The Naproche system: Proof-checking mathematical texts in controlled natural language

Marcos Cramer

University of Luxembourg
marcos.cramer@uni.lu
http://www.naproche.net

Abstract. The Naproche system is a system for linguistically analysing and proof-checking mathematical texts written in a controlled natural language, i.e. a subset of the usual natural language of mathematical texts defined through a formal grammar. This paper gives an overview over the linguistic and logical techniques developed for the Naproche system. Special attention is given to the dynamic nature of quantification in natural language, to the phenomenon of implicit function introduction in mathematical texts, and to the usage of definitions for dynamically extending the language of a mathematical text.

Key words: Naproche, mathematical language, CNL, proof-checking, implicit function introduction, definitions

1 Introduction

The *language of mathematics*, i.e. the special language that is used in mathematical journals and textbooks, has some unique linguistic features, on the syntactic, on the semantic and on the pragmatic level: For example, on the syntactic level, it can incorporate complex symbolic material into natural language sentences. On the semantic level, it refers to rigorously defined abstract objects, and is in general less open to ambiguity than most other text types. On the pragmatic level, it reverses the expectation on assertions, which have to be implied by the context rather than adding new information to it.

The work presented in this paper has been conducted in the context of the *Naproche project*, an interdisciplinary project between mathematical logic and computational and formal linguistics at the universities of Bonn and Duisburg-Essen (see [6] and section 1.4 of [11]). The Naproche project is guided by the vision of a computer program that could check the correctness of mathematical proofs written in the natural language of mathematics. Given that reliable processing of unrestricted natural language input is out of the reach of current technology, the project focuses on the attainable goal of using a *controlled natural language* (*CNL*), i.e. a subset of a natural language defined through a formal grammar, as input language to such a program. We have developed a prototype of such a computer program, the *Naproche system*. This paper presents the linguistic and logical theoretical framework of the Naproche system and its CNL.

The main application that we have in mind for the Naproche system is to make *formal mathematics* more natural and hence more accessible to the average mathematician. *Formal mathematics* is a branch of mathematics that aims at developing substantive parts of mathematics in a purely formal way. This is usually done with the help of computer programs, and the usual input language of formal mathematics systems have more resemblance with programming languages than with the natural language of mathematics. We think that this is one of the reasons why formal mathematics is not widely used by mathematicians outside the circles of the relatively small formal mathematics community. We hope to close the gap between the formal mathematics community and the rest of the mathematical community by developing a formal mathematics system that allows for a much more natural input language.

Before running the proof-checking algorithm for checking the logical correctness of a given input text, the Naproche system transforms the input text into a logical representation of its content, called a *Proof Representation Structure* (*PRS*). PRSs are an extension of *Discourse Representation Structures* (*DRSs*), enriched in such a way as to represent some special characteristics of the language of mathematics [6]. In this paper, as in [11], we use *Dynamic Predicate Logic* (*DPL*) and its extension *Proof Text Logic* (*PTL*) instead of DRSs and PRSs as the basis for the theoretical exposition. Just like Discourse Representation Theory, Dynamic Predicate Logic is a formal system aimed at capturing the dynamic nature of natural language quantification. But unlike Discourse Representation Theory, it has a close syntactical resemblance to standard systems of first-order predicate logic.

The logical and linguistic theory developed in the context of the Naproche project has been presented in detail in the author's thesis [11]. This paper gives an overview over this theory, with a special focus on the phenomenon of implicit function introduction in mathematical texts, exemplified by constructs of the form "for every x there is an $f(x)$ such that ...", on the usage of definitions for dynamically extending the language of a mathematical text, and on quantifiers in bi-implications and reversed implications that have to be treated in a similar way as in donkey sentences.

Section 2 gives an overview over the language of mathematics, with a special focus on the features of it which will be relevant for the discussion in this paper. Section 3 introduces and formally defines Dynamic Predicate Logic (DPL). In section 4, we give an overview over the controlled natural language implemented in the Naproche system. In section 5, we discuss some difficulties in parsing the symbolic expressions found in mathematical texts and sketch the solution to these problems that is implemented in the Naproche system. Section 6 discusses the phenomenon of implicit dynamic function introduction, and section 7 defines an extension of DPL called *Typed Higher-Order Dynamic Predicate Logic* (THODPL) which formalizes this phenomenon. In section 8, we explain the proof checking algorithm implemented in the Naproche system. In section

9, we discuss how definitions are used to dynamically expand the language of mathematics, and how this is implemented in Naproche using the methods provided by THODPL. Section 10 discusses the interpretation of quantifiers in biimplications and reversed implications that have to be treated in a similar way as in donkey sentences. Section 11 presents related work and section 12 concludes the paper.

Throughout the paper we discuss various extensions to DPL. All of these extensions are included in Proof Text Logic (PTL), which is comprehensively presented in chapter 5 of [11].

2 The language of mathematics

Just as other sciences, mathematics has developed its own specialized language. This language has many registers, but in this paper we will focus on the written registers of undergraduate and graduate textbooks.

As an example of the language of mathematics, we cite a text fragment from [29].

> **Definition 5.8.** *A linearly independent set (resp. sequence) whose elements generate a given vector space is called a basis (resp. ordered basis) of that space.*
>
> *Examples*
> 1. The empty set is a basis of the zero-space.
> 2. $(E_1, \ldots, E_m)$ is an ordered basis of $\mathbf{F}^m$. We call it the *canonical* basis.
> 3. The polynomials: 1, X, X^2, ... form a basis (or an ordered basis) of the space of polynomials.
>
> In our three-dimensional geometric representation, any three non-coplanar vectors form a basis.
> The following theorem gives a more useful characterization of bases.
>
> **Theorem 5.6.** *Let $\mathcal{V}$ be a nontrivial vector space, $\mathcal{X}$ a subset of $\mathcal{V}$. Then $\mathcal{X}$ is a basis if, and only if, each vector of $\mathcal{V}$ has a unique representation as a linear combination of elements of $\mathcal{X}$.*
>
> *Proof.* What we have to prove is that the linear independence of a set of generators $\mathcal{X}$ is equivalent to the uniqueness of the representation. If $\mathcal{X}$ is not linearly independent, the representation is, in general, surely not unique, since we have $a_1\mathbf{x}_1 + \cdots + a_n\mathbf{x}_n = \mathbf{0} = 0\mathbf{x}_1 + \cdots + 0\mathbf{x}_n$, where the $\mathbf{x}$'s are distinct elements of $\mathcal{X}$ and the a's are not all zero. Conversely, suppose that some vector $\mathbf{v}$ has two distinct representations as a linear combination of elements of $\mathcal{X}$. Then we have $\mathbf{v} = a_1\mathbf{x}_1 + \cdots + a_n\mathbf{x}_n = b_1\mathbf{x}_1 + \cdots + b_n\mathbf{x}_n$, where the $\mathbf{x}$'s are distinct elements of $\mathcal{X}$ and $a_i \neq b_i$ for at least one i. Consequently $(a_1 - b_1)\mathbf{x}_1 + \cdots + (a_n - b_n)\mathbf{x}_n = \mathbf{0}$, and the $\mathbf{x}$'s are linearly dependent.

As the example illustrates, the language of mathematics incorporates the syntax and semantics of the general natural language. Hence it takes over its complexity and some of its ambiguities. However, mathematical texts are distinguished from common language texts by several characteristics. Below we give a list of some of the most perspicuous characteristics of mathematical texts. Some of the features mentioned are also found in general language, but are much more prevalent in the language of mathematics than in general language.

– Mathematical texts 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. In the proof to theorem 5.6 in the above text fragment, the sentence beginning with "Conversely, suppose" introduces the assumption that some vector $\mathbf{v}$ has two distinct representations as a linear combination of elements of $\mathcal{X}$. The claims that follow are understood to be relativized to this assumption. When the assumption gets retracted at the end of the proof, it allows one to conclude one of the two implications needed for the bi-implicational claim of the theorem.
– Mathematical texts are highly structured, and their structure is often made explicit. 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.
– The language is adaptive: Definitions add new symbols and expressions to the vocabulary and fix their meaning.
– On the pragmatic level, the expectation on assertions is reversed: Assertions have to be implied by the context rather than adding new information to it.
– Proof steps are commonly justified by referring to results in other texts, or previous passages in the same text. So there is a large amount of intertextual and intratextual references (often in a standardized form).
– Furthermore, mathematical 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.

A thorough linguistic analysis of the language of mathematics was provided by Mohan Ganesalingam in his Ph.D. thesis [15, pp. 17-38].

3 Dynamic Predicate Logic

Our formal treatment of mathematical reasoning and our controlled natural language for mathematical texts is based on an extension of *Dynamic Predicate Logic* (see [17]). *DPL* is a logical system for formalizing some of the dynamic features of natural language. Its syntax is identical to that of standard first-order predicate logic (*PL*), but its semantics is defined in such a way that the dynamic nature of natural language quantification is captured in the formalism. Consider the following example sentence and formulae:

(1) If a farmer owns a donkey, he beats it.[1]

(2) *PL*: $\forall x \forall y\ (\mathit{farmer}(x) \land \mathit{donkey}(y) \land \mathit{owns}(x, y) \rightarrow \mathit{beats}(x, y))$

(3) *DPL*: $\exists x\ (\mathit{farmer}(x) \land \exists y\ (\mathit{donkey}(y) \land \mathit{owns}(x, y))) \rightarrow \mathit{beats}(x, y)$

The standard way of translating (1) into *PL* is (2). In *DPL*, (1) can also be translated by the formula (3), which is more faithful to the structure of (1). Note that in *PL*, (3) is not a sentence, since the final occurrences of x and y are free. In *DPL* on the other hand, a variable may be bound by a quantifier even if it is outside its scope. The semantics of *DPL* is defined in such a way that (3) is equivalent to (2) in *DPL*. Hence we can conclude that in *DPL*, (3) captures the meaning of (1) while being more faithful to its syntax than (2).

The natural language quantification used in mathematical texts also exhibits these dynamic features, as can be seen in the following quotation from [19, p. 36]:

> If a space X retracts onto a subspace A, then the homomorphism $i_* : \pi_1(A, x_0) \rightarrow \pi_1(X, x_0)$ induced by the inclusion $i : A \hookrightarrow X$ is injective.

3.1 DPL semantics

We present DPL semantics in a way slightly different but logically equivalent to its definition in [17]. Structures and assignments are defined as for PL: A structure S specifies a domain $|S|$ and an interpretation a^S for every constant, function or relation symbol a in the language. An S-assignment is a function from variables to $|S|$. G_S is the set of S-assignments. Given two assignments g, h, we define $g[x]h$ to mean that g differs from h at most in what it assigns to the variable x. Given a DPL term t, we recursively define

$$[t]^g_S = \begin{cases} g(t) & \text{if } t \text{ is a variable,} \\ t^S & \text{if } t \text{ is a constant symbol,} \\ f^S([t_1]^g_S, \ldots, [t_n]^g_S) & \text{if } t \text{ is of the form } f(t_1, \ldots, t_n). \end{cases}$$

In [17], DPL semantics is defined via an interpretation function $[\![\bullet]\!]_S$ from DPL formulae to subsets of $G_S \times G_S$. We instead recursively define for every $g \in G_S$ an interpretation function $[\![\bullet]\!]^g_S$ from DPL formulae to subsets of G_S:[2]

1. $[\![\top]\!]^g_S := \{g\}$

[1] This example sentence is one of a number of standard examples from the linguistic literature about dynamic quantification, which are usually called *donkey sentences*. Donkey sentences were originally introduced by Peter Geach [16].

[2] This can be viewed as a different currying of the uncurried version of the interpretation function in [17].

2. $[\![t_1 = t_2]\!]_S^g := \{h | h = g \text{ and } [t_1]_S^g = [t_2]_S^g\}$[3]
3. $[\![R(t_1, \ldots, t_2)]\!]_S^g := \{h | h = g \text{ and } ([t_1]_S^g, \ldots, [t_2]_S^g) \in R^S\}$
4. $[\![\neg\varphi]\!]_S^g := \{h | h = g \text{ and there is no } k \in [\![\varphi]\!]_S^h\}$
5. $[\![\varphi \wedge \psi]\!]_S^g := \{h | \text{there is a } k \text{ s.t. } k \in [\![\varphi]\!]_S^g \text{ and } h \in [\![\psi]\!]_S^k\}$
6. $[\![\varphi \to \psi]\!]_S^g := \{h | h = g \text{ and for all } k \text{ s.t. } k \in [\![\varphi]\!]_S^h, \text{ there is a } j \text{ s.t. } j \in [\![\psi]\!]_S^k\}$
7. $[\![\exists x\varphi]\!]_S^g := \{h | \text{there is a } k \text{ s.t. } k[x]g \text{ and } h \in [\![\varphi]\!]_S^k\}$

$\varphi \vee \psi$ and $\forall x\varphi$ are defined to be a shorthand for $\neg(\neg\varphi \wedge \neg\psi)$ and $\exists x\top \to \varphi$ respectively.

4 The Naproche CNL

The Naproche CNL is a controlled natural language for mathematical texts. Texts written in the Naproche CNL can be interpreted unambiguously by the Naproche system, which can check their logical correctness using the *proof checking algorithm* described in section 8 below.

For those parts of the language of mathematics that are part of the common language, like the basic functioning of nouns, verbs, adjectives, prepositions and their syntactic combinations into noun phrases, verb phrases etc., the Naproche CNL is based on Attempto Controlled English [13]. The details of this basic layer of the Naproche CNL are not the focus of this paper; for details about it, see chapter 7 of [11].

This basic layer of the Naproche CNL has been enriched by a number features and constructs that are characteristic of the language of mathematics. Some of these – namely symbolic mathematics, implicit dynamic function introduction, mathematical definitions and the dynamic interpretation of quantifiers in bi-implications and reverse implications – will be discussed in more detail in subsequent sections (sections 5-10). For the rest of this section, we will briefly discuss some further such features and constructs of the Naproche CNL.

The Naproche system first translates an input text into a *Proof Representation Structure* (PRS), and the proof-checking algorithm is defined on PRSs. In this paper, just as in [11], we use the DPL extension PTL instead of PRSs. The details of the translation of Naproche CNL texts into PTL can be found in chapter 7 of [11].

4.1 Macro-grammar

There are syntactical rules that govern the overall structure of a text. For example, there is a syntactic construct called *theorem-proof block* that consists of the following five parts in this order:

– A sentence of the form “Theorem.”, which marks the beginning of the theorem-proof block

[3] The condition $h = g$ in cases 2, 3, 4 and 6 implies that the defined set is either $\emptyset$ or $\{g\}$.

- A sequence of sentences of a certain form that express the content of the theorem
- A sentence of the form “Proof:”, which marks the beginning of the proof
- A sequence of sentences of a certain form that express the proof of the theorem
- A sentence of the form “Qed.”, which marks the end of the proof

Syntactical rules of this kind make up the *macro-grammar* of the Naproche CNL. Another example of a macro-grammatical block are *case distinction blocks*.

One issue that needed to be solved when defining the macro-grammar of the Naproche CNL is the retraction of local assumptions. As in the natural language of mathematics, in the Naproche CNL a local assumption is marked by a keyword like “assume”, “suppose” or “let”. Local assumptions are assumed to hold only for a limited fragment of the text. We say that at the place, where they stop to be assumed, they get *retracted*. In the natural language of mathematics, the retraction of an assumption is usually not marked explicitly and can only be inferred from an understanding of the proof. In a CNL, however, we need a reliable method for retracting assumptions. We have therefore introduced the rule that the keyword “thus”, which can be used at the beginning of a sentence, retracts the most recently introduced not yet retracted assumption. Furthermore, an assumption gets retracted when the macro-grammatical block inside of which it was introduced ends. For example, an assumption introduced inside a proof gets retracted when the proof is ended with “Qed.”. We call the text fragment from the assumption up to the place where it gets retracted the *scope* of the assumption.

The macro-grammar of the Naproche CNL is described in detail in section 7.2 of [11].

4.2 Representing complete texts in PTL

Since DPL formulae are usually used to represent single statements and not complete texts, we need to explain how PTL as an extension of DPL can be used to represent complete texts.

For this, we include in PTL a construct for representing theorem-proof blocks. It has the form $\mathit{thm}(\bullet, \bullet)$: Its first argument contains the theorem assertion, and its second argument contains the proof of the theorem.

We will discuss in section 9 how definitions can be represented in PTL. So we now only need to explain how texts and text fragments without definitions and theorem-proof blocks can be represented in PTL: The concatenation of sentences is usually rendered by conjoining their respective translations with $\wedge$. A special case are axioms and local assumptions: When an axiom appears in a text, the part of the text starting at the axiom is translated by a formula of the form $\varphi \to \vartheta$, where φ is the translation of the axiom and ϑ is the translation of the text following the axiom. The translation of local assumptions is similar, only that one has to take into account the scope of the assumption: The scope of an assumption is translated as $\varphi \to \vartheta$, where φ is the translation of the assumption and ϑ is the translation of the rest of the scope of the assumption. The translation of the scope of an assumption is embedded into the translation of a complete text as if the whole scope of the assumption were a single sentence.

4.3 Metalinguistic expressions

The Naproche CNL allows for expressions of the form “If case 2 holds, then …”. Here “case 2” is a metalinguistic noun phrase that anaphorically refers back to a previously introduced case.

Furthermore, the Naproche CNL allows for a sentence of the form “At most one of the following statements holds:” followed by a list of sentences. Here “the following statements” is a metalinguistic noun phrase that cataphorically refers to the statements that follow it. Different kinds of logical relations between statements can be expressed in this way, using terms like “at most one”, “at least one”, “holds”, “does not hold” and “are inconsistent”.

4.4 Such-that clauses

One conspicuous feature of mathematical texts is the abundance of *such-that clauses*. These are subclauses that similarly to normal relative clauses specify a noun phrase further. Normally the noun phrase is a variable or an apposition of a contentful noun phrase and a variable, and the variable is used in the such-that clause in order to refer back to the specified noun phrase:

(4) A contains some integer k such that $k^2 > 4$.

In this example, the such-that clause “such that $k^2 > 4$” specifies further the noun phrase “some integer k”, which is an apposition of the noun phrase “some integer” and the variable k.

One problem that arises when introducing such-that clauses into a controlled natural language is that the end of such-that clauses is not explicitly marked, which can be a cause of ambiguity:

(5) A does not contain an integer k such that $k^2 > 4$ and A is finite.

This sentence can be read as the conjunction of “A does not contain an integer k such that $k^2 > 4$” and “A is finite”, or as a sentence containing the such that clause “such that $k^2 > 4$ and A is finite”. In the Naproche CNL we use the principle that the reading given to such a sentence is the one where the such-that clause goes on as long as possible, i.e. the second of the two readings just mentioned. Similar disambiguation principles are also applied to other possible sources of ambiguity of the Naproche CNL.

4.5 Complex and plural noun phrases

Complex and plural noun phrases can be interpreted either *collectively* and *distributively*:

(6) 12 and 25 are coprime.

(7) 2 and 3 are prime numbers.

(7) is interpreted as “2 is prime and 3 is prime”: This is called a *distributive* reading of the complex noun phrase “2 and 3”, because the predicate “prime” distributes over the two conjuncts of the complex noun phrase. (6) is interpreted as “12 is coprime to 25”, with a *collective* reading of “12 and 25”.

The same noun phrase may have to be interpreted collectively with respect to one predicate and distributively with respect to another:

(8) p_1 and p_2 are distinct primes.

Here “p_1 and p_2” has to be interpreted collectively with respect to the predicate “distinct” and distributively with respect to the predicate “prime”.

A similar ambiguity arises with respect to the scope of quantifiers in sentences with complex or plural noun phrases:

(9) A and B contain some prime number.

(9) can mean either that A contains a prime number and B contains a (possibly different) prime number, or that there is a prime number that is contained in both A and B. In the first case we say that the scope of the noun phrase conjunction “A and B” contains the quantifier “some”, whereas in the second case we say that the scope of “some” contains the noun phrase conjunction. We call the first reading the *wide-conjunction-scope* reading and the second the *narrow-conjunction-scope* reading.

Sometimes certain considerations of reference or variable range force one of the two readings, as in (10) and (11).

(10) x and y are integers such that some odd prime number divides $x + y$.

(11) x and y are prime numbers p such that some odd prime number q divides $p + 1$.

(10) only has a narrow-conjunction-scope reading, because the existentially introduced entity is linked via a predicate (“divides”) to a term (“$x + y$”) that refers to the coordinated noun phrases individually. (11) on the other hand only has a wide-conjunction-scope reading, because the variable p must range over the values of both x and y, and q depends on p.

In mathematical texts, a wide-conjunction-scope reading is usually preferred over a narrow-conjunction-scope reading unless the narrow-conjunction-scope reading is forced as in the case of (10).

We have implemented a *plural interpretation algorithm* which disambiguates between collective and distributive readings and between wide-conjunction-scope and narrow-conjunction-scope readings in a way that generally agrees with the natural reading in mathematical texts. The details of this algorithm are described in [10] and in section 7.6 of [11].

5 Symbolic mathematics in the Naproche CNL

One of the conspicuous features of the language of mathematics is the way it integrates mathematical symbols into natural language material. We use the term *symbolic mathematics* for all the expressions found in mathematical texts formed from mathematical symbols. One could be tempted to think that symbolic mathematics would be easy to describe formally and hence easy to incorporate in a controlled natural language. However, it turns out that the language of symbolic mathematics is extremely rich and flexible, and that its syntax gives rise to potential ambiguities that can only be disambiguated using contextual knowledge.

Already at first sight, a whole variety of syntactic rules are encountered for forming complex terms and formulae out of simpler ones; a basic classification of these was provided by [27]:

- There are infix operators that are used to combine two terms to one complex term, e.g. the $+$ symbol in $m+n$ or $\frac{1}{x}+\frac{x}{1+x}$.
- There are suffix operators that are added after a term to form another term, e.g. the ! symbol in $n!$.
- There are prefix operators that are added in front of a term to form another term, e.g. sin in $\sin x$.
- There are infix relation symbols used to construct a formula out of two terms, e.g. the $<$ symbol in $x<2$.

As noted by [15], "this simple classification is adequate for the fragment Ranta is considering, but does not come close to capturing the breadth of symbolic material in mathematics as a whole." It does not include notations like $[K:k]$ for the degree of a field extension, it does not allow infix operators to have an internal structure, like the $*_G$ in $a *_G b$ for denoting multiplication in a group G, nor does it account for the common way of expressing multiplication by concatenation, as in $a(b+c)$.

Another kind of prefix operator not mentioned by Ranta is the one that requires its argument(s) to be bracketed, e.g. f in $f(x)$. (Of course, the argument of a prefix operator like sin might also be bracketed, but generally this is done only if the argument is complex and the brackets are needed for making sure the term is disambiguated correctly.) This is even the standard syntax for applying functions to their arguments, in the sense that a newly defined function would be used in this way unless its definition already specifies that it should be used in another way.

The expression $a(x+y)$ can be understood in two completely different ways, depending on what kind of meaning is given to a: If a is a function symbol and $x+y$ denotes a legitimate argument for it, then $a(x+y)$ would be understood to be the result of applying the function a to $x+y$. If on the other hand a, x and y are – for example – all real numbers, then $a(x+y)$ would be understood as the product of a and $x+y$. Now whether a is a function or a real number should have been specified (whether explicitly or implicitly) in the preceding text. So we can conclude that the disambiguation of symbolic expressions requires information from the preceding text, and this information might have been provided in natural language rather than in a symbolic way.

In the Naproche CNL, we have incorporated a large part of the rich flexibility of symbolic mathematics. The potential ambiguities arising from this flexible syntax are resolved using a combination of three criteria:

1. A type system for symbolic mathematics: Every subterm and subformula of a mathematical expression is assigned a type, and when combining function symbols with argument terms to form more complex terms, the system checks that the argument terms have the type that is expected for the function symbol in question.
2. Presupposition checking: Some functions may only take arguments satisfying certain presuppositions as arguments. For example, the second argument of the division function must be non-zero. For a mathematical expression with multiple potential readings, readings whose presuppositions are fulfilled are always preferred over readings whose presuppositions are not fulfilled, so that presupposition checking contributes to the disambiguation process.
3. In the rare case that the two previously mentioned methods do not fully disambiguate a term, preference is given to a reading that involves more recently introduced notation.

The details of the type system and the overall disambiguation process are described in [8] and in section 7.4 of [11].

6 Implicit dynamic introduction of function symbols

Functions are often dynamically introduced in an implicit way in mathematical texts. For example, [28] introduces the additive inverse function on the reals as follows:

(12) For each a there is a real number $-a$ such that $a + (-a) = 0$.

Here the natural language quantification "there is a real number $-a$" *locally* (i.e. inside the scope of "For each a") introduces a new real number to the discourse. But since the choice of this real number depends on a and we are universally quantifying over a, it *globally* (i.e. outside the scope of "For each a") introduces a function "$-$" to the discourse.

The most common form of implicitly introduced functions are functions whose argument is written as a subscript, as in the following example:

(13) Since f is continuous at t, there is an open interval I_t containing t such that $|f(x) - f(t)| < 1$ if $x \in I_t \cap [a, b]$. [28]

If one wants to later explicitly call the implicitly introduced function a function, the standard notation with a bracketed argument is preferred:

(14) Suppose that, for each vertex v of K, there is a vertex $g(v)$ of L such that $f(st_K(v)) \subset st_L(g(v))$. Then g is a simplicial map $V(K) \to V(L)$, and $|g| \backsimeq f$. [20]

Implicitly introduced functions generally have a restricted domain and are not defined on the whole universe of the discourse. For example in (14), g is only defined on vertices of K and not on vertices of L. Implicit function introduction can also be used to introduce multi-argument functions. For example, subtraction on the reals could be introduced by a sentence of the following form:

(15) For all reals a, b, there is a real $a - b$ such that $(a - b) + b = a$.

For the sake of simplicity and brevity, we restrict ourselves to unary functions for the rest of this paper.

If the implicit introduction of functions is allowed without limitations, one can derive a contradiction:

(16) For every function f, there is a natural number $g(f)$ such that

$$g(f) = \begin{cases} 0 & \text{if } f \in \mathit{dom}(f) \text{ and } f(f) \neq 0, \\ 1 & \text{if } f \notin \mathit{dom}(f) \text{ or } f(f) = 0. \end{cases}$$

Then g is defined on every function, i.e. $g(g)$ is defined. But from the definition of g, $g(g) = 0$ iff $g(g) \neq 0$.

This contradiction is due to the *unrestricted function comprehension* that is implicitly assumed when allowing implicit introductions of functions without limitations. Unrestricted function comprehension could be formalized as an axiom schema as follows:

Unrestricted function comprehension
For every formula $\varphi(x, y)$, the following is an axiom:

$$\forall x \; \exists y \; \varphi(x, y) \rightarrow \exists f \; \forall x \; \varphi(x, f(x))$$

The inconsistency of unrestricted function comprehension is analogous to the inconsistency of unrestricted set comprehension, i.e. Russell's paradox.

This paradox can be avoided by using a type theory. In the next section we will sketch a system that formalizes implicit dynamic function introduction by extending DPL to *Typed Higher-Order Dynamic Predicate Logic*. It is also possible to formalize implicit dynamic function introduction in an untyped *Higher-Order Dynamic Predicate Logic*, which has the consistency strength of ZFC (see [11]).

7 Typed Higher-Order Dynamic Predicate Logic

In this section, we extend DPL to a system called *Typed Higher-Order Dynamic Predicate Logic* (THODPL), which formalizes implicit dynamic function introduction, and also allows for explicit quantification over functions. Note that the extensions to DPL introduced in this section are also included in PTL.

THODPL has variables typed by the types of Simple Type Theory [3]. In the examples below we use x and y as variables of the basic type i, and f as a variable of the function type $i \to i$. A complex term is built by well-typed application of a function-type variable to an already built term, e.g. $f(x)$ or $f(f(x))$.

The distinctive feature of THODPL syntax is that it allows not only variables but any well-formed terms to come after quantifiers. So (17) is a well-formed formula:

(17) $\forall x\ \exists f(x)\ R(x, f(x))$

(18) $\forall x\ \exists y\ R(x, y)$

(19) $\exists f\ (\forall x\ R(x, f(x)))$

The semantics of THODPL is to be defined in such a way that (17) has the same truth conditions as (18). But unlike (18), (17) dynamically introduces the function symbol f to the context, and should hence be equivalent to (19).

We now sketch how these desired properties of the semantics can be achieved. In THODPL semantics, an assignment assigns elements of $|S|$ to variables of type i, functions from $|S|$ to $|S|$ to variables of type $i \to i$ etc. Additionally, an assignment can also assign an object (or function) to a complex term. For example, any assignment in the interpretation of $\exists f(x)\ R(x, f(x))$ has to assign some object to $f(x)$. The definition of $g[x]h$ can now naturally be extended to a definition of $g[t]h$ for terms t. The definition of $[t]^g_S$ has to be adapted in the natural way to account for function variables.

Just as in the case of DPL semantics, we recursively define an interpretation $[\![\bullet]\!]^g_S$ from THODPL formulae to subsets of G_S (the cases 1-5 of the recursive definition are as before):

6. $[\![\varphi \to \psi]\!]^g_S := \{h | h$ differs from g in at most some function variables $f_1, \ldots, f_n$ (where this choice of function variables is maximal), and there is a variable x such that for all $k \in [\![\varphi]\!]^g_S$, there is an assignment $j \in [\![\psi]\!]^k_S$ such that $j(f_i(x)) = h(f_i)(k(x))$ for $1 \leq i \leq n$, and if $n > 0$ then $k[x]g\ \}$
7. $[\![\exists t \varphi]\!]^g_S := \{h |$there is a k s.t. $k[t]g$ and $h \in [\![\varphi]\!]^k_S\}$

In order to make case 6 of the definition more comprehensible, let us consider its role in determining the semantics of (17), i.e. of $\exists x \top \to \exists f(x)\ R(x, f(x))$: $[\![\exists f(x)\ R(x, f(x))]\!]^k_S$ is the set of assignments j satisfying $R(x, f(x))$ (i.e. for which $[\![R(x, f(x))]\!]^j_S$ is non-empty) such that $j[f(x)]k$. $[\![\exists x \top]\!]^g_S$ is the set of assignments k such that $k[x]g$. So by case 6 with $n = 1$,

$$\begin{aligned}[\![\exists x \top \to \exists f(x)\ R(x, f(x))]\!]^g_S &= \{h | h[f]g \text{ and there is a variable } x \text{ such that for all } k \text{ such that } k[x]g, \text{ there is an assignment } j \text{ satisfying } R(x, f(x)) \text{ such that } j[f(x)]k \text{ and } j(f(x)) = h(f)(k(x)), \text{ and } k[x]g\} \\ &= \{h | h[f]g \text{ and for all } k \text{ such that } k[x]g, \text{ there is an assignment } j \text{ satisfying } R(x, f(x)) \text{ such that } j[f(x)]k \text{ and } j(f(x)) = h(f)(k(x))\}\end{aligned}$$

$$= \{h|h[f]g \text{ and for all } k \text{ such that } k[x]h,\ k \text{ satisfies } R(x, f(x))\}$$
$$= [\![\exists f\ (\forall x\ R(x, f(x)))]\!]_S^g$$

8 Proof-checking mathematical texts in the Naproche system

The Naproche system checks the logical correctness of mathematical texts written in the *Naproche CNL*. By "checking the logical correctness" we mean that it tries to establish all proof steps found in the text based on the information gathered from previous parts of the text, in a similar way as a mathematician reads a (logically self-contained) mathematical text if asked not to use his mathematical knowledge originating from other sources.

The Naproche system contains two main modules, the linguistic module which translates an input Naproche CNL text into a PRS, and the proof checking algorithm, which checks whether the PRS represents a logically correct text. Since we are working with the DPL extension PTL instead of PRSs in this paper, we will describe how the proof checking algorithm checks PTL formulae.

In the explanations in this section we are assuming that the input text does not refer to any partial functions and does not contain definite descriptions (i.e. does not contain the word "the"). Partial functions and definite descriptions trigger *presuppositions*. The way these are handled in the proof-checking algorithm is described in [9] and [12].

For checking single proof steps, the Naproche system makes use of state-of-the-art *automated theorem provers* (ATPs) for standard first-order logic. Given a set of *premises*[4] Γ and a *conjecture* φ, an ATP tries to find either a proof that the Γ logically implies φ, or build a model for $\Gamma \cup \{\neg\varphi\}$, which shows that they do not imply it. A conjecture together with a set of premises handed to an ATP is called a *proof obligation*. We denote the proof obligation consisting of premises Γ and conjecture φ by $\Gamma \vdash^? \varphi$.

The proof checking algorithm keeps track of a list of first-order premises considered to be true. This list gets updated continuously during the checking process. Each assertion is checked by an ATP based on the currently active premises.

Before we explain the details of the proof checking algorithm, we illustrate its functioning on an example. Suppose that the Naproche system has to check the text in (20). (21) is the PTL translation of (20), i.e. the input to the proof checking algorithm.

[4] In the ATP community, the term "axiom" is usually used for what we call "premise" here; the reason for our deviation from the standard terminology is that in the context of our work the word "axiom" means a completely different thing, namely an axiom stated inside a mathematical text that is to be checked by the Naproche system. The premises that we are considering here can originate either from axioms, from definitions, from assumptions or from previously proved results.

(20) Assume n is a square number. So there is a k such that $n = k^2$. If n is even, then k is even, i.e. 4 divides n.

(21) $(\text{square}(n) \rightarrow \exists k\ n = k^2 \wedge (\text{even}(n) \rightarrow (\text{even}(k) \wedge \text{divides}(4, n))))$

Suppose further that Γ is the set of premises that is active on encountering this text fragment (i.e. Γ is the set of premises that the proof-checking algorithm has learned from previous text parts). Then the proof checking algorithm will check (21) by sending the three proof obligations (22), (23) and (24) to an ATP:

(22) $\Gamma, \text{square}(n) \vdash^? \exists k\ n = k^2$

(23) $\Gamma, \text{square}(n), n = k^2, \text{even}(n) \vdash^? \text{even}(k)$

(24) $\Gamma, \text{square}(n), n = k^2, \text{even}(n), \text{even}(k) \vdash^? \text{divides}(4, n)$

Let us first explain the functioning of the algorithm in terms of how it works on a Naproche CNL text like (20): Reading in the assumption "Assume n is a square number", the proof checking algorithm adds "square(n)" to the premise list. It then checks the rest of the text, which is completely inside the scope of this assumption, using this extended premise list. (22) is the proof obligation that checks the sentence "So there is a k such that $n = k^2$". Having checked the existence of a k with $n = k^2$, $n = k^2$ is then added to the premise list for checking the final sentence. Note that we do not add the checked existentially quantified formula $\exists k\ n = k^2$ to the premise list, but its Skolemized form $n = k^2$. This ensures that the k in this new premise corefers with the variable k in the final sentence. (Adding $\exists k\ n = k^2$ to the premise list would be like adding $\exists m\ n = m^2$ for some unused variable m.) When checking the final sentence, the translation "even(n)" of the antecedent "n is even" of the implication is added to the premise list, and the two parts of the consequence are checked one after the other.

However, as explained before, the proof checking algorithm does not directly work on the Naproche CNL input, but on its PTL translation. For understanding how the proof checking algorithm produces the proof obligations (22), (23) and (24) out of the input (21), one needs to know the rules for checking atomic formulae, implications, conjunctions and existentially quantified sentences explained formally below.

We use Γ to denote the list of premises considered true before encountering the formula in question, and Γ' to denote the list of premises considered true after encountering the formula in question. For any given PTL formula φ, we denote by $FI(\varphi)$ the *formula image* of φ, which is a list of first-order formulae representing the content of φ; the definitions of $FI(\varphi)$ and of the checking algorithm are mutually recursive, as specified below.

- If φ is atomic, check $\Gamma \vdash^? \varphi$ and set Γ' to be Γ, φ.
- If φ is of the form $\varphi_1 \wedge \varphi_2$, check φ_1 with premise list Γ and φ_2 with the premise list that is active after checking φ_1; set Γ' to be the premise list that is active after checking φ_2.

- If φ is of the form $\varphi_1 \to \varphi_2$, check φ_2 with premise list $\Gamma \cup FI(\varphi_1)$ and set Γ' to be $\Gamma \cup \{\forall x_1, \ldots, x_n \bigwedge FI(\varphi_1) \to \psi \mid \psi \in FI(\varphi_2)\}$, where $x_1, \ldots, x_n$ are the variables which are introduced by an existential quantifier in φ_1 and whose occurrences in φ_2 can be bound by this quantifier.
- If φ is of the form $\neg\psi$, check $\Gamma \vdash^? \neg \bigwedge FI(\psi)$ and set Γ' to be $\Gamma, \neg \bigwedge FI(\psi)$.
- If φ is of the form $\varphi_1 \vee \varphi_2$, check $\Gamma \vdash^? \bigwedge FI(\varphi_1) \vee \bigwedge FI(\varphi_2)$ and set Γ' to be $\Gamma, \bigwedge FI(\varphi_1) \vee \bigwedge FI(\varphi_2)$.
- If φ is of the form $\exists t\ \psi$, check $\Gamma \vdash^? \exists x\ \bigwedge FI(\psi\frac{x}{t})$[5] for some fresh variable x and set Γ' to be $\Gamma \cup FI(\psi)$.
- If φ is of the form $thm(\varphi, \psi)$, first check ψ with premise list Γ and then check φ with premise list $\Gamma \cup FI(\psi)$; set Γ' to be $\Gamma \cup FI(\varphi)$.

For computing $FI(\varphi)$, the algorithm proceeds analogously to the checking of φ, only that no proof obligations are sent to the ATP: The updated premise lists are still computed, and $FI(\varphi)$ is defined to be $\Gamma' - \Gamma$, where Γ is the active premise list before processing φ and Γ' is the active premise list after processing φ.

We should briefly discuss how the definitions of the $\varphi_1 \to \varphi_2$ and $\exists t\ \psi$ cases of the proof checking algorithm are used in checking implicit dynamic function introductions. Remember the three example formulae whose relationship we described in section 7:

(17) $\forall x\ \exists f(x)\ R(x, f(x))$

(18) $\forall x\ \exists y\ R(x, y)$

(19) $\exists f\ (\forall x\ R(x, f(x)))$

For checking $\forall x\ \exists f(x)\ R(x, f(x))$[6], the conjecture $\exists y\ R(x, y)$ will be checked under the active premise list (where y is some fresh variable). The premise list is then extended by $\forall x\ (\top \to R(x, f(x)))$, which is trivially equivalent to $\forall x\ R(x, f(x))$. So when checking (17), it behaves like (18), whereas the premise that is added after having checked it is equivalent to the premise added after having checked (19). This corresponds to the semantic relationship between these formulae discussed in section 7. Note that in this way, functions can be introduced without having to check a proof obligation whose conjecture explicitly asserts the existence of a function.

We have proven two soundness and two completeness theorems for the proof checking algorithm: In each case one theorem compares the algorithm to the semantics of PTL and one theorem compares it to the semantics of standard first-order predicate logic. See chapter 6 of [11] for details.

[5] $\psi\frac{x}{t}$ is the formula obtained from ψ by replacing all occurrences of t by x.
[6] In PTL just as in DPL, $\forall x\ \varphi$ is an abbreviation for $\exists x\ \top \to \varphi$.

9 Definitions in mathematical texts

A further very conspicuous feature of the language of mathematics is its *adaptivity*[7] through definitions: The language is constantly expanded through the use of definitions, which introduce new natural-language or symbolic expressions and fully specify their meaning. This expansion of the language should not be confused with the change of language over time: What we mean is an expansion of the language used for one particular text and – related to this – an expansion of the language in the mind of a mathematician reading such a text.

The introduction of new technical terms through definitions does, of course, also exist in other specialized languages. But, as Ganesalingam [15] has pointed out, there are two important differences between definitions in mathematics and in other fields: Firstly, mathematical definitions contain no vagueness and hence perfectly specify the semantics of the defined expression. Secondly, in advanced mathematics all newly introduced terms are introduced through definitions, and mathematicians even go back to less advanced mathematics and rigorously define all terms used there.

We can distinguish expansions of the natural-language lexicon of the the language of mathematics and extensions of the symbolic part of the language. (25) is an example of a definition expanding only the natural-language lexicon:

(25) **Definition 1.1.5** A set D is *dense in the reals* if every open interval (a, b) contains a member of D. [28, p. 6]

(26) expands both the natural-language lexicon (by the word "sum") and the symbolic part of the language (by a construct of the form "$\bullet + \bullet + \cdots + \bullet$"):

(26) **Definition** Suppose R is a ring and A_1, A_2, $\ldots$, A_m are ideals of R. Then the *sum* $A_1 + A_2 + \cdots + A_m$ is the set of all $a_1 + a_2 + \cdots + a_m$ with $a_i \in A_i$. [5, p. 108]

Ganesalingam [15] considers the expansion of the symbolic part of the language as an expansion of the *syntax* of this symbolic part. This is certainly a very sensible interpretation at a certain level of abstraction in the understanding of the term "syntax". We, however, prefer to take a more abstract view of syntax, under which this expansion of the symbolic part of the language can be viewed as an expansion of the lexicon, just as in the case of the expansion of the lexicon of the textual part of the language. For example, under this interpretation, the definition in (26) adds a lexical item of the form "$\bullet + \bullet + \cdots + \bullet$" to the lexicon of the symbolic part of the language. The syntax of the language under this interpretation must contain rules that specify what form definitions can take, what properties the symbolic lexical items have depending on the form of the definition, and in what way these properties influence how different items of the symbolic lexicon can be combined to symbolic expressions. In this way this abstract syntax indirectly specifies how definitions change what form symbolic

[7] The use of the term *adaptivity* for this feature of the language of mathematics is due to Ganesalingam [15].

expressions following the definition can take. Thus this abstract syntax specifies a more concrete syntax (i.e. a *syntax* in the way Ganesalingam used the term) for every position in a mathematical text, depending on which previously stated definitions are *accessible*, i.e. may be made use of.

In the Naproche CNL, definitions are formed according to special rules. For fixing the meaning of the defined word or symbolic expression, we equate a *definiendum*, whose meaning is to be specified, to a *definiens*, which specifies the meaning of the definiendum. The definiendum is either the symbol that is being introduced, or the word or symbol that is being introduced applied to some dummy variables, which may also appear in the definiens. If the definiendum and definiens represent propositions, they are equated by a bi-implication ("iff" or "if and only if"); else they are equated by an inflected form of the verb "to be". So we distinguish between *bi-implicational definitions* and *copula definitions.*

Here are some examples of bi-implicational definitions in the Naproche CNL:

1. Define n to be even if and only if there is some k such that $n = 2k$.
2. Define a real r to be an integer iff there is a natural number n such that $r = n$ or $r = n \cdot (-1)$.
3. Define a line L to be parallel to a line M iff there is no point on L and M.
4. Define m and n to be coprime iff $(m, n) = 1$.
5. Define an integer m to divide an integer n iff there is an integer k such that $km = n$.
6. Define $R(x, y, z)$ iff $x = y = z$ or $4x < 2y < z$.
7. Define $m|_k n$ iff there is an $l < k$ such that $m \cdot l = n$.

Here are some example of copula definitions:

1. Define c to be $\sqrt{\frac{\pi}{6}}$.
2. Define $f(x)$ to be x^2.
3. Define $f_x(y, z)$ to be $x(2y - 5z)$.
4. For defining ! at x', define $x'!$ to be $x' \cdot x!$.

For details about the rules for forming definitions in the Naproche CNL, see section 7.3.7 of [11].

PTL does not have any explicit notation for definitions. Definitions can be considered to extend the language by the symbolic construct or word that they are defining. But in a similar way, existentially quantified statements could be considered to extend the language due to the dynamic nature of the existential quantifier: The variable we quantify over becomes a possible antecedent for later uses of the same variable; so in a sense we have extended the language by that variable.

In definitions without dummy variables, we make direct use of this analogy by rendering the definition by an existentially quantified PTL formula:[8]

(27) Define c to be $a + g(a)$.

[8] All examples in this section are considered to appear in a context in which a unary classical function symbol g, a binary infix function symbol $+$ and the variable a are accessible, whereas x is not accessible.

(28) $\exists c\; c = +(a, g(a))$

The translation (28) of (27) existentially introduces the PTL variable c that corresponds to the defined symbol c. In the proof checking algorithm, this existential PTL formula makes the premise $c = +(a, g(a))$ available for proving later assertions, just as would be expected for definition (27).

There is one point in the proof checking algorithm where one might think that the analogy between definitions and existential PTL formulae breaks down: The existential PTL formula triggers a proof obligation with an existential conjecture; in our example, this means that $\exists c\; c = +(a, g(a))$ has to be proven to follow from the active premise list. For a definition, on the other hand, one would not expect any proof obligation, since it does not make an assertion but just expands the language. But note that $\exists c\; c = +(a, g(a))$ is a very simple logical tautology that can be immediately recognised as a logical tautology by any ATP. So this difference between existential assertions and definitions is not a real issue. Hence we can say that our choice to translate definitions with existential PTL formulae is justified.

Now in the case that a definition contains dummy variables, its translation is not an existential formula, but an implication whose antecedent introduces the dummy variable and whose consequence contains an existential claim:

(29) Define $f(x)$ to be $x + x$.

(30) $\exists x\; \top \rightarrow \exists f(x)\; f(x) = +(x, x)$

The principle of implicit dynamic function introduction in PTL (see sections 6 and 7) now ensures that the translation (30) dynamically introduces the function symbol f as a possible antecedent for subsequent parts of a PTL formula containing (30) as a subformula. The conjecture of the proof obligation which (30) triggers is still an existential claim that trivially follows from the active premise list, namely $\exists x\; (x = +(x, x))$.

For treating bi-implicational definitions analogously, we need to consider a relation to be a function whose codomain is the set $\{\top, \bot\}$ of Booleans. Let us see the analogy on a concrete example:

(31) Define an integer x to be even iff $x = g(x)$.

(32) $\exists x\; \mathit{integer}(x) \rightarrow \exists \mathit{even}(x)\; (\mathit{even}(x) = \top \leftrightarrow x = g(x))$

The PTL translation (32) of (31) now triggers a proof obligation with conjecture $\exists y\; (y = \top \leftrightarrow x = g(x))$, which again is a trivial logical tautology. The premise that is added to the premise list after checking this definition is $\mathit{even}(x) = \top \leftrightarrow x = g(x)$, as one would expect for a bi-implicational definition.

10 Bi-implications and reversed implications

In bi-implications and reversed implications, a phenomenon similar to that of the donkey sentences discussed in section 3 can be observed:

(33) 2 divides an integer x iff x is even.

(34) $\forall x\ (integer(x) \to (divide(2,x) \leftrightarrow even(x)))$

The natural interpretation of (33) in PL is (34), i.e. the variable x introduced in an indefinite noun phrase in the left part ("antecedent") of the bi-implication is interpreted as globally universally quantified. This corresponds to the interpretation of mathematical *donkey sentences* like (35), where a variable introduced in an indefinite noun in the antecedent of a usual implication is interpreted as globally universally quantified:

(35) If a space X retracts onto a subspace A, then the homomorphism $i_* : \pi_1(A, x_0) \to \pi_1(X, x_0)$ induced by the inclusion $i : A \hookrightarrow X$ is injective.

The phenomenon appears in the same way in reversed implications:

(36) 2 divides an integer x if x is even.

(37) $\forall x\ (integer(x) \to (even(x) \to divide(2,x)))$

Reversed implications – unlike bi-implications – are common not only in the language of mathematics but also in general language use. Nevertheless, there has to our knowledge not been any systematic study of this donkey-sentence-like phenomenon for reversed implications. It might be the case that in common language it can be reduced to the phenomenon of *generic readings* of indefinite noun phrases (see [4] for an overview over generic interpretation of noun phrases). A generic reading differs semantically from a universal reading in that it is restricted to *typical* members of a category. But in the language of mathematics this typicality restriction of generic readings is generally dropped and generic noun phrases are interpreted in the same way as universal quantifiers. Hence, for the purpose of our interpretation of reversed implications and bi-implications in the Naproche CNL, we can ignore this possible linguistic difference between usual donkey sentence and the phenomenon discussed here.

In the discussion that follows we will, for simplifying the exposition, concentrate on bi-implications, with the understanding that everything we say could just as well be said about reversed implications.

In the case of usual implications, the donkey-sentence phenomenon is treated by the interpretation of $\exists$ and $\to$ in PTL and hence does not need to be treated separately in the Naproche-CNL-to-PTL translation. The analogous phenomenon for bi-implications, however, presents some problems which the usual donkey sentences do not present, and which have been the motivation for treating this issue not within the semantics of the purely formal language PTL, but within the Naproche-CNL-to-PTL translation.

The first problem is that the conjunct $integer(x) \land divide(2,x)$ in the PTL translation $\exists x\ (integer(x) \land divide(2,x))$ of "2 divides an integer x" has to be split up in order to attain the intended interpretation (34): $integer(x)$ has to restrict the universal quantification, while $divide(2,x)$ has to become one argument of the logical bi-implication. So we cannot make use of the semantics $\exists x\ (\mathit{integer}(x) \wedge \mathit{divide}(2,x))$ in a compositional way.

Indefinite noun phrases always give rise to PTL formulae of the form $\exists x\ (\varphi \wedge \psi)$, where φ results from the indefinite noun phrase itself and ψ results from other expressions in the semantic scope of the noun phrase (e.g. from the verb phrase whose subject is the noun phrase in question). Of course each of φ and ψ may again be a conjunction, but the bracketing of the complex conjunction which $\varphi \wedge \psi$ is in that case tells us which parts come from the indefinite noun phrase itself and which one from other expressions. The solution to the first problem is that the part φ which results from the indefinite noun phrase itself is used to restrict the universal quantification over x, whereas the part ψ which results from other expressions becomes part of the logical bi-implication.

The second problem is that not every existential quantification in the left part of a bi-implication is to be interpreted as a universal quantifier outside the scope of the bi-implication. Consider sentence (38), whose natural interpretation in PL is (39) and not (40):

(38) For all n, n divides a prime number iff $n = 1$ or n is prime.

(39) $\forall n\ (\exists p\ (\mathit{prime}(p) \wedge \mathit{divide}(n,p)) \leftrightarrow n = 1 \vee \mathit{prime}(n))$

(40) $\forall n\ \forall p\ (\mathit{prime}(p) \wedge \mathit{divide}(n,p) \leftrightarrow n = 1 \vee \mathit{prime}(n))$

If we interpreted all existential quantifications in the left part of a bi-implication in a universal way, we would get the interpretation (40), which however is not equivalent to the natural interpretation (39).

In the case of usual implications, this problem does not arise, since $\exists x\ P(x) \rightarrow Q$ is equivalent to $\forall x\ (P(x) \rightarrow Q)$. Since the analogous formulae involving bi-implications – $\exists x\ P(x) \leftrightarrow Q$ and $\forall x\ (P(x) \leftrightarrow Q)$ – are not equivalent, the phenomenon now discussed for bi-implications inherently causes problems that the donkey sentences cannot cause.

The basic idea for solving this second problem is simple: We give a universal interpretation only to those existential quantifiers introduced in the left part of the bi-implication which serve as anaphoric antecedents for an expression in the right part ("succedent") of the bi-implication. However, there is a problem with this basic solution. Consider for example sentence (41):

(41) An integer k such that $k^2 - 1$ is prime divides an integer l iff $l^2 - 1$ is prime.

(42) $\forall l\ (\mathit{integer}(l) \rightarrow (\exists k\ (\mathit{integer}(k) \wedge \mathit{prime}(k^2-1) \wedge \mathit{divide}(k,l)) \leftrightarrow \mathit{prime}(l^2-1)))$

(43) $\forall k\ \forall l\ (\mathit{integer}(k) \wedge \mathit{prime}(k^2-1) \wedge \mathit{integer}(l) \rightarrow (\mathit{divide}(k,l) \leftrightarrow \mathit{prime}(l^2-1)))$

According to the basic solution just proposed, only l would be interpreted in a universal way, as in (42). But since l is introduced after k in the left part of the bi-implication, it is felt to be somehow dependent on k, which makes it very unnatural to give it wider scope than k. Hence the interpretation (43) which gives wide scope and a universal interpretation to both k and l is naturally preferred.

Hence we modify our solution to the second problem as follows: If at least one of the existential quantifiers introduced in the left part of the bi-implication serves as anaphoric antecedent for an expression in the right part, we give a universal interpretation to all existential quantifiers introduced in the left part of the bi-implication preceding or identical to an existential quantifier serving as anaphoric antecedent for an expression in the right part.

The formal definition of this translation of Naproche CNL sentences involving bi-implications to PTL formulae is defined in section 7.5.9 of [11].

11 Related Work

The first thorough analysis of the language of mathematics using techniques from formal linguistics was provided by Aarne Ranta in a number of papers from the mid-1990s [23–27]. Ranta analysed syntactic categories of both symbolic and textual mathematics within the framework of Constructive Type Theory [22]. In the practical application of his analysis, he put a larger emphasis on the conversion of logical representation into the natural language of mathematics than on the conversion in the other direction that is needed in proof-checking mathematical texts.

In 2009, Mohan Ganesalingam published a Ph.D. thesis on the language of mathematics [14], republished as a book in 2013 [15]. His thesis contains a very thorough and detailed analysis of this language, with an emphasis on its formal semantics, but also with aspects of pragmatics and philosophy of mathematics. In the introduction to his thesis, he stated that his thesis "is part of a long-term project to build programs that do mathematics in the same way as humans do" [14, p. 9]. The analysis in the thesis can be seen as providing the theoretical bedrock for one part of this envisioned program, namely the part that translates the natural language input into a formal semantic representation language.

One of the aspects of the language of mathematics that Ganesalingam has studied in detail and has shed light on is the aspect of adaptivity through definitions discussed in section 9 above. The issue that Ganesalingam studies most thoroughly is that of disambiguating mathematical language, both its textual and its symbolic parts. To handle potential ambiguities, he develops an ingenious novel type system for typing the objects that a given mathematical text refers to. The potential types available at any point in a text are extracted from preceding definitions.

Concerning systems that can check mathematical texts written in a natural input language, there have – before Naproche – been two different and independent approaches, one by Claus Zinn in his doctoral project, and one by the *Evidence Algorithm* project in Kiev.

For his doctoral project, Claus Zinn developed the system *Vip*, a prototypical proof-checker for natural language mathematics [30]. He did not define a CNL, but attempted to parse any input written in the language of mathematics. As a consequence, he had to gear his system towards a single text, for which he chose *An Introduction to the Theory of Numbers* by Hardy and Wright [18]. The approach did not scale well: Only two proofs from this book were both parsed and checked successfully by Vip. Ganesalingam [15] argued that Zinn's linguistic analysis is "heavily tailored for his two proofs", and that it is "of a comparably shallow kind".

Zinn's thesis also provided a detailed analysis of reasoning patterns in mathematical proofs. In his system, the proof-checking was heavily reliant on Alan Bundy's concept of *proof plans* [2]. The idea is that one analyses families of related proofs in order to identify common reasoning patterns in them, which are formally represented in *proof plan schemata*. These guide future proofs of the same family, and enable an automated system to fill in the details that the proof author has omitted. The heavy reliance on proof plans, however, has increased the extent to which the system is tailored towards particular proofs.

The *Evidence Algorithm* project was initiated by Victor Glushkov in the early 1960s in Kiev with the goal to develop a computer system that could check mathematical proofs written in a powerful input language that is close to the natural mathematical language. After developing a prototype called *System for Automated Deduction* (*SAD*), the project came to a halt between the early 1980s an 1998. From 1998 until 2008, Andrei Paskevich reimplemented and improved SAD for his master and doctor projects.

The input language of SAD is less natural than our controlled natural language, but still significantly more natural than the input languages of standard systems for formal mathematics like Mizar, HOL, Isabelle and Coq. The SAD system is up to now the most successful system for producing automatically checkable formal mathematics that can be read by humans almost like natural mathematical texts. Examples of more advanced results formalised in SAD include the Chinese Remainder Theorem, the result that square roots of prime numbers are irrational, the Cauchy-Bunyakovsky-Schwartz inequality for real vectors and Furstenberg's topological proof of the infinitude of primes [21]. Just like Naproche, SAD makes use of an automated theorem prover (ATP) for checking single proof steps. It implements a number of techniques for making the proof obligations sent to the ATP more tractable. For example, definitions are normally not given to the ATP explicitly as premises, but are used for a stepwise definition expansion.

12 Conclusion

For the development of the Naproche system, a number of novel linguistic and logical techniques had to be developed and integrated. In this paper, we have sketched some of these techniques, with special emphasis on the phenomenon of implicit function introduction in mathematical texts, on the usage of definitions for dynamically extending the language of a mathematical text, and on quantifiers in bi-implications and reversed implications that have to be treated in a similar way as in donkey sentences.

The Naproche system has been tested on a number of mathematical texts. The most involved of these have been the first chapter of Landau's *Grundlagen der Analysis* (see chapter 8 of [11]) and the beginning of Euclid's *Elements* with an axiomatization of Euclidean geometry based on the system E by Avigad, Dean and Mumma [1]. While the parsing and disambiguation of these texts works well, there are still limitations to the proof checking: Some proof steps can currently not be checked by the ATP to which we send the proof obligation. These limitations could be reduced by implementing a premise selection – a prototype of which was implemented in a previous version of Naproche [7] – and by using similar techniques as SAD for making the proof obligations more tractable for the ATP.

References

1. Avigad, J., Dean, E., Mumma, J.: A formal system for Euclid's Elements. Review of Symbolic Logic (2009)
2. Bundy, A.: The use of explicit proof plans to guide inductive proofs. In: Lusk, R., Overbeek, R. (eds.) Proceedings of the 9th Conference on Automated Deduction. pp. 111–120. Springer (1988)
3. Church, A.: A formulation of the simple theory of types. Journal of Symbolic Logic 5, 56–68 (1940)
4. Cohen, A.: Genericity. Linguistische Berichte 10, 59–89 (2002)
5. Connell, E.H.: Elements of Abstract and Linear Algebra (1999), `http://www.math.miami.edu/~ec/book/author.html`
6. Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., Veldman, J.: The Naproche Project – Controlled Natural Language Proof Checking of Mathematical Texts. In: Fuchs, N.E. (ed.) Controlled Natural Language, LNAI 5972. pp. 170–186. Springer (2010)
7. Cramer, M., Koepke, P., Kühlwein, D., Schröder, B.: Premise Selection in the Naproche System. In: Giesl, J., Hähnle, R. (eds.) Automated Reasoning, LNAI 6173. pp. 434–440. Springer (2010)
8. Cramer, M., Koepke, P., Schröder, B.: Parsing and Disambiguation of Symbolic Mathematics in the Naproche System. In: Davenport, J., Farmer, W., Rabe, F., Urban, J. (eds.) Intelligent Computer Mathematics, LNAI 6824. pp. 180–195. Springer (2011)
9. Cramer, M., Kühlwein, D., Schröder, B.: Presupposition Projection and Accommodation in Mathematical Texts. In: Pinkal, M., Rehbein, I., Schulte im Walde, S., Storrer, A. (eds.) Semantic Approaches in Natural Language Processing: Proceedings of the Conference on Natural Language Processing 2010 (KONVENS). pp. 29–36. Universaar (2010)
10. Cramer, M., Schröder, B.: Interpreting Plurals in the Naproche CNL. In: Rosner, M., Fuchs, N.E. (eds.) Controlled Natural Language, LNAI 7175. pp. 43–52. Springer (2012)
11. Cramer, M.: Proof-checking mathematical texts in controlled natural language. Ph.D. thesis, University of Bonn (2013)
12. Cramer, M.: Modelling the usage of partial functions and undefined terms using presupposition theory. In: Geschke, S., Löwe, B., Schlicht, P. (eds.) Infinity, Computability and Metamathematics – Festschrift celebrating the 60th birthdays of Peter Koepke and Philip Welch, pp. 71–88. College Publications (2014)

13. Fuchs, N.E., Höfler, S., Kaljurand, K., Rinaldi, F., Schneider, G.: Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. In: Eisinger, N., Maluszynski, J. (eds.) Reasoning Web, First International Summer School 2005, LNCS 3564. pp. 213–250. Springer (2005)
14. Ganesalingam, M.: The Language of Mathematics. PhD thesis, University of Cambridge (2009)
15. Ganesalingam, M.: The Language of Mathematics – A Linguistic and Philosophical Investigation. Springer, Berlin (2013)
16. Geach, P.T.: Reference and Generality. An Examination of Some Medieval and Modern Theories. Cornell University Press, Ithaca, NY (1962)
17. Groenendijk, J., Stokhof, M.: Dynamic Predicate Logic. Linguistics and Philosophy 14(1), 39–100 (1991)
18. Hardy, G.H., Wright, E.M.: An Introduction to the Theory of Numbers. Oxford University Press, Oxford, 4 edn. (1960)
19. Hatcher, A.: Algebraic Topology. Cambridge University Press, Cambridge (2002)
20. Lackenby, M.: Topology and Groups (2008), `http://people.maths.ox.ac.uk/lackenby/tg050908.pdf`
21. Lyaletski, A., Verchinine, K., Paskevich, A.: System for Automated Deduction (examples) (2008), `http://nevidal.org/help-txt.en.html#examples`
22. Martin-Löf, P.: Intuitionistic Type Theory. Bibliopolis, Naples (1984)
23. Ranta, A.: Type theory and the informal language of mathematics. In: Barendregt, H., Nipkow, T. (eds.) Types for Proofs and Programs, LNCS 806. pp. 352–365. Springer (1994)
24. Ranta, A.: Syntactic categories in the language of mathematics. In: Dybjer, P., Nordström, B., Smith, J. (eds.) Types for Proofs and Programs, LNCS 996. pp. 162–182. Springer (1995)
25. Ranta, A.: Context-relative syntactic categories and the formalization of mathematical text. In: Berardi, S., Coppo, M. (eds.) Types for Proofs and Programs, LNCS 1158. pp. 231–248. Springer (1996)
26. Ranta, A.: Structure grammaticales dans le français mathématique I. Mathématiques, Informatique et Sciences Humaines pp. 5–56 (1997)
27. Ranta, A.: Structure grammaticales dans le français mathématique II (suite et fin). Mathématiques, Informatique et Sciences Humaines (139), 5–36 (1997)
28. Trench, W.: Introduction to Real Analysis. Prentice Hall, Upper Saddle River, NJ (2003)
29. Wolfenstein, S.: Introduction to linear algebra and differential equations. Holden-Day, San Francisco (1969)
30. Zinn, C.: Understanding Informal Mathematical Discourse. PhD thesis, Friedrich-Alexander-Universität Erlangen Nürnberg (2004)