The Role of Formalization and Z Notation

Arthur Ryman, arthur.ryman@gmail.com

March 4, 2018

Abstract

The notion of *type* was introduced into set theory by Russel in an effort to establish a firm foundation for logic. Types are now a standard feature of many computer programming languages. Z Notation is a formal language based on typed set theory for specifying computer programs. Since Z Notation is a formal language, a Z specification can itself be automatically checked for type errors. Writing valid Z specifications requires that the author explicitly define all terms. This encourages a style in which complex terms are gradually built up from simpler, previously defined terms. This article proposes to use Z Notation to formally specify mathematical structures, even if there is no intension of using them in computer programs. The hoped for benefits of doing so are that the exercise of formalization will lead the author to a better understand of the subject matter, resulting in a clearer, higher quality, result.

1 Types

Computer languages can be divided into two major groups, namely those that are strongly typed and those that are weakly typed. Strongly-typed languages, such as Java, have complex type systems and require that all expressions and variable declarations have well-defined types. Weakly-typed languages, such as Javascript, have simpler type systems and allow greater flexibility in expressions and the use of variables. Although no compiler can possibly tell you if your program is correct, a compiler for a strongly-typed language can at least detect type errors. Correcting these errors at compile-time is often quicker than doing so at execution-time.

There are many similarities between mathematics and programming. Both require the definition and use of named objects that are built up from simpler objects. The concept of type applies to both. Bertrand Russell discovered typed set theory in his efforts to solve some problems in the foundations of mathematics long before computers were a reality. Theoretical computer science now makes extensive use of types. Mathematics can be used to specify the requirements for programs. Specifications act as a bridge between mathematics and programs. Z Notation is a strongly-typed formal specification language. However, Z Notation can be used to formalize mathematics even if the goal is not to program a computer.

2 Formalization

Z Notation can be used to precisely define mathematic objects and a Z specification can be type checked by a program. I have found that the exercise of expressing mathematics using Z Notation is a great way make sure that concepts are well-defined. This is especially helpful, although somewhat labour-intensive, when dealing with complex mathematical structures. I propose to formalize all definitions using Z Notation and to validate the document using the *f*UZZ type checker.

I believe that the effort of precisely defining every needed concept pays off. Consider how Cédric Villani [Vil17] described the mathematical writing style of Alexander Grothendieck:

> Let me quote a famous mathematical text by Alexander Grothendieck, one of the most famous mathematicians of all times, who wrote long writings. In one of them he talks about the metaphor *de la noix*. You know the parable, the metaphor, of the nut to explain the difference between his style and the style of his fellow mathematician Jean-Pierre Serre. Both of them working in the same area of mathematics, but very different styles. Grothendieck said, "Imagine that we have a nut to open. The style of Serre would be take a hammer and, Bang!, smash the nut. My style would be to take the nut and put it in a sea of acid so that it would be dissolved very slowly, the crust of the nut, without noticing anything." Yes, experts tell us that the style of Grothendieck is like everything is very incremental from one step to another to another by very tiny steps so that you have the impression that nothing occurs and we are really making no progress, and at the end it's proven, there it is, the big theorem is is done.

Similarly, Pierre Deligne [Art+16] said:

> From Grothendieck I have also learned not to take glory in the difficulty of a proof: difficulty means we have not understood. The ideal is to be able to paint a landscape in which the proof is obvious. I admire how often he succeeded in reaching this ideal.

The message is clear: a slow, methodical build-up of concepts is a good thing in mathematics.

3 Given Sets and Signatures

Although the integers are built-in to Z Notation, the real numbers are not. In principle, one could first build up the rational numbers from the integers, and then the real numbers from the rational numbers, but that would take a lot of time and not really accomplish much. I'll assume that the real numbers are sufficiently well-understood and do not need

a complete formalization. Instead, I'll define real numbers as a given set and then declare the types and signatures of the usual constants and operations of real arithmetic.

4 Names and Symbols

Integers and reals are distinct types in the sense of typed set theory. But distinct types are disjoint sets. This implies that the 0 element of the integers is not the same object as the 0 element of the reals. If fact, they aren't even comparable within a Z specification since doing so is a type error and the specification would therefore not be valid. The name `0` is built-in to Z Notation as the name of the 0 element of the integers. A Z specification must therefore introduce a new name to refer to the 0 element of the reals. However, working mathematicians regard the integers as a subset of the reals and therefore have no need to deal with duplicate names. As Henri Poincaré [Poi14] said:

> I think I have already said somewhere that mathematics is the art of giving the same name to different things. It is enough that these things, though differing in matter, should be similar in form, to permit of their being, so to speak, run in the same mould.

How then can we make a Z specification valid while still honouring normal mathematical practice? The way out of this difficulty is that if a mathematician would normally regard two distinct objects as the same then we give then distinct Z Notation names, but display them using the same typographic symbol. To paraphrase Poincaré:

> Formalizing mathematics is the art of giving different names, but the same symbol, to different things. It is enough that these things, though differing in type, should be similar in typography, to permit of their being, so to speak, run in the same mould.

For example, Mike Spivey [Spi00] describes three typographic symbols defined in the *f*UZZ package that are each used to display two distinct objects:

> A few symbols have two names, reflecting two different uses for the symbol in Z:
>
> - The symbol ⨾ is called `\semi` when it is used as an operation on schemas, and `\comp` when it is used for composition of relations.
> - The symbol \ is called `\hide` as the hiding operator of the schema calculus, and `\setminus` as the set difference operator.
> - The symbol ↾ is called `\project` as the schema projection operator, and `\filter` as the filtering operator on sequences.

> Although the printed appearance of each of these pairs of symbols is the same, the type checker recognizes each member of the pair only in the appropriate context.

This approach keeps both the Z type checker and the mathematician reader happy. For example, in my formalization of the reals:

- The symbol 0 is called `0` when it is used as an integer, and `\zeroR` as a real number.
- The symbol + is called `+` as the addition operator on the integers, and `\addR` as the addition operator on the reals.

The mathematician reader will always be able to infer the actual type of an object from its context.

References

[Poi14] Henri Poincaré. In: *Science and Method*. Trans. by Francis Maitland. Digitized by the Internet Archive in 2008 with funding from Microsoft Corporation. Thomas Nelson and Sons, London, 1914. Chap. I.II. The Future of Mathematics, p. 34. URL: `http://www-history.mcs.st-andrews.ac.uk/Extras/Poincare_Future.html`.

[Spi00] Mike Spivey. *The fuzz Manual*. Second Edition. The Spivey Partnership, 2000. URL: `http://spivey.oriel.ox.ac.uk/mike/fuzz/fuzzman.pdf`.

[Art+16] Michael Artin et al. "Alexandre Grothendieck 1928-2014, Part 1". In: *Notices of the AMS* 63.3 (Mar. 2016), pp. 242–265. URL: `https://www.ams.org/publications/journals/notices/201603/rnoti-p242.pdf`.

[Vil17] Cédric Villani. *Cedric Villani: The Hidden Beauty of Mathematics - Spring 2017 Wall Exchange*. 2017. URL: `https://youtu.be/v-d0ruh0CHU?t=1823`.