Interpreting Mathematical Texts in Naproche-SAD

Adrian De Lon, Peter Koepke(✉), and Anton Lorenzen

Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, Germany
koepke@math.uni-bonn.de

Abstract. Naproche-SAD is a *natural proof assistant* based on the controlled natural input language ForTheL. Integrating ForTheL into LaTeX allows to leverage type setting commands for the disambiguation and structuring of mathematical texts, with high-quality mathematical typesetting coming for free. A new generic parsing mechanism allows the translation of texts into other formal languages besides the original first-order internal format of Naproche-SAD. We can generate correct Lean code from ForTheL statements which may be useful for writing readable fabstracts.

1 Natural Proof Assistants

Leading proof assistants have enabled spectacular successes like fully formal and certified proofs of the four-color theorem or of the Kepler conjecture. On the other hand proof assistants have so far not been widely adopted in mathematical practice since their input languages look like conventional programming languages, use unfamiliar foundations and require a lot of detail that seem to be mathematically irrelevant (see also [15]).

To facilitate the use of formal methods in the mathematical community at large proof assistants should employ:

1. input languages which are close to the mathematical vernacular, including symbolic expressions;
2. familiar text structurings that support, e.g., the axiomatic definition-theorem-proof approach;
3. underlying logics that correspond to strong foundations in set theory or type theory;
4. automatic handling of tedious formalization details that are usually left implicit;
5. strong automatic proof checking to approximate proof granularities found in the mathematical literature.

In principle, these points were already addressed in the early years of interactive theorem proving, e.g., in the Mizar project [7], see also [4]. Other proof assistants have implemented Mizar-like proof languages with declarative proof structures [14]. Note, however, that the Mizar language is a restricted formal language that is *not* part of commonly used mathematical English.

To reach an even higher degree of naturality, a small number of experimental proof assistants accept proofs in controlled natural languages (CNL) which are fully formal subsets of common natural English (with symbolic mathematical terms). Moreover, input texts may be structured just like proof texts in the published mathematical literature. This development should eventually lead to systems that one may term *natural proof assistants*. In this paper we highlight some aspects of points 1–3 in view of recent improvements [12] to the previous Naproche-SAD release [3]. More technical details are contained in an informal system description that we are also submitting to this conference.

2 Naproche-SAD and ForTheL

The Evidence Algorithm project (EA) which was started by V. Glushkov was inspired by the idea of a system to assist actual mathematical work [13]. It was centered around the development of a controlled natural language for mathematics called ForTheL (Formula Theory Language). The project culminated in the implementation of the proof assistant SAD (System for Automated Deduction) in the PhD work of Andrei Paskevich [11].

Independently, the Naproche (Natural Proof Checking) initiative [10] developed a controlled natural language on top of classical first-order logic, with an emphasis on techniques from formal linguistics. The PhD thesis of Marcos Cramer demonstrated that formal grammars and discourse representation theory could deal adequately and efficiently with mathematical proof texts [1].

A few years ago Naproche has adopted and extended the ideas and algorithms of SAD (see [2], [5]) because of SAD's superior logical setup and performance. Naproche-SAD accepts and proof-checks texts like

Definition 1. *A natural number p is prime iff $p \neq 0, 1$ and for every k such that $k \mid p$, we have $k = p$ or $k = 1$.*

Theorem 1 (Euclid's lemma). *If p is prime and $p \mid m \cdot n$ then $p \mid m$ or $p \mid n$.*

By stripping away pretty-printing commands in the LaTeX source of this text fragment one obtains a valid text in the ForTheL proof language for Naproche-SAD. This can be done by a simple filter or by hand. The stripped text proof-checks in Naproche-SAD within the context of a larger file that formalizes a sufficient amount of arithmetic and contains a standard proof of the theorem. Such formalizations are representative of a growing library of proof-checked ForTheL texts from various fields of mathematics [9].

ForTheL, the proof language of Naproche-SAD, is a controlled natural language (CNL). Its design goes back to the 1980s and was based on extensive studies of published mathematical texts. It was found that a large part of mathematical language can be simply built up from fixed patterns which consist of (multiple) words and symbols. Formal production rules of the ForTheL grammar are based on patterns without further analysis of its constituent tokens. On the other hand ForTheL gives a lot of freedom for the creation of patterns, allowing rather arbitrary ASCII sequences as tokens.

For the above sample text, the pattern "natural number $(-)$" with a slot $(-)$ for some other term can be introduced by a language extension of the form

```
Signature. A natural number is a notion.
```

Internally this generates a unary predicate `aNaturalNumber()` which can be addressed by phrases like "x is a natural number" or "for all natural numbers". Note that after "natural number" is parsed there is no attempt to break this down into "natural" and "number". This corresponds to the mathematical practice of taking "natural number" as an "atomic" notion whose meaning cannot be derived from meanings of "natural" and "number".

3 ForTheL and LaTeX

Mathematical typography and typesetting is a prominent part of mathematical culture. An iconic formula like $e^{i\pi} = -1$ uses non-Latin letters (π) and a two-dimensional arrangement of letters to denote certain constants and operations. Typography and typesetting carry semantic information that is utilizable in mathematical text processing.

These days mathematicians routinely do their own typesetting using LaTeX or related software. LaTeX has become the universal format for editing and exchanging mathematics. It provides fonts and symbols to distinguish many mathematical objects and notions. Environments like `\begin{theorem} ... \end{theorem}` mark statements to be proved and define scopes for assumptions and variable declarations. The original ForTheL language has only primitive theorem and proof environments. Special symbols have to be simulated by "ASCII-art".

Therefore we are integrating ForTheL into LaTeX, extending features of the original Naproche input language. This work uses some previous experiences with the Naproche input language. A grammatically correct ForTheL text is supposed to be a valid LaTeX file in the context of appropriate document classes and packages. Further benefits will be achieved by semantically enriched versions of LaTeX.

The previous version of Naproche-SAD [3] used a parser which employed an internal parser combinator library. We are replacing the ASCII syntax with a LaTeX-based syntax. The old parser had some logical transformations interwoven with the parsing process, assuming that the target would only be first-order logic. We have now separated the parser module from further logical processing. This required a change of internal representations. Whereas the old parser produced blocks of tagged first-order formulas, we replaced this with a higher-level abstract syntax tree, which is not committed to any particular foundational framework and is also amenable to type-theoretic semantics.

Text Mode and Math Mode. A characteristic feature of ordinary mathematical language is the intuitive distinction between ordinary text and specific mathematical terms and phrases. In LaTeX, this is reflected by commands for switching between text mode and math mode. This distinction is often crucial for disambiguations like between the article "a" and the mathematical variable "a", recognizable by different typesettings. In the old parser, patterns definitions such as "the closure of X as a metric space" resulted either in the unintended pattern "the closure of (−) as (−) metric space" or gave a nondescript parser error.

Another example is that the phrase "vector space" may be parsed as the structure vector space or as a vector named space, provided that both vectors and vector spaces have been defined previously. This led to surprising errors, requiring awkward rephrasings to fix. With the new syntax, variables only occur within math environments which removes many ambiguities.

Ease of Learning and Compatibility. A significant advantage of the new syntax is that most mathematicians are already comfortable with using LaTeX, which eases the learning curve for the CNL. We can also re-use some of the existing tooling around LaTeX: editors, syntax-highlighters, bibliography managers, metadata extractors, etc.

Generating Documents. We provide a custom LaTeX package that makes a CNL text a valid LaTeX document. This way we get prettyprinted documents for free! Furthermore we allow patterns to contain LaTeX commands. The above sample text is prettyprinted from the following *ForTheL* source:

```
\begin{definition}
  A natural number $p$ is prime iff $p \neq 0, 1$
  and for every $k$ such that $k \divides p$,
  we have $k = p$ or $k = 1$.
\end{definition}
\begin{theorem}[Euclid's lemma]
  If $p$ is prime and $p \divides m\mul n$
  then $p \divides m$ or $p \divides n$.
\end{theorem}
```

Expression Parsing. The old syntax made no distinction between symbolic expressions and word patterns, both were parsed as patterns. This approach was flexible, allowing free mixing of words and symbols in patterns. The downside was that parsing of symbolic expressions was complicated and had no mechanism for operator precedences. With the new distinction between math and text content, it seems natural to investigate alternative approaches. We are currently experimenting with more traditional precedence-based expression parsers.

Expression parsing is complicated by allowing relator chaining, as in "$a < b < c = d$". Some operators (e.g. logical connectives) should have lower precedence than relators, and some should have higher precedence (e.g. arithmetic operations). We address this by maintaining two operator tables, along with the list of relators, and parsing expressions in three steps.

Introduction of Grammatical Number. Naproche-SAD used to have no concept of grammatical number, treating singular and plural forms completely synonymously. This can lead to ambiguities. For example, treating "is"/"are" synonymously in "the maximum of x and y is/are smaller than z" creates an ambiguity; with the first interpretation being "(the maximum of x and y) is smaller than z" and the second interpretation being "(the maximum of x) and y are smaller than z", where the maximum is understood as an operation on a list or set. This ambiguity can be resolved with grammatical number.

4 ForTheL and Types

ForTheL is a language with soft types which are called *notions*. One can introduce notions like *integer* and modify them with adjectives like *positive*. In the current Naproche-SAD system dependent notions in n parameters are translated into $(n+1)$-ary relation symbols and processed in classical first-order logic.

Since many established interactive theorem provers are based on type theories, it appears natural to translate ForTheL into (dependent) type theory. Parsing of ForTheL texts should yield an internal representation that can be translated alternatively into FOL or type theory.

Translating to Lean. Lean [8] is a proof assistant with a growing library of mathematical texts [6] from a wide variety of undergraduate courses up to some formalizations of research mathematics. We have implemented a translation from the new syntax to Lean definitions. We include some predefined commands mapping to basic definitions from Leans stdlib and mathlib, such as `\naturals`, `\rationals`, `\divides`, etc.

The above sample text renders as the correct and idiomatic Lean fragment:

```
def prime (p : ℕ) : Prop :=
  p ≠ 0 ∧ p ≠ 1 ∧ (∀ k, has_dvd.dvd k p -> k = p ∨ k = 1)
theorem euclids_lemma {p} {n} {m} : prime p ∧ has_dvd.dvd p (m * n)
  -> has_dvd.dvd p m ∨ has_dvd.dvd p n := omitted
```

The Transformation. We translate pattern definitions to definitions of propositions, and theorems to Lean-theorems. Premises become arguments and the statement the result type of the theorem. Patterns inside the premises and statement will not be unrolled, but rather refer to the Lean definitions defined previously (like `prime` above). We use the optional argument of the amsthm environments (in this case Euclid's lemma) to pick an appropriate Lean name, with a fallback to `thm0`, `thm1`, etc.

5 Future Work

The naturalness of an interactive system is the result of a large number of small natural features. We shall continue to enhance Naproche-SAD in this direction.

One important example is the handling of common type coercions: Given a rational number q and a natural number n, the expression $q = n$ type-checks in Lean thanks to implicit coercions, while $n = q$ does not, as Lean cannot "undo" the specialization of $=$ to the natural numbers. Having coercions depend on the order of arguments is undesirable for natural language texts. While it is of course possible for users to supply coercions manually, we plan on addressing this issue by adding a system that manages subtyping relations. We are also evaluating the possibility of translating ForTheL proofs to Lean tactics.

References

1. Cramer, M.: Proof-checking mathematical texts in controlled natural language. Ph.D. thesis, University of Bonn (2013)
2. Frerix, S., Koepke, P.: Automatic proof-checking of ordinary mathematical texts. In: CICM Informal Proceedings (2018). http://ceur-ws.org/Vol-2307/paper13.pdf
3. Frerix, S., Wenzel, M., Koepke, P.: Isabelle/Naproche (2019). https://sketis.net/2019/isabelle-naproche-for-automatic-proof-checking-of-ordinary-mathematical-texts
4. Harrison, J., Urban, J., Wiedijk, F.: Interactive theorem proving. In: Gabbay, D.M., Siekmann, J., Woods, J. (eds.) Computational Logic of the Handbook of the History of Logic, vol. 9, pp. 266–290. Elsevier, Amsterdam (2014)
5. Koepke, P.: Textbook Mathematics in the Naproche-SAD System. In: CICM Informal Proceedings (2019). http://cl-informatik.uibk.ac.at/cek/cicm-wip-tentative/FMM4.pdf
6. Lean community: The Lean mathematical library. https://github.com/leanprover-community/mathlib
7. Mizar. http://mizar.org/
8. de Moura, L., Kong, S., Avigad, J., van Doorn, F., von Raumer, J.: The Lean theorem prover. In: Automated Deduction - CADE-25 (2015)
9. Naproche community: A ForTheL Library. https://github.com/naproche-community/FLib
10. Naproche. https://korpora-exp.zim.uni-duisburg-essen.de/naproche/
11. Paskevich, A.: Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques. Ph.D. thesis, Université Paris 12 (2007)
12. Prototype CNL. https://github.com/adelon/nave
13. Glushkov, V.M.: Some problems in the theories of automata and artificial intelligence. Cybern. Syst. Anal. **6**, 17–27 (1970). https://doi.org/10.1007/BF01070496
14. Wenzel, M.: Isabelle/Isar - a versatile environment for human-readable formal proof documents. Ph.D. thesis, TU Munich (2002)
15. Wiedijk, F.: The QED manifesto revisited. In: From Insight to Proof, Festschrift in Honour of Andrzej Trybulec, pp. 121–133 (2007)