Roadmap

Warrick Macmillan

February 12, 2021

1 Introduction

The central concern of this thesis is the syntax of mathematics, programming languages, and their respective mutual influence, as conceived and practiced by mathematicians and computer scientists. From one vantage point, the role of syntax in mathematics may be regarded as a 2nd order concern, a topic for discussion during a Fika, an artifact of ad hoc development by the working mathematician whose real goals are producing genuine mathematical knowledge. For the programmers and computer scientists, syntax may be regarding as a matter of taste, with friendly debates recurring regarding the use of semicolons, brackets, and white space. Yet, when viewed through the lens of the propositions-as-types paradigm, these discussions intersect in new and interesting ways. When one introduces a third paradigm through which to analyze the use of syntax in mathematics and programming, namely Linguistics, I propose what some may regard as superficial detail, indeed becomes a central paradigm, with many interesting and important questions.

To get a feel for this syntactic paradigm, let us look at a basic mathematical example: that of a group homomorphism, as expressed in a variety of sources.

Definition 1 *In mathematics, given two groups, $(G, *)$ and $(H, \cdot)$, a group homomorphism from $(G, *)$ to $(H, \cdot)$ is a function $h : G \to H$ such that for all u and v in G it holds that*

$$h(u * v) = h(u) \cdot h(v)$$

Definition 2 *Let $G = (G, \cdot)$ and $G' = (G', *)$ be groups, and let $\phi : G \to G'$ be a map between them. We call ϕ a* ***homomorphism*** *if for every pair of elements $g, h \in G$, we have*

$$\phi(g * h) = \phi(g) \cdot \phi(h)$$

Definition 3 *Let G, H, be groups. A map $\phi : G \to H$ is called a* group homomorphism *if*

$$\phi(xy) = \phi(x)\phi(y) \text{ for all } x, y \in G$$

(Note that xy on the left is formed using the group operation in G, whilst the product $\phi(x)\phi(y)$ is formed using the group operation H.)

Definition 4 *Classically, a group is a monoid in which every element has an inverse (necessarily unique).*

We inquire the reader to pay attention to nuance and difference in presentation that is normally ignored or taken for granted by the fluent mathematician.

If one want to distill the meaning of each of these presentations, there is a significant amount of subliminal interpretation happening very much analagous to our innate lingusitic ussage. The inverse and identity are discarded, even though they are necessary data when defning a group. The order of presentation of information is incostent, as well as the choice to use symbolic or natural language information. In (3), the group operation is used implicitly, and its clarification a side remark.

Details aside, these all mean the same thing–don't they? This thesis seeks to provide an abstract framework to determine whether two lingusitically nuanced presenations mean the same thing via their syntactic transformations.

These syntactic transformations come in two flavors : parsing and linearization, and are natively handled by a Logical Framework (LF) for specifying grammars : Grammatical Framework (GF).

We now show yet another definition of a group homomorphism formalized in the Agda programming language:

```
monoidHom : {ℓ : Level}
              → ((monoid' a _ _ _ _ _ _) (monoid' a' _ _ _ _ _ _) : Monoid' {ℓ} )
              → (a → a') → Type ℓ
monoidHom
  (monoid' A ε _•_ left-unit right-unit assoc carrier-set)
  (monoid' A₁ ε₁ _•₁_ left-unit₁ right-unit₁ assoc₁ carrier-set₁)
  f
  = (m1 m2 : A) → f (m1 • m2) ≡ (f m1) •₁ (f m2)
```

While the first three definitions above are should be linguistically comprehensible to a non-mathematician, this last definition is most certainly not. While may carry degree of comprehension to a programmer or mathematician not exposed to Agda, it is certainly comprehensible to a computer : that is, it typechecks on a computer where Cubical Agda is installed. While GF is designed for multilingual syntactic transformations and is targeted for natural language translation, it's underlying theory is largely based on ideas from the compiler communities. A cousin of the BNF Converter (BNFC), GF is fully capable of parsing progamming languages like Agda! And while the above definition is just another concrete syntactic presentation of a group homomorphism, it is distinct from the natural language presentations above in that the colors indicate it has indeed type checked.

While this example may not exemplify the power of Agda's type checker, it is of considerable interest to many. The typechecker has merely assured us that monoidHom, is a well-formed type. The type-checker is much more useful than is immediately evident: it delegates the work of verifying that a proof is correct, that is, the work of judging whether a term has a type, to the computer. While it's of practical concern is immediate to any exploited grad student grading papers late on a Sunday night, its theoretical concern has led to many recent developments in modern mathematics. Thomas Hales solution to the Kepler Conjecture was seen as unverifiable by those reviewing it. This led to Hales outsourcing the verification to Interactive Theorem provers HOL Light and Isabelle, during which led to many minor corrections in the original proof which were never spotted due to human oversight.

Fields Medalist Vladimir Voevodsky, had the experience of being told one day his proof of the Milnor conjecture was fatally flawed. Although the leak in the proof was patched, this experience of temporarily believing much of his life's work invalid led him to investigate proof assintants as a tool for future thought. Indeed, this proof verification error was a key event that led to the Univalent Foundations Project [1].

While Agda and other programming languages are capable of encoding definitions, theorems, and proofs, they have so far seen little adoption, and in some cases treated with suspicion and scorn by many mathematicians. This isn't entirely unfounded : it's a lot of work to learn how to use Agda or Coq, software updates may cause proofs to break, and the inevitable errors we humans are instilled in these Theorem Provers. And that's not to mention that Martin-Löf Type Theory, the constructive foundational project which underlies these proof assistants, is rejected by those who dogmatically accept the law of the excluded middle and ZFC as the word of God.

What these theorem provers give the mathematician is confidence that her work is correct, and even more importantly, that the work which she takes for granted and references in her work is also correct. The task before us is then one of religious conversion. And one doesn't undertake a conversion by simply by preaching. Foundational details aside, this thesis is meant to provide a blueprint for the syntactic reformation that must take place.

It doesn't ask the mathematician to relinquish the beautiful language she has come to love in expressing her ideas. Rather, it asks her to make a compromise for the time being, and use a Controlled Natural Language (CNL) to develop her work. In exchange she'll get the confidence that Agda provides. Not only that, she'll be able to search through a library, to see who else has possibly already postulated and proved her conjecture. This grandiose vision is not original, The Formal Abstracts Project

This may be a grandiose vision for the future of mathematics,

the computer scientist

undertake the c

It is therefore natural for this thesis, which seeks Here we must

```
module Id where

  data _≡_ {A : Set} (a : A) : A → Set where
    r : a ≡ a

  infix 20 _≡_

  J : {A : Set}
      → (D : (x y : A) → (x ≡ y) → Set)
      -- → (d : (a : A) → (D a a r ))
      → ((a : A) → (D a a r ))
      → (x y : A)
      → (p : x ≡ y)
      ------------------------------------
      → D x y p
  J D d x .x r = d x
```

2 Goals and Challenges

3 Approach

References

[1] The Univalent foundations program and N.J.) Institute for advanced study (Princeton. *Homotopy Type Theory: Univalent Foundations of Mathematics*. 2013.