THE ROLE OF FORMALIZATION AND Z Notation

ARTHUR RYMAN

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 intention 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 understanding of the subject matter, resulting in a clearer, higher quality, result.

Contents

Date: June 30, 2022.

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 mathematical 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 [4] 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." And 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 done.

Similarly, Pierre Deligne [1] 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 into 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. In 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 [2] 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 [3] 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.

5. Theorems, Examples, and Remarks

In mathematical writing it is good practice to follow a definition with some examples. When an example is given, there is a proof obligation associated with it, namely that whatever the author is asserting about the example is in fact true. Similarly, when the author makes a remark or states a theorem, there is also a proof obligation. In each of these cases, the author makes a statement and asserts it to be true, but may not offer any proof.

Z Notation allows the author to add constraints to a specification. However, if the constraint is a logical consequence of the preceding definitions then it adds nothing to the specification. Therefore, we can freely add examples, remarks, and theorems to a specification without changing its meaning. These statements should be placed in a `\zed` box that is enclosed in an appropriate LaTeX theorem-like environment.

For example, the following statement is true and therefore does not change the meaning of the specification but will be type-checked by *f*UZZ, .

Example.

$1 + 1 = 2$

The benefit of placing theorem-like statements in `\zed` boxes is that the *f*UZZ type checker will check their type-correctness.

5.1. **Generic Theorem-Like Statements.** It will often be the case that a theorem-like statement will involve generic constructs. Although Z Notation provides a mechanism for introducing formal generic parameters into the definition of generic constants and schemas, it does not provide such a mechanism for constraints. We can work around this limitation by declaring a set of global given sets which are understood to be used in theorem-like statements where they are taken to represent arbitrary sets. The article on topological spaces introduces the global given sets X, Y, and Z for this purpose.

Testing hyperlinks to another PDF document:

```
\url{https://agryman.github.io/mathz/complex-numbers.pdf}
```

https://agryman.github.io/mathz/complex-numbers.pdf

```
\url{complex-numbers.pdf}
```

complex-numbers.pdf

```
\url{./complex-numbers.pdf}
```

./complex-numbers.pdf

For example, we can remark that every element x of an arbitrary set X is a member of that set.

Remark.

$\forall\, x : \mathsf{X} \bullet x \in \mathsf{X}$

6. Simulating Standard Mathematical Notation

The expression (a, b) usually denotes an ordered pair in mathematical writing. Sometimes an author asks us to interpret (a, b) as an open interval of the real number line. In this case we recognize that the symbols used in the expression (a, b) do not have their usual meanings and we interpret them accordingly. The notation (a, b) for denoting an open interval of the real number line is arguably very compact, convenient, and easy to understand. In contrast, the Z Notation function application interval(a, b), defined in the article on real numbers, may strike the reader as somewhat verbose and clumsy, albeit precise.

6.1. **Prefix, Infix, and Postfix Operators in fuzz.** There is a way to achieve some of the compactness of standard mathematical notation in Z while preserving its precision and type-correctness. The approach is to take advantage of the *f*uzz type checker's ability to define infix and postfix operator symbols. In the grammar used by *f*uzz, postfix operators have higher precedence than prefix operators, and prefix operators have higher precedence than infix operators. Infix operators are assigned numerical priorities of 1 through 6 with higher priorities taking precedence over lower ones. We can use these relative precedences to define operators that allow us to construct expressions that resemble usual mathematical writing.

6.2. **Defining Intervals with Operators.** First, we use LaTeX bold styling to provide a visual cue that helps the reader distinguish the interval $(a \mathbin{,} b)$ from the usual ordered pair (a, b). Next, we interpret the symbols as prefix, infix, and postfix operators. The symbol $($ is a prefix operator that maps a to $(a$, the open interval bounded below by a. Similarly, $)$ is a postfix operator that maps b to $b)$, the open interval bounded above by b. Finally, we interpret the symbol $\mathbin{,}$ as an infix operator that forms the intersection of the two intervals.

$$
\begin{array}{|l}
(\,: \mathbb{R} \longrightarrow \mathbb{P}\,\mathbb{R} \\
\_) : \mathbb{R} \longrightarrow \mathbb{P}\,\mathbb{R} \\
\_ \mathbin{,} \_ : \mathbb{P}\,\mathbb{R} \times \mathbb{P}\,\mathbb{R} \longrightarrow \mathbb{P}\,\mathbb{R} \\
\hline
\forall\, a : \mathbb{R} \bullet (a = \{\, x : \mathbb{R} \mid a < x \,\} \\
\forall\, b : \mathbb{R} \bullet b) = \{\, x : \mathbb{R} \mid x < b \,\} \\
\forall\, a, b : \mathbb{R} \bullet (a \mathbin{,} b) = \{\, x : \mathbb{R} \mid a < x < b \,\}
\end{array}
$$

This design is a step in the right direction, but it doesn't prevent the writer from abusing the notation. For example, the following paragraph looks odd but makes perfect sense to *f*uzz.

$\forall\, a, b : \mathbb{R} \bullet (a \mathbin{,} b) = b) \mathbin{,} (a$

When we are asked to interpret (a, b) as an interval, we are, in a sense, being asked to parse a sentence in a new mathematical mini-language that defines intervals. If we want to enforce the usual syntax for intervals, we can introduce new types to represent fragments of the expression so that the operators can only be applied to fragments in some prescribed order.

To illustrate this point, let's expand the example to include closed intervals $[a, b]$ as well as semi-closed intervals $(a, b]$ and $[a, b)$. Regard $(a$ and $[a$ as open and closed right half lines, and $b)$ and $b]$ as open and closed left half lines. Only allow a right half line to be combined with a left half line.

6.3. **Right Half Lines.** Let *RightHalfLine* denote the type of syntactic fragments that define open and closed right half-lines.

$$\mathit{RightHalfLine} ::= (\langle\!\langle \mathbb{R} \rangle\!\rangle \mid [\langle\!\langle \mathbb{R} \rangle\!\rangle$$

Example.

$(0 \in \mathit{RightHalfLine}$

$[1 \in \mathit{RightHalfLine}$

Let $\mathit{rightSet}(R)$ denote the set of real numbers that the right half-line syntactic fragment R represents.

$$
\begin{array}{|l}
\mathit{rightSet} : \mathit{RightHalfLine} \longrightarrow \mathbb{P}\,\mathbb{R} \\
\hline
\forall\, a : \mathbb{R} \bullet \mathit{rightSet}((a) = \{\, x : \mathbb{R} \mid a < x \,\} \\
\forall\, a : \mathbb{R} \bullet \mathit{rightSet}([a) = \{\, x : \mathbb{R} \mid a \leq x \,\}
\end{array}
$$

For example

$0 \notin rightSet((0)$

$1 \in rightSet([1)$

6.4. **Left Half Lines.** Let *LeftHalfLine* denote the type of syntactic fragments that define open and closed left half-lines.

$LeftHalfLine ::= (\_))\langle\langle\mathbb{R}\rangle\rangle \mid (\_])\langle\langle\mathbb{R}\rangle\rangle$

For example

$0) \in LeftHalfLine$

$1] \in LeftHalfLine$

Let $leftSet(L)$ denote the set of real numbers that the left half-line syntactic fragment L represents.

$$
\begin{array}{|l}
leftSet : LeftHalfLine \longrightarrow \mathbb{P}\,\mathbb{R} \\
\hline
\forall\, b : \mathbb{R} \bullet leftSet(b)) = \{\, x : \mathbb{R} \mid x < b \,\} \\
\forall\, b : \mathbb{R} \bullet leftSet(b]) = \{\, x : \mathbb{R} \mid x \leq b \,\}
\end{array}
$$

For example

$0 \notin leftSet(0))$

$1 \in leftSet(1])$

6.5. **Intersection of Half Lines.** Let $R\,,\,L$ denote the set of real numbers in the intersection of the sets represented by the half-line syntactic fragments R and L.

$$
\begin{array}{|l}
\_\,,\,\_ : RightHalfLine \times LeftHalfLine \longrightarrow \mathbb{P}\,\mathbb{R} \\
\hline
\forall\, R : RightHalfLine;\ L : LeftHalfLine \bullet \\
\qquad R\,,\,L = rightSet\ R \cap leftSet\ L
\end{array}
$$

For example

$0 \in [0\,,\,1)$

6.6. **Improvements.** After reading the above, my reaction is that it would be clearer to first define the relevant objects using meaningful names and then define the symbols as abbreviations for them.

TODO:

- Fix up the above discussion.
- Use the example environment for examples.
- Improve the Summary. It should summarize the whole article, not just how to simulate mini-languages.

6.7. **Summary.** A lot of standard mathematical notation can be written directly using the built-in capabilities of Z Notation and *f*UZZ. However, mathematics contains many specialized notations including those for intervals, probability, and quantum mechanics (Dirac bra-ket notation). One can view these specialized notations as mathematical mini-languages. As demonstrated above for the case of intervals, it is possible to simulate these notations in Z by introducing new syntactic types along with infix and postfix operators that construct, combine, and reduce fragments of this syntax. The general pattern is for the opening and closing symbols to correspond to prefix and postfix operators that construct fragments, and for internal symbols to correspond to infix operators that combine and then finally reduce fragments. Reduction occurs when the final infix operator is applied and maps the completed mini-language sentence to a some usual mathematical type.

References

[1] 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.

[2] 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.

[3] Mike Spivey. *The fUZZ Manual*. Second Edition. The Spivey Partnership, 2000. URL: https://spivey.oriel.ox.ac.uk/wiki/images-corner/c/cc/Fuzzman.pdf.

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

Email address, Arthur Ryman: arthur.ryman@gmail.com