AN ARGUMENT FOR CONTROLLED NATURAL LANGUAGES IN MATHEMATICS

THOMAS HALES

1. Introduction

At the recent Big Proof 2 conference in Edinburgh,[1] I realized that a case must be made for developing a controlled natural language for mathematics. There is little consensus on this issue, and mathematicians and computer scientists tend to line up on opposite sides. Some are outright dismissive of the idea. Little research is happening. While making the case below (giving the mathematician's side), I'll also review and respond to some of the computer scientists' objections.

I thank Arnold Neumaier and Peter Koepke for many discussions and for getting me interested in this topic.

1.1. Controlled Natural Languages (CNL).

By controlled natural language for mathematics (CNL), we mean an artificial language for the communication of mathematics that is (1) designed in a deliberate and explicit way with precise computer-readable syntax and semantics, (2) based on a single natural language (such as Chinese, Spanish, or English), and (3) broadly understood at least in an intuitive way by mathematically literate speakers of the natural language.

The definition of controlled natural language is intended to exclude invented languages such as Esperanto and Logjam that are not based on a single natural language. Programming languages are meant to be excluded, but a case might be made for TEX as the first broadly adopted controlled natural language for mathematics.

Perhaps it is best to start with an example. Here is a beautifully crafted CNL text created by Peter Koepke and Steffen Frerix.[2] It reproduces a theorem and proof in Rudin's *Principles of mathematical analysis*

Date: June 14, 2019.

[1] Big Proofs, ICMS

[2] AITP 2019 proceedings, page 84

almost word for word. Their automated proof system is able to read and verify the proof.

Theorem 1 (text in Naproche-SAD system)**.** *If $x \in \mathbb{R}$ and $y \in \mathbb{R}$ and $x > 0$ then there is a positive integer n such that $n \cdot x > y$.*

Proof. Define $A = \{n \cdot x \mid n \text{ is a positive integer}\}$. Assume the contrary. Then y is an upper bound of A. Take a least upper bound α of A. $\alpha - x < \alpha$ and $\alpha - x$ is not an upper bound of A. Take an element z of A such that not $z \leq \alpha - x$. Take a positive integer m such that $z = m \cdot x$. Then $\alpha - x < m \cdot x$ (by 15b).

$$\alpha = (\alpha - x) + x < (m \cdot x) + x = (m + 1) \cdot x.$$

$(m + 1) \cdot x$ is an element of A. Contradiction. Indeed α is an upper bound of A. □

In my view, this technology is undervalued. I feel like a stock advisor giving tips here, but I see it as an opportune time to invest heavily in CNLs.

2. The argument

Here is an outline of the argument that I will develop in the paragraphs that follow.

(1) Technology is still far from being able to make a semantic reading of mathematics as it is currently written.
   (a) Machine learning techniques (in particular, deep neural networks) are still far from a semantic reading of mathematics.
   (b) Linguistic approaches are still far from a semantic reading of mathematics as it is currently written.
(2) Mathematicians are still far from the mass adoption of proof assistants.
   (a) Adoption has been gradual.
   (b) Structural reasons hinder the adoption of proof assistants.
(3) There is value in bridging the gap between (1) and (2).
(4) CNL technology works now and can help to bridge the gap.

1. Technology is still far from making a semantic reading of mathematics as it is currently written. In my view, the current champion is WolframAlpha, in its ability to answer natural language queries about mathematics.

1a. Machine learning (neural networks) are still far from a semantic reading of mathematics. Recent avances in machine learning, such as AlphaZero, have been spectacular. However, some experts point to major unsolved technological problems.

I visited Christian Szegedy's group at Google in Mountain View and April and attended their talks at AITP in Austria the same month.[3] They have ambitions to *solve math* (to use their mind-boggling phrase) in the coming years. Their ambitions are inspiring, the resources and research methodologies at Google are grandiose, but I would not personally bet on their time frame.

Their current focus is to use deep neural networks to predict a sequence of tactics that will successfully lead to new formal proofs of lemmas. These are all lemmas that have been previously formalized by John Harrison in his HOL Light libraries on real and complex analysis. As things stand now, other long-established technologies (such as hammers) have had similar success rates in formal proof rediscovery. Thus far, Google is not ahead of the game, but they are progressing quickly.

David Saxton (at DeepMind) spoke at the Big Proof 2 conference in May.[4] He described a new large machine-learning dataset of school-level problems in areas such as arithmetic, algebra, and differential calculus.[5] Currently the machine learning success rates are poor for many of these categories. The dataset is meant to stimulate machine learning research on these topics.

Deep neural networks have not been successful thus far in learning simple algorithms. In particular, deep neural networks cannot currently predict with much accuracy the answer to elementary sums of natural numbers. What is $437 + 156$? Can a machine learn the algorithm? School-level math problems require more than parsing mathematical text. Nevertheless, parsing of mathematical text remains an unsolved challenge. They write in the introduction to their recent paper:

> "One area where human intelligence still differs and excels compared to neural models is discrete compositional reasoning about objects and entities, that 'algebraically generalize' (Marcus, 2003). Our ability to generalise

[3] AITP 2019 proceedings, page 19
Christian Szegedy's research page

[4] David Saxton et al. "Analysing Mathematical Reasoning Abilities of Neural Models"

[5] Github mathematics dataset

> within this domain is complex, multi-faceted, and patently different from the sorts of generalisations that permit us to, for example, translate new sentence of French into English."

Here is an example of a challenging problem for machine learning from their paper. What is $g(h(f(x)))$, where $f(x) = 2x + 3$, $g(x) = 7x - 4$, and $h(x) = -5x - 8$?"

Based on their paper, it seems to me that semantic parsing of research level mathematics is still years away.

1b. Linguistic approaches are still far from a semantic reading of mathematics as it is currently written. The most widely cited work here is Mohan Ganesalingam's thesis "The language of mathematics." The thesis uses Discourse Representation Theory from linguistics to analyze mathematical language. Ganesalingam analyzes the language of mathematics as it is written in real mathematical publications, in all of its organic complexity.

There is far too little research in this area. Here is his assessment.[6]

> Mathematics differs sufficiently from human languages that little linguistic theory can be used directly; most of the material must be rebuilt and reconstructed. But linguistics, at the very least, gives us general ideas about how we can formally analyse language: it gives us a mindset and a starting place. Thus, in very broad terms, our work may be regarded as 'the application of linguistics to mathematical language'. This is almost untrodden ground; the only linguist to analyse mathematics, Aarne Ranta, notes that
>
> > Amazingly little use has been made of [mathematical language] in linguistics; even the material presented below is just an application of results obtained within the standard linguistic fragment [...]. Ranta (1994)

Furthermore,

> We demonstrate that no linguistic techniques can remove the ambiguity in mathematics. We instead borrow

[6]M. Ganesalingam, Principia conference

> from mathematical logic and computer science a notion called the 'type' of an object... and show that this carries enough information to resolve ambiguity. However, we show that if we penetrate deeply enough into mathematical usage, we hit a startling and unprecedented problem. In the language of mathematics, what an object is and how it behaves can sometimes differ: numbers sometimes behave as if they were not numbers, and objects that are provably not numbers sometimes behave like numbers.

As I understand, Mohan's work was mostly theoretical and although there were once plans to do so, it has not been implemented in any available software.

2. Mathematicians are still far from the mass adoption of proof assistants.

2a. Adoption has been gradual. I have watched first-hand the gradual adoption of proof assistants.

In 2001 when I first took up formalization, I knew only a few mathematicians who were even aware of proof assistants (Dan Bernstein, Carlos Simpson, Doran Zeilberger, ...). That was the year I moved to Pittsburgh, and CMU already had a long tradition of formal proof in among logicians and computer scientists (Dana Scott, Peter Andrews, Frank Pfenning, Bob Harper, Jeremy Avigad, and so forth).

In 2008, Bill Casselman and I were guest editors of a special issue of the Notices of the AMS on formal proof that helped to raise awareness among mathematicians. This included a cover article by Georges Gonthier on the formal proof of the four color theorem. (All the authors of this special issue were together again at the BigProof2 conference.)

Homotopy type theory (HoTT) became widely known throug a special program at the Institute for Advanced study in 2012-2013, organized by V. Voevodsky, T. Coquand, and S. Awodey. A popular book on HoTT resulted from that program. Other programs followed, such as the IHP program on the semantics of proof and certified programs in 2014.

Major publishers in mathematics now publish on formal proofs. There have been notable success stories (formalizations of the Feit-Thompson theorem and the Kepler conjecture). Equally, there have been major

industrial success stories (the CompCert verified compiler and the SeL4 microkernel verification).

In the last few years, several more mathematicians (or perhaps dozens if we include students) have become involved with Lean (but not hundreds or thousands).

Formal proofs are now regular topics online in places such as reddit and mathoverflow.

In summary, the landscape has changed considerably in the last twenty years, but it is still nowhere near mass adoption.

2b. Structural reasons hinder the adoption of proof assistants. There are reasons for lack of mass adoption. The reason is not (as some computer scientists might pretend) that mathematicians are too old or too computer-phobic to program.

If mathematicians are not widely adopting formalization, it is because they have already sized up what formalization can do, and they realize that no matter its long term potential, it has no immediate use to them. Homotopy type theory and related theories are closely watched by mathematicians even if not widely used.

I realize that I am an exception in this regard, because I had a specific real-world situation that was solved through formalization: the referees were not completely able to certify the correctness of the Kepler conjecture and eventually they reached a point of exhaustion. It fell back on my shoulders to find an alternative way to certify a proof the Kepler conjecture.

Wiedijk has pinpointed the most important reason that proof assistants have not caught on among mathematicians: in a proof assistant, the obvious ceases to be obvious.

Kevin Buzzard has spoken about his frustrations in trying to get Lean to accept basic facts that are obvious to every mathematician. Why is $1 \neq 2$? He has faced further frustrations in dealing with mathematically irrelevant distinctions made by proof assistants. For example, different constructions of a localization of a ring can lead to canonically isomorphic (but unequal) objects, and it is no small matter to get the proof assistant to treat them as the same object.

3. There is value in bridging the gap between (1) and (2). I'll quote a few mathematicians.[7]

> "I would like to see a computer proof verification system with an improved user interface, something that doesn't require 100 times as much time as to write down the proof. Can we expect, say in 25 years, widespread adoption of computer verified proofs?" -J. Lurie

> "I hope [we will eventually be able to verify every new paper by machine.]. Perhaps at some point we will write our papers... directly in some formal mathematics system." - T. Tao

Timothy Gowers was part of a panel discussion at BigProof1 conference in 2017.[8] He went through a list of automation tools that he would find useful and another list of tools that he would not find useful at all. One dream was to develop an automated assistant that would function at the level of a helpful graduate student. The senior mathematician would suggest the main lines of the proof, and the automated grad student would fill in the details. He also suggested improved search for what is in the literature, when you do not know the name of the theorem you are searching for, or even whether it exists. For this, full formalization is not needed.

Michael Kohlhase and others in his group have advocated "flexiformal" mathematics, as an interpolation between informal and formalized mathematics.[9] They are also developing technology to translate between different proof assistants (or formal languages), once the mathematics has been represented in one of them.

The Logipedia project also implements general methods to translate mathematics between formal languages. For this to happen, the mathematical content must first be represented in some formal way. The two main technologies for this project are Logical Frameworks (Dedukti) and Reverse Mathematics.

An intended application of a large corpus of mathematics in a CNL would be big data sets for machine learning projects. Currently there

[7] Quoted in Ursula Martin's BigProof2 conference slides from the Breakthrough Prize interviews, 2014

[8] Gowers, panel discussion, Big Proof, 2017 (minutes 15–26, mp3)

[9] Iancu, Thesis 2017, Flexiformal, Flexiforms

is far too little data that aligns natural language mathematics with formal mathematics.

My dream is to someday have an automated referee for mathematical papers.

4. CNL technology works and can help to bridge the gap. I believe strongly in the eventual mechanization of large parts of mathematics. According to a divide and conquer strategy for accomplishing this, we want to strike at the midpoint between current human practice and computer proof. In my view, CNLs do that. A million pages of this style of CNL is achievable.

4a. CNL has a long history. A brief history of the CNL *Forthel (an acronym of FORmal THEory Language)* is sketched in K. Vershinin's note.[10] Koepke told me that Forthel (that is, fortel') means trick in Russian and that this is an intended part of the acronym.

Research started with V. Glushkov in the 1960s. A system of deduction was part of the system from the beginning (even if my interest is primarily in the language). A description of the language first appeared in the Russian paper "On a language for description of formal theories" Teoretischeskaya kibernetika, N3, 1970.

Research continued with V. Bodnarchuk, K. Vershinin and his group through the late 1970s. Most of the early development was based in Kiev.

The language was required to have a formal syntax and semantics. To allow for new definitions, "the thesaurus of the language should be separated from the grammar to be enrichable. On the other hand the language should be close to the natural language of mathematical publications."

The development of the Forthel language development seems to have been intermittant. Andri Paskevich's thesis in Paris gives a description of the language and deduction system as of the end of 2007.

Afterwards, development (renamed as Naproche-SAD) shifted to Bonn with Peter Koepke and his master's student Steffan Frerix. The current Haskell code is available on github.[11] In this document, I will refer to Forthel and Naproche-SAD as synonyms, although Forthel is the name

[10] Vershinin, "ForTheL – the language of formal theories"
[11] Github, Naproche-SAD

of the controlled natural language, whereas Naproche-SAD includes a reasoning module connected to the E theorem prover.

In May 2019, Makarius Wenzel implemented a a mode of Isabelle/PIDE for Naproche-SAD.[12] It can be downloaded.[13] This is currently the best way to interact with Naproche-SAD.

4b. The Forthel language has an elegant design. The text we produce is intentionally verbose and redundant. The aim is to produce texts that can be read by mathematicians without any specialized training in the formalization of mathematics. We want mathematicians (say after reading a one page description of our conventions) to be able to read the texts.

The ability to understand a CNL is an entirely different matter than to write or speak a CNL. The CNL might have invisible strict grammatical rules that must be adhered to. As Stephen Watt put it after my BigProof2 lecture, there is a big difference between watching a movie and directing a movie.

Our main references are Paskevich's paper on Forthel *(The syntax and semantics of the ForTheL language, 2007)*, his thesis at Paris XII and the Naproche-SAD source code.[14].

Naproche-SAD achieves readability through a collection of tricks. The linguistics behind Naproche-SAD is actually quite elementary, and it is remarkable how English friendly it is with so little linguistics. One trick is the generous use of stock phrases (or canned phrases) that the user can supply.

Another trick is the use of free variants (synonyms), such as [set/sets], which declares that the words *set* and *sets* are fully interchangeable. There are no grammatical notions of gender, case, declension, conjugation, agreement, etc. in Naproche. The needed forms are introduced as variants and the system does not care which are used.

Another trick is the use of filler words. These are words that are ignored by Naproche-SAD. For example, in many contexts indefinite and definite articles *a, an, the* are disregarded. They are there only for human readability.

[12] Wenzel, arXiv paper, 2019

[13] Isabelle-Naproche download

[14] See *(Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques, 2007)* and github, Naproche-SAD

The Naproche-SAD parser is written in Haskell. It is basically a simplified knock-off of the Haskell parsec parser library.

The grammar is not context-free (CFG). Thus, some parser generators (such as $LR(k)$ parsers) are not appropriate for our project.

Although the grammar is extensible, we can still describe the language by a finite list of production rules in BNF format. The collection of nonterminals cannot be changed. However, some of the nonterminals are designated as *primitive*. Primitive nonterminals differ from ordinary nonterminals in that new replacement rules can be added (by the user) to a primitive nonterminal. In this way, as mathematical text progresses, the grammar can be enriched with new constructions and vocabulary.

4c. Forthel language design principles provide a template for future CNLs. This is work in progress. The point is that Forthel design is sufficiently generic that the language can be readily adapted to support back-end semantics of Lean's dialect of CiC.

For this project, we recommend using the Haskell parsec parser library. Parsec is written as a library of parser combinators, with lazy evaluation, in a continuation style, with state stored in a parser monad.

If Lean4 has sufficiently powerful parsing capabilities, we can eventually port the project into Lean4. By using Haskell for our initial implementation, we make porting easier.

Actually, we combine two different parsing technologies: parsec style combinators and a top-down-operator precedence parser. The top-down-operator precedence parser is used for purely symbolic formulas. We describe below the rule used to extract sequences of symbolic lexemes that are shipped to the top-down-operator precedence parser. By including a top-down-operator precedence parser, we are able to add new symbolic notations with fine-grained levels of precedence.

We expect the grammar to be ambiguous. The parser should be capable of reporting ambiguous user input.

It seems to me that the biggest difficulty in writing a CNL for Lean will be the "last mile" semantics of Lean. By this, I mean the automatic inserting the correct coercions and casts along equality to produce a pre-expression that can be successfully elaborated and type checked by Lean. (I expect that automatically generated pre-expressions will frequently be off by casts along equality.)

3. Objections and counterarguments

At the BigProofs2 conference, several researchers made the argument that controlled natural languages are a step backwards from precise programming languages.

3.1. **We should not abandon centuries of mathematical notational improvements.** A sneering comparison was made between CNLs and the medieval practice of writing formulas and calculations out long-form in natural language. Significant progress in mathematics can be attributed to improved symbolic representations for formulas and calculations. It would be terrible to be forced in our language to write $\pi/\sqrt{18}$ as *pi divided by the square root of eighteen.*

I agree. In fact, our controlled natural language is designed as a superset of Lean syntax. Lean is both a language of mathematics and a pure functional programming language similar to Haskell. Our augmented syntax loses none of that. We continue to write $\pi/\sqrt{18}$ as such.

Nobody advocates abandoning notations that mathematicians find useful. A fully developed CNL includes all the great notational inventions of history such as decimal, floating point and other number formats, polynomial notation, arithmetic operations, set notation, matrix and vector notation, equations, logical symbolism, calculus, etc.

A CNL is also more than that, by providing precise semantics for mathematics in natural language.

Knuth, who has thought long and hard about mathematical notation, instructs that it is "especially important" not to use the symbols

$$\therefore,\ \Rightarrow,\ \forall,\ \exists,\ \ni;$$

"replace them by the corresponding words. (Except in works on logic, of course.)"[15] The symbols are less fluent than the corresponding words. Knuth gives this example of a bad writing style:

$$\exists_{n_0 \in N_0} \forall_{p \in P}\ p \geq n_0 \Rightarrow f(p) = 0.$$

Another bad example from Knuth appears below in the section on examples.

3.2. **We should not return to COBOL.** Continuing with the criticism of CNL, some argue that CNLs are an unpleasant step in the direction of COBOL. Among the BigProof2 conference crowd, COBOL

[15] Knuth, math writing, page 1. pdf page 3.

evokes Dijkstra's assessment "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence." Still today, there is a strong distaste among some theoretical computer scientists (and principled programming language designers) of designs based on natural language.

Like COBOL, a CNL is self-documenting, English-oriented, and more verbose than ordinary programming languages. However, the features of COBOL that really incite wrath have nothing to do with CNLs: the GOTO statement, lack of modularity, no function calls, no type checking on subroutine calls, no libraries, and so forth.

3.3. **Lean is already readable.** Some say that Lean is already readable and that a CNL is therefore wasted effort.

I find the level of readability of Lean to be similar to that of Coq. It is somewhat less readable than scripts in Mizar and Isabelle/HOL/Isar.

For those trained in Lean, the statements of definitions and theorems are readable, but most users find it necessary to jump incessantly from file to file within an environment such as VSCode (or emacs) because relevant context is spread through many files.

Generally, even those who write the scripts in Lean find it necessary to jump around in this manner to make sense of their own scripts. It is wonderful to have tools that allow us to jump around, but when that becomes the only possible mode of interaction, our thinking about mathematics becomes fragmented. (I may be old-fashioned in this worry of fragmentation.)

Proofs in almost all proof assistants are unreadable (in any sort of linear text-oriented fashion). Lean is no exception.

My impression is that almost no mathematician would be willing to invest the necessary effort to read Lean scripts.

Here are some samples of Lean code.[16] I do not mean to single out the authors of these files for enhanced criticism. The files are meant to be representative of Lean.

In a CNL, there is no semantic penalty for writing the fundamental theorem of algebra as follows.

[16] Lean manifolds
Lean Fundamental Theorem of Algebra

Assume that f is a polynomial over $\mathbb{C}$ of positive degree. Then there exists a complex number z that is a root of f.

3.4. **Documentation should be generated from source code, not vice versa.** Another argument is that we should generate natural language documentation from the formal text rather than the other way around. There are many successful examples of this approach, such as Mizar to LaTeX translation.[17] These are useful tools, and I have no criticism of this.

The direction of code generation depends on the purpose. The source format is generally speaking the cleanest representation. A human will be directly involved in writing the source format. If the primary purpose is formalization, then formal scripts should be the source format from which others are generated. (Subtle differences in representation of mathematical concepts code can make a big difference in how painful the formal proofs are.) If the primary purpose is mathematical communication, then human-oriented language should be the source format. (Humans can be easily annoyed by the stilted language generated from formal representations.) Who should we ask for tolerance, the human or the computer?

Other variations appear in practice. In some projects there are two source formats and semantics-preserving translation from one to the other. For example, computer software can have a specification from which executable code might be automatically extracted, but for reasons of efficient execution it might be better to run an optimized handcrafted version of the code that is verified to be equivalent to the extracted code.

Another approach was used in the Flyspeck project, which used an English text as the blueprint for the formal proof scripts. Both the blueprint and proof scripts were written by humans, and the alignment between them was also a human effort. I wish to avoid that labor-heavy style in the formal abstracts project.

I have some experience in automatic code generation. In the proof of the Kepler conjecture, the C++ code to rigorously test nonlinear inequalities by interval arithmetic was automatically generated from the formal specification of the inequalities in HOL Light.

[17] XSL-Based Translator of Mizar to LaTeX

3.5. **The foundational back ends of CNLs are too weak.** The Forthel language compiles down to first order logic. The reasoning in the CNL example from Rudin in Section 1.1 is not based on axiomatic reasoning using the axioms of Zermelo-Fraenkel set theory. Rather, the authors made up their own system of axioms that were designed to prove exactly what they wanted to prove. For this reason, the example is really just a toy example that is foundationally ungrounded. (Yet it is a working technology that makes us see what is concretely realizable.)

None of the successful proof assistants are based on first order logic. Most of the widely used proof assistants are based on some form of type theory (simple type theory or the calculus of inductive constructions). Proof assistants based on set theory (such as Mizar) add a layer of soft typing on top of the set theory.

I agree with this criticism. I believe that a more powerful back end is needed for a CNL if it is to be widely adopted. I propose the Lean theorem prover as a back end. This will require some changes to the grammar of the CNL.

3.6. **We already have too many proof assistants and incompatible research directions.** Another CNL will further fragment research on formalization.

Interoperability is important. It emerged as a notable theme at the BigProofs2 conference. I would hope that we can achieve some interoperability with other projects such as Flexiformalized math and OMDoc, Logipedia and Dedukti, and the Lean theorem prover math libraries.

A long term project might be to design a CNL that is sufficiently universal to work with different proof asssistants on the back end.

In fact, there is too little activity in the CNL space right now. I hope that we can be compatible with Arnold Neumaier's group and Peter Koepke's.

4. EXAMPLES

The examples in this section are hypothetical. They are based on CNL grammar rules that have not been programming into a computer. They illustrate what we hope soon to obtain. The changes to the existing production rules of Forthel are relatively minor.

4.1. **the Kepler conjecture as an example.** Here is the formal statement of the Kepler conjecture as it appears in the HOL Light proof assistant.

```
[VERSION 1]
(!V. packing V
            ==> (?c. !r. &1 <= r
                ==> &(CARD(V INTER ball(vec 0,r))) <=
                pi * r pow 3 / sqrt(&18) + c * r pow 2))
```

There are several elements that hamper readability. It is all in ascii, including unpleasant ascii alternatives: ! for $\forall$, ? for $\exists$, <= for $\leq$, r pow 3 for r^3, sqrt for $\sqrt{}$, pi for π, INTER for $\cap$, and ==> for $\Rightarrow$.

Replacing the ascii, we obtain the Kepler conjecture in this slightly improved form:

[VERSION 2]

$$\begin{aligned}&(\forall\ V.\ \text{packing}\ V\\ &\quad\Rightarrow(\exists\ c.\ \forall\ r.\ \&1 \leq r\\ &\qquad\Rightarrow \&(\text{CARD}(V\ \cap\ \text{ball}(\text{vec}\ 0, r))) \leq\\ &\qquad\quad \pi * r^3/\sqrt{(\&18)} + c * r^2))\end{aligned}$$

This still looks like an ugly formula written by a novice. A mathematician would understand that $\sqrt{18}$ is a real number, without writing the unfamiliar operator & that casts the natural number 18 to the real number. It is more idiomatic so say *for every packing* V than the symbol-laden phrase $\forall\ V.$ packing $V \Rightarrow \ldots$, and so forth. Let's try again. (In this final version, we are switching from simple type theory as back-end semantics to the calculus of inductive constructions.)

[VERSION 3]

For every packing V, there exists a real number c such that for all real numbers $r \geq 1$, the number of points of V in the open ball $\mathbf{B}(0, r)$ is at most

$$\frac{\pi\ * r^3}{\sqrt{18}} + c\ * r^2.$$

Version 3 is by far the most readable of the three versions. This is what a CNL looks like in practice. It uses formulas where they are useful, but it avoids the needless symbolism of version 2.

Version 3 is very close to the wording used for a general mathematical audience in the official publication on the formal proof of the Kepler conjecture. Too much of the work of formalization consists of taking mathematics in a readable form (version 3) and reformatting it in a barely readable form (version 1). It simply is not true that version 1 is in any sense superior to version 3. A controlled natural language makes version 3 every bit as formal as version 1. It is easy to see why mathematicians would prefer version 3.

Version 3 has a consistent lexical structure, with numeric constants

$$0,\ 1,\ 2,\ 3,\ 18$$

variables (single letters)

$$V,\ c,\ r,$$

words (including specially declared single letters)

$$\text{for},\ \text{every},\ \text{there},\ \mathbf{B},\ \pi,\ \ldots$$

delimiters

$$(\ )$$

punctuation

$$,\ .$$

symbols

$$\geq,\ *,\ \sqrt{\ },\ /,\ \hat{}$$

etc. The lexemes are combined using production rules to give precise meaning.

For example, the phrase *for all real numbers* $r \geq 1$*, blah-blah* of version 3 follows a production rule for the symbolic statement nonterminal

```
- symbolicStatement   →   for all quantifierProp, symbolicStatemant
```

In turn, the subphrase *real numbers* $r \geq 1$ follows the production rule for the nonterminal `quantifierProp`. In this instance, syntax processing will recognize r as the bound variable, *real numbers* r as a type annotation $(r : \mathbb{R})$, and $r \geq 1$ as a subtyping predicate on $\mathbb{R}$.

A controlled natural language differs from a natural language because these production rules have been chosen by the language designer, and the text must exactly conform to these rules. A realistic approximation to mathematical English can be achieved by a few hundred carefully crafted production rules.

4.2. **Knuth's exercise on technical writing.** Knuth's course on writing math contains a useful exercise on technical writing.[18]

The context for the exercise is the following. We warn that what follows is intentionally atrocious writing style.

N denotes the nonnegative integers, N^n denotes the set of n-tuples of nonnegative integers, and $A_n = \{(a_1, \ldots, a_n) \in N^n \mid a_1 \geq \cdots a_n\}$. If $C, P \subset N^n$, then $L(C, P)$ is defined to be $\{c+p_1+\cdots+p_m \mid c \in C, m \geq 0, \text{ and } p_j \in P \text{ for } 1 \leq j \leq m\}$. We want to prove that $L(C, P) \subseteq A_n$ implies $C, P \subseteq A_n$.

The exercise is to rewrite the following proof originally written by a sophomore.

$L(C, P) \subset A_n$
$C \subset L \Rightarrow C \subset A_n$
Spse $p \in P, p \notin A_n \Rightarrow p_i < p_j$ for $i < j$
$c + p \in L \subset A_n$
$\therefore c_i + p_i \geq c_j + p_j$ but $c_i \geq c_j \geq 0,\ p_j \geq p_i \therefore (c_i - c_j) \geq (p_j - p_i)$
but $\exists$ a constant $k \ni c + kp \notin A_n$
let $k = (c_i - c_j) + 1 \quad c + kp \in L \subset A_n$
$\therefore c_i + kp_i \geq c_j + kp_j \Rightarrow (c_i - c_j) \geq k(p_j - p_i)$
$\Rightarrow k - 1 \geq k \cdot m \quad k, m \geq 1 \quad$ Contradiction
$\therefore p \in A_n$
$L(C, P) \subset A_n \Rightarrow C, P \subset A_n$ and the
lemma is true.

We will rewrite the proof in the style of a controlled natural language (that in principle can be parsed directly into a sequence of statements in Lean). We do not claim that Lean can correctly reconstruct a proof from these statements (but that would be an interesting research problem). Here goes.

Lemma 2. *Assume* $(i,\ j,\ n : \mathbb{N})$, $(P,\ C : \textit{set over } \mathbb{N}^n)$. *Let*

$$A(n) := \{a : \mathbb{N}^n \mid \textit{for all } i \leq j,\ a_i \leq a_j\}.$$

Let P^* *denote the submonoid of* $\mathbb{N}^n$ *generated by any* P. *Let*

$$C + P^* := \{c + b \mid c \in C,\ b \in P^*\}.$$

[18] Knuth's writing course

Then for every n, for all $C \neq \emptyset$ and $C + P^* \subseteq A(n)$, we have $C, P \subseteq A(n)$.

Proof. We claim that $C \subseteq A(n)$. Indeed, for all $c \in C$,

$$c = c + 0 \in C + P^* \subseteq A(n).$$

Next, we claim that $P \subseteq A(n)$. Pick any $p \in P$ and any $i \leq j$. Take $c \in C \neq \emptyset$. For all $(k : \mathbb{N})$, we have $c + k \cdot p \in C + P^* \subseteq A(n)$ and hence

$$c_i + k * p_i \leq c_j + k * p_j.$$

By the Archimedean property of $\mathbb{N}$, we get $p_i \leq p_j$. Hence $p \in A(n)$. □

I want a million pages of mathematics written in this style of CNL and compiled into Lean.