2008

A Language for Mathematical Knowledge Management

Steven Kieffer
Simon Fraser University

Jeremy Avigad
Carnegie Mellon University

Harvey Friedman
Ohio State University - Main Campus

A language for mathematical knowledge management

Steven Kieffer[1], Jeremy Avigad[2], and Harvey Friedman[3⋆]

[1] Simon Fraser University
[2] Carnegie Mellon University
[3] Ohio State University

Abstract. We argue that the language of Zermelo Fraenkel set theory with definitions and partial functions provides the most promising bedrock semantics for communicating and sharing mathematical knowledge. We then describe a syntactic sugaring of that language that provides a way of writing remarkably readable assertions without straying far from the set-theoretic semantics. We illustrate with some examples of formalized textbook definitions from elementary set theory and point-set topology. We also present statistics concerning the complexity of these definitions, under various complexity measures.

1 Introduction

With the growing use of digital means of storing, communicating, accessing, and manipulating mathematical knowledge, it becomes important to develop appropriate formal languages for the representation of such knowledge. But the scope of "mathematical knowledge" is broad, and the meaning of the word "appropriate" will vary according to the application. At the extremes, there are competing desiderata:

- At the *foundational level*, one wants a small and simple syntax, and a precise specification of its semantics. In particular, one wants a specification as to which inferences are valid.
- At the *human level*, one wants to have mathematical languages that are as easy to read and understand as ordinary mathematical texts, yet also admit a precise interpretation to the foundational level.

For ordinary working mathematicians, the foundational interpretation is largely irrelevant, but some sort of formal semantics is necessary if the information encoded in mathematical texts is to be used and manipulated at the formal level. Of course, one solution is simply to pair each informal mathematical assertion with a formal translation, but then there is the problem of obtaining the formal translations and ensuring that they match the intention of the informal text. As a result, it is more promising to use semi-structured languages that integrate

⋆ Carnegie Mellon Technical Report CMU-PHIL-181. Work by Avigad and Friedman partially supported by NSF grant DMS-0700174.

features of both the foundational and human levels. This results in a smooth spectrum of languages in between the two extremes. At intermediate "expert user" levels, one may want a language whose structure is close to that of the underlying foundational framework, yet is as humanly readable as possible.

To complicate matters, there are features of mathematical knowledge that are not captured at the level of assertions: mathematical language is used to communicate definitions, theorems, proofs, algorithms, and problems, among other things. At the level of a mathematical theory, language is also used to communicate relationships between these different types of data. The formal information that is relevant will vary depending on the application one has in mind, be it database access and search, theorem proving, formal verification, etc.

Here we will be primarily concerned with mathematical assertions as they are used to state definitions and theorems.[4] If one is looking for a foundational framework that is robust enough to subsume those used by most systems of MKM, it is hard to beat the language of set theory: we know of no foundational system other than Quine's New Foundations that cannot be interpreted in the language of set theory in such a way that inferences are reduced to inferences in Zermelo-Fraenkel set theory with the axiom of choice (ZFC), or some plausible extension (say, with large universes of sets). To be clear, we are not denying the importance of other frameworks for more specific purposes. For example, the theory of real closed fields is appropriate to representing many constraint problems, and constructive frameworks are better suited to certain forms of algorithmic reasoning. It is also important to find ways of sharing the additional information that comes with the use of these more restricted frameworks. We are simply singling out set theory as a unifying framework for expressing what assertions in the various local frameworks have in common.

We extend the foundational framework in two ways. First, we allow for explicit definitions of new predicates and functions on the universe of sets. And, second, we allow function symbols to denote functions that are only partially defined, using a logic of partial terms. We call the resulting formal system $DZFC$. As we observe in Section 2, this system is easily shown to be conservative over ZFC. We argue that these extensions are *not* just a matter of syntactic sugar, but, rather, are essential to adequate representation of the mathematical data: there is a difference between assertions using defined terms and their expanded versions, and, in mathematical terms, $1/0$ really is an undefined quantity. Thus $DZFC$ is our proposal for a foundational language and its semantics.

Our main goal here is to show that the distance between this foundational level and ordinary mathematical text is not as far as is commonly supposed, by presenting a syntactically-sugared version of set theory, PST, that is simulta-

[4] In passing, we note that computational proof assistants like Mizar [13], HOL [7], Isabelle [12], Coq [3] and HOL light [8] all provide languages that can be used to describe mathematical proofs. Of these, the Mizar and Isabelle/Isar languages model human proof languages most closely. The Isar effort [16] shows that the proof language is somewhat orthogonal to the assertion language; that is, Isar can be instantiated to various foundational frameworks, subject only to minor constraints.

neously close to both. On the one hand, we show that our language is easily parsed and translated to *DZFC*. On the other hand, by automatically replacing symbolic expressions with user-provided natural language equivalents, we obtain output that is humanly readable, and, although not exactly literary, recognizably faithful to the original mathematical texts.

We support this last claim with examples from Suppes's *Axiomatic set theory* [14] and Munkres's *Topology* [11]. In each case, we present our formal input with both *DZFC* and our natural language translations. Indeed, the appendices to Kieffer [9] provide a corpus of 341 definitions, taken from Chapters 2–6 of Suppes and Sections 12–38 of Munkres. Examples of the natural language translations can be found in Appendix B, below. These examples show that *PST* offers a promising target semantics for mathematical markup languages, like OMDoc [10].

To illustrate the utility of *PST*, we describe two pieces of software that take advantage of both the formal structure of the definitions and their proximity to the informal text. First, we describe statistical studies of the complexity of definitions in our corpus, measured in various ways. Our analysis shows, not surprisingly, that expanding definitions to the pure language of set theory yields formulas that are huge. Perhaps more surprisingly, quantifier complexity of definitions remains remarkably low, even when they are expanded to *DZFC*. We also describe software that makes it possible to explore definitional dependencies, expanding and compressing nodes via a graphical interface. To be sure, data like this can be mined from contemporary formal verification efforts.[5] But mathematical developments are often changed significantly in the process of formalization; what distinguishes the data presented here is the extent to which it faithfully represents the informal texts it is supposed to model.

Our "user-friendly" version of set theory is based on Friedman [6]; see also an earlier version in Friedman [5]. Most of the work described here, including the implementation of the parser, the entering of the data from Suppes's and Munkres's books, and associated software, constitute Kieffer's MS thesis [9], written under Avigad's supervision. The thesis and code described here, as well as additional samples of the natural language translations, can be found via Avigad's web page.[6]

2 *ZFC* with definitions and partial terms

It is widely acknowledged that Zermelo-Fraenkel axiomatic set theory with the axiom of choice, *ZFC*, is robust enough to accommodate ordinary mathematical arguments in a straightforward way. The most notable exceptions are category-theoretic arguments which rely on the existence of large universes with suitable closure properties; but these can be formalized in extensions of *ZFC* with suitable large cardinal axioms, or by restricting the closure properties of the universes in question.

[5] See, for example, the MPTP challenges, http://www.cs.miami.edu/~tptp/MPTPChallenge/
[6] Specifically, see http://www.andrew.cmu.edu/user/avigad/Papers/mkm/.

In this section, we describe a conservative extension *DZFC* of *ZFC*. This theory incorporates two features that allow for a more direct and natural mathematical modeling:

- it accommodates partially defined functions, and hence undefined terms; and
- it allows the introduction of new function and predicate symbols to stand for explicitly defined functions and predicates.

We describe each of these extensions, in turn.

To start with, *DZFC* is based on a free logic, with a special predicate $E(t)$. This is usually written $t\downarrow$, and can be read "t is defined" or "t denotes." The axioms governing the terms are presented as the "logic of partial terms" in Beeson [2], E^+ logic in Troesltra and Schwichtenberg [15]; see also the very helpful explanation and overview in Feferman [4]. The basic idea is that variables in the language range over objects in the intended domain (in our case, sets), but, as function symbols may denote partial functions, some terms fail to denote. So, for example, the axioms for universal instantiation are given by $\forall x\ \varphi(x) \wedge t\downarrow\ \rightarrow\ \varphi(t)$. The basic relation symbols of *ZFC*, which we take to be $\in$ and $=$, are assumed only to hold between terms that denote; thus we have axioms $s \in t \rightarrow s\downarrow \wedge t\downarrow$ and $s = t \rightarrow s\downarrow \wedge t\downarrow$. Partial equality $s \simeq t$ is defined as usual by the axiom $s \simeq t \leftrightarrow (s\downarrow \vee t\downarrow \rightarrow s = t)$.

Next, the syntax of ordinary set theory is extended to include definition descriptions, *à la* Russell. Formally, for each formula $\varphi(x)$, the expression $(\iota x)\varphi(x)$ is a term whose free variables are just those of φ, other than x. These terms are governed by the axioms

$$y = (\iota x)\varphi(x) \leftrightarrow \forall z\ (\varphi(z) \leftrightarrow z = y).$$

Thus in *DZFC* one can show that $(\iota x)\varphi(x)$ is defined if and only if there is a unique y satisfying $\varphi(y)$, in which case, $(\iota x)\varphi(x)$ is equal to that y.

Finally, one is allowed to introduce new function symbols and relation symbols to abbreviate formulas and terms. That is, for each formula $\varphi(x, \bar{y})$, one can introduce a new function symbol $f(\bar{y})$ with the axiom

$$f(\bar{y}) \simeq (\iota x)\varphi(x, \bar{y}),$$

and for every formula $\psi(\bar{y})$ one can introduce a new relation symbol $R(\bar{y})$ with the axiom

$$R(\bar{y}) \leftrightarrow \psi(\bar{y}).$$

It is not hard to show that adding the usual axioms of set theory to this framework yields a conservative extension:

Theorem 1. *DZFC is a conservative extension of ZFC.*

The proof amounts to an interpretation of partial functions and elimination of definitions that is by now standard; details can be found in [15, 9]. Note, however, that the usual method of eliminating defined function symbols and relation symbols by replacing them by their definiens can result in an exponential increase in length.

3 The language of practical set theory, *PST*

We now describe a more flexible language, *Practical set theory*, or PST, designed by Friedman. This language has two key features:

- The language incorporates a healthy amount of syntactic sugar, making it possible to express ordinary mathematical definitions and assertions in a natural way.
- The language is easily and efficiently translatable to $DZFC$.

In this section we describe some of the features of PST and the translation to $DZFC$. A full and precise specification of the PST and its $DZFC$ semantics can be found in [9, 6], where it was called the *Language of Proofless Text*, or LPT. The claims of naturality will be supported with examples in the next section and in Appendix B.

The starting point for PST is the usual syntax of first-order logic. We adopt conventions to distinguish between variables, defined functions, and relations; application of a defined relation REL to terms $t_1, \ldots, t_k$ is written with square brackets $REL[t_1, \ldots, t_k]$, while application of a defined function Fun is written with parentheses, $Fun(t_1, \ldots, t_k)$. The usual language of first-order logic is augmented with a significant amount of "syntactic sugar," to make the expression of mathematical notions as convenient as possible. These include the following.

Function application for sets. Any term may be used as though it were a function, of any arity (including "infix"). For example, one may quantify a variable f, and then proceed to use it as though it were a function. In PST, $f(x)$ denotes the unique u such that the ordered pair $\langle x, u\rangle$ is in f, assuming there is such u. The following definition of the unary predicate $\texttt{FCN}$ therefore asserts that f is a function if it is a set of ordered pairs $\langle x, u\rangle$ in which no x occurs more than once as the first component of a pair.

DEFINITION FS.2.58: 1-ary relation $\texttt{FCN}$. $\texttt{FCN}[f] \leftrightarrow f = \{\langle x, y\rangle : f(x) = y\}$.

Finite sets and tuples. In the previous example, we saw a finite tuple; namely, the ordered pair $\langle x, y\rangle$. Tuples of any finite length are terms in PST.

A finite set can be denoted by simply listing all of its elements. For example, in defining the Wiener-Kuratowski ordered pair, we may use the term $\{\{a\}, \{a, b\}\}$.

Set-builder notation. The example above illustrates the use of set-builder notation. In PST, the term $\{t : \varphi\}$ denotes the set of all values of $t(x_1, \ldots, x_n)$, where the variables $x_1, \ldots, x_n$ occurring in t range over tuples satisfying $\varphi(x_1, \ldots, x_n)$. Note that this involves an essential use of partiality; for example, in the intended semantics, the term $\{x : x = x\}$ is undefined.

Suppose we wish to define $\texttt{Image}(f)$ to be the set of all $f(x)$ such that $x \in \texttt{Dom}(f)$. The expression

$$\texttt{Image}(f) \simeq \{f(x) : x \in \texttt{Dom}(f)\}$$

is not what we want, because f on the right-hand side is taken to be a bound variable ranging over the universe of sets. Instead, *PST* has us write

$$\texttt{Image}(f) \simeq \{f(x) : x \in \texttt{Dom}(f),\ f \text{ fixed}\}$$

to indicate that the expression depends on a fixed value of f.

Defined function symbols. We use an exclamation mark in place of Russell's ι as a definite description operator. It is used in the next example, where we define an infix function, $+_{\mathbb{Q}}$, for addition on the rational numbers. Every infix function is given a *precedence* number, for use in determining order of operations.

DEFINITION FS.5.25: Infix function $+_{\mathbb{Q}}$. $x +_{\mathbb{Q}} y \simeq (!z)(x, y, z \in \mathbb{Q} \wedge (\exists a, b, c)$ $(a \in x \wedge b \in y \wedge c \in z \wedge a +_{SUB} b = c))$. Precedence 40.

A definition may be composed of any number of "If ... then ..." clauses, and may end with one "Otherwise ..." clause, which allows definition by cases, as in the example below. In this example the 'Otherwise' clause introduces a condition under which the function is undefined. For this we use the predicate $\uparrow$, and this allows for the definition of partial functions.

DEFINITION FS.2.3: 1-ary function $\texttt{Dom}$. If $\texttt{BR}[R]$ then $\texttt{Dom}(R) \simeq \{x : (\exists y)(x\ R\ y)\}$. Otherwise $\texttt{Dom}(R)\uparrow$.

Defined relation symbols. As with functions, we may define infix relations, as in the definition of $<$ on the rational numbers, below.

DEFINITION FS.5.24: Infix relation $<_{\mathbb{Q}}$. $x <_{\mathbb{Q}} y \leftrightarrow (\exists z, w)(x, y \in \mathbb{Q} \wedge z \in x \wedge$ $w \in y \wedge z <_{\texttt{SUB}} w)$.

Lambda notation. PST includes a lambda operator which can be used to bind variables and thereby denote functions. In the example below, we define a binary function called `Cartespow` (for "Cartesian power"). This function maps a pair of sets A, B to the set A^B; i.e., a product of B-many copies of A. The definition relies on a previously defined function, `Cartesprod` (for "Cartesian product"), a binary function taking a map f and a set C to the product over $c \in C$ of the sets $f(c)$. The definition of `Cartespow` uses lambda abstraction to define the constant function $b \mapsto A$ on the fly, to serve as the first argument to `Cartesprod`.

DEFINITION MunkTop.19.2.5: 2-ary function $\texttt{Cartespow}$. $\texttt{Cartespow}(A, B) \simeq$ $\texttt{Cartesprod}((\lambda b \in B)(A), B)$.

Infix relation chains. Infix relations may be chained together in the usual way, as with the $<_{\mathbb{R}}$ relation in the example below.

DEFINITION MunkTop.13.3.a.basis: 0-ary function `Stdrealtopbasis`.
`Stdrealtopbasis` $\simeq \{U \subseteq \mathbb{R} : (\exists a, b \in \mathbb{R})(U = \{x \in \mathbb{R} : a <_{\mathbb{R}} x <_{\mathbb{R}} b\})\}$.

Bounded quantifiers. Quantified variables and variables used in set-builder notation may be bounded by any infix relation, as in the example above.

The translation from *PST* to *DZFC* is not difficult. Since our grammar for *PST* is not LL, we used the ACCENT[7] compiler-compiler, which implements Earley's algorithm. The latter can parse any context-free grammar in cubic time, and runs in quadratic time when the grammar is unambiguous [1].

Appendix A contains a number of examples of *PST* definitions, together with their translations to *DZFC*. In each case, we present the *PST* input, a LaTeXrepresentation of that input generated by the parser, and the translation to *DZFC*. A much larger corpus of examples — 183 definitions from Suppes's *Axiomatic Set Theory* [14] and 148 definitions from Munkres's *Topology* [11] — can be found in [9]. In practice, the translation took at most a few seconds to process a file containing a dozen large definitions. Comparing the (Latex) *DZFC* output with the (Latex version of the) *PST* input yields a factor of about 0.91, which is to say, the *DZFC* translations are actually slightly shorter.

4 Natural language output

The examples of *PST* input in the last section are readable, but not attractive. It is hard to remember meaning of symbols "BR" or "TOPSP"; it would help to have phrases like "is a binary relation" or "is a topological space." In fact, even for logical connectives like $\wedge$, natural language equivalents like "and" are generally easier to read. In an ordinary mathematical language text, however, words are not always favored over symbols. For example, defined functions are usually given symbols: $gcd(x, y)$ instead of "the greatest common divisor of x and y." Binary relations like $=$ and $<$ are usually preferred to "equal to" and "less than." On the other hand, unary relations often represent concepts that are expanded to words, as shown by the examples above.

In light of these observations, we chose to output natural language equivalents for the connectives, and allow the user to input natural language equivalents for defined symbols. For example, with the entry

```
TOPSP:2@
  reln:$(#^0,#^1)$ is a %e?topological space%ee?@
  negn:$(#^0,#^1)$ is not a topological space@
  plur:%$(#^0,#^1)$% are topological spaces@
  nplu:%$(#^0,#^1)$% are not topological spaces@@
```

[7] http://accent.compilertools.net/

the user can specify the natural language that should be used in place of the `TOPSP` relation.

In some cases, either symbols or a natural language equivalent can be used, as in $\{x \in \mathbb{N} \mid \ldots\}$ or "the set of $x \in \mathbb{N}$ such that ...." It is usually awkward to have natural language occur as a subterm of a symbolic expression; for example, consider "1 + the greatest common divisor of x and y." Thus we incorporate a monotonicity rule: once a subterm of a term has been expanded to natural language, natural language versions are favored from then on. This choice yields, for example, $\{x \in \mathbb{N} \mid a < x < b\}$, but also "the set of x in $\mathbb{N}$ such that $a < x < b$ and x is even."

Accordingly, the user supplies two clauses for a defined function or relation for which symbols are preferred over words:

```
\wp:1@
  symb:$\wp(#^0)$@
  word:the power set of #0@@
```

whereas if words are the desired default then just one clause is needed:

```
Stdrealtop:0@
  word:the standard topology on $\mathbb{R}$@@
```

Appendix B provides examples of natural language output. We emphasize that these were generated directly from the *PST* input, using the additional natural language data, supplied by the user, described above. Although the definitions are not exactly literary, they are surprisingly readable, and close to ordinary mathematical text. It is certainly the case that additional heuristics could be used to render the output more attractive, and additional markup from the user would result in improvements. In other words, there is a lot more that can be done along these lines; our claim here is only that *PST* offers an auspicious start.

5 Exploring definitions

Among the benefits of having a database of definitions is the ability to explore those definitions interactively. We designed two simple programs with which to demonstrate some of the possibilities.

Our first program allows the interactive display and manipulation of directed acyclic graphs (dags) of conceptual dependencies, as depicted in Figure 1.

With a second program we gathered statistics on these graphs. Associated to each definition is the dag of all definitions on which it depends; by the *size* of this dag we mean the number of vertices, and by the *depth* of this dag we mean the length of its longest directed path. Table 1 shows the maximum and mean values for all definitions in our database.

Additional statistics, including data on the quantifier complexity of definitions in our corpus, can be found in Appendix C.

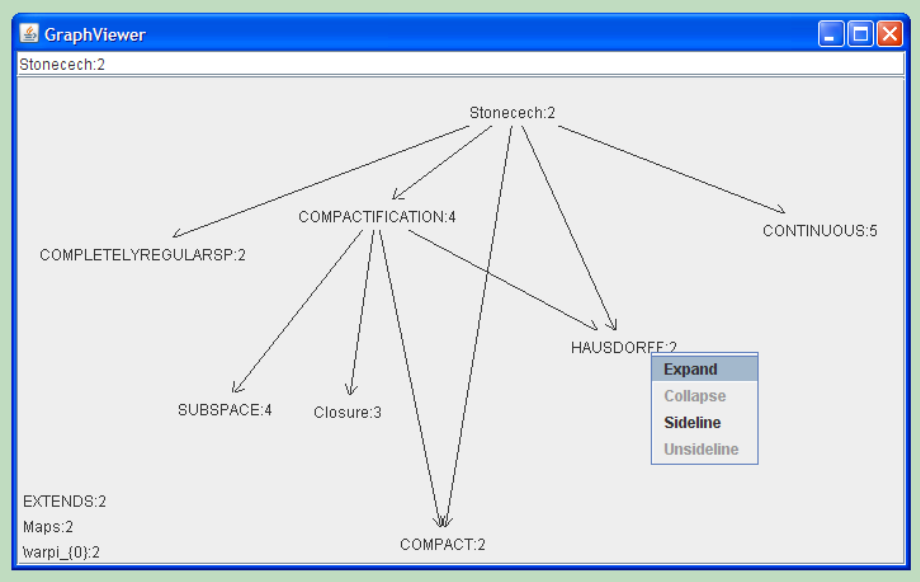

Fig. 1. Exploring the definition dag for the Stone-Čech compactification.

Table 1. Max and mean dag sizes and depths

		Max	Mean
All	Depth	32	10.77
	Size	110	29.56
Suppes	Depth	26	10.09
	Size	77	25.91
Munkres	Depth	32	12.25
	Size	110	36.01

6 Conclusions

We have argued that one should adopt a language close to definitional set theory as a uniform language to support communication and exchange of mathematical results. The particular language we describe here, *Practical set theory*, fares well in that regard: it is easy and natural to work with, providing a high-degree of readability while remaining close to a clear foundational semantics.

Appendix A: Examples of *PST* input and *DZFC* translations

We consider a few examples of formal definitions, highlighting the naturality of *PST* over *DZFC*. (The ϖ_0 function appearing in the *DZFC* translations is a function defined to take (a, b) to the Wiener-Kuratowski ordered pair $\{\{a\}, \{a, b\}\}$.)

Example 1. Here the description operator is used in *PST* to bind an ordered pair, so that we are able to refer to "the unique ordered pair $\langle Y, T' \rangle$ such that...." This translates to a much clumsier expression in *DZFC*, requiring two additional bound variables.

PST input:

```
DEFINITION MunkTop.29.4: 2-ary function Oneptcompactification.
If TOPSP[X,T] then Oneptcompactification(X,T) \simeq
(!<Y,T'>)(
  COMPACTIFICATION[Y,T',X,T] \wedge Y \less X \approx_{C} 1_{N}
).
```

PST rendered in LaTeX:

DEFINITION MunkTop.29.4: 2-ary function `Oneptcompactification`. If `TOPSP`$[X, T]$ then `Oneptcompactification`$(X, T) \simeq (!\, \langle Y, T' \rangle)$ (`COMPACTIFICATION`$[Y, T', X, T] \wedge Y \backslash X \approx_{\mathtt{C}} 1_{\mathtt{N}}$).

DZFC translation:

`Oneptcompactification`$(X, T) \simeq (\iota y_0)$(`TOPSP`$[X, T] \wedge y_0 \simeq (\iota x_0)(\exists Y, T')(x_0 = \varpi_0(Y, T') \wedge$ (`COMPACTIFICATION`$[Y, T', X, T] \wedge \approx_{\mathtt{C}} [\backslash(Y, X), 1_{\mathtt{N}}])))$

Example 2. Next observe what happens in *DZFC*, where we cannot match the brevity of expression used in our definition of the `FCN[f]` predicate in *PST* (which says that `f` is a function).

PST input:

```
DEFINITION FS.2.58: 1-ary relation FCN. FCN[f] \iff
f = {<x,y> : f(x) = y}.
```

PST rendered in LaTeX:

DEFINITION FS.2.58: 1-ary relation FCN. $\texttt{FCN}[f] \leftrightarrow f = \{\langle x, y\rangle : f(x) = y\}$.

DZFC translation:

$\texttt{FCN}[f] \leftrightarrow f = (\iota z_0)(\forall y_0)(y_0 \in z_0 \leftrightarrow (\exists x, y)(y_0 = \varpi_0(x, y) \wedge ((\iota x_0)(\varpi_0(x, x_0) \in f) = y)))$

Example 3. Here we see how important the lambda operator is:

PST input:

```
DEFINITION MunkTop.19.2.5: 2-ary function Cartespow. Cartespow(A,B)
\simeq Cartesprod((\lambda b \in B)(A),B).
```

PST rendered in LaTeX:

DEFINITION MunkTop.19.2.5: 2-ary function Cartespow. $\texttt{Cartespow}(A, B) \simeq \texttt{Cartesprod}((\lambda b \in B)(A), B)$.

DZFC translation:

$\texttt{Cartespow}(A, B) \simeq \texttt{Cartesprod}((\iota z_0)(\forall y_0)(y_0 \in z_0 \leftrightarrow (\exists b, x_0)(y_0 = \varpi_0(b, x_0) \wedge x_0 = (A) \wedge b \in B)), B)$

Appendix B: Examples of the natural language translations

In some cases our natural language generating program pst2nl produces output that is quite close to what a human being might write. For example, from the following *PST* input,

DEFINITION MunkTop.13.2: 2-ary function Basisgentop. If $\texttt{TOPBASIS}[\mathscr{B}, X]$ then $\texttt{Basisgentop}(\mathscr{B}, X) \simeq (!\mathscr{T} \subseteq \wp(X))((\forall U \subseteq X)(U \in \mathscr{T} \leftrightarrow (\forall x \in U)(\exists B \in \mathscr{B})(x \in B \wedge B \subseteq U)))$.

we get the following NL (natural language) output:

Definition: If $\mathscr{B}$ is a basis for a topology on X then *the topology on X generated by $\mathscr{B}$* is the unique $\mathscr{T} \subseteq \wp(X)$ such that for every $U \subseteq X$, $U \in \mathscr{T}$ if and only if for every $x \in U$, there exists $B \in \mathscr{B}$ such that $x \in B$ and $B \subseteq U$.

What is more common is that the output of pst2nl reads nicely except for a "run-on" sound, resulting from insufficient punctuation. For example:

Definition: If R is a strong simple order on X then *the basis for the order topology on (X, R)* is the set of U such that there exist a, $b \in X$ such that $U = (a, b)$ or a is a first element in X and $U = [a, b)$ or b is a last element in X and $U = (a, b]$.

Heuristics, combined with additional user markup, could eventually be incorporated to help improve the flow and punctuation of the translations. We have implemented one easy improvement already, whereby adjacent assertions of a common predicate are combined into a single assertion using plural form. Thus, from the *PST* input,

DEFINITION MunkTop.12.4.a: 3-ary relation `FINERTOP`. If $\texttt{TOPSP}[X, \mathscr{T}] \wedge \texttt{TOPSP}[X, \mathscr{T}']$ then $\texttt{FINERTOP}[\mathscr{T}', \mathscr{T}, X] \leftrightarrow \mathscr{T}' \supseteq \mathscr{T}$.

we obtain:

Definition: If $(X, \mathscr{T})$ and $(X, \mathscr{T}')$ are topological spaces then $\mathscr{T}'$ is *finer* than $\mathscr{T}$ on X if and only if $\mathscr{T}' \supseteq \mathscr{T}$.

We consider a final example.

DEFINITION MunkTop.13.3.c: 0-ary function `Krealtop`. $\texttt{Krealtop} \simeq \texttt{Basisgentop}(\texttt{Stdrealtopbasis} \cup \{V \subseteq \mathbb{R} : (\exists W \in \texttt{Stdrealtopbasis})$ $(V = W \backslash \{\texttt{Incl}_{\texttt{FrR}}(1_{\texttt{N}}/n) : n \in \mathbb{N}\})\}, \mathbb{R})$.

The NL output is as follows:

Definition: *The K-topology on* $\mathbb{R}$ is the topology on $\mathbb{R}$ generated by the standard basis for a topology on $\mathbb{R}$ union the set of $V \subseteq \mathbb{R}$ such that there exists W in the standard basis for a topology on $\mathbb{R}$ such that $V = W \setminus \{1/n : n \in \mathbb{N}\}$.

There are two sets mentioned in this definition: the set of $V \subseteq \mathbb{R}$ such that ..., and the set of $1/n$ such that .... According to the "monotonicity rule" described in Section 4, the latter is rendered in symbols since it has no subterm in words; the former is rendered in words since its subterm, "the standard basis for a topology on $\mathbb{R}$" has no symbolic form, and is displayed in words by default.

Another feature of `pst2nl` is apparent in this last example, where the word "in" appears before "the standard basis...." We get this preposition rather than the incorrect phrase "is in," thanks to the final clause in the user-supplied natural language equivalents for the $\in$ relation:

```
\in:infix@
  symb:#0 $\in$ #1@
  nsym:#0 $\not\in$ #1@
  reln:#0 is %e?in%ee? #1@
  negn:#0 is not in #1@
  plur:%#0% are in #1@
  nplu:%#0% are not in #1@
  prep:#0 in #1@@
```

Finally we note that the user is free to suppress artifacts of formalization, in the NL output. In the *PST* above there is an inclusion function $\texttt{Incl}_{\texttt{FrR}}$, and the number 1 is subscripted as $1_{\texttt{N}}$. None of this shows up in the NL output.

Appendix C: Data on quantifier complexity and length

Our database of definitions entered in *PST* consists of 183 definitions from Suppes's *Axiomatic Set Theory* [14] and 148 definitions from Munkres's *Topology* [11].

Quantifier complexity data. For each definition in our database, we measured quantifier complexity in eight different ways. In the first place, we considered both alternating quantifier depth, and non-alternating. Secondly, we considered each definition in four different states: (1) as given in *PST*; (2) as translated into *DZFC*; (3) the *expanded* version of the *DZFC*, that is, with all definienda replaced by their definiens, recursively, until the process halts; and (4) a *partially expanded* version of the *DZFC* in which certain low-level, foundational definienda were left unexpanded, namely: the union, intersection, and set difference operations, the ordered pair, and powerset functions, the empty set, and the subset and superset relations. The maximum and mean depths are presented in Table 2.

Table 2. Max and mean quantifier depths

	Max	Mean
PST	4	0.66
unexpanded *DZFC*	5	1.31
fully expanded *DZFC*	1235	78.68
partially expanded *DZFC*	552	38.54
PST alternating	3	0.63
unexpanded *DZFC* alternating	5	1.18
fully expanded *DZFC* alternating	422	36.19
partially expanded *DZFC* alternating	239	22.16

It has been said that among actually occurring definitions in mathematics texts, the maximum alternating quantifier depth is three. Insofar as *PST* comes close to what actually occurs in textbooks, the maximum alternating depth of 3 tends to confirm this conjecture.

Note that the maximum depth after translating into *DZFC* goes up to 5. This reflects what we saw in Appendix A, where a definition that used no quantifiers in *PST* turned out to require them after translation into *DZFC*.

The maximum depth of 1235 for a fully expanded definition confirms the necessity of using definitions to package information into manageable chunks. Meanwhile, the contrast between the total expansion maximum, and the partial expansion maximum of 552, demonstrates that the lowest, most foundational definitions, lend quite a bit of this complexity.

The ratio $78.68/36.19 \approx 2.17$ of the mean fully expanded depth to the mean fully expanded alternating depth suggests that quantifiers often occur in runs of two, before alternating, when definitions are written in pure set theory. The

somewhat lower ratio of $38.54/22.16 \approx 1.74$ for the partially expanded cases indicates the extent to which the lowest-level concepts contribute to this doubling of consecutive quantifiers.

The mean depth for *PST* alternating (again, what comes closest to what we ordinarily think of as quantifier depth in textbooks) shows that, while the maximum is three, the most common depths are 0 and 1. The exact number of occurrences are presented in Table 3.

Table 3. Quantifier depth frequencies in *PST*

	Occurrences	
Depth	*PST*	*PST* alternating
0	178	178
1	118	120
2	30	35
3	14	8
4	1	0

Length data. As was expected, there is rapid blowup in the size of definitions when they are expanded. In collecting our data we set a maximum of $2^{31} - 1$ before we stopped counting, and this maximum was often reached.

In particular, since the development of the real numbers taken from Suppes [14] involves such deep definition trees, any definition mentioning the real numbers will have enormous expanded length. For example, the definition of the basis for the standard topology on the reals (see Section 3) is just 303 symbols long after initial translation into *DZFC*, but blows up to over $2^{31} - 1$ symbols after expansion.

The longest definition we formalized from Suppes [14] was 526 symbols, and the longest from Munkres [11] was 714.

References

1. Alfred V. Aho and Jeffrey D. Ullman *The Theory of Parsing, Translation, and Compiling, volume 1.* Prentice-Hall, Englewood Cliffs, N. J., 1972.
2. Michael J. Beeson. *Foundations of Constructive Mathematics.* Springer, Berlin, 1985.
3. Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: The calculus of inductive constructions.* Springer, Berlin, 2004.
4. Solomon Feferman. Definedness. *Erkenntnis*, 43(3):295–320, 1995. Varia with a Workshop on the Foundations of Partial Functions and Programming (Irvine, CA, 1995).
5. H. Friedman and R. C. Flagg. A framework for measuring the complexity of mathematical concepts. *Adv. in Appl. Math.*, 11(1):1–34, 1990.
6. Harvey Friedman. Proofless text. Manuscript, September 29, 2005.

7. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher-order logic*. Cambridge University Press, 1993.
8. John Harrison. HOL light: a tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*, pages 265–269, 1996.
9. Steven Kieffer. A language for mathematical knowledge management. Master's thesis, Carnegie Mellon University, 2007.
10. Michael Kohlhase. *OMDoc: An open markup format for mathematical documents*, volume 4810 of *LNAI*. Springer, Berlin, 2006.
11. James R. Munkres. *Topology*. Prentice Hall, Upper Saddle River, N.J., second edition.
12. Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Berlin, 2002.
13. P. Rudnicki. An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*. Chalmers University of Technology, Bastad, 1992.
14. Patrick Suppes. *Axiomatic Set Theory*. Van Nostrand, Princeton, 1960.
15. A. S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, Cambridge, second edition, 2000.
16. Makarius Wenzel. Isabelle/Isar — a generic framework for human-readable proof documents. *Studies in Logic, Grammar, and Rhetoric*, 10(23), 2007. From Insight to Proof — Festschrift in Honour of Andrzej Trybulec, edited by R. Matuszewski and A. Zalewska.