Originally published in: Fuchs, Norbert E. (Ed.): Controlled Natural Language. Workshop on Controlled Natural Language, CNL 2009, Marettimo Island, Italy, June 8-10, 2009. Revised Papers. - Berlin [et al.]: Springer, 2010. Pp. 170-186. (Lecture Notes in Computer Science 5972)

The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts

POSTPRINT

Marcos Cramer, Bernhard Fisseni, Peter Koepke, Daniel Kühlwein, Bernhard Schröder, and Jip Veldman

University of Bonn and University of Duisburg-Essen
{cramer,koepke,kuehlwei,veldman}@math.uni-bonn.de,
{bernhard.fisseni,bernhard.schroeder}@uni-due.de
http://www.naproche.net

Abstract. This paper discusses the semi-formal language of mathematics and presents the Naproche CNL, a controlled natural language for mathematical authoring. Proof Representation Structures, an adaptation of Discourse Representation Structures, are used to represent the semantics of texts written in the Naproche CNL. We discuss how the Naproche CNL can be used in formal mathematics, and present our prototypical Naproche system, a computer program for parsing texts in the Naproche CNL and checking the proofs in them for logical correctness.

Keywords: CNL, mathematical language, formal mathematics, proof checking.

1 Introduction

The Naproche project[1] (NAtural language PROof CHEcking) studies the semi-formal language of mathematics (SFLM) as used in mathematical journals and textbooks from the perspectives of linguistics, logic and mathematics. A central goal of Naproche is to develop and implement a controlled natural language (CNL) for mathematical texts which can be transformed automatically into equivalent formulae of first-order logic using methods of computational linguistics.

One possible application of this mathematical CNL is to make formal mathematics more readable to the average mathematician; this application is fundamental to most other applications and therefore is currently the focus of Naproche development. In order to show the feasibility of this goal, we develop the prototypical *Naproche system*, which can automatically check texts written in the Naproche CNL for logical correctness. We test this system by reformulating parts of mathematical textbooks and the basics of mathematical theories in the Naproche CNL and automatically checking the resulting texts.

[1] Naproche is a joint initiative of PETER KOEPKE (Mathematics, University of Bonn) and BERNHARD SCHRÖDER (Linguistics, University of Duisburg-Essen). The Naproche system is technically supported by GREGOR BÜCHEL from the University of Applied Sciences in Cologne.

Once Naproche has sufficiently broad coverage, we also plan to use it as a tool that can help undergraduate students to learn how to write formally correct proofs and thus get used to (a subset of) SFML. This possible application of Naproche also makes it more evident why we are emphasising a lot on the naturalness of our CNL.

We first discuss the relevant features of SFLM and then discuss the Naproche CNL, Proof Representation Structures and the translation to first order logic. Finally, we describe how proof checking in Naproche can be of use to the field of formal mathematics and discuss further development of Naproche.

2 The Semi-Formal Language of Mathematics

As an example of the semi-formal language of mathematics (SFLM), we cite a proof for the theorem "$\sqrt{2}$ *is irrational*" from HARDY-WRIGHT's introduction to number theory [8].

> If $\sqrt{2}$ is rational, then the equation $a^2 = 2b^2$ is soluble in integers a, b with $(a, b) = 1$. Hence a^2 is even, and therefore a is even. If $a = 2c$, then $4c^2 = 2b^2$, $2c^2 = b^2$, and b is also even, contrary to the hypothesis that $(a, b) = 1$.

SFLM incorporates the syntax and semantics of the general natural language, so that it takes over its complexity and some of its ambiguities. However, SFLM texts are distinguished from common language texts by several characteristics:

- They combine natural language expressions with mathematical symbols and formulae, which can syntactically function as noun phrases or sub-propositions.
- Constructions which are hard to disambiguate are generally avoided.
- Mathematical symbols can be used for disambiguation, e.g. by use of variables instead of anaphoric pronouns.
- Assumptions can be introduced and retracted. For example, the proof cited above is a proof by contradiction: At the beginning, it is assumed that $\sqrt{2}$ is rational. The claims that follow are understood to be relativised to this assumption. Finally the assumption leads to a contradiction, and is retracted to conclude that $\sqrt{2}$ is irrational.
- Mathematical texts are highly structured. At a global level, they are commonly divided into building blocks like definitions, lemmas, theorems and proofs. Inside a proof, assumptions can be nested into other assumptions, so that the scopes of assumptions define a hierarchical proof structure.
- Definitions add new symbols and expressions to the vocabulary and fix their meaning.
- Proof steps are commonly justified by referring to results in other texts, or previous passages in the same text. So there are intertextual and intratextual references.
- Furthermore, SFML texts often contain commentaries and hints which guide the reader through the process of the proof, e.g. by indicating the method of proof ("by contradiction", "by induction") or giving analogies.

3 The Naproche CNL

The Naproche CNL is currently a small but functional subset of SFLM which includes some of the above mentioned characteristics of SFLM. We first discuss the syntax of the Naproche CNL, and finally present a CNL version of the example proof from Section 2. A more detailed specification of the CNL syntax can be found in [5].

In the Naproche CNL we distinguish between *macrostructure* (general text structure) and *microstructure* (grammatical structure in a sentence).

3.1 Macrostructure

A Naproche text is structured by structure markers: *Axiom*, *Theorem*, *Lemma*, *Proof*, *Qed* and *Case*. For example, a theorem is presented after the structure marker *Theorem*, and its proof follows between the structure markers *Proof* and *Qed*. In a proof by case distinction, each case starts with the word *Case*, and the end of a case distinction can be marked by a sentence starting with *in both cases* or *in all cases*.

Assumptions are always introduced by an assumption trigger (e.g. *let*, *consider*, *assume that*). A sentence starting with *thus* always retracts the most recently introduced assumption that hasn't yet been retracted. When finishing a proof with a *qed*, all assumptions introduced in the proof get retracted.

Definitions can be used to define the meanings of constants, function symbols, relation symbols, verbs, adjectives and nouns. They can be marked by the words *definition* or *define*.

Here is an extract from our reformulation of EDMUND LANDAU's *Grundlagen der Analysis* [12], with all structure markers marked in bold font:

> **Axiom 3**: For every x, $x' \neq 1$.
> **Axiom 4**: If $x' = y'$, then $x = y$.
> **Theorem 1**: If $x \neq y$ then $x' \neq y'$.
> **Proof**: **Assume that** $x \neq y$ and $x' = y'$. Then by axiom 4, $x = y$. **Qed**.
> **Theorem 2**: For all x $x' \neq x$.
> **Proof**: By axiom 3, $1' \neq 1$. **Suppose** $x' \neq x$. Then by theorem 1, $(x')' \neq x'$. **Thus** by induction, for all x $x' \neq x$. **Qed**.
> **Definition 1**: **Define** $+$ recursively:
> $x + 1 = x'$. $x + y' = (x + y)'$.

Additionally we present an example of a proof by case distinction in the Naproche CNL:

> Theorem: No square number is prime.
> Proof: Let n be a square number.
> **Case 1**: $n = 0$ or $n = 1$. Then n is not prime.
> **Case 2**: $n \neq 0$ and $n \neq 1$. Then there is an m such that $n = m^2$. Then $m \neq 1$ and $m \neq n$. m divides n, i.e. n is not prime.
> So **in both cases** n is not prime. Qed.

3.2 Microstructure

Mathematical terms and formulae can be combined with natural language expressions to form statements, definitions and assumptions: The use of nouns, adjectives, verbs, natural language connectives (e.g. *and*, *or*, *i.e.*, *if*, *iff*) and natural language quantification (e.g. *for all*, *there is*, *every*, *some*, *no*) works as in natural English, only with some syntactical limitations similar to those in Attempto Controlled English [7]. Sentences in a proof can start with words like *then*, *hence*, *therefore* etc. Mathematical terms can function as noun phrases, and mathematical formulae can function as sentences or sub-clauses. The language of mathematical formulae and terms that can be used in the Naproche CNL is a first order language with function symbols, including some syntactic sugar that is common in the mathematical formulae and terms found in SFLM.

3.3 Coverage

The Naproche CNL in its present state does not cover all constructs of SFLM, but most texts can be rewritten in the Naproche CNL in such a way that they resemble the original text and still read like SFLM.

In our biggest test so far, we translated the first chapter of LANDAU's *Grundlagen der Analysis* [12] (17 pages) into the Naproche CNL. The resulting text can be seen online [4], and stays close to the original. Another example of a reformulation in the Naproche CNL can be seen in the following subsection.

We are continuously extending the language to allow more and more SFLM constructs.

3.4 $\sqrt{2}$ in Naproche CNL

The irrationality proof can be reformulated in the Naproche CNL as follows:

> Theorem: $\sqrt{2}$ is irrational.
> Proof:
> Assume that $\sqrt{2}$ is rational. Then there are integers a, b such that $a^2 = 2 \cdot b^2$ and $gcd(a, b) = 1$. Hence a^2 is even, and therefore a is even. So there is an integer c such that $a = 2 \cdot c$. Then $4 \cdot c^2 = 2 \cdot b^2$, $2 \cdot c^2 = b^2$, and b is even. Contradiction. Qed.

Naproche uses LaTeX input, so the actual input used would be the LaTeX code that was used to typeset the above text. We are also working on the alternative of using the mathematical WYSIWYG editor TeXmacs [15].

4 Proof Representation Structures

Texts written in the Naproche CNL are translated into *Proof Representation Structures* or PRSs (see [10], [11]). PRSs are Discourse Representation Structures (DRSs, [9]), which are enriched in such a way as to represent the distinguishing characteristics of SFLM discussed in Section 2.

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[2]. Similar to DRSs, we can display PRSs as "boxes" (Figure 1).

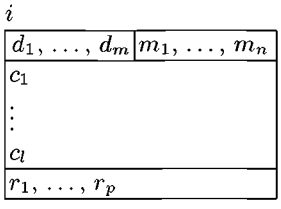


Fig. 1. A PRS with identification number i, discourse referents $d_1, ..., d_n$, mathematical referents $m_1, ..m_k$, conditions $c_1, ..., c_l$ and textual referents $r_1, ..r_p$

Mathematical referents are the terms and formulae 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 formulae 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 we can refer to them from other points in the discourse. The textual referents indicate the intratextual and intertextual references.

Just as in the case of DRSs, PRSs and PRS conditions are defined recursively: Let $A, B, B_1, \ldots, B_n$ be PRSs, $d, d_1, \ldots, d_n$ discourse referents and m a mathematical referent. Then

- for any n-ary predicate p (e.g. expressed by adjectives and noun phrases in predicative use and verbs in SFLM), $p(d_1, \ldots, d_n)$ is a condition.
- $holds(d)$ is a condition representing the claim that the formula referenced by d is true.
- $math\_id(d, m)$ 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.
- $A \Rightarrow B$ is a condition, representing an assumption (A) and the set of claims made inside the scope of this assumption (B).
- $A \Leftrightarrow B$ is a condition, representing a logical equivalence.
- $A \Leftarrow B$ is a condition, representing a reversed conditional (i.e. of the form "... if ...")
- $A \vee B$ is a condition, representing an inclusive disjunction.
- $>< (A_1, \ldots, A_n)$ is a condition, representing an exclusive disjunction, i.e. the claim that precisely one of $A_1, \ldots, A_n$ holds.

[2] The order of the conditions in a PRS reflects the argument structure of a proof and is relevant to the PRS semantics. This was in part inspired by ASHER's SDRT [1].

- $<>\ (A_1, \ldots, A_n)$ is a condition, representing the claim that at most one of $A_1, \ldots, A_n$ holds.
- $A := B$ is a condition, representing a definition of a predicate.
- $f : A \to B$ is a condition, representing a definition of a function or constant (where A fixes the scope of the function and B specifies its defining equality or property).
- *contradiction* is a condition, representing a contradiction.

4.1 PRS Construction

The algorithm creating PRSs from CNL proceeds sequentially [10]:

- It starts with the empty PRS.
- Structure markers open special *structure PRSs*.
- An assumption triggers a condition of the form $A \Rightarrow B$, where A contains the assumption and B contains the representation of all claims made inside the scope of that assumption.
- Representations of single sentences are produced in a way similar to a standard threading construction algorithm (see [2]).

We clarify both the PRS construction algorithm and the functions of the various kinds of PRS conditions by presenting examples which show how the most important PRS conditions are constructed from input text:

Negation conditions

Negation conditions work just as in Discourse Representation Theory:

n is not prime

$\Downarrow$

$\neg$ [| n is prime]

The downward arrow in this and the following examples roughly corresponds to one step of the PRS construction algorithm. The "n is prime" in this PRS would still be processed further.

Implication conditions

PRS conditions of the form $A \Rightarrow B$ are called *implication conditions*. One function of them is to represent conditionals of the form *If ... then ...*:

If $x' = y'$, then $x = y$.

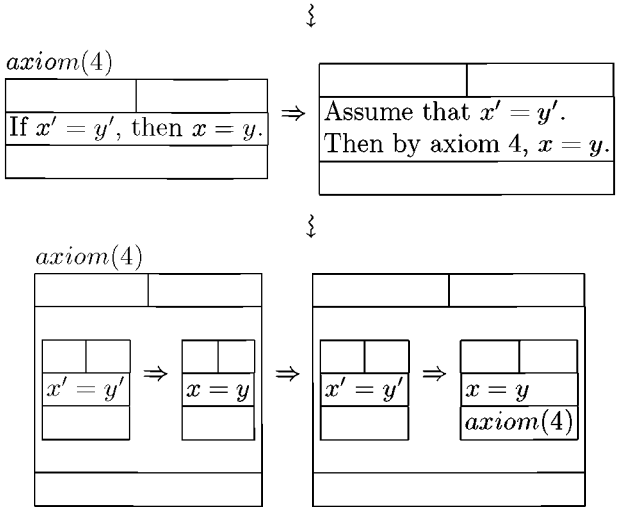
$\Downarrow$

[| $x' = y'$] $\Rightarrow$ [| $x = y$]

When an assumption is introduced in a proof, and then something is derived from that assumption, this is also represented by an implication condition: The assumption is placed in the left PRS of the condition, and all proof steps until the assumption gets retracted are placed in the right PRS. Axioms are treated like assumptions that never get retracted.

The following example shows how different linguistic constructions can yield implication conditions, and how these can be nested inside one another:

Axiom 4: If $x' = y'$, then $x = y$.
Assume that $x' = y'$. Then by axiom 4, $x = y$.

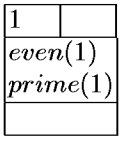


Note that the *axiom*(4) entry in the second PRS is not a condition, but a textual referent to a premise (see 5.3).

Predicate conditions

Predicate conditions work just like in DRT:

There is an even prime.



Note that we use natural numbers $(1, 2, 3, \ldots)$ as discourse referents.

***math_id* conditions**

As in DRT, all objects that are talked about are represented by discourse referents. Additionally, the mathematical terms (e.g. the variables) used in the text

are listed in the PRS as mathematical referents. It is therefore necessary to indicate which discourse referent refers to the same object as a certain mathematical referent. For this we use *math_id* conditions:

There are integers m, n such that m divides n.

$\wr$

1, 2	m, n
$integer(1)$	
$math\_id(1, m)$	
$integer(2)$	
$math\_id(2, n)$	
$divide(1, 2)$	

holds conditions

When a mathematical formula is used, this gets represented by a *holds* condition, which indicated the truth of the formula:

Then $m \neq 1$ and $m \neq n$.

$\wr$

1, 2	$m \neq 1$, $m \neq n$
$math\_id(1, m \neq 1)$	
$holds(1)$	
$math\_id(2, m \neq n)$	
$holds(2)$	

Definition conditions

There are two special kinds of conditions for definitions. Here we concentrate on just on of these two kinds; conditions of this kind are of the form "$A := B$" and represent definitions by biconditionals:

Define m to be square iff there is an integer n such that $m = n^2$.

$\wr$

1	m
$math\_id(1, m)$	
$square(1)$	

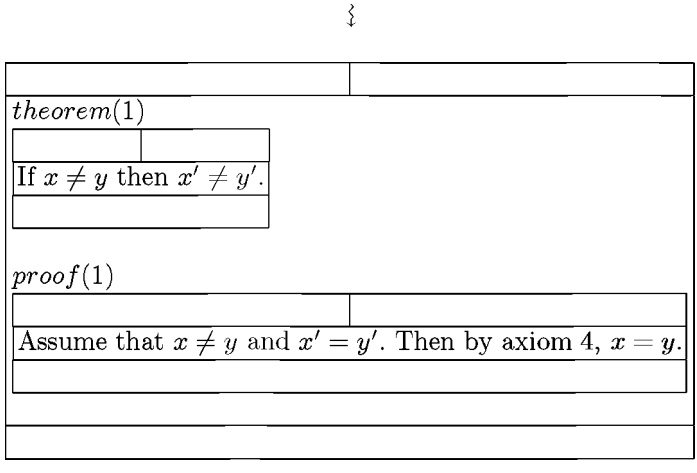
$:=$

2, 3	n, $m = n^2$
$math\_id(2, n)$	
$math\_id(3, m = n^2)$	
$holds(3)$	

Bare PRSs as conditions

Contrary to the case of DRSs, a bare PRS can be a direct condition of a PRS. This allows us to represent in a PRS the structure of a text divided into building blocks (definitions, lemmas, theorems, proofs) by structure markers.

Theorem 1: If $x \neq y$ then $x' \neq y'$.
Proof: Assume that $x \neq y$ and $x' = y'$. Then by axiom 4, $x = y$. Qed.



$\sqrt{2}$ is irrational

The PRS constructed from the example proof,"$\sqrt{2}$ is irrational.", is shown in Figure 2.

4.2 Accessibility in PRSs

Unlike for DRSs, accessibility for PRSs is not just a relation between PRSs, but also a relation between PRS conditions, and between PRSs and PRS conditions. As in the case of DRSs, accessibility can be defined via a subordination relation:

Let Θ_1 and Θ_2 be PRSs or PRS conditions.[3] Θ_1 directly subordinates Θ_2 if

- Θ_1 is a PRS, and Θ_2 is the first condition of Θ_1, or
- Θ_1 and Θ_2 are PRS conditions of a PRS B, and Θ_1 appears before Θ_2 in the list of conditions of B, or
- Θ_1 is a PRS condition of the form $\neg\Theta_2$, or

[3] Θ_1 and Θ_2 should be understood to be occurrences of PRSs or PRS conditions. For example, if a PRS has as its condition list $\langle c_1, c_2, c_1 \rangle$, then c_2 subordinates the second occurrence of c_1, but not its first occurrence. So strictly speaking it is necessary to speak about occurrences ("this occurrence of c_2 subordinates the second occurrence of c_1").

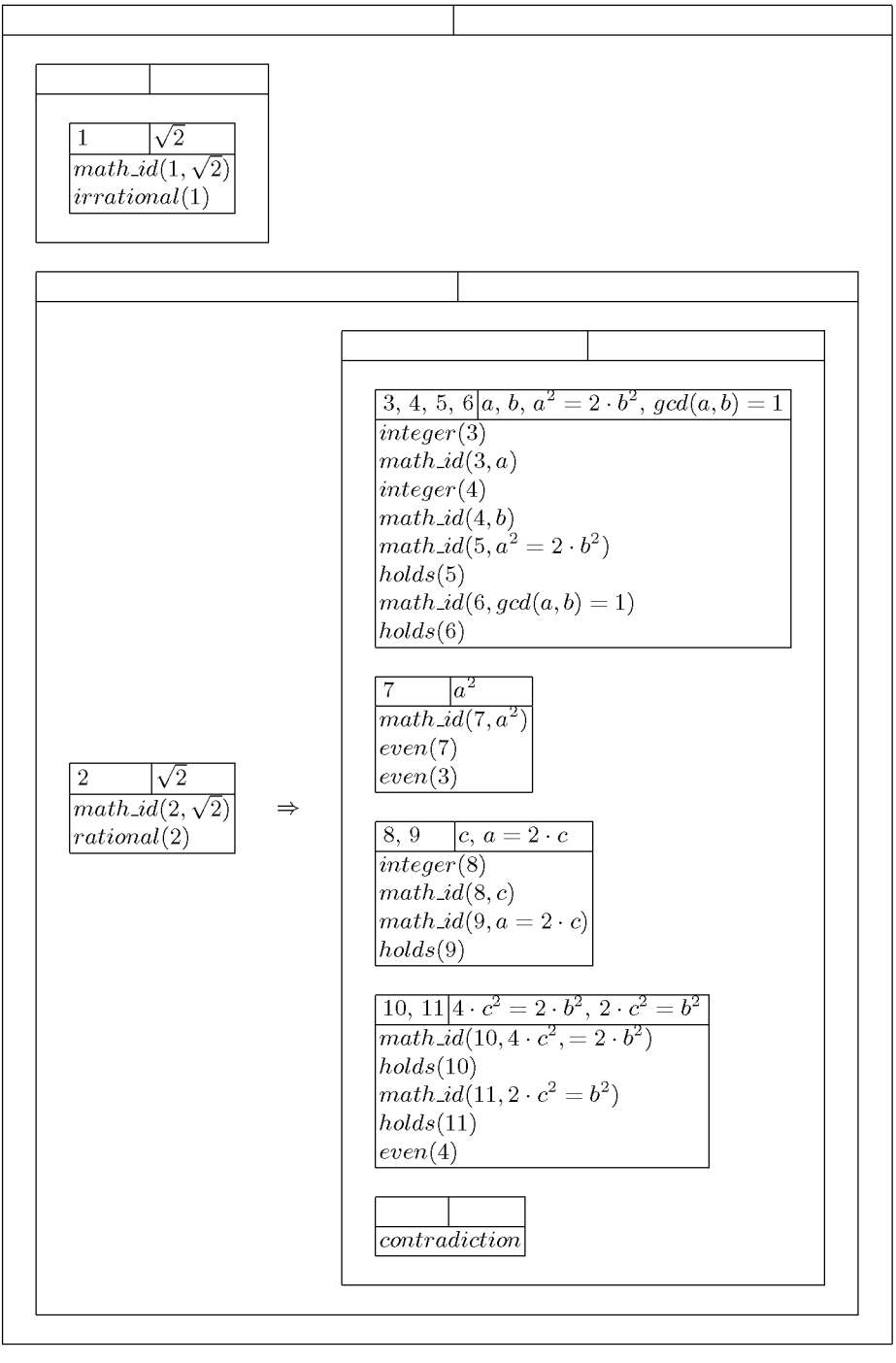


Fig. 2. The PRS corresponding to proof of "$\sqrt{2}$ is irrational.", depicted without PRS identifiers and textual references.

- Θ_1 is a PRS condition of the form $\Theta_2 \Rightarrow B$, $\Theta_2 \Leftrightarrow B$, $\Theta_2 \Leftarrow B$, $\Theta_2 \vee B$, $B \vee \Theta_2$, $>< (\Theta_2, B_1, \ldots, B_n)$, $<> (\Theta_2, B_1, \ldots, B_n)$, $\Theta_2 := B$ or $f : \Theta_2 \rightarrow B$, or
- $\Theta_1 \Rightarrow \Theta_2$, $\Theta_1 \Leftrightarrow \Theta_2$, $\Theta_1 \Leftarrow \Theta_2$, $\Theta_1 := \Theta_2$ or $f : \Theta_1 \rightarrow \Theta_2$ is a PRS condition, or

The relation "Θ_1 is accessible from Θ_2" is the transitive closure of the relation "Θ_1 subordinates Θ_2".

A discourse referent d or a mathematical referent m is said to be accessible from Θ, if there is a PRS B that is accessible from Θ and that contains d or m in its list of discourse or mathematical referents.

In the PRS of the $\sqrt{2}$ example (Figure 2), we can derive from the above definitions that the discourse referent 2 is accessible from all the PRSs and PRS conditions inside the PRS that is to the right of the "$\Rightarrow$". The discourse referent 7 and the mathematical referent a^2 are only accessible in the PRSs and conditions that are graphically below the line where they are introduced. On the other hand the discourse referent 1 is only accessible in the PRS in which it gets introduced, as this PRS does not subordinate any other PRSs.

4.3 Motivations for the Accessibility Constraints

For those complex PRS conditions that are analogous to standard DRS conditions (i.e. $\neg A$, $A \Rightarrow B$ and $A \vee B$), the accessibility constraints are equivalent to the standard accessibility constraints in DRT: In all three cases the discourse referents introduced in A and B are not accessible for the later discourse; in $A \Rightarrow B$ the discourse referents from A are accessible in B, but not so for $A \vee B$.

The fact that discourse referents introduced in one of the argument PRSs of a complex PRS condition should not be accessible after that condition naturally extends to those complex conditions that are specific to PRSs ($A \Leftrightarrow B$, $A \Leftarrow B$, $>< (A_1, \ldots, A_n)$, $<> (A_1, \ldots, A_n)$, $A := B$, $f : A \rightarrow B$).

For example, after the definition "Define m to be square iff there is an integer n such that $m = n^2$.", neither n nor m should be accessibe. The corresponding PRS condition has the form $A := B$ and can be seen in Figure 3. A introduces a discourse referent for m which is also used in B (since m is used to the right of the "iff"). This example also shows that the discourse referents introduced by A should still be accessible in B, as they are according to our above definition of accessibility.

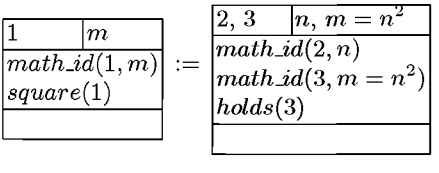


Fig. 3. The PRS condition corresponding to the sentence "Define m to be square iff there is an integer n such that $m = n$

Similarly, there are natural examples that show that also in conditions of the form $A \Leftrightarrow B$, $A \Leftarrow B$ and $f : A \rightharpoonup B$ the discourse referents of A should be accessible in B:

- "A natural number n is even if and only if $n + 1$ is odd."
- "A natural number is even if it is not odd."[4]
- "For x a positive real, define $\sqrt{x}$ to be the unique positive real y such that $y^2 = x$."

The condition $>< (A_1, \ldots, A_n)$ expresses an exclusive disjunction, so it is natural that accessibility is blocked between the disjuncts just as in the case of the inclusive disjunction condition $A \vee B$. The similar condition $<> (A_1, \ldots, A_n)$ expresses the weaker statement that at most one of the "disjuncts" must hold, but it linguistically behaves very similar to the statement that precisely one of them holds, so that the same accessibility constraints apply.

4.4 PRS Semantics

Similarly to the case of the semantics of first order formulae, defining the semantics of PRSs means fixing the criteria under which a PRS is satisfied in a certain *structure*. A *structure* is a set together with relations and functions defined on the elements of the set.

The difference to the semantics of first order logic is that PRS semantics is defined dynamically[5]: Each PRS condition changes the *context* for the subsequent PRS conditions. A *context* is defined to be a triple consisting of a structure and two kinds of variable assignments[6], one for discourse referents and one for mathematical referents.

- Any definition condition extends the structure in this context triple, so that it contains the newly defined symbol.
- Any PRS condition that is a simple PRS extends the two assignments in the triple, so that they are also defined on the discourse referents and mathematical referents introduced in that PRS.
- Any other kind of PRS conditions doesn't actually change the context, but must satisfy the context in which it is interpreted.

For a PRS to be satisfied in a certain structure $\mathfrak{A}$, it must be possible to sequentially modify the context $[\mathfrak{A}, \emptyset, \emptyset]$ by each condition in the PRS.

[4] Concerning this "reversed conditional" as well as the biconditional above, we should mention that we do not take these examples to be linguistically analogous to donkey sentences. Indeed, these cases should probably be seen as cases of generic interpretation of "a", whereas the "a" in the antecedent of donkey sentences is normally not considered to be a case of generic interpretation.

[5] This dynamic definition of PRS semantics is partly based on the dynamic definition of DRS semantics found in [2].

[6] A *variable assignment* is a function that assigns elements of the domain of the structure to variables, in our case to discourse referents or mathematical referents.

The details of the PRS semantics can be found in [6]. Note that textual referents are not included in the PRS semantics. They are used in the Logic module of the Naproche system (See 5.3).

4.5 Translating PRSs into First-Order Logic

PRSs, like DRSs, have a canonical translation into first-order logic. This translation is performed in a way similar to the DRS to first-order translation described in the book by BLACKBURN and BOS [2]. For example, in both DRS and PRS translations to first-order logic, the introduction of discourse referents is translated into first-order logic using existential quantifiers, unless the discourse referents are introduced in a DRS/PRS A of a condition of the form $A \Rightarrow B$, in which case a universal quantifier is used in the translation. There are two main differences between the PRS to first order logic translation and DRS to first-order logic translations that need discussion:

- *math_id* conditions are not translated into sub-formulae of the first-order translation. A *math_id* condition, which binds a discourse referent n to a term t, triggers a substitution of n, by t, in the translation of subsequent conditions. A *math_id* condition, which binds a discourse referent n to a formula φ, causes a subsequent $holds(n)$ condition to be translated to φ.
- Definition conditions are also not translated into sub-formulae of the translation. Instead, a condition of the form $A := B$ triggers a substitution of the relation symbol it defines by the first-order translation of B in subsequent conditions.

Cramer [6] presents a rigorous definition of this translation as well as a proof that this translation is sound, i.e. that a PRS is satisfied in a structure if and only if its translation into first order logic is satisfied in that structure.

5 Formal Mathematics in Naproche CNL

Currently our primary goal is to use the Naproche CNL in the field of *formal mathematics*.

5.1 Formal Mathematics

Mathematicians usually publish *informal proofs*, i.e. proofs for which there are no formal criteria of correctness. Informal proofs contrast with *formal proofs*, in which only syntactical rules can be applied and hence every logical step is explicit.

Formal mathematics aims at carrying out all of mathematics in a completely formal way, i.e. formalise all mathematical statements in strictly formal languages and carry out all proofs in formal proof calculi.

However, formal proofs tend to be lengthy and difficult to read. This problem can be limited by the use of formal mathematics system: These are computer systems which facilitate the formalisation of mathematical proofs. In prominent

examples of formal mathematics systems like *Mizar* [13] and *Coq* [3], this is achieved by allowing the user to use a more readable formal language and to leave out some of the simpler steps of a derivation, which can be filled in by a computer. However, even proofs written in these systems are still not very readable for an average mathematician, mainly because the syntax is more similar to a programming language than to SFML and because they contain much information that human readers consider redundant.

One possible application of the Naproche CNL is to use it in an even more natural system for formal mathematics. As a prototypical version of such a system, we have developed the *Naproche system*.

5.2 The Naproche System

The Naproche system can parse texts written in the Naproche CNL, build the appropriate PRSs, and check them for correctness using automated theorem provers. Apart from several small examples in group theory and set theory, we reformulated the first chapter of EDMUND LANDAU's *Grundlagen der Analysis* [12] in the Naproche CNL, and automatically checked it for correctness using the Naproche system.

The Naproche system consists of three main modules: The User Interface, the Linguistics module and the Logic module. Figure 4 shows an overview of the architecture of the Naproche system.

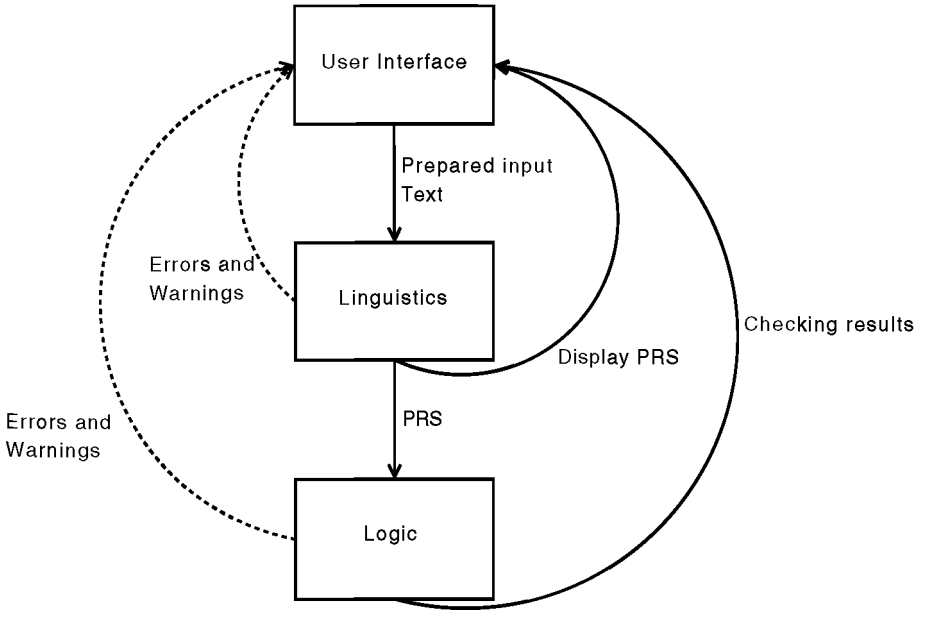


Fig. 4. The architecture of the Naproche system

The User Interface is the standard communication gateway between the user and the Naproche system. The input text is entered using the User Interface, and the other modules report back to the User Interface.

The user enters the input text, written in the Naproche CNL, on the interface, which transforms the input 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 plug-in for the WYSIWYG editor TeXmacs [15].

The Linguistics module creates a PRS from the CNL text. The text is parsed using a Prolog-DCG parser that builds up the PRS during the parsing process. The created PRS can either be given to the Logic module, or reported back to the User Interface. With the help of automated theorem provers, the Logic module checks PRSs for logical correctness. The result of the check is sent to the User Interface.

5.3 Checking PRSs

The checking algorithms keeps a list of first order formulae it considers to be true, called the *list of 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 system creates obligations which have to be discharged by an automated theorem prover[7]. For example, a *holds* condition is accepted under the premises θ if the automated theorem prover can derive the formula, which is bound to the discourse referent of the *holds* conditions, from θ.

The more premises are available, the harder it gets for the automated theorem prover to discharge an obligation. Therefore we implemented a premises selection algorithm that tries to select only the relevant premises for an obligations. This is where the the textual referents of a PRS are used. If a PRS contains a textual referent, the premises corresponding to that referent are taken to be relevant for the corresponding obligation.

If all obligations of a condition can be discharged, then the condition is accepted. After a conditions is accepted, the active premises get updated, and the next condition gets checked. A PRS is accepted by Naproche if all its conditions are accepted consecutively.

If the PRS created from a Naproche CNL text is accepted by our checking algorithm, one can conclude that the text is correct (of course under the assumption that the automatic theorem prover only creates correct proofs).

5.4 Speed and Scalability

One important issue for every algorithm is speed and scalability.

[7] We access automated theorem provers, currently Otter and E Prover, through GEOFF SUTCLIFF's SystemOnTPTP [14].

Single sentences are parsed in milliseconds, a text of 30 sentences takes about one second. Our biggest example, the Landau text, consists of 326 sentences. The corresponding PRS is created in 11 seconds.

The obligation creation algorithm needs less than a second for small texts (less than 30 sentences), and 4 seconds for the Landau PRS.

Discharging an obligation heavily depends on the automated theorem prover used, the available premises and the actual query. Using the E Prover, version 1.0, all obligations created from the Landau text can be discharged in 80 seconds.

All tests were carried out on a 2 GHz Intel Pentium M with 2 GB Ram.

6 Conclusion

We discussed the particular properties of the Semi-Formal Language of Mathematics and presented the Naproche CNL as a controlled, and thus automatically tractable, subset of SFLM. We defined an extension of Discourse Representation Structures, called Proof Representation Structures, which is adapted so as to represent the relevant content of texts written in the Naproche CNL. Some theoretical properties of Proof Representation Structures were discussed in this paper, but a more exhaustive treatment can be found in the works we refered to.

Since we currently focus on applications in the field of formal mathematics, our implementation, the Naproche system, is geared to the needs of this application. The system translates Naproche CNL texts into Proof Representation Structures, which are then checked for logical correctness; if they are correct, this amounts to the original text being correct.

We believe that the naturalness of the Naproche CNL could make formal mathematics more feasible to the average mathematician.

7 Related and Future Work

Claus Zinn developed a system for processing SFLM texts, which he calls *informal mathematical discourse* [17]. He used an extended version of DRSs, which he also called Proof Representation Structures, to represent the meaning of a text. The PRS definitions of Naproche and Zinn are different, however.

We are collaborating with the VeriMathDoc project [16], which is developing a mathematical assistant system that naturally supports mathematicians in the development, authoring and publication of mathematical work. We are working towards common formats for exchanging mathematical proofs, e.g. for corpora of mathematical texts which help to assess to what extent Naproche can and should be extended to optimally suit writers-readers of SFML and the purpose of automated verification.

We shall extend the Naproche controlled natural language to include a richer mathematical formula language and more argumentative mathematical phrases and constructs. Once the Naproche CNL is sufficiently extensive and powerful, we shall reformulate and check proofs from various areas of mathematics in a way "understandable" to men and machines.

References

1. Asher, N.: Reference to Abstract Objects in Discourse (1993)
2. Blackburn, P., Bos, J.: Working with Discourse Representation Theory: An Advanced Course in Computational Linguistics (2003)
3. Coq Development Team: The Coq Proof Assistant Reference Manual: Version v8.1 (July 2007), http://coq.inria.fr
4. Carl, M., Cramer, M., Kühlwein, D.: Landau in Naproche, ch. 1, http://www.naproche.net/downloads/2009/landauChapter1.pdf
5. Cramer, M.: The Controlled Natural Language of Naproche in a nutshell, http://www.naproche.net/wiki/doku.php?id=dokumentation:language
6. Cramer, M.: Mathematisch-logische Aspekte von Beweisrepräsentationsstrukturen, Master's thesis, University of Bonn (2008), http://naproche.net/downloads.shtml
7. Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., Schneider, G.: Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines
8. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers, 4th edn. (1960)
9. Kamp, H., Reyle, U.: From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language. Kluwer Academic Publisher, Dordrecht (1993)
10. Kolev, N.: Generating Proof Representation Structures for the Project Naproche, Magister thesis, University of Bonn (2008), http://naproche.net/downloads.shtml
11. Kühlwein, D.: A calculus for Proof Representation Structures, Diploma thesis, University of Bonn (2008), http://naproche.net/downloads.shtml
12. Landau, E.: Grundlagen der Analysis, 3rd edn. (1960)
13. Matuszewski, R., Rudnicki, P.: Mizar: the first 30 years. Mechanized Mathematics and Its Applications 4(2005) (2005)
14. Sutcliffe, G.: System Description: System on TPTP. In: CADE, pp. 406–410 (2000)
15. Texmacs Editor website: http://www.texmacs.org/
16. VeriMathDoc website: http://www.ags.uni-sb.de/~afiedler/verimathdoc/
17. Zinn, C.: Understanding Informal Mathematical Discourse, PhD thesis at the University of Erlangen (2004), http://citeseer.ist.psu.edu/233023.html