The Naproche System

D. Kühlwein, M. Cramer, P. Koepke, and B. Schröder

Mathematical Institute, University of Bonn
German Linguistics, University of Duisburg-Essen
{cramer,koepke,kuehlwei}@math.uni-bonn.de
bernhard.schroeder@uni-due.de
http://www.naproche.net

Abstract. The Naproche project (Natural language Proof Checking) was initiated by Bernhard Schröder and Peter Koepke at the University of Bonn to focus on an interdisciplinary study of the semi-formal language of mathematics. A central goal of Naproche is to develop a controlled natural language (CNL) for mathematical texts and adapted proof checking software which checks texts written in the CNL for syntactical and mathematical correctness. The project is still at a prototypical stage, further information is available at *www.naproche.net*.
This paper describes the Naproche system, an implementation of the ideas developed by the Naproche project. The Naproche system accepts LaTeX-style texts, consisting of mathematical formulas imbedded in a controlled natural language. Texts written in the controlled natural language are parsed using techniques from computational linguistics and transformed into first-order formulas. The formulas are given to an automatic theorem prover which checks whether each formula of an argument is a logical consequence of the preceding formulas or axioms.

Key words: Controlled natural language, formal mathematics, discourse representation theory, automated theorem proving

1 Introduction

Considering that mathematics has a reputation of being an exact science, it is interesting to note that one of the main concepts of mathematics, the mathematical proof, is somewhat vaguely defined. What exactly is a mathematical proof? Firstly, one can distinguish between two kinds of mathematical proofs: Formal and informal proofs.

Formal proofs are finite derivations in a calculus. They are sequences of mathematical symbols and do not contain natural language elements. Given a calculus, there is a definite answer whether or not a text is a formal proof.

An informal proof is a mixture of natural language and mathematical symbols. Most proofs in mathematical textbooks and journals are informal. Contrary to formal proofs, there is no clear criteria whether a text constitutes an informal proof or not.

Ever since Alfred Whitehead's and Bertrand Russells's work Principia Mathematica [16] most mathematicians agree that it would be possible, even if extremely tedious, to find a formal proof for every theorem. With this in mind, one could interpret informal proofs as abbreviations for formal proofs.

Facilitating methods from (computational) linguistics, computer mathematics and mathematics, the Naproche project (NAtural language PROof CHEcking), a joint initiative of Peter Koepke (Mathematics, University of Bonn) and Bernhard Schröder (Linguistics, University of Duisburg-Essen), studies the interplay between formal and informal proofs.

1.1 The Naproche System

As part of the Naproche project, we develop the Naproche system, a program which aims to automatically translate informal proofs into their formal counterparts.

Of course, one first has to define what exactly a translation of an informal proof is. Our approach is to see an informal proof as a blueprint for a formal proof. The major stepstones in the derivation are given, and all that is left to do is to flesh out each single step. To do this, we first transform the informal proof into a sequence of first order logic statements. Once an input text is translated, we use automated theorem provers (ATPs) to fill out the gaps. Proofs created by ATPs are basically derivations in a calculus, and therefore this gives us a method of creating a formal from an informal proof.

Given that this translation from natural language into first order logic is sound, this procedure also gives a method of checking the correctness of informal proofs.

There are several challenges which one has to face when trying to implement such a program. Firstly, one has to teach the computer to automatically interpret statements in natural language correctly. The fact that natural language is often very ambiguous and that there are usually several different ways of stating the same fact only adds to the difficulty. Another problem is the definition of the translation algorithm from informal proofs into first order logic. And finally, one has find a way to extend the first order translation of a text to a full formal proof.

In order to handle the ambiguity of natural language texts, we developed a controlled natural language (CNL) for mathematics, the Naproche CNL, which has a clearly defined translation from texts written in this language into first order formulas. We use an adapted version of Discourse Representation Structures (DRS), which we call Proof Representation Structures (PRS), to extract the first order representation of a Naproche CNL text. To fill the gaps in the proof, we implemented a checking algorithm for PRS that creates the appropriate proof obligations for the ATP.

We will present each part of the Naproche system in more detail.

1.2 Related Work

While, to the authors knowledge, there is no other group that focuses on the connection between informal and formal proofs, there are already several proof checking systems that also emphasize the readability of their respective input language.

A. TRYBULEC's Mizar [7] is arguably the most prominent. It was started in 1973, and by today several non-trivial mathematical theorems have been proved (e.g GÖDEL's completeness theorem [2]). An active community continues to formulate and prove theorems in Mizar. The results are published regularly in the journal *Formalized Mathematics*.

The Isabelle [8] team is working on Isar [15], a "human-readable structured proof language". The System for Automated Deduction (SAD, [12]) checks texts that are written in its input language, ForThel [13], for correctness.

We are collaborating with the VeriMathDoc project [1], which includes the PLATO program [14]. Their goal is to develop a mathematical assistant system that naturally supports mathematicians in the development, authoring and publication of mathematical work.

2 The Architecture of the Naproche System

The Naproche system consists of three main modules: The User Interface, the Linguistics module and the Logic module. Figure 1 shows an overview of the architecture of the Naproche system.

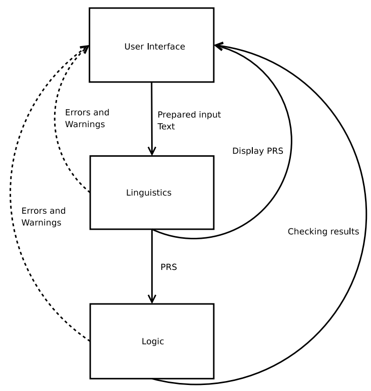

Fig. 1. The architecture of the Naproche system.

The User Interface is the standard communication gateway between the user and the Naproche system. The input text is given to the User Interface, and the other modules report back to the User Interface. The module transforms the input text into a prolog list and hands it over to the Linguistics module. Currently, our User Interface is web-based and can be found on the Naproche website, *www.naproche.net*. We are also working on a plugin for the WYSIWYG editor $\text{T}_\text{E}\text{X}_{\text{MACS}}$ [11]. The input language is described in section 3.

The Linguistics module creates a PRS from the input text. Any errors or warnings which occur during the creation are reported back to the User Interface. The created PRS can either be given to the Logic module, or reported back to the User Interface. Section 4 explains PRSs in greater detail.

With the help of ATPs, the Logic module checks PRSs for logical correctness. As in the Linguistics module, errors or warnings which occur during the creation are reported back to the User Interface. The result of the check is sent to the User Interface. A brief overview of the checking algorithm is given in section 5.

3 The Naproche CNL

Natural language is usually full of ambiguities and heavily context dependent. This makes extracting the semantics of an arbitrary text written in common English a very hard, if not impossible, task. Controlled natural languages are subsets of natural languages, which usually try to reduce complexity and eliminate ambiguity while preserving the the relevant parts of the expressiveness of the original language. They give the user the means to 'easily' process and work with text written in such a language.

Attempto Controlled English (ACE) [3] showed that this concept can be very successful. Texts written in ACE read like normal English while having a unique first order representation which can be used for further tasks, e.g. reasoning or queries over the text.

The Naproche CNL aims to be the mathematical equivalent of ACE, i.e. to be as intuitive and expressive as the semi-formal language of mathematics as used in textbooks and journals, while defining an unambiguous translation into first order logic.

In the current version of the Naproche CNL, a text is structured by structure markers: *Axiom*, *Theorem*, *Lemma*, *Proof* and *Qed*. For example, a theorem is presented after the structure marker *Theorem*, and its proof follows between the structure markers *Proof* and *Qed*.

First order formulas can be combined with natural language expressions to form the usual mathematical constructs like statements, definitions, implications and assumptions.

Assumptions are always opened by an assumption trigger (e.g. *let*, *consider*, *assume that* ...), and closed by a sentence starting with *thus* or by a *Qed*. Definitions always start with *define*. Assertions are made by a first order formula or a naturally predicated term (e.g. *x is an ordinal*), optionally preceeded a statement trigger (e.g. *then*, *hence*, *therefore*, *so*, ...). Additionally, negation, conjunction, disjunction, quantification and implication may be expressed in natural language.

Since most mathematicians use LaTeX for writing papers, we tried to keep the Naproche CNL very close to original LaTeX code.

As an example, we present a short proof written in the Naproche CNL.

```
Define $Trans(x)$ if and only if $\forall u, v (u \in v \and
v \in x) \implies  u\in x$.
Define $Ord(x)$ if and only if $Trans(x) \and \forall y (y \in x
\implies Trans(y))$.

Theorem.
For all $x$, $y$, if $x \in y$ and $Ord(y)$ then $Ord(x)$.

Proof.
Suppose $x \in y$ and $Ord(y)$. Then $Trans(x)$.
Assume that $u \in x$. Then $u \in y$.
Hence $Trans(u)$. Thus $Ord(x)$.
Qed.
```

Note that formal quantification and implication is used in the definitions, whereas the statement of the theorem is written using natural language quantification and implication.

Substituting \and with \wedge and \implies with \rightarrow gives LaTeX compilable text:

Define Trans(x) if and only if $\forall u, v(u \in v \wedge v \in x) \rightarrow u \in x$.
Define Ord(x) if and only if Trans(x) $\wedge \forall y(y \in x \rightarrow$ Trans(y)).

Theorem.
For all x, y, if $x \in y$ and Ord(y) then Ord(x).
Proof.
Suppose $x \in y$ and Ord(y). Then Trans(x). Assume that $u \in x$. Then $u \in y$. Hence Trans(u). Thus Ord(x).
Qed.

4 Proof Representation Structures

While extracting the semantics of a CNL is definitely easier than extracting the semantics of a text in unrestricted natural language, it is still no trivial task. The more sophisticated the CNL gets, the more advanced approaches need to be used.

Computational linguistics has developed several techniques to automatically extract the semantics of a natural language text. We adapted the Discourse Representation Structures (DRS, [4]) from Discourse Representation Theory, which are also used in the *Attempto* project [3], for our needs, and called the result *Proof Representation Structures*[1] (PRSs, see [5], [6]). PRSs are Discourse Representation Structures that are enriched in such a way as to represent the distinguishing characteristics of the semi-formal language of mathematics.

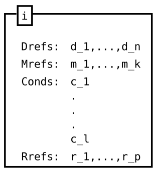

Fig. 2. A PRS with identification number i, discourse referents $d_1, ..., d_n$, mathematical referents $m_1, ..m_k$, conditions $c_1, ..., c_l$ and textual referents $r_1, ..r_p$

A PRS has five constituents: An identification number, a list of discourse referents, a list of mathematical referents, a list of textual referents and an ordered list of conditions. Similar to DRSs, we can display PRSs as “boxes” (See fig. 2).

Mathematical referents are the terms and formulas which appear in the text. As in DRSs, discourse referents are used to identify objects in the domain of the discourse. However, the domain contains two kinds of objects: mathematical objects like numbers or sets, and the symbols and formulas which are used to refer to or make claims about mathematical objects. Discourse referents can identify objects of either kind.

PRSs have identification numbers, so that they can be referred to at some later point. The textual referents indicate the intratextual and intertextual references.

Just as in the case of DRSs, PRSs and PRS conditions have to be defined recursively: Let A, B be PRSs, f be a function symbol, $X, X_1, \dots, X_n$ discourse referents, Y a mathematical referent, and let *Word* denote an English noun, adjective or verb. Then

- $holds(X)$ is a condition representing the claim that the formula referenced by X is true.
- $math\_id(X, Y)$ is a condition which binds a discourse referent to a mathematical referent (a formula or a term).
- A is a condition.
- $\neg A$ is a condition, representing a negation.
- $A := B$ is a condition, representing a definition.
- $A => B$ is a condition, representing an implication or universal quantification.
- $A <=> B$ is a condition, representing an equivalence statement.

[1] Note that even though Claus Zinn [17] also uses the term *Proof Representation Structure*, the definitions of Naproche and Zinn are different.

- $A <= B$ is a condition, representing a B if A statement.
- $A \vee B$ is a condition, representing a disjunction.
- $A ==> B$ is a condition, representing an assumption.
- $f :: A => B$ is a condition, representing a function definition.
- *contradiction* is a condition, representing a contradiction.
- $predicate(X, Word)$ is a condition, representing a natural language statement with one argument.
- $predicate(X_1, X_2, Word)$ is a condition, representing a natural language statement with two arguments.

Note that contrary to the case of DRSs, a bare PRS can be a direct condition of a PRS. This allows to represent the nested structure of mathematical texts in a PRS: The different building blocks of a text (axioms, definitions, lemmas, theorems, proofs), that are denoted by structure markers, correspond to sub-PRSs (See fig. 3 for an example). The hierarchical structure of assumptions is represented by nesting conditions of the form $A ==> B$: A contains an assumption, and B contains the representation of all claims made inside the scope of that assumption.

The algorithm creating PRSs from CNL proceeds sequentially: It starts with the empty PRS. Each sentence or structure marker in the discourse updates the PRS according to an algorithm similar to a standard DRS construction algorithm but taking the nesting of assumptions into account. [5]

The PRS constructed from the example proof is shown in figure 3.

5 Checking PRS

The checking algorithms keeps a list of first order formulas it considers to be true, called premises, which gets continuously updated during the checking process.

To check a PRS, the algorithms considers the conditions of the PRS. The conditions are checked sequentially and each condition is checked under the currently active premises. According to the kind of condition, the Naproche systen creates obligations which have to be discharged by an automated theorem prover. If all obligations of a condition can be discharged, then the condition is proved to be valid. After a conditions is deemed valid, the active premises get updated, and the next condition gets checked. A PRS is accepted by Naproche if all its conditions are valid.

In our example PRS (Fig. 3), the first condition is a definition. As this is the first condition of the PRS, the sequence of currently active premises is empty. Since definitions do not have to be checked, all we have to do is store the first order representation of this definition as a premise for future use. So after this condition is processed, the sequence of currently active premises contains the formula $\forall x\, Trans(x) \leftrightarrow (\forall u, v(u \in v \wedge v \in x) \rightarrow u \in x)$.

The second conditions is also a definition and therefore gets treated the same way. The sequence of active premises after this condition is

$$\begin{gathered}[\forall x\, Trans(x) \leftrightarrow (\forall u, v(u \in v \wedge v \in x) \rightarrow u \in x),\\ \forall x\, Ord(x) \leftrightarrow (Trans(x) \wedge \forall y(y \in x \rightarrow Trans(y)))]\end{gathered}$$

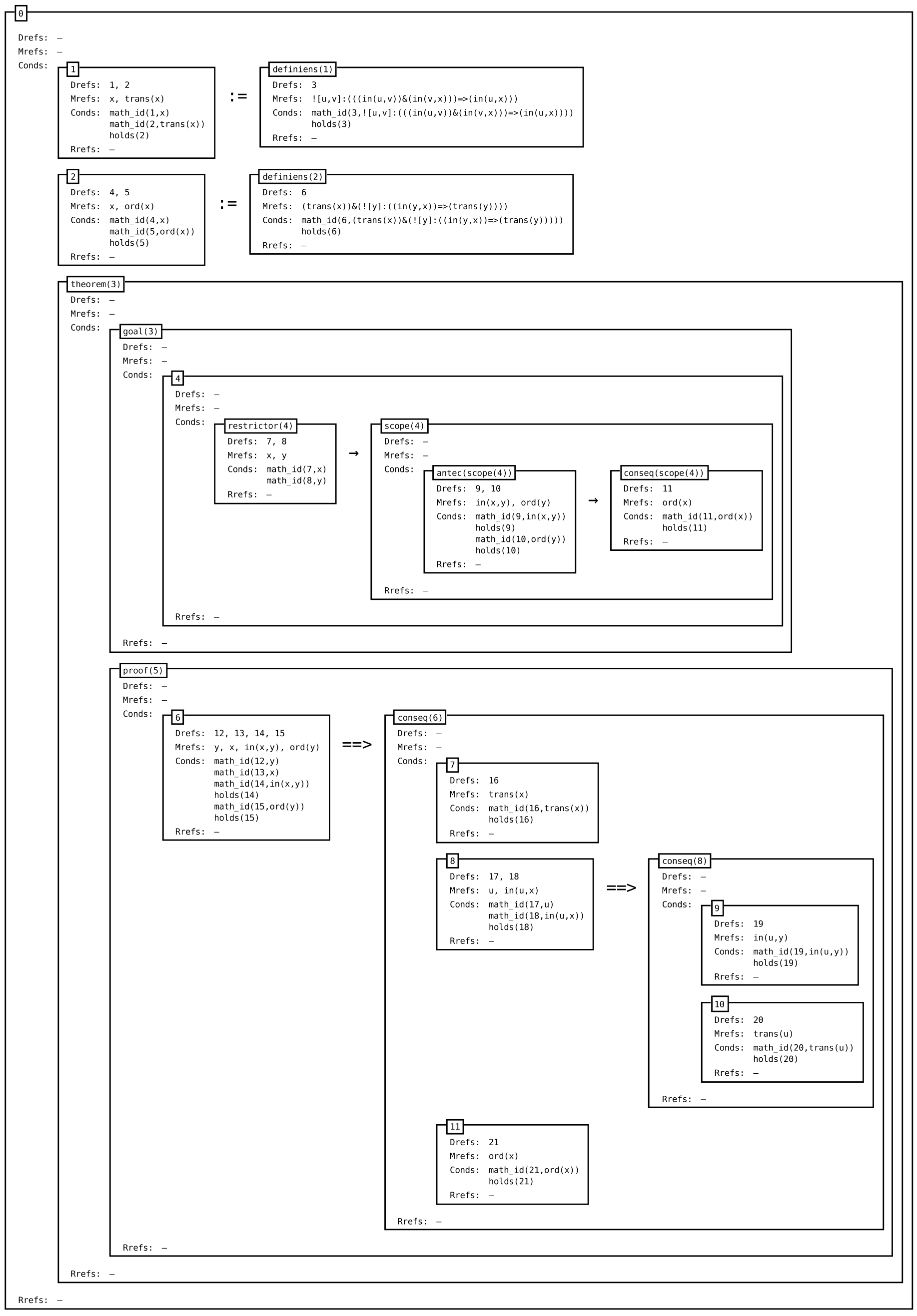

Fig. 3. The PRS for the proof of: For all x, y, if $x \in y$ and $Ord(y)$ then $Ord(x)$.

The next condition in our example PRS is a theorem condition. A theorem conditions is a PRS with two sub-PRS, the goal PRS and the proof PRS. The statement of the theorem is stored in the goal PRS, the proof for the theorem is in the proof PRS. The algorithm first checks the proof PRS and then uses the updated premises to check the goal PRS.

The first condition in the proof PRS is an assumption. Again, all we have to do is to update the list of active premises. It now contains the following formulas:

$$\begin{aligned}&[\forall x\, Trans(x) \leftrightarrow (\forall u, v(u \in v \wedge v \in x) \rightarrow u \in x),\\ &\forall x\, Ord(x) \leftrightarrow (Trans(x) \wedge \forall y(y \in x \rightarrow Trans(y))),\\ &x \in y, Ord(y)]\end{aligned}$$

Next comes the statement $Trans(x)$. The algorithm tries to prove this statement from the active premises. For this, the premises as well as the statement are transformed into the TPTP first order format [9] and an ATP query is created:

```
fof(1, axiom, ![Vx]:((trans(Vx))<=>
  (![Vu,Vv]:(((in(Vu,Vv))&(in(Vv,Vx)))=>(in(Vu,Vx)))))).
fof(2, axiom, ![Vx]:((ord(Vx))<=>
  ((trans(Vx))&(![Vy]:((in(Vy,Vx))=>(trans(Vy))))))).
fof(3, axiom, in(vx,vy)).
fof(4, axiom, ord(vy)).
fof(1, conjecture, trans(vx)).
```

Geoff Sutcliffe's service tools for the TPTP library (e.g. SystemsOnTPTP [10]) are used to interact with the ATP. The result is given as an output to the user, $Trans(x)$ gets added to the active premises and the checking algorithm proceeds to the next condition.

The remaining conditions are checked in a similar fashion. The detailed procedure as well as considerations about the completeness and the correctness of the algorithm can be found in [6].

6 Results

The Naproche system parses the Naproche CNL, creates the appropriate PRSs, and checks them for correctness using automated theorem provers. The Burali-Forti paradox, a well known mathematical theorem, as well as several basic statements in group theory, were formulated in the Naproche CNL, and checked for correctness with the Naproche system. A web-based interface to the Naproche system can be found on our homepage *www.naproche.net.*

7 Future Work

There are two major points that we would like to improve in upcoming versions of the Naproche system:

Firstly, we will extend the Naproche controlled natural language to include a rich mathematical formula language and many argumentative mathematical phrases and constructs. And secondly, we shall improve the Naproche-ATP interaction. We want to study which of the premises were actually used when discharging a proof obligation and whether the systems can be tailored to do inferences in the 'size' of human proofsteps.

Once the Naproche system is sufficiently extensive and powerful, proofs from various areas of mathematics can be reformulated and checked in a way understandable to men and machines.

References

1. VeriMathDoc, URL http://www.ags.uni-sb.de/ afiedler/verimathdoc/.
2. Patrick Braselmann and Peter Koepke. Gödels Completeness Theorem. *Formalized Mathematics*, 13, 2005.
3. Norbert E. Fuchs, Stefan Höfler, Kaarel Kaljurand, Fabio Rinaldi, and Gerold Schneider. Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines, 2005.
4. H. Kamp and U. Reyle. *From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Languge*. Kluwer Academic Publisher, 1993.
5. Nickolay Kolev. Generating Proof Representation Structures for the Project NAPROCHE. Master's thesis, University of Bonn, 2008.
6. Daniel Kuehlwein. A Calculus for Proof Representation Structures. Master's thesis, University of Bonn, 2008.
7. Roman Matuszewski and Piotr Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4:2005, 2005.
8. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
9. G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
10. Geoff Sutcliffe. System Description: SystemOn TPTP. In *CADE*, pages 406–410, 2000.
11. J. van der Hoeven. GNU TeXmacs: A free, structured, wysiwyg and technical text editor. In Daniel Flipo, editor, *Le document au XXI-ième siècle*, volume 39–40, pages 39–50, 14–17 mai 2001. Actes du congrès GUTenberg.
12. Konstantin Verchinine, Alexander Lyaletski, and Andrei Paskevich. *System for Automated Deduction (SAD): a tool for proof verification*, volume 4603 of *Lecture Notes in Computer Science*, pages 398–403. Springer, July 2007.
13. Konstantin Vershinin and Andrey Paskevich. ForTheL - the language of formal theories. *International Journal of Information Theories and Applications*, 7(3):120–126, 2000.
14. M. Wagner, S. Autexier, and C. Benzmüller. PLATO: A Mediator between Text-Editors and Proof Assistance Systems. In S. Autexier and C. Benzmüller, editors, *7th Workshop on User Interfaces for Theorem Provers (UITP'06)*, volume 174(2) of *Electronic Notes on Theoretical Computer Science*, pages 87–107. Elsevier, August 2006.
15. Makarius Wenzel. *From Insight to Proof - Festschrift in Honour of Andrzej Trybulec*, chapter Isabelle/Isar - a generic framework for human-readable proof documents. 2007.

16. Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1962.
17. Claus Zinn. *Understanding Informal Mathematical Discourse*. PhD thesis, Friedrich-Alexander-Universitt Erlangen Nürnberg, 2004.