Inc. *Eclipse Epsilon*. 2023. URL: https://github.com/eclipse/epsilon (cit. on p. 2).

[4] Eclipse Foundation, Inc. *Xtext*. 2023. URL: https://github.com/eclipse/xtext (cit. on p. 2).