Generating Proof Representation Structures in the Project NAPROCHE

Magisterarbeit
zur Erlangung des Grades eines
Magister Artium M.A.

vorgelegt
der
Philosophischen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität
zu Bonn

von

Nickolay Mitkov Kolev
aus
Sofia, Bulgarien

An Eides statt versichere ich, dass die Arbeit

Generating Proof Representation Structures
for the Project Naproche

von mir selbst und ohne jede unerlaubte Hilfe angefertigt wurde, dass sie noch keiner anderen Stelle zur Prüfung vorgelegen hat und dass sie weder ganz, noch im Auszug veröffentlicht worden ist. Die Stellen der Arbeit—einschließlich Tabellen, Karten, Abbildungen usw.—, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, habe ich in jedem einzelnen Fall als Entlehnung kenntlich gemacht.

Bonn, den

Nickolay Kolev

Danksagung

Ich möchte mich im Vorfeld bei dem Betreuer dieser Arbeit, Prof. Dr. Bernhard Schröder, für die tatkräftige Unterschtützung bedanken.

Des Weiteren möchte ich mich bei den Mitgliedern der Naproche Arbeitsgruppe (Prof. Dr. Peter Koepke, Jip Veldman, Merlin Carl und Bernhard Fisseni) für die angenehme Zusammenarbeit bedanken.

Außerdem möchte ich dem Mathematischen Institut der Universität Bonn für die finanzielle Unterstützung meiner Mitarbeit an der Naproche Implementation danken.

Contents

Chapter 1

Introduction

One should not aim at being possible to understand,
but at being impossible to misunderstand.
Quintilian, circa 100 AD

20th century developments in the foundations of mathematics have shown that virtually all of present day mathematical proofs can be exhaustively expressed in a suitable formal system. However, for reasons of readability and practicality, mathematicians often produce their work. This renders such work insusceptible to automatic verification and poses the requirement that its correctness be checked by a trusted social mechanism of peer review.

1.1 Motivation

Naproche (*Na*tural language *Pro*of *Che*cking) is a research project started at the University of Bonn by Prof. Dr. Bernhard Schröder (then Institute for Communication Research, IfK) and Prof. Dr. Peter Koepke (Mathematical Instiute). It aims at providing a system within which mathematical texts written in a controlled natural language can be automatically checked for integrity and correctness. The project is intended as an aid to both authors, who wish to check their work before submitting it, and to reviewers who wish to automatically check work submitted to them. The prototypical users of the system are students of mathematics and related fields such as computer science and physics, who need to turn in mathematical proofs as assignments, and teachers and teaching assistants who need to check large numbers of such proofs.

From a user's point of view, the system is simple. It can be integrated as a plug-in into the

WYSIWYG (what-you-see-is-what-you-get) TEXMACS editing platform (van der Hoeven, 1998) and allows the checking of mathematical proofs at the press of a button. User interaction is minimal—invoking the system on a mathematical text from within TEXMACS produces either a *"Proof accepted."* message if the proof verification succeeded, or a diagnostic message hinting at the error which prevented a successful check. Under the hood, however, the NAPROCHE system is complex and multi-layered. The mathematical text undergoes a number of format transformations, gets tokenised and split into sentences and finally a formal representation of it is constructed. This formal representation is in turn passed on to an automatic proof checker for the actual verification[1] . It is with these transformations in general and with the generation of the formal representation in particular that this work is concerned.

1.2 Overview

The main subject of this work is the generation of formal representations of informal mathematical proofs. In order to approach this task correctly, we shall first try to define the domain we are working in by giving a brief overview of the history and practice of mathematical proofs and compare formal and informal proofs from both a mathematical and a linguistic point of view (chapter 2). We shall then focus on informal proofs and describe their structure and medium of expression, or what is widely referred to in literature as "the language of mathematics". We shall argue that this "language of mathematics", or, as we suggest it should be more appropriately called, "language of mathematical texts", can be viewed as a mixture of a particular ethnic natural language and a universal symbolic language. We shall then introduce the "NAPROCHE language"—a controlled language for writing mathematics—and show how different language constructs map to specific PRSs. We shall look at mechanisms for the generation of formal discourse representations in general and focus on Discourse Representation Theory (DRT) and the formal structures it uses called Discourse Representation Structures (DRS) (chapter 3). DRSs were conceived with natural language discourse in mind and fall short on dealing with some of the specific properties of mathematical texts (section 5.1). We shall therefore propose an extension to the makeup and method of construction of DRSs, yielding Proof Representation Structures (PRSs)—the formal structures used within the NAPROCHE project (chapter 5). Finally, we will review some related work (chapter **??**) and touch on possible directions for the future development of

[1] Currently the NAPROCHE plug-in for TEXMACS uses a home-grown checker written in Prolog by Prof. Dr. Peter Koepke. It is, however, the intention of the project's authors for the system to be able to interface with any available proof checker such as Coq, Otter, HOL or Isabelle.

the NAPROCHE project (chapter 7).

Chapter 2

Mathematical proof

The main subject of study of this work is the mathematical text as a means for the presentation of a mathematical proof. It is therefore only appropriate that we begin our discussion by looking at what proofs are.

2.1 What is a proof?

A proof in mathematics may refer to the process of discovering a justification for a certain mathematical hypothesis or to the discourse produced as a result of the description of such a process. For the purpose of this work we shall adopt the second meaning. Accordingly, by "mathematical text" we shall mean the written manifestation of this discourse. Spoken mathematical lectures and texts which may be considered as being of a mathematical nature *per se*, such as philosophical, methodological or historical overviews, will not be considered subject of this work.

Despite the central role which proofs play in the practise of mathematics, there seems to exist no universally accepted method for constructing proofs or even a general consensus on what a proof actually is. The answers to the second question do however, seem to converge around two distinct claims. The first of those two claims, which we shall call the *traditionalist* view, is that proofs are rhetorical devices which present a convincing argument that a mathematical statement is true. The second claim, which we shall call the *formalist* view, is that proofs are chains of logical reasoning steps carried out exclusively within a formal system which culminate at a certain mathematical statement of which it is then said to have been proven. Those two approaches to proofs entail very different definitions of "proof correctness" and accordingly different methods of proof verification. Both will

be discussed in detail in section 2.3. To understand how it came to be that two different notions, and indeed, standards of proof came to co-exist, we need to look at the evolution of mathematical reasoning.

2.2 The evolution of mathematical proofs

The idea that mathematical statements require deductive justification comes from the ancient Greeks. Pythagoras was the first to deductively prove the theorem bearing his name, although the statement of the theorem had been known for at least 500 years. Aristotle devised the axiomatic method and captured in writing the *syllogisms*—a number of inference rules which lie at the heart of deductive reasoning. Using the axiomatic method, one is allowed to define objects only in terms of already defined objects and derive new statements only from statements one has already shown to be true. To avoid an unending chain of reasoning one needs an inventory of objects and statements which need no such rigid justification—*primitive objects* and *axioms*. Euclid's *Elements* was the first large body of work in mathematics (and, arguably, all of science) which made use of the axiomatic method and the application of deductive thinking. In the course of time mathematicians discovered numerous, albeit non-fatal, flaws with Euclid's reasoning[1]. A much more radical shortcoming, at least from the present day's point of view, is that Euclid argumented exclusively in a natural language which is inherently wrought with ambiguity and susceptible to imprecision. It was only through the work of Galileo and Leibniz some 1900 years later that mathematical language acquired its now common form of a mixture of natural language and symbolic expressions. Despite these shortcomings the *Elements* set a paradigm on how mathematics is to be written and was regarded as an example of rigorous proof *par excellence* until well into the 19^{th} century.

The 19^{th} century saw the advent of the *modern* axiomatic method. The vital new characteristic of this method when compared with Aristotle's was that the statements a mathematician uses should not only follow from statements which have previously been shown to be true, but that they should do so in a purely logical manner, without the need for interpretation. Instrumental in this regards was George Bolle's *Calculus of Logic* (Boole, 1848) in which he described a formal system for logical and set-theoretical reasoning. Boole's calculus lacked quantifiers, however, and was therefore only suitable for propositional reasoning. Gottlob Frege's introduction of quantifiers and the publication of his

[1] Both Gauss and Leibniz pointed out that Euclid made use of terms and properties of objects, such as the notion of *betweenness* without justifying them. Those notions are intuitive, but a rigorous application of the axiomatic method requires that they be justified nontheless. For more details see Morris Kline's *Mathematics: The Loss of Certainty* (Kline, 1982).

Begriffsschrift on formal logic (Frege, 1879) marked a cornerstone in the development of mathematical reasoning. Frege argued that a logical derivation of the mathematical axioms was possible and that in order to do so, one needs to work exhaustively within a formal system. A natural consequence of this claim was the requirement for a *formal language* on which to build-up the formal system. Frege insisted that ordinary every day language is not suited to this purpose:

> *. . . [es kommt] aber nicht darauf an, daß man sich von der Wahrheit des Schlußsatzes überzeuge, womit man sich in der Mathematik meistens begnügt; sondern man muß sich auch zum Bewußtsein bringen, wodurch diese Überzeugung gerechtfertigt ist, auf welchen Urgesetzen sie beruht. Dazu sind feste Geleise erforderlich, in denen sich das Schließen bewegen muß, und solche sind in den Wortsprachen nicht ausgebildet.*[2]

The strive for reduction of mathematical foundations to pure logic went to extremes at the beginning of the 20^{th} century. Mathematics was approached on a completely abstract level, devoid from any recourse to meaning. To quote A. N. Whitehead: "Mathematics is thought moving in the sphere of complete abstraction from any particular instance of what it is talking about" (Whitehead, 1925). Bertrand Russel introduced the Theory of Types in 1903 and pursued his own type-based logic. Together with Whitehead, Russel worked on the establishment of a formal foundation of mathematics, which culminated in the monumental *Principia Mathematica* (Whitehead & Russell, 1910, 1912, 1913). In *Principia*, Russel and Whitehead provided many formal derivations of important theorems in set theory and arithmetic and so demonstrated that vital parts of working mathematics can be specified completely in a formal logic. Moreover, by employing a much clearer notation than the two-dimensional system of lines of Frege's *Begriffsschrift*, *Principia* made evident the expressive power of modern predicate logic. Unfortunately, many mathematicians found Russel and Whitehead's treatment of mathematical logic left a lot to be desired in a number of crucial aspects. E.g., Gödel pointed out that:

> *[. . .] this first comprehensive and thorough going presentation of a mathematical logic and the derivation of mathematics from it is so greatly lacking in formal precision in the foundations (contained in *1-*21 of* Principia*), that it presents in this respect a considerable step backwards as compared with Frege. What is missing, above all, is a precise statement of the syntax of the formalism.*[3]

[2] Gottlob Frege, *Kleine Schriften* (Frege, 2006), cited after (Zinn, 2004b).

[3] Kurt Gödel, *Russel's mathematical logic* (Gödel, 1944), cited after (Harrison, 1996).

The *Principia* also a demonstration that, while logical formalisation was possible, it was certainly impractical:

> *Within the mathematical community, the view of mathematics as logical and formal was elaborated by Bertrand Russell [...] in the first years of this century. [He] saw mathematics as proceeding in principle from axioms or hypotheses to theorems by steps, each step easily justifiable from its predecessors by a strict rule of transformation, the rules of transformation being few and fixed. The* Principia Mathematica *was the crowning achievement of the formalists. It was also the deathblow for the formalist view. There is no contradiction here: Russell did succeed in showing that ordinary working proofs can be reduced to formal, symbolic deductions. But he failed, in three enormous, taxing volumes, to get beyond the elementary facts of arithmetic.*[4]

Both Frege and Russel devised their formal systems in a bottom-up manner—starting with axioms and building new statements only according to the rules of logic, thus seemingly guaranteeing that the systems were consistent. In the 1920s David Hilbert rejected this logicist approach to general axiomatisation and worked on what was later to become known as the Hilbert program—a strive to formalise existing mathematical theories to a finite set of axioms and then prove that the resulting system is consistent using only finitary methods. Finitary methods are intuitive operations on finitary objects—"extralogical concrete objects that are intuitively present as immediate experience prior to all thought". By recursively finding proofs for simpler and simpler systems, all mathematics could be reduced to a non-primitive number theory, such as Peano Arithmetic.

> *We have thus reduced Peano's three primitive ideas to ideas of logic: we have given definitions of them which make them definite, no longer capable of an infinity of different meanings, as they were when they were only determinate to the extent of obeying Peano's five axioms.*
>
> *[...]*
>
> *Assuming the number of individuals in the universe is not finite, we have now succeded not only in defining Peano's three primitive ideas, but in seeing how to proce his five primitive propositions, by means of primitive ideas and propositions belonging to logic. It follows that all pure mathematics, in so far as it is deducible from the theory of natural numbers, is only a prolongation of logic. The extension of this result to those modern branches of mathematics which are not deducible from the theory of natural numbers offers no difficulty of principle...*[5]

[4] (Millo et al., 1979)

[5] Bertrand Russel, *Introduction to Mathematical Philosophy* (Russell, 1919)

The central question was, therefore, whether the system one uses in each recursion step is itself consistent.

Kurt Gödel's work on Incompleteness seemed like a lethal blow to the formalist doctrine. Gödel's First and Second Incompleteness Theorems proved that the effort to formalise *all* of mathematics is inevitably bound to fail. Gödel showed that every formal system which is complex enough to include arithmetic, finitely specified and consistent is also incomplete, i.e. there will always exist true statements which can be expressed in the language of the formal system, but cannot be proven to be true by using the reasoning rules of the system. It has, however, been established that it is possible to express all *working* mathematics in a formal system, e.g. in Zermelo-Fraenkel set theory plus the Axiom of Choice (ZFC) combined with first-order logic (van Heijenoort, 1967).

This is undoubtedly part of the reason why Hilbert's program and the standard it set for mathematical rigour influenced many 20^{th} century mathematicians. It certainly inspired Nicolas Bourbaki[6] to also pursue mathematics "from the beginning, [by giving] complete proofs". Bourbaki set out to write books on the foundations of mathematics, covering only material that "any mathematics graduate student should know" and which was not "active area of current research in mathematics" (Krantz, 2007). Bourbaki worked in a fixed formal system of axiomatic set theory of which he stated "[on] these foundations, [...] I can build up the whole of mathematics"[7] seemingly ignoring Gödel's findings:

> *[...] Bourbaki had grasped the positive worth of the work of Hilbert and his school, and welcomed the idea of the reduction of the question of correctness of mathematics to a set of rules, but nevertheless persisted, even after Gödel's work showed that Hilbert's program could never be completed, in thinking of logic and set theory as stu one settled in Volume One and then forgot about.*
>
> *[...]*
>
> *He regards the foundational crisis of the beginning of the century as having been resolved by Hilbert's formalist doctrine that the correctness of a piece of mathematics is a question of its following certain rules, and not a question of its interpretation.*[8]

Despite the fact the he provided a description of the formal the system he was working in, Bourbaki did not limit himself to the language of that system for the recording of presented proofs for "reasons of readability and practicality":

[6] Nicolas Bourbaki was the *nom de plume* of a group of French mathematicians which formed in the 1930s at the École Normale Supérieure.

[7] Nicolas Bourbaki, *The Theory of Sets* (Bourbaki, 1968), cited after (Harrison, 1996).

[8] A.R.D. Mathias, *The Ignorance of Bourbaki* (Mathias, 1992)

If formalized mathematics were as simple as the game of chess, then once our chosen formalized language had been described there would remain only the task of writing out our proofs in this language, [...] But the matter is far from being as simple as that, and no great experience is necessary to perceive that such a project is absolutely unrealizable: the tiniest proof at the beginning of the Theory of Sets would already require several hundreds of signs for its complete formalization. [...] formalized mathematics cannot in practice be written down in full, [...] We shall therefore very quickly abandon formalized mathematics.[9]

Bourbaki proofs cannot be checked on purely syntactic grounds (which would have been the case had he adhered to the formal language he described in such copious detail). The final arbiter of proof validity is, therefore, not mechanical verification, but rather the fact that each proof could *in theory* be expressed in this formal language.

2.3 Formal and informal proof

We chose to end the brief overview of the history of mathematical reasoning with Nicolas Bourbaki's views as they are rather symptomatic of the general attitude towards mathematical work within a formal system. Indeed, the fact that proofs *can* be reduced to formal symbolic deduction seems to have had little influence on the way "ordinary mathematics" is practised today.

At the beginning of this chapter we hinted at the two different notions of proof: proof as a chain of logical inferences and proof as a sufficiently convincing argument. It is now time to give those two notions names—*formal* and *informal* proof, offer precise definitions and focus on the implications each of the two approaches have on the methods of proof verification.

2.3.1 Formal proof

Consider the following definition of proof in (Mendelson, 1997):

A formal theory $\boldsymbol{L}$ is defined when the following conditions are satisfied:

1. A countable set of symbols is given as the symbols of $\boldsymbol{L}$. A finite sequence of symbols of $\boldsymbol{L}$ is called an *expression* of $\boldsymbol{L}$.

[9] Nicolas Bourbaki, *The Theory of Sets* (Bourbaki, 1968), cited after (Harrison, 1996).

2. There is a subset of expressions of $\boldsymbol{L}$ called the set of *well-formed formulas (wfs)* of $\boldsymbol{L}$. There is usually an effective procedure to determine whether a given expression is a wf.

3. There is a set of wfs called the set of *axioms* of $\boldsymbol{L}$. Most often one can effectively decide whether a given wf is an axiom; in such as case $\boldsymbol{L}$ is called an *axiomatic* theory.

4. There is a finite set $\boldsymbol{R_1, \ldots, R_n}$ of relations among wfs, called *rules of inference*. For each $\boldsymbol{R_i}$ there is a unique positive integer $\boldsymbol{J}$ such that, for every set of $\boldsymbol{j}$ wfs and each wf $\boldsymbol{B}$, one can effectively decide whether the given $\boldsymbol{j}$ wfs are in the relation $\boldsymbol{R_i}$ to $\boldsymbol{B}$, and, if so, $\boldsymbol{B}$ is said to *follow from* or to be a *direct consequence of* the given wfs by virtue of $\boldsymbol{R_i}$.

A *proof* in $\boldsymbol{L}$ is a sequence $\boldsymbol{B_1, \ldots, B_k}$ of wfs such that, for each $\boldsymbol{i}$, either $\boldsymbol{B_i}$ is an axiom of $\boldsymbol{L}$ or $\boldsymbol{B_i}$ is a direct consequence of some preceding wfs in the sequence by virtue of one of the rules of inference of $\boldsymbol{L}$.

A *theorem* of $\boldsymbol{L}$ is a wf $\boldsymbol{B}$ of $\boldsymbol{L}$ such that $\boldsymbol{B}$ is the last wf of some proof in $\boldsymbol{L}$. Such a proof is called a *proof of* $\boldsymbol{B}$ in $\boldsymbol{L}$. A wf $\boldsymbol{A}$ is said to be the consequence in $\boldsymbol{L}$ of a set of $\boldsymbol{\Gamma}$ of wfs if and only if there is a sequence $\boldsymbol{B_i, \ldots, B_k}$ of wfs such that $\boldsymbol{\Phi}$ is $\boldsymbol{B_k}$ and, for each $\boldsymbol{i}$, either $\boldsymbol{B_i}$ is an axiom or $\boldsymbol{B_i}$ is in $\boldsymbol{\Gamma}$, or $\boldsymbol{B_i}$ is a direct consequence by some rule of inference of some of the preceding wfs in the sequence. Such a sequence is called a *proof* (or *deduction*) of $\boldsymbol{\Phi}$ *form* $\boldsymbol{\Gamma}$. The members of $\boldsymbol{\Gamma}$ are called *hypotheses* or *premises* of the proof.[10]

Across Mendelson's definition we find the properties of proofs of we already encountered in the brief historical digression. To summarise: a proof is formal if is carried out exclusively within a formal system (called formal theory in (Mendelson, 1997)). This constitutes two properties of proofs vital to our discussion:

— Proofs are expressed in a formal language. The use of the term *formal language* is perhaps misleading considering the traditional dichotomy of "formal" and "natural" languages. The language of a formal system may well include words and syntactic constructions from natural languages, as long as the following conditions are met: a) the natural language words are included in the alphabet/lexicon of the formal language; b) the algorithms for recognition of well-formed formulas in the formal language are able to recognise the syntactic rules "borrowed" from the natural language; and c) the words and syntactic constructions coming from the natural language are unambiguous. There is nothing special

[10] Elliott Mendelson, *Introduction to Mathematical Logic* (Mendelson, 1997), cited after (Zinn, 2004b).

about the "symbols" and syntactic rules of a formal language that would prevent natural language words and phrases from becoming part of the formal language, as long as the semantics of the formal language remains well-defined. Indeed, this is just the approach we take with the Naproche language—it includes a mixture of symbolic and natural language, yet the lexicon and grammar of the natural language are controlled and unambiguous. More details will be presented in the dedicated chapter 4.

— Proofs are expressed exhaustively. This means that every single expression of the proof sequence is a well-formed formula and is there as a consequence of the application of one of the system's inference rules, i.e. no reasoning steps have been left and no additional "noise" is present. By noise we mean expressions which are not part of the reasoning process (such as hints and emphasis common with informal proofs). This means that one can manipulate formal proofs on purely syntactic grounds, without recourse to meaning or the need for interpretation. For the implications of this fact on the mechanism of proof verification, consider the following treatment by (Jones, 2007):

> *The purpose of proof is to provide a standard of demonstration which is independent of intuition, at least insofar as the validation of proofs is concerned. Though intuition may be essential in the discovery of a proof, once discovered, it should be possible to verify the correctness of a proof in a completely mechanical way.*
>
> *What is meant here by a mechanical verification is that given the rules and axioms of the logical system in which the proof is conducted, and the definitions upon which the proof depends, the checking of the correctness of the proof is simply a matter of syntactic manipulations which can be checked in principle without any understanding of the subject matter of the proof.*

It is the utter explicitness of formal proofs which enables their verification on purely syntactic grounds. Unfortunately the very same explicitness makes formal proofs extremely taxing for human readers. This was equally true at the beginning of the 20^{th} century for Russel and his own work[11] and for the formal proofs created for automated proofs checkers and those produced by automated theorem provers.

The features we highlighted above—the language, the factor of readability, and the verification mechanism—are of special interest to us, considering the goal of the Naproche system to allow authors to write human readable proofs and verify them. We shall summarise them in a table (2.1), which will be updated with the information on informal and Naproche proofs in the next sections.

[11] "*[Principia Mathematica]* demands extremely close attention from the reader, and there can be few who made the effort. Indeed, (Russell, 1968) remarks that his own intellect 'never quite recovered from the strain of writing it.'" (Harrison, 1996)

	Formal
language	symbolic, formally defined
readability	hardly/not readable
verification	mechanical, on syntactic grounds, interpretation not required

Table 2.1: Language, readability and verification mechanism for formal proofs.

2.3.2 Informal proof

Despite the existence of numerous guides on writing mathematics, e.g. (Steenrod et al., 1973) and (Krantz, 1997), there is no definitive processing model for informal proof texts. This lies in part in the lack of definition of the language of informal proofs (see (Trybulec & Święczkowska, 1991), (Wolska & Kruijff-Korbayová, 2004a) and (Wolska & Kruijff-Korbayová, 2004b)). This necessitates the existence of a social mechanism for verification. Consider (Millo et al., 1979):

> *In mathematics, the aim is to increase one's confidence in the correctness of a theorem, and it's true that one of the devices mathematicians could in theory use to achieve this goal is a long chain of formal logic. But in fact they don't. What they use is a proof, a very different animal. [...] First of all, the proof of a theorem is a message. [...] We believe that, in the end, it is a social process that determines whether mathematicians feel confident about a theorem [...] .*

Contrast the definition of proof by Mendelson we presented in the previous section with that of Krantz (Krantz, 2007): "Heuristically, a proof is a rhetorical device for convincing someone else that a mathematical statement is true or valid". "Convincing" implies that proofs have an intended audience which is not only a recipient, but also, by some process of critical thinking, responsible for the "social" verification of the proof.
William Thurston elaborates on this "social process" in (Thurston, 1994):

> *However, we should recognize that the humanly understandable and humanly checkable proofs that we actually do are what is most important to us, and that they are quite different from formal proofs. For the present, formal proofs are out of reach and mostly irrelevant: we have good human processes for checking mathematical validity.*
>
> *[...]*

Mathematicians can and do fill in gaps, correct errors, and supply more detail and more careful scholarship when they are called on or motivated to do so. Our system is quite good at producing reliable theorems that can be solidly backed up. It's just that the reliability does not primarily come from mathematicians formally checking formal arguments; it comes from mathematicians thinking carefully and critically about mathematical ideas.

The main criterion in proof verification becomes the "social acceptability" of the proof rather that the adherence to a formal system and accuracy with respect to that system's language and rules. This means that mathematicians can rely on the fact that their peers posses certain background knowledge and familiarity with the operational methods of the particular field of mathematics. They can forego explicit logical reasoning in parts of the proof, omitting certain steps, and rely on the fact that the audience will be able to, by intuition or experience, "fill in the gaps".

The rather vague notion of social acceptability is described in (Hanna, 1983):

1. *They understand the theorem, the concepts embodied in it, its logical antecedents, and its implications. There is nothing to suggest it is not true;*
2. *The theorem is significant enough to have implications in one or more significant branches of mathematics (and is thus important and useful enough to warrant detailed study and analysis);*
3. *The theorem is consistent with the body of accepted mathematical results;*
4. *The author has an unimpeachable reputation as an expert in the subject matter of the theorem.*
5. *There is a convincing mathematical argument for it (rigorous or otherwise), of a type they have encountered before.*

If there is a rank order of criteria for admissibility, then these five criteria all rank higher than rigorous proof.

[…]

[The] mathematician is much more interested in the message embodied in the proof than its formal codification and syntax. The mechanics of proof are seen as a necessary but ultimately less significant aspect of mathematics. Certainly being able to follow the steps of a proof is not the same as understanding it.[12]

Adding informal proofs to the matrix of proof properties yields table 2.2:

[12] G.Hanna, *Rigorous proof in mathematics education* (Hanna, 1983), cited after (Zinn, 2004b)

	Formal	*Informal*
language	symbolic, formally defined	mixture of natural and symbolic, not formally defined
readability	hardly/not readable	readable
verification	mechanical, on syntactic grounds, interpretation not required	social, interpretation required; mechanical verification not possible

Table 2.2: Language, readability and verification mechanism for formal and informal proofs.

2.4 Requirements for Naproche proofs

Having seen how formal and informal proofs compare with regard to their language, readability and verification, we shall now place Naproche proofs in the comparison matrix.

The mathematical proofs which the Naproche system accepts display a fusion of both formal and informal characteristics. The language they are written in is, just like that of informal proofs, a mix of natural and symbolic language which makes them readable by humans and thus suitable for a "social" process of verification. Yet the language is controlled, having a fixed lexicon and grammar, and unambiguous, which means that the proofs are also machine verifiable.

The Naproche system also provides a mechanism which allows authors to write hints and explanations (say, to provide a proof outline) such as are typical of informal proofs. These "comments" of a sort are only helpful to human readers, yet they are not part of the proof proper and are silently discarded when the proof is submitted for automated verification.

Table 2.3 shows the updated comparison matrix of proofs types.

	Formal	*Informal*	*Naproche*
language	symbolic, formally defined	mixture of natural and symbolic, not formally defined	mixture of natural and symbolic, formally defined
readability	hardly/not readable	readable	readable
verification	mechanical, on syntactic grounds, interpretation not required	social, interpretation required; mechanical verification not possible	mechanical and social possible

Table 2.3: Language, readability and verification mechanism for formal, informal, and Naproche proofs.

Chapter 3

Discourse Representation Theory

Discourse understanding has been one of the main directions of research in computational linguistics and numerous theories have sprouted as a result of this research—File Card Semantics (Heim, 1982), Dynamic Predicate Logic (Groenendijk & Stokhof, 1991), Update Semantics (Veltman, 1996) and (Groenendijk et al., 1996) and Discourse Representation Theory (Kamp, 1981) among others. As we are concerned with mathematical discourse, it is only natural to look at those theories as a useful starting point.

We chose Discourse Representation Theory (DRT) as it is one of the oldest and certainly the most prominently established theories of discourse understanding. It has seen numerous implementations and has, since its inception, been adapted to explain a number of different natural language phenomena such as anaphora and quantification (Black, 1993).

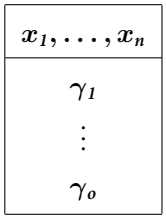
DRT provides a basic data structure—the Discourse Representation Structure (DRS)—which servers a two-fold purpose: it is used to represent content and to provide context.

3.1 Discourse Representation Structures

3.1.1 Formal definition of DRSs

For the following definition, let $x_1, \ldots, x_n$ be discourse referents, let $\gamma_1, \ldots, \gamma_m$ be conditions, let R be a relation symbol of arity n, and let B and B_1 be DRSs.

Definition 1: Discourse Representation Structures.

1. 

is a DRS.

2. $R(x_1, \ldots, x_n)$ is a condition.

3. $x_1 = x_2$ is a condition.

4. $\neg B$ is a condition.

5. $B \vee B_1$ is a condition.

6. $B \rightarrow B_1$ is a condition.

7. Nothing else is a DRS or a condition.

3.2 Accessibility

The notion of accessibility states which discourse referents are available for resolution. The accessibility of discourse referents is defined in terms ot the accessible relation between DRSs. A discourse referent x is accessible from a sub-DRS B if and only if x is introduced in some DRS B' and B is accessible from B'. The accessibility between DRSs is defined as follows:

Definition 2: A DRS B_i is accessible from a DRS B_j if and only if

1. B_i equals B_j; or
2. B_i *subordinates* B_j.

Definition 3: A DRS B_i subordinates a DRS B_j if and only if

1. B_i immediately subordinates B_j; or
2. there is some PRS B such that B_i subordinates B and B subordinates B_j.

Definition 4: For the DRSs B_i and B_j, B_i immediately subordinates B_j if and only if

1. B_i contains a condition of the form $\neg B_j$; or
2. B_i contains a condition of the form $B_j \vee B$ or $B \vee B_j$ for some DRS B; or
3. B_i contains a condition of the form $B_j \rightarrow B$ for some PRS B; or
4. B contains a condition $B_i \rightarrow B_j$ for some DRS B.

3.3 DRS construction

DRS construction is an iterative, incremental process. In the original version of DRT processing the discourse $s_1, s_2, \ldots, s_n$ where $s_1, s_2, \ldots, s_n$ is the list of sentences in the discourse proceeds as follows: the parse tree of s_1 is transformed according to DRS construction rules into a DRS K_1 which serves as the context of s_2. The algorithm halts when the end on the list of sentences is reached. This basic principle has remained, yet there have been numerous reformulations to address compositionality (Bos et al., 1994), the notion of sentences as transformation of information states (Johnson & Klein, 1986), etc. For a thorough overview of the implications of the different approaches see (Black, 1993).

Chapter 4

The Naproche language

The Naproche language is the language in which Naproche proofs are written in. It is a controlled language with well-defined semantics and represents a mixture of selected English natural language expressions extracted from common mathematical practise and a symbolic language which allows the use of simple mathematical expressions. We shall confine the discussion in this chapter to the syntax of the Naproche language and define its semantics in terms of the structures it is used to represent (chapter 5).

The Naproche language is currently a template language, i.e. it is defined as a set of templates in which we use so called trigger phrases to infer the type of sentence. The basic building block is the sub-statement. What this means will become clear in the listing of the templates below. For them we shall use a pseudo-DCG notation. The expressions in brackets are fixed parts of the templates.

The Naproche language supports five types of sentences—structure markers, statements, definitions, assumptions and assumption closing sentences.

4.1 Structure markers

Structure markers are not part of the proof proper, i.e. they play no role in the reasoning recorded within the proof. Rather, they are used to denote the structure of the proof. The structure markers currently supported by the Naproche system are *"Theorem."*, *"Lemma."*, *"Proof."* and *"Qed."*

4.2 Statements

Statements have the general form of

```
statement --> (statement_trigger), sub-statement.
```

Currently supported statement triggers are: *then, hence, recall that, but, in particular, observe that, together we have* and *so*.

4.3 Definitions

Definitions have the general form of

```
define --> (define), sub-statement, if and only if, sub-statement.
```

or

```
define --> (define), sub-statement, iff, sub-statement.
```

4.4 Assumptions

Assumptions have the general form of

```
assumption --> (assumption_trigger), sub-statement.
```

Currently supported statement triggers are: *let, consider, assume that* and *assume for a contradiction that*.

4.5 Assumption closing sentences

Assumption closing statements have the general form of

```
closing -> (closing_trigger), sub-statement.
```

Currently, the only supported closing trigger is *thus*.

4.6 Other expression types

The NAPROCHE language also supports the expression of natural language quantification and implication. Those are defined as follows.

Natural language quantification:

```
(for all), sub-statement, ',', sub-statement.
```

Implication:

```
sub-statement, (implies), sub-statement.
```

Chapter 5

Proof Representation Structures

Having provided the linguistic description of the Naproche system, it is time to focus on its processing model. In this chapter we shall describe the structure we devised to support the formalisation of Naproche texts—the Proof Representation Structure (prs)[1]. Following a brief summary of the requirements we set out to fulfil by developing the prs specification (section 5.1), we shall give a bird's eye overview of the role prss play within the Naproche project (section 5.2) and give their formal definition (section 5.4). We shall then focus on the all-important accessibility relation (section 5.6) and describe how specific expressions in the Naproche language map to different prs instances (chapter 6).

5.1 Limitations of drt with regard to proof texts

So far we have established that in order to pass an informal mathematical text to a proof verification system, we need to convert it to some kind of formal structure first. We claim that semantic theories designed for accomplishing this task for natural language narrative text provide a good basis, yet fail to capture crucial properties specific to mathematical texts. So the question becomes "can we extend natural language semantic theories to support the analysis of said properties?". We claim that the answer to this question is *yes*. And since we deemed drt particularly fit for our purposes (chapter 3),

[1] The name Proof Representation Structure coincides with the name of a similar structure described by Claus Zinn in his doctoral thesis (Zinn, 2004b). In addition to their names, both structures share the same motivation ("How do we represent informal mathematical discourse in a formal manner?") and the same background (both are extensions of Discourse Representation Structures). Despite this similarities the Naproche prss and Zinn's prss are different in both their composition and their construction mechanism. Zinn's work will be further described in chapter **??**.

we will look at extending its underlying formal structure—the DRS—in both composition, semantics and methods of construction.

Before we describe the extensions we propose, let's recap the requirements for the NAPROCHE formal structure. The following list of requirements is both general, in the sense that any NLP system for mathematical text should fulfil them, as well as NAPROCHE-specific, driven by the motivation behind the project itself—to provide a useful teaching aid for students of mathematics and related disciplines.

Text and proof structure Mathematical proofs are highly structured, unlike narrative text. Proofs are commonly divided into discrete theorems, which are themselves composed of a goal (*the proposition we want to prove*) and a body (*the steps of the proof itself*). Proof building blocks may also be nested, usually by including one or more auxiliary proofs (lemmas) inside a theorem. On an even lower level, specific linguistic expressions—such as assumption-conclusion pairs and case differentiations—lead to similar natural bracketing.

Nesting and bracketing in this manner are crucial for the comprehension of proofs by readers, as well as their verification by automated proof checkers. It is therefore necessary to preserve such structures when formalising a proof.

Sequentiality The order in which the sentences of which a proof consists of are interpreted should be mirrored in the final formal representation. Proof verification systems rely on this order to check whether each consecutive proposition follows logically from the preceding ones. DRT's goal of processing the whole discourse to a sort of a "net" meaning representation by merging sub-structures whenever possible results in a loss of individual sentences, and thus of their order, and must therefore be abandoned.

Reuse of variable names The natural nesting and bracketing outlined at the beginning of this list, and the strive to minimise the number of variable names used within a proof (in order to increase readability and clarity), means that variables are possibly bound by different existential conditions at any point of the proof. Basic examples of this are variable definitions, which are usually not shared between the different theorems in a proof, and variables for which a case differentiation is performed (say, considering three possibilities for a number—it can be positive, negative or equal to 0).

This calls for a highly dynamic reading of variable conditions which allows us to add binding conditions when assumptions about a variable are made, as well as to retract those conditions when the assumption loses scope.

Definitions* Definitions provide a mechanism for proofs to dynamically modify the very language they are written in. They introduce new language constructs which must be added to the language processing modules on-the-fly.

Intertextual and intratextual references* Proofs commonly make use of other, already proven, propositions. A reference by number or name (if one is available and commonly agreed upon) is usually inserted to show this relationship. A formal structure suitable for representing proofs should, therefore, account for a way of uniquely identifying proofs, as well as a way for resolving references made to them.

Localisation We should be able to easily identify the exact location of the original proof text which is responsible for the creation of each individual sub-structure in our formal representation. This is mainly necessary for the generation of meaningful error messages, no matter whether they are visual (e.g. the highlighting of an erroneous sentence) or textual. Error reporting is desirable for both linguistic errors, which are usually caught during the initial parsing, and logical errors which are identified by the proof checking system.

5.2 Overview

To address the requirements outlined in the previous section we propose a new formal structure called *Proof Representation Structure*. It inherits its basic form from the Discourse Representation Structure and extends it with new slots and a novel construction mechanism.

A PRS is a heterogeneous quintuple $(\boldsymbol{i}, \boldsymbol{D}, \boldsymbol{M}, \boldsymbol{C}, \boldsymbol{R})$ where

— $\boldsymbol{i}$ is a unique id;

— $\boldsymbol{D}$ is a set of discourse referents;

— $\boldsymbol{M}$ is a set of mathematical referents;

— $\boldsymbol{C}$ is a set of conditions;

— and $\boldsymbol{R}$ is a set of references to parts of the same text or other texts.

* Provisions for this feature have been made in the definition of the formal structure and its associated construction mechanisms. However, it is not fully implemented in current the codebase.

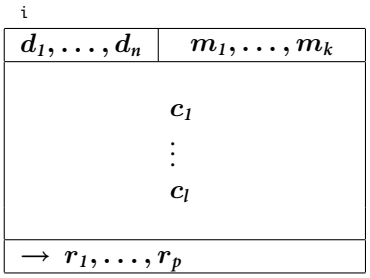


Figure 5.1: The basic form of a PRS. i is the id, $d_1, \ldots, d_n$ are discourse referents, $m_1, \ldots, m_k$ are mathematical referents, $c_1, \ldots, c_l$ are conditions and $r_1, \ldots, r_p$ are textual references.

We will often use a graphical representation of such tuples as a box-like structure, like the one in fig. 5.1. When the set of references is empty, which is the case for most formalised sentences, we will leave out the bottom-most field in order to save space (fig 5.2). This graphical notation is particularly useful when displaying nested structures* .

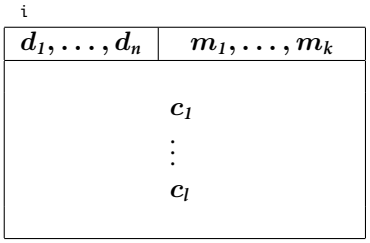


Figure 5.2: A PRS with no refereces. Note that the references field is missing at the bottom. For the remaining elements, the conditions of fig. 5.1 apply.

* If one imagines the PRS tuple as a physical structure, the analogy of a labelled chest of drawers comes to mind; the id being the label, the four remaining elements being drawers or slots one has to fill. This terminology is common within the NAPROCHE working group and we shall adopt it here, using *label* interchangeably with *id* and speaking of the references drawer/slot, the conditions drawer/slot, etc.

5.3 Two roles for PRSs

The PRS was devised, on a lower lever, with the representation of sentence meaning in mind. The formalisation of each sentence is stored a single PRS or an operator-bound pair of PRSs, and an updated accessibility relation guarantees that discourse referent resolution works correctly across PRS boundaries (see section 5.6). In order to accommodate the proof structure and semantic bracketing we spoke of in the requirements list at the beginning of this chapter as well, the early drafts of the NAPROCHE codebase used an additional structure to represent the nested composition of proofs (listing 5.1).

```
1 proof(
2   theorem(goal(...)
3           body(
4             ...,
5             lemma(goal(...)
6                   body(...)),
7             ...)))
```

Listing 5.1: Initial draft of a proof representation in pseudo-Prolog notation. This one shows a proof consisting of a single theorem with a nested lemma. Ellipses represent sequences of discrete sentence PRSs.

To avoid the overhead caused by dealing with two different structures, and thus having to write two different parsers—one for the proof structure and one for the PRSs representing sentences, we later decided to abandon this approach in favour of re-using a special case of PRSs with pre-defined ids, empty discourse referents and mathematical referents slots. The "contents" of a logical proof section could be represented in a simple manner by including them in the conditions slot of the special-case PRS representing that section. Due to the box-like nature of PRSs, this approach is feasible and has so far proved to be sufficient. We shall refer to these special-case PRSs as *composition* PRSs to be able to differentiate them from *sentence* PRSs.

The predefined ids for composition PRSs begin with the type of the sub-structure in question, followed by an underscore and a running integer for unique identfication. Thus, a PRS which represents a whole proof has an id of the form `proof_n`, a PRS containing a theorem—an id like `theorem_n`, etc., where `n` is an unique integer. More details on ids will be provided in section 5.5.

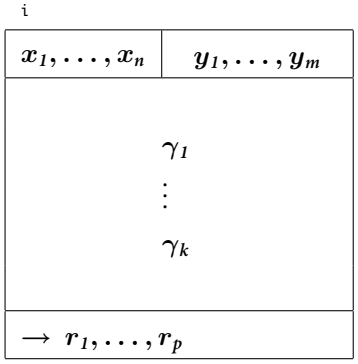
PRS and sub-PRS

So far we used the name PRS when referring to the type of structure we are using. The convention agreed to by the NAPROCHE working group states that when talking about instances of this structure, we use the name PRS only when referring to a complete proof representation, and the name sub-PRS when referring to any structure within that top-level PRS. We shall, however, retain the name PRS when describing the formal properties of the structure and come back to this distinction when discussing the construction mechanisms in chapter 6.

5.4 Formal Definition

For the following definition, let $x_1, \ldots, x_n$ be discourse referents, let $y_1, \ldots, y_m$ be mathematical referents, let $\gamma_1, \ldots, \gamma_k$ be conditions, let R be a relation symbol of arity n, let $r_1, \ldots, r_p$ be references to other parts of the same text or to other texts, and let B and B_1 be PRSs.

Definition 5: Proof Representation Structures.

1. 

is a PRS.
2. $R(x_1, \ldots, x_n)$ is a condition.
3. B is a condition.
4. $\neg B$ is a condition, representing a negated statement.
5. $B := B_1$ is a condition, representing a definition.
6. $B \rightarrow B_1$ is a condition, representing an implication or a universal quantification.
7. $B \Rightarrow B_1$ is a condition, representing an assumption.
8. Nothing else is a PRS or a condition.

5.5 The anatomy of a PRS

Let's look at the individual constituents of a PRS. In this section we will confine ourselves to describing *what* the building blocks of a PRS are and reserve a further section to the discussion of *how* they come into being.

5.5.1 Id

The id of a PRS is a token (represented in Prolog by an atom) which uniquely[2] identifies this PRS within the representation structure. The original motivation behind "labelling" PRSs in this manner is the ability to pinpoint the exact location in the original proof text which is responsible for the creation of the sub-PRS in question[3].

Each PRS which was created as a direct consequence of formalising a particular sentence, inherits the id of that sentence. The sentence id in turn is composed of the line it is to be found on in the original text (this is provided by the $\TeX_{\text{MACS}}$ export plug-in), a dash and a running number representing the position of the sentence in that line. So the third sentence in the line with id `1.2.3` would be `1.2.3-3`, that of the fourth one would be `1.2.3-4`, etc. (fig. 5.3). This covers the basic case, when a sentence is represented by a single PRS.

For sentences which result in a complex construction of two PRSs connected by an operator, such as assumptions, definitions and implications, the right hand PRS inherits the sentence id, and the left hand PRS is assigned an artificial id consisting of a type name, an underscore and a running integer, which is guaranteed to be unique within the entire proof representation. E.g., an assumption in the third sentence of line `1.2.3` would result in `1.2.3-3` $\Rightarrow$ `consec_1` (for the sake of clarity, we shall use the framed strings to represent ids only, the rest of the PRSs is left out). In the left hand side PRS id *consec* stands for consequence. Similarly, a definition will generate `1.2.3-3` := `definiens_1`. Implications generate a higher-level PRS with two sub-PRSs inside. Those have ids which begin with `prefix_` and `matrix_` for the left hand-side and the right-hand side respectively. The higher-level PRS inherits the sentence id, the sub-PRSs get the above prefixes and a running index.

[2] The id is guaranteed to be unique within a single discourse representation even if this contains multiple proofs. The ids are not, however, guaranteed to be unique between different runs of the formalisation algorithms if the input text is the same. If, at some point, the NAPROCHE system is to become distributed and save formalisations for re-use and combination, the sensible thing to do would be to add augment ids with a generated Universally Unique Identifiers (UUIDs).

[3] Recall that the aim of the NAPROCHE system is to aid authors in writing correct proofs; a major part of this aid resides in the ability to highlight erroneous parts of the proof when something goes wrong. An error message which references the exact sentence which caused a linguistic (parsing) or a logical error is inherently more useful than a message which merely states that such an error occurred.

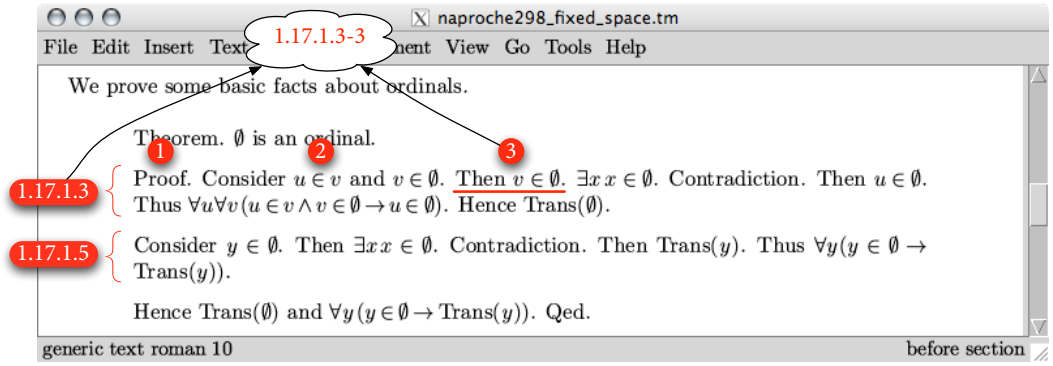


Figure 5.3: Id composition for statements. Line ids are in rounded triangles, the running number of sentences within a line in circles. The resulting id for the PRS representation of the underlined sentence is shown at the top.

The third type of id is reserved for composition PRSs which denote the higher-level proof structure. All consist of a prefix and a text-wide unique integer. We have five prefixes to represent the five types of composition PRSs, their names follow the names of the structures they represent—`proof_`, `theorem_`, `lemma_`, `goal_` and `body_`.

5.5.2 Discourse referents

The role of discourse referents in PRSs follows the one we described for DRSs in section 3.1.1.

Discourse referents in the current NAPROCHE implementation are represented by running integers.

5.5.3 Mathematical referents

Just as discourse referents, sets of mathematical referents are also generated from content sentences only, and thus only occur in sentence PRSs. They are a set of all sentence constituents which are marked up as "math" in the TEX$_{\text{MACS}}$ source and all free variables[4] contained within these constituents. Consider the sentences in (1):

(1) a. Hence $\boldsymbol{x}$ is an ordinal.

b. Let $\boldsymbol{u} \in \boldsymbol{v}$ and $\boldsymbol{v} \in \boldsymbol{x}$.

[4]Currently, when considering which variables are free and which are bound, only quantification which occurs within formulae is considered. Natural language quantification, such as *For all integers* $\boldsymbol{x}$*…* and *There is no* $\boldsymbol{x}$ *such that…*, is handled on a different level (see chapter 6).

c. Thus there is no x such that $\forall u(u \in x \leftrightarrow \mathbf{Ord}(u))$.

The sentence PRS for (1a) would contain x as the sole mathematical referent as it is the only part of the sentence marked-up as "math" in the source file and the variable x is not being quantified over within that sentence. The mathematical referents for the sentence in (1b) will be $u \in v$, $u \in x$, u, v and x. First, the formulae $u \in v$ and $v \in x$ are accepted into the slot, the sets of their free variables (u and v, and v and x respectively) are merged and the result is also added to the slot. For sentence (1c) the mathematical referents would be x and $\forall u(u \in x \leftrightarrow \mathbf{Ord}(u))$. u is universally bound in the second formula and is thus omitted from the set of referents.

5.5.4 Conditions

The conditions acceptable in a PRS depend on the type of expression the PRS represents. Recall from the previous chapter that we divide sentences in the NAPROCHE language into distinct groups depending on their semantic contribution. We have structuring sentences, such as "Theorem.", "Lemma." and "Proof." and what we termed content sentences—simple statements, definitions, implications (which we re-use to represent natural language quantification) and assumptions. Furthermore, we described how content sentences are either statements or can be treated as multiple statements which are connected by operators to reflect the nature of the original sentence.

So we can reduce the problem of describing which conditions are allowed in which PRS type to two basic cases—PRSs which represent statements and PRSs which represent some sort of structure.

Conditions for PRSs representing statements

For statement PRSs the allowed conditions are of the type $R(x_1, \ldots, x_i)$ or $x_m = x_n$ where $x_i, \ldots, x_j$, x_m and x_n are discourse referents associated with the same PRS or accessible from it (accessibility will be discussed at length in section 5.6). Two predefined relations which are very common in NAPROCHE formalisations are the binary `math_id` and the unary `holds`.

Any PRS which introduces a new mathematical referent (because it is part of the sentence the PRS in question will eventually stand for), automatically associates it with a discourse referent. The relationship between the mathematical referent and the discourse referent is expressed by the `math_id` relation. Recall example (1a). The variable x would cause an x to be inserted into the drawer of mathematical referents. Provided there is no accessible discourse referent which can be resolved to x, a new discourse referent, say 1, will be added to the appropriate drawer. The relation which binds the two together, `mathid(1, x)`, is then inserted into the conditions drawer. Subsequent encounters

of x will only add a mathematical referent, as a discourse referent bound to x is already accessible, making a repetition of the statement for the relation between 1 and x is unnecessary. From this point on, this is the default behaviour of the construction mechanism until the variable x goes out of scope.

The unary `holds` is used to denote that a formula encountered within a certain sentence is believed to be true at this particular point of the proof. It takes a discourse referent as its single parameter. Consider sentence (2):

(2) Hence $\forall u(u \in x \leftrightarrow \mathbf{Ord}(u))$.

After putting a discourse referent and a mathematical referent into the respective slots, a further condition is added, stating that the formula is true at this point. The resulting PRS looks like the one in fig. 5.4.

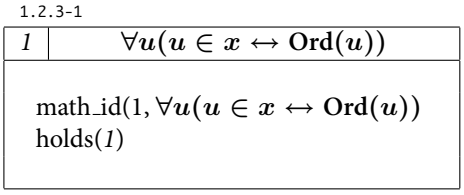


Figure 5.4: A partial PRS demonstrating the use of the `holds` condition.

Conditions for composition PRSs

For PRSs representing structure, the only allowed conditions are other PRSs and operator-connected PRS pairs with some fairly straightforward constraints imposed on the nesting. Those constraints are not of a technical nature; rather, they mirror the established conventions for writing mathematical proofs[5].

The PRS which effect high-level (proof, theorem, lemma) structuring are composition PRSs (see section 5.3) with the following constraints:

[5]It is common to prove a lemma in the course of a proving a theorem and thus nest the lemma inside the theorem body. Nesting a theorem inside another theorem, however, is not allowed. Also, when writing a proof, *what* is to be proven is stated before the steps showing *how* it can be proven. Such rules are describes in a number of manuals on mathematical style such as (Krantz, 1997) and (van Gasteren, 1990).

1. A proof PRS[6] must subordinate [7] at least one theorem PRS.
2. A proof PRS may not subordinate another proof PRS.
3. A theorem PRS contains exactly two conditions in the following order—a goal PRS and a body PRS.
4. A theorem PRS cannot subordinate another theorem PRS.
5. A lemma PRS may be included anywhere in the body PRS of a theorem PRS.
6. A lemma PRS contains exactly two conditions in the following order—a goal PRS and a body PRS.
7. A lemma PRS may not subordinate a theorem PRS or another lemma PRS.
8. No other constraints apply.

Pairing statement PRSs

We represent definitions, implications and assumptions by pairing lower-level statement PRSs using three pre-defined operators—the assignment operator ($:=$), the single arrow ($\rightarrow$) and the double arrow ($\Rightarrow$). To demonstrate what the pairs look like, consider the following examples:

(3) Define $\mathbf{Trans}(\boldsymbol{x})$ if and only if $\forall \boldsymbol{u} \forall \boldsymbol{v} (\boldsymbol{u} \in \boldsymbol{v} \wedge \boldsymbol{v} \in \boldsymbol{x} \rightarrow \boldsymbol{u} \in \boldsymbol{x})$.

(4) $\boldsymbol{x} \in \boldsymbol{y}$ and $\boldsymbol{y}$ is an ordinal implies $\boldsymbol{x}$ is an ordinal.

(5) For all $\boldsymbol{x}$, not $\boldsymbol{x} \in \boldsymbol{x}$.

(6) Assume that $\neg \exists \boldsymbol{x} \boldsymbol{x} \in \emptyset$.

(3) is a definition. The corresponding PRS is shown in fig. 5.5.
(4) is an implication. The corresponding PRS is shown in fig. 5.6.
We represent natural language quantification as an implication. Both interpretations are dynamically and statically equivalent, so the re-use of the representation structure is justified. The corresponding PRS for (5) is shown in fig. 5.7.

Assumptions such as (6) are represented in a form like the one in fig. 5.8.

Much more details on the form of conditions will be discussed in the next chapter, where we shall look at the methods of construction for PRSs.

[6] We shall use the "proof PRS" to mean a PRS representing a whole proof text, a "theorem PRS" to mean a PRS representing a theorem structure, etc.

[7] The definition of the *subordinate* relation with regard to PRSs will be discussed section 5.6.

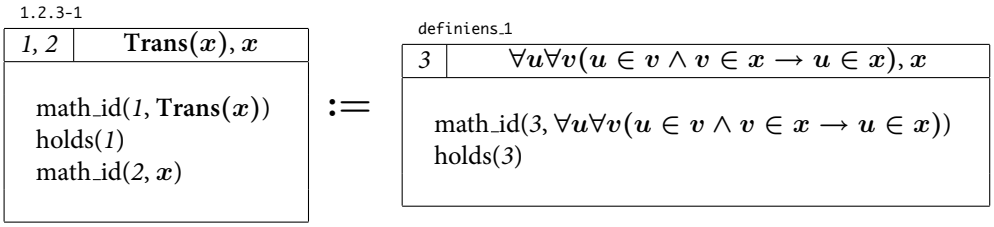


Figure 5.5: A PRS demonstrating a definition formalisation.

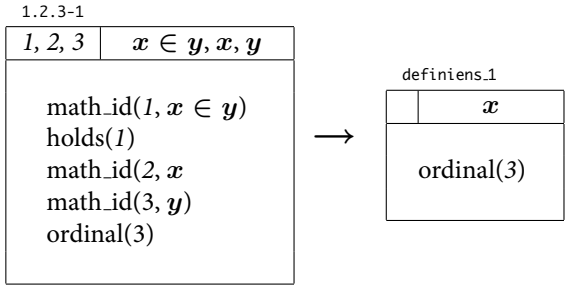


Figure 5.6: A PRS demonstrating an implication formalisation.

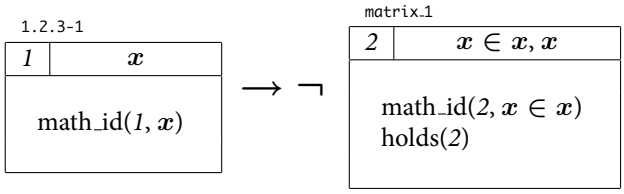


Figure 5.7: A PRS demonstrating natural language quantification.

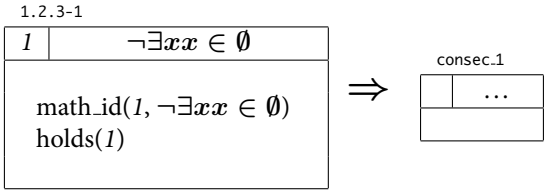


Figure 5.8: A PRS demonstrating an assumption representation.

5.5.5 Textual references

The textual references slot of with each PRS is a set of ids which refer to other parts of the same proof or, if the discourse we are formalising contains multiple proofs, to parts of other proofs. Those parts

are expected to be theorems and lemmas, in order to support the form of the most common references which occur in informal proofs—"By theorem N" and "By lemma N" where N is usually a running number of the theorems, resp. lemmas in a text.

It would not be hard to extend the definition of this slot to support the inclusion of references to named theorems and lemmas, providing those names (and the proofs associated with them) are available to the proof checking software we are interfacing with.

5.6 Accessibility in a PRS

The accessibility relation for DRSs we discussed in section 3.2 applies unaltered to PRSs as well:

Definition 6: A PRS $\boldsymbol{B}_i$ is accessible from a PRS $\boldsymbol{B}_j$ if and only if

1. $\boldsymbol{B}_i$ equals $\boldsymbol{B}_j$; or
2. $\boldsymbol{B}_i$ *subordinates* $\boldsymbol{B}_j$.

The same holds for the *subordinates* relation:

Definition 7: A PRS $\boldsymbol{B}_i$ subordinates a PRS $\boldsymbol{B}_j$ if and only if

1. $\boldsymbol{B}_i$ immediately subordinates $\boldsymbol{B}_j$; or
2. there is some PRS $\boldsymbol{B}$ such that $\boldsymbol{B}_i$ subordinates $\boldsymbol{B}$ and $\boldsymbol{B}$ subordinates $\boldsymbol{B}_j$.

To accommodate for the fact that we use a sub-PRS per sentence and to handle the richer language of PRSs, we need to alter the *immediately subordinates* relation. It is defined as follows:

Definition 8: For the PRSs $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$, $\boldsymbol{B}_i$ immediately subordinates $\boldsymbol{B}_j$ if and only if

1. $\boldsymbol{B}_i$ contains the conditions $\boldsymbol{B}_j$; or
2. $\boldsymbol{B}_i$ contains a condition of the form $\neg\boldsymbol{B}_j$; or
3. $\boldsymbol{B}$ contains the conditions $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$ such that $\boldsymbol{index}(\boldsymbol{B}_i) < \boldsymbol{index}(\boldsymbol{B}_j)$ for some $\boldsymbol{B}$, where $\boldsymbol{index}(\boldsymbol{B}_i)$ and $\boldsymbol{index}(\boldsymbol{B}_j)$ are the positions of $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$ in the list of conditions of $\boldsymbol{B}$; or
4. $\boldsymbol{B}_i$ contains a condition of the form $\boldsymbol{B}_j := \boldsymbol{B}$ or $\boldsymbol{B} := \boldsymbol{B}_j$ for some PRS $\boldsymbol{B}$; or

5. $\boldsymbol{B}_i$ contains a condition of the form $\boldsymbol{B}_j \rightarrow \boldsymbol{B}$ or $\boldsymbol{B} \rightarrow \boldsymbol{B}_j$ for some PRS $\boldsymbol{B}$; or
6. $\boldsymbol{B}_i$ contains a condition of the form $\boldsymbol{B}_j \Rightarrow \boldsymbol{B}$ or $\boldsymbol{B} \Rightarrow \boldsymbol{B}_j$ for some PRS $\boldsymbol{B}$; or
7. $\boldsymbol{B}$ contains a condition of the form $\boldsymbol{B}_i := \boldsymbol{B}_j$ or for some PRS $\boldsymbol{B}$; or
8. $\boldsymbol{B}$ contains a condition of the form $\boldsymbol{B}_i \rightarrow \boldsymbol{B}_j$ or for some PRS $\boldsymbol{B}$; or
9. $\boldsymbol{B}$ contains a condition of the form $\boldsymbol{B}_i \Rightarrow \boldsymbol{B}_j$ or for some PRS $\boldsymbol{B}$.

The accessibility relation is, just as in DRT, useful only insofar as it poses constraints on the list of available discourse referents available for resolution.

Chapter 6

Constructing PRSs

So far we have described the language and expression types of Naproche proofs and the formal structure we want to use to represent them. What remains to be defined are the rules of the Naproche processing model, i.e. the methods for constructing formal representations from sentences and entire discourses.

In the chapter on DRT we mentioned that there exist a number of strategies of building DRSs. Because the Proof Representation Structure is a rather straightforward extension of the DRS, a generation algorithm for PRSs can reuse these methods provided the necessary modifications are made to support the richer PRS language and the novel accessibility relation for discourse referents.

For the implementation of the Naproche system we choose a functional style of construction, similar to that of λ-DRT, as it remains close to the notion of sentences as functions from PRSs to PRSs. In order to ensure a more modular architecture, we define the task of constructing a complete PRS from a proof representing discourse as a *two-layer iterative* process. We shall call the two layers micro-construction and macro-construction.

Micro-construction ("micro" because it operates on the sentence level) is the process of generating an *update function* based on the parse tree of a single sentence, considering the previously processed discourse as context. The update function can perform a number of operations on PRSs, including shifting the pointer to the currently active sub-PRS and the insertion of new sub-PRSs and even whole sub-PRS constructions.

Macro-construction ("macro" because it operates on the discourse level) is the process of iteratively applying the update functions to a context PRS, yielding a new PRS which in turn serves as a context for the generation of a new update function from the next sentence and so on until the list of

sentences is exhausted.

We shall start by giving a (necessarily crude at this point) description of the construction algorithm (section 6.1), discuss the notion of *active* PRS (section 6.2.1) and why we need it, and then turn to the details of micro-construction (section 6.3) to see how we can build up sub-PRSs from NAPROCHE language expressions.

6.1 A birds-eye view of the PRS construction algorithm

1. We start with an empty PRS B_0 and a list $s_1, \ldots, s_n$ of sentences.
2. (micro) We parse s_1 and, based on type of s_1, the parse tree and in the context of B_0, construct an *update function* f_{s_1} which takes a PRS as an argument and yields a richer, modified PRS.
3. (macro) The application of f_{s_1} on B_0 yields B_1.
4. (micro+macro) We construct an update function f_{s_2} from s_2 and apply it to B_1 which yields B_2.
5. Applying f_{s_n} to B_{n-1} yields B_n.

   Formally expressed: $B_n = f_{s_n}(\ldots(f_{s_2}(f_{s_1}(B_0)))\ldots)$, where $B_n = B_{s_1,\ldots,s_n}$ is the PRS for the complete proof represented by the discourse $s_1, \ldots, s_n$.

6.2 Preliminaries

Before we look at the construction process in detail we need to discuss some preliminary issues which directly influence the current NAPROCHE implementation. Because of the natural scoping of proofs, we need to define the notion of an *active sub-PRS* and a memoisation mechanism for keeping track of accessible discourse referents.

6.2.1 Active sub-PRS

Recall that proofs are not flat, but rather hierarchically nested. This means that we have to re-construct a nested structure from a flat list of sentences. Moreover, when processing each subsequent sentence, we not only have to interpret it in the context of the PRS we have built up to this point, but also respect the scope the sentence is nested in within the original proof and incorporate any scope changes the sentence may introduce. To facility scope management we introduce the notion of an *active sub-PRS*. The active sub-PRS is a meta-property of the PRS which is essential only for the process of PRS

construction. Is not needed for the further manipulation of the final PRS and is therefore silently discarded before the verification proper.

To manage multi-level nestings, say, several assumptions in succession, we actually implement a *stack of active sub-PRSs*. Sentences which introduce a new nested scope level add to the history stack, e.g. assumptions push the sub-PRS representing their succedent onto the stack. Sentences which return the scope to a previous level push one or more elements off of the stack, e.g. assumption closing sentences pop one element, qed-sentence pop all the elements needed so as to return to the scope of the current lemma or theorem.

6.2.2 Memoising accessible discourse referents

From the definition of accessibility (section 5.6) it is clear that we can take a straightforward declarative approach to the resolution algorithm. This would, however, require "walking" the PRS tree for each sentence. This is the approach taken in standard DRT and it presents no problem there, because the nesting of sub-structures in DRSs is relatively small (one of the goals of DRS construction is to keep the universe of discourse referents in the top level of the DRS as much as possible). With PRSs however, the multiple levels of nesting and the fact that we construct at least one sub-PRS per sentence lead to a very fragmented discourse referent universe so walking the tree for each iteration becomes impractical.

Therefore we employ memoisation, keeping a set of currently accessible discourse referents together with the objects assigned to them for each sub-PRS. This also enables us to return the set of discourse referents to a previous state when we change the active sub-PRS by popping from the history stack. As is the case for the active sub-PRS these sets of accessible discourse referent is not needed for the proof verification and are discarded once the construction process in complete.

Instead of encoding these two meta properties in the PRS itself we chose to embed them in a richer structure which *includes* the PRS. This structure is a triple $\{\boldsymbol{B}, \boldsymbol{H}, \boldsymbol{A}\}$ where $\boldsymbol{B}$ is the PRS, $\boldsymbol{H}$ is the history stack and $\boldsymbol{A}$ is a list of the per sub-PRS accessible discourse referent and their associations. We can now update the definition of the construction algorithm—for each formalisation, we are starting with an empty instance of this richer structure, i.e. consisting of an empty PRS $\boldsymbol{B_0}$, a history stack consisting of just $\boldsymbol{B_0}$ and an empty list of accessible discourse referents. Listing 6.1 shows the Prolog code (save for an omitted portion of error checking) for the main loop of the construction algorithm.

```
1 discourse_to_prs(Sentences, PRS) :-
2   Initial = id~proof_1..
```

```
            drefs~[]..
            mrefs~[]..
            conds~[]..
            rrefs~[],
  In = id_stack~[proof_1]..accessibles~[]..prs~Initial,
  add_sentences(Sentences, In, PRS).

add_sentences([], In, In).
add_sentences([sentence(id(Id), Content)|Rest], In, Out) :-
  In = accessibles~Accessibles,
  phrase(sentence(Id, In, TmpOut), Content),
  add_sentences(Rest, TmpOut, Out).
```

Listing 6.1: The main loop of the NAPROCHE PRS construction algorithm. If you are not familiar with the tilde and double period notation, have a look at section B.5.

6.3 Micro-construction

The micro-construction algorithm works as follows:

1. Recognise the sentence type according to the templates in our grammar, i.e. is the sentence a definition, an assumption, etc.
2. Identify the individual sub-statements in the sentence.
3. Iterate over the sub-statements, constructing a temporary sub-PRS for each one.
4. Process the temporary sub-PRSs to a PRS representing the whole sentence according to the rules associated with the appropriate template.
5. Create the update function. Broken down procedurally, the update function consists of at least one of three possible instructions:
   (a) Insert a sub-PRS or a complex, operator connected sub-PRS construction into the active sub-PRS!
   (b) Update the list of accessible discourse referent associations!
   (c) Update the history stack!

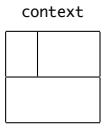
Let us look at a few introductory examples to illustrate the low-level construction mechanism. Consider the sentences (7). The numbers in brackets represent the ids of the sentences.

(7) a. [1] Observe that $\boldsymbol{u} \in \boldsymbol{x}$ and $\boldsymbol{x} \in \boldsymbol{y}$.

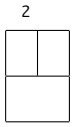
b. [2] Hence $\boldsymbol{u} \in \boldsymbol{y}$.

(7b) is an example of the simplest sentences it is possible to express in the NAPROCHE language—it consists of a single mathematical expression. We will follow the application of the micro-construction algorithm to this sentence step by step, assuming an empty context.

1. We start off with an empty PRS with a fictional id of `context`.

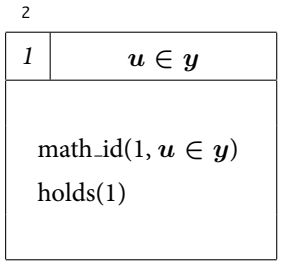


2. The trigger "hence" identifies the sentence as a statement. We match it against the template
`statement --> "hence" + sub-statement.`
and create a new PRS to accommodate the semantic contribution of the sub-statement, passing it the id of the sentence.

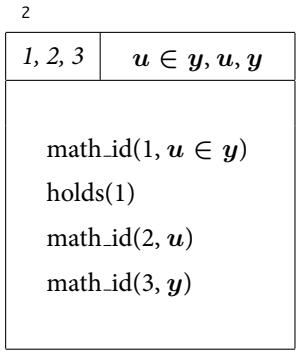


3. The sub-statement is $\boldsymbol{u} \in \boldsymbol{y}$—a single mathematical expression.

4. We insert $\boldsymbol{u} \in \boldsymbol{y}$ into the list of mathematical referents. We check whether we have already associated this expression with a discourse referent by looking it up in the list of accessible discourse referent associations. The list is empty so, naturally, no available association can be found. Therefore, we insert a new discourse referent (say, the integer 1) into the discourse referents drawer and a condition binding the expression to the referent (using the `math_id` predicate) into the conditions drawer. We add the association to the previously empty list of associations. Because $\boldsymbol{u} \in \boldsymbol{y}$ is an expression (as opposed to a single variable mention), we also add a `holds` condition.



5. We identify the free variables in the sub-statement—u and y. We insert them into the mathematical referents drawer and perform the same look up of discourse referents, and, finding none, create new ones and associate them with the variables. The newly created associations are added to the running list of associations.



6. This completes the parsing of the sentence. We have created a new sub-PRS for its contribution and have a list of discourse referent-mathematical object associations.

7. The rule for statements says that the update function they create contains the following instructions:

   (a) Insert the sub-PRS for the statement into the currently active sub-PRS!

   (b) Update the associations list!

8. Applying the update function to our initial context PRS we get the PRS in fig. 6.1.

   Conjuncted sub-statements are represented in a single sub-PRS by first creating a temporary sub-PRS for each of the sub-statements and then merging them. One thing to note is that the list of the accessible discourse referent associations is updated and passed from the one temporary sub-PRS to the next. Fig. 6.2 shows the representation of sentence 7a.

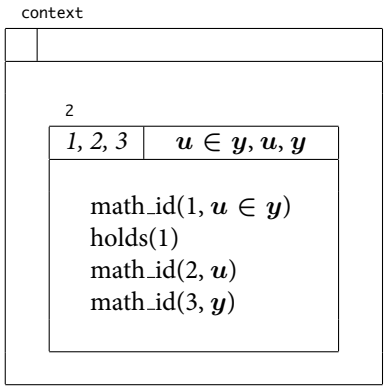


Figure 6.1: The result of inserting "Hence $u \in y$." into an empty context PRS.

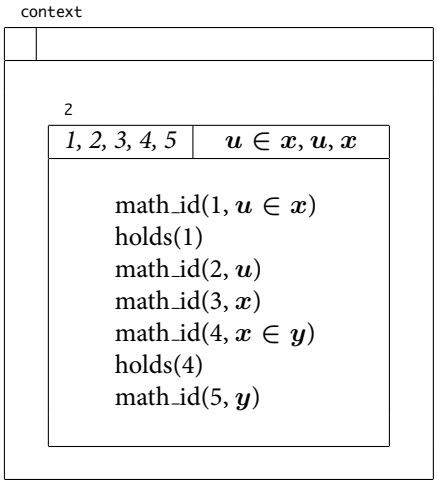


Figure 6.2: The result of inserting "Observe that $u \in x$ and $x \in y$." into an empty context PRS.

6.4 Macro-Construction

In this section we shall look at the the representation of sample discourses and the construction of complex sub-PRSs (such as those for definitions, assumptions, etc.) as they are best illustrated in a discourse context.

We shall refrain from discussing the management of accessible discourse referents as it can easily be inferred by the careful reader from the definition of accessibility (section 5.6), the remark on memoisation (section 6.2.2) and the detailed description of the low-level construction mechanism

above.

For the purpose of brevity we shall introduce a new notation for the graphical PRS representations.

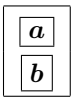
All sentences in the NAPROCHE language are defined in terms of sub-statements. We can use single letter variables to represent sub-statements and "boxed" single letter variables to represent sub-PRSs. This relation is defined as follows: if $\boldsymbol{a}$ is a sub-statement, then $\boxed{\boldsymbol{a}}$ is the sub-PRS which represents a single statement which consists only of $\boldsymbol{a}$.

We shall use the letters $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$ and $\boldsymbol{e}$ to signify sub-statements.

6.4.1 Statements

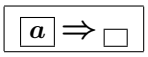
Each statement contributes a single sub-PRS to the currently active sub-PRS. Thus, upon encountering a statement $\boldsymbol{a}$, we include $\boxed{\boldsymbol{a}}$ at the bottom of the active PRS. For an initially empty PRS this means

For a PRS already containing the statement $\boldsymbol{a}$—  —adding the statement $\boldsymbol{b}$ results in

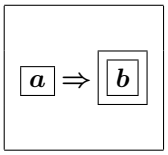


6.4.2 Assumptions

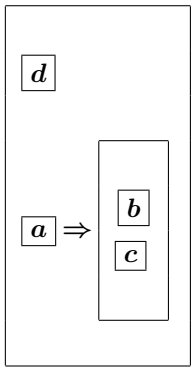
Assumptions introduce a new sub-PRS pair connected by the $\Rightarrow$ operator. The left-hand side sub-PRS contains the sub-statement of the assumption, the right-hand side one is initially empty. Each assumption opens its own new scope and thus forces all following sentences to contribute to the right-hand side of the implication until an assumption closing sentence is encountered, i.e. adding an assumption pushes the right-hand sub-PRS onto the history stack. The artificially added right-hand sub-PRS (artificial because it does not stem from a complete sentence) is assigned an id of the form `consec_` plus a running integer. The bare-bones example of a single assumption *Let* $\boldsymbol{a}$. results in



The discourse *Let* $\boldsymbol{a}$. $\boldsymbol{b}$. results in

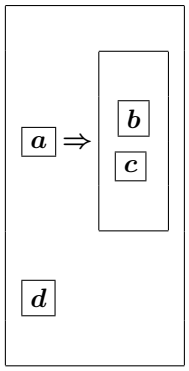


A more complex example: ***d***. *Let* ***a***. ***b***. ***c***.



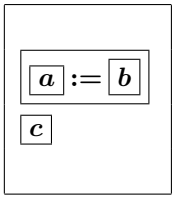
6.4.3 Assumption closing sentences

Assumption closing sentences terminate the scope of the currently active assumption by popping its "consequence" sub-PRS off of the history stack. As assumptions and assumption closing sentences are symmetrically nested, this would also be the last "consequence" box which is still open. The statement constituent of the assumption closing sentence is then added *after* the assumption sub-PRS. The following shows the representation of the discourse *Let* ***a***. ***b***. ***c***. *Thus* ***d***.



6.4.4 Definitions

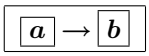
A definition introduces a new definition sub-PRS with the id of the sentence containing the definition which is then immediately closed. This sub-PRS contains a single condition consisting of two sub-PRSs connected by the `:=` operator—one for the definiendum and one for the definiens. They receive ids of the form `definiendum` plus a running integer and `definiens_` plus a running integer respectively. The left-hand side sub-PRS contains the first sub-statement in the definition, the right-hand side—the second one. The following shows the representation of the discourse *Define* ***a*** *iff* ***b***. ***c***.



6.4.5 Natural language implications

Natural language implications introduce a new sub-PRS which inherits the id of the sentence. This contains a single pair of sub-PRSs connected by the ⟶ operator. The left-hand side sub-PRS receives an id of the form `antecedent_` plus a running integer, the right-hand sub-PRS—an id of the form `succedent_` plus a running integer.

The following shows the representation of the sentence ***a*** *implies* ***b***.



This concludes our discussion on the construction mechanism for PRSs.

Chapter 7

Conclusion

In this work we described one possible method of generating a formal representation from informal mathematical discourse for the Naproche project. We began by introducing Naproche and clearly stating its goals as a motivation for our research. Then we looked at mathematical proofs, giving a brief overview of the evolution of mathematical reasoning and comparing formal and informal proofs. We also described the implications that formal and informal methods in mathematics have on the language, readability and the process of verification of the written manifestation of the proving process—the mathematical text as a vehicle for the communication of proofs. We proposed the introduction of a controlled language with well-defined semantics and a novel method of formalisation as a means of capturing proofs in form that is both human and machine readable and therefore verifiable.

We introduced the Naproche controlled language, describing its syntax. We looked at an established theory for the generation of formal representation of narrative natural language text—Discourse Representation Theory (drt)—and the basic data structure it provides—the Discourse Representation Structure (drs). The drs provides a well-defined semantics for representing natural language discourse, yet it fails to capture many vital properties specific to mathematical discourse—the inherent hierarchical structuring, the sequentiality of propositions, the reuse of variables and the references to other parts of the proof or to other proof texts.

We proposed an extension of the drs, designed specifically to capture those properties—the Proof Representation Structure (prs). We defined the language of prss and demonstrated its ability to store all of the properties outlined above. In the last chapter, we looked at the algorithm of construction of prss from mathematical discourse, describing a novel two-phase iterative process of

generation.

The practical part of this work included the development of a system capable of preprocessing mathematical text in an XML format, tokenising it, and generating its PRS representation. The system implementation was done in Prolog.

7.1 Possible improvements

The NAPROCHE system is still in its early stages of development. Certain immaturities leave room for improvement.

The NAPROCHE system is intended to be embedded in the $\TeX_{\text{MACS}}$ editor and invoked on text within the editor window. The write-test loop is admittedly a clumsy way of ensuring correctness. A more elegant way of handling this would be the extension of $\TeX_{\text{MACS}}$ to support smart look-ahead editing, by giving hints to users as to which expressions are allowed in each context, much like the ECOLE editor does (Schwitter & Tilbrook, 2007).

The grammar of the NAPROCHE language is currently extremely simple, providing coverage for only a minimal corpus. It should clearly be extended to support more expression types and a richer lexicon.

The main shortcoming of the NAPROCHE grammar, however, is that it does not parse mathematical expressions beyond the determination of their free variables. This results in different PRSs for expressions written in a different manner, but expressing the same meaning (e.g. the sentence *For all* $\boldsymbol{x}$, $\boldsymbol{x} < \boldsymbol{succ}(\boldsymbol{x})$ will be represented differently than $\forall \boldsymbol{x} \boldsymbol{x} < \boldsymbol{succ}(\boldsymbol{x})$). We currently rely on the interface to the proof checker to resolve those to an identical structure.

Also concerning the parsing of mathematical expressions: the names of variables, operators and predicates are hardwired in the lexicon, there is currently no way to determine them heuristically. Possible methods are described in (Zinn, 2004a).

On the implementation side, one evident drawback is the management of accessible discourse referents—our memoisation approach leads to duplication, as we keep the set of accessible discourse referents per sub-PRS. We use more memory for the sake of speed and ease of management. One possible way of solving this would be the use difference structures.

Appendix A

The expected proof structure

```
<text>              ::= [<open-assumption>]*, [<definition>]*, [<proof>]*;

<proof>             ::= "theorem.",
                        [<free-assumption>]*,
                        [<definition>]*,
                        [statement]+,
                        "proof.",
                        [<close-dassumption> | statement | <lemma>]+,
                        "qed.";

<lemma>             ::= "lemma.",
                        [<assumption>]*,
                        [<definition>]*,
                        [statement]+,
                        "proof.",
                        [<close-dassumption> | statement]+,
                        "qed.";

<open-assumption>   ::= <assumption>;

<closed-assumption> ::= <assumption>,
                        [statement]*,
```

```
23                          <assumption-close>;
24
25  <definition>        ::= "define",
26                          statement,
27                          ["iff" | "if and only if"],
28                          statement.
29
30  <assumption>        ::= [ "assume that" |
31                            "assume for a contradiction that" |
32                            "let" |
33                            "consider" ],
34                          statement;
35
36  <assumption-close>  ::= "thus",
37                          statement;
```

Listing A.1: The expected structure of proofs in the NAPROCHE language in EBNF.

Appendix B

Implementation

As part of my participation in the Naproche project I developed a sample implementation of the prs generation algorithms which was successfully tested on the available corpus. The implementation consists of a pre-processor for the Naproche xml format (described in section B.2), a `texmacs` module responsible for the conversion of xml data into a Prolog data structure (sec. B.3), a `prs` module which provides a framework for working with prss and a prs-to-html converter to aid the visualisation of prss. Refer to fig. B.1 for a schematic representation of the Naproche implementation architecture. I will discuss each of the different data states and the modules responsible for their transformation in the following sections. To illustrate the transformations, I will use a small portion of text as an example throughout. The text consists of the first two assumptions of the proof of John von Neumann's Theory of Ordinals, as written up by Peter Koepke and later revised by Jip Veldman. The sample text reads:

(8) Assume that $\neg\exists x\, x \in \emptyset$.
Assume that for all x, not $x \in x$.

B.1 The input document format

The TeXmacs internal format is similar in structure to a tree-like mark-up format such as xml or sgml and is saved as a plain text ASCII encoded file.

```
1 <\quotation>
2     Assume that <with|mode|math|\<neg\>\<exists\>x x\<in\>\<emptyset\>>.
```

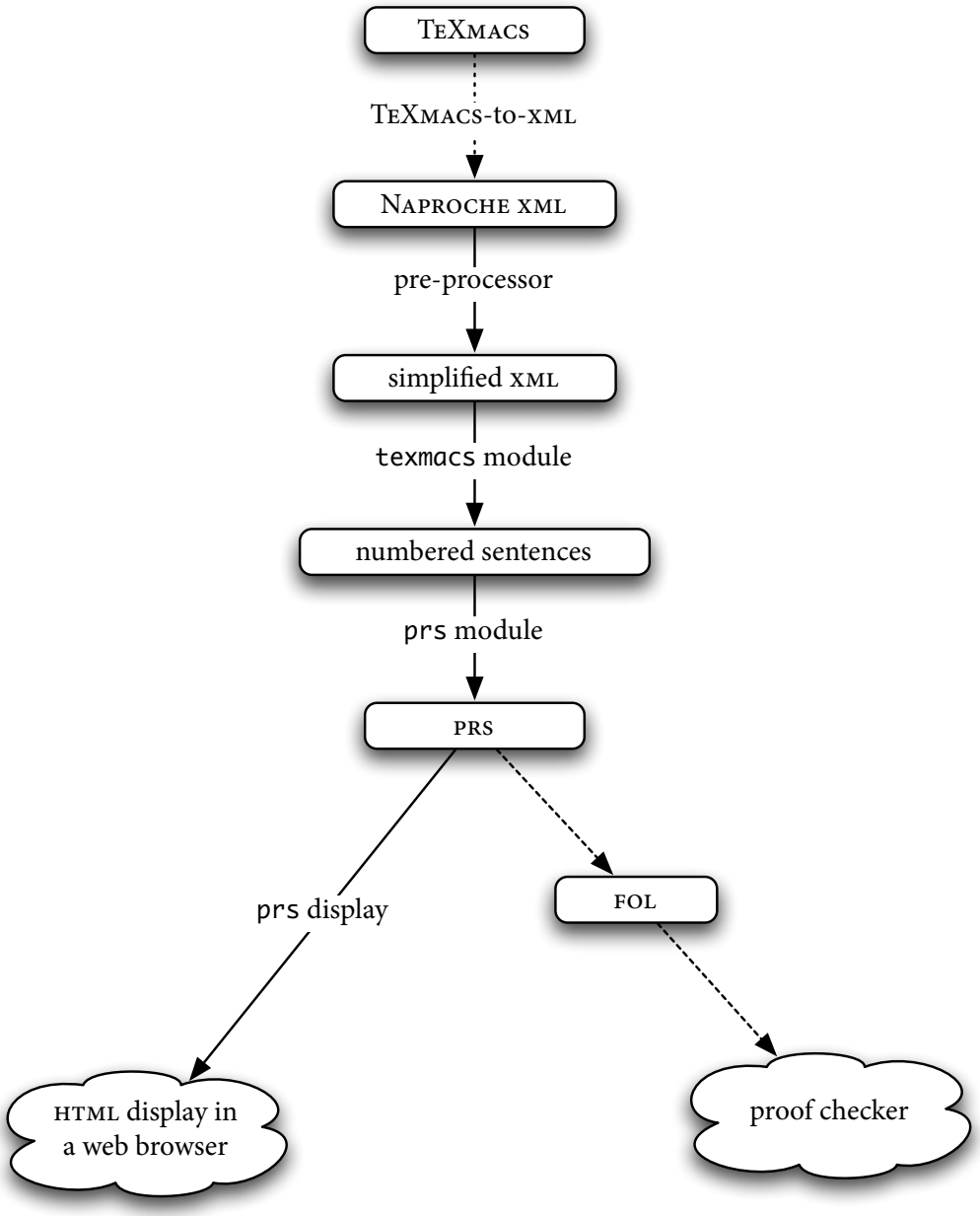


Figure B.1: The architecture of the Naproche implementation. Bubbles denote different data states, arrows denote transformations. Arrow labels show the piece of code responsible for the transformation. Solid arrows stand for modules implemented by the author, dashed arrows—for the work of other project collaborators.

```
    Assume that for all <with|mode|math|x>, not <with|mode|math|x\<in\>x>.
</quotation>
```

Listing B.1: An example of the TeXmacs internal format

Please note that all mathematical symbols are encoded as elements; thus $\in$, $\exists$ and $\emptyset$ are respectively represented by `\<in\>`, `\<exists\>` and `\<emptyset\>`.

Michael Klein has written a Scheme plug-in for TeXmacs which converts this internal format into well formed XML. Listing B.2 shows the result of the conversion.

```
<quotation id="1.1.9">
    <tm-par id="1.1.9.1">
        <concat id="1.1.9.1.1">
            <text id="1.1.9.1.1.1">Assume that </text>
            <math xmlns="http://www.w3.org/1998/Math/MathML" id="1.1.9.1.1.3">
                <mtext id="1.1.9.1.1.3"><neg id="1.1.9.1.1.3#0"/><exists id="1.1.9.1.1.3#1"/>
                x x<in id="1.1.9.1.1.3#4"/><emptyset id="1.1.9.1.1.3#5"/></mtext>
            </math>
            <text id="1.1.9.1.1.5">.</text>
        </concat>
        <concat id="1.1.9.1.3">
            <text id="1.1.9.1.3.1">Assume that for all </text>
            <math xmlns="http://www.w3.org/1998/Math/MathML" id="1.1.9.1.3.3">
                <mtext id="1.1.9.1.3.3">x</mtext>
            </math>
            <text id="1.1.9.1.3.5">, not </text>
            <math xmlns="http://www.w3.org/1998/Math/MathML" id="1.1.9.1.3.7">
                <mtext id="1.1.9.1.3.7">x<in id="1.1.9.1.3.7#1"/>x</mtext>
            </math>
            <text id="1.1.9.1.3.9">.</text>
        </concat>
    </tm-par>
</quotation>
```

Listing B.2: The TeXmacs XML export format

B.2 The pre-processor

The pre-processor is a Ruby[1] script which takes the path of a file in the TeXmacs XML export format as an argument, transforms the DOM tree and writes the result to STDOUT. Three kinds of transformations are performed:

— Nested `math/mtext` elements are converted to `math` elements.

— Namespace declarations are stripped from `math` elements[2].

— Named elements within `math` elements are converted to their Unicode equivalents and encoded as 4-bit hexadecimal entities. The conversion is made according to a separate, manually maintained conversion table.

I have also provided a driver script which sources the pre-processor, performs the necessary transformations, writes the result to a temporary file and invokes the Prolog program. This is the script which the TeXmacs plug-in is calling.

The pre-processor utilises the Hpricot[3] gem for Xpath searching and DOM tree transformations.

```
<quotation id="1.1.9">
  <tm-par id="1.1.9.1">
    <concat id="1.1.9.1.1">
      <text id="1.1.9.1.1.1">Assume that </text>
      <math>&#xac;&#x2203;xx&#x2208;&#x2205;</math>
      <text id="1.1.9.1.1.5">.</text>
    </concat>
    <concat id="1.1.9.1.3">
      <text id="1.1.9.1.3.1">Assume that for all </text>
      <math>x</math>
      <text id="1.1.9.1.3.5">, not </text>
      <math>x&#x2208;x</math>
      <text id="1.1.9.1.3.9">.</text>
    </concat>
  </tm-par>
```

[1] Ruby <http://www.ruby-lang.org> is an object-oriented scripting language.

[2] This is currently desirable as the namespaces of `math` elements in the TeXmacs export format declare the contents of those elements to be valid MathML. This is, however, unfortunately not the case.

[3] Hpricot <http://code.whytheluckystiff.net/hpricot/> is an HTML/XML processing library for Ruby.

```
16 </quotation>
```

Listing B.3: The XML document after pre-processing

B.3 The `texmacs` module

The `texmacs` module serves as an interface between documents in the transformed (simplified) XML format (described in section B.2) and the internal format expected by the `prs` module. It exports the single predicate `texmacs_sentences/2` which is defined as follows:

texmacs_sentences(File, Sentences) ⟵

Sentences is the list of significant sentences contained in the document saved under the path *File*.

Internally, the `texmacs` module provides code for a number of steps to facilitate the conversion. First, the document at path *File* is read and converted to a complex Prolog term using the XML processing module described in section B.6. Significant portions are selected[4] and their contents tokenised and split into sentences. The splitting algorithm is at present rudimentary, splitting sentences at the periond (character code 46). Sentences are also associated with an unique id which allows for a sentence to be traced to an exact location in the source document. The id consists of the id of the line on which the sentence begins, a dash, and a running index. Thus, the sentence with id 1.2.3.4-5 is the fifth running sentence in the line with id 1.2.3.4.

```
1 [ sentence(id('1.1.9.1.1-1'), [assume, that, math("\u0XAC\u2203xx\u2208\u2205")]),
2   sentence(id('1.1.9.1.3-1'), [assume, that, for, all, math("x"), (','), not, math("x\u2208x")]) ]
```

Listing B.4: The example sentences transformed to Prolog terms. Symbols are represented as Unicode strings, mathematical terms and formulas are wrapped in `math` predicates.

[4] Authors of Naproche documents are expected to explicitly mark the portions of the text they wish to be checked. This provides a simple mechanism for separating the actual proof text from meta information, such as comments or clarifying remarks, which, though informative for human readers, are of no significance to the prover. At present, authors need to place all significant text in the TeXmacs *quotation* environment. It is planned, however, that the Naproche plug-in for TeXmacs also provide a designated *naproche* environment. This would require minor changes to the `texmacs` module.

B.4 The `prs` module

The `prs` module provides a framework for working with PRSs. This includes common operations on these structures like constructing, modifying, searching and displaying. Included with the `prs` module is also an example grammar in DCG format which is tailored to the available examples. The main predicate this module exports in `discourse_to_prs/2`:

discourse_to_prs(Discourse, PRS) ⟵

> *Discourse* is a list of sentences as provided by the `texmacs` module; parsing them results in a PRS which is unified with *PRS*. The predicate succeeds if all sentences in *Discourse* can be successfully parsed.

`discourse_to_prs/2` sets up an empty PRS and traverses the list of sentences provided as its first parameter, adding them sequentially to the PRS. Should a sentence fail to parse correctly according to the provided grammar, the parsing is halted and a diagnostic error message including the id of the faulty sentence is written to STDOUT.

B.5 PRS representation

Throughout the current NAPROCHE implementation PRSs are represented as feature-value (f-v) structures with the following features (text in parentheses shows the actual feature name as used in the code): id (`id`), discourse referents (`drefs`), mathematical references (`mrefs`), conditions (`conds`) and text references (`rrefs`). Prolog does not provide a built-in mechanism for f-v structures, so I used Michael Covington's GULP4 (Covington, 1994) extension. It extends Prolog with a syntax and mechanism for encoding f-v structures. In GULP, features and values are separated by a tilde (~), and f-v pairs are joined by a double period (..). An empty PRS in GULP looks like this:

```
1 id~_..       % an uninstantiated id
2 drefs~[]..   % an empty list of discourse referents
3 mrefs~[]..   % an empty list of mathematical references
4 conds~[]..   % an empty list of conditions
5 rrefs~[]     % an empty list of text referents
```

Listing B.5: An empty PRS in GULP syntax.

Nested sub-PRSs are included in the list of conditions. For performance reasons the list of conditions is built-up backwards (new conditions are added to the front of the list). Any code which processes the PRSs further should thus reverse the list to restore the correct order.

B.6 The `xml` module

The `xml.pl` module used for reading XML encoded proof texts was written by John Fletcher and released in the public domain[5]. I made some minor modifications to ensure compatibility with recent versions of SWI-Prolog. A version of `xml.pl` is included in the NAPROCHE codebase.

[5]The most recent version of `xml.pl`, along with the licence under which it is released, can be found at `http://www.zen37763.zen.co.uk/xml.pl.html`.

List of Figures

Listings

Bibliography

Black, Alan. *Some Different Approaches to DRT*, 1993.

Boole, George. *The Calculus of Logic*. In: *The Cambridge and Dublin Mathematical Journal*, 3 (1848).

Bos, J., Mastenbroek, E., McGlashan, S., Millies, S. & Pinkal, M. *A compositional DRS-based formalism for NLP-applications*. In: *Proceedings of the International Workshop on Computational Semantics*, pp. 21--31. Tilburg, 1994.

Bourbaki, Nicolas. *Theory of sets*. Addison-Wesley, 1968.

Covington, Michael. *GULP 3.1: An Extension of Prolog for Unification-Based Grammar*, 1994.

Frege, Gottlob. *Begriffsschrift. Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879.

Frege, Gottlob. *Kleine Schriften*. Vdm Verlag Dr. Müller, 2006.

Groenendijk, J. & Stokhof, M. *Dynamic Predicate Logic*. In: *Linguistics and Philosophy*, 14 (1991):39--100.

Groenendijk, J., Stokhof, M. & Veltman, F. *This might be it*, 1996.

Gödel, Kurt. *Russell's mathematical logic*. In: *The Philosophy of Bertrand Russell*. Northwestern University, 1944.

Hanna, G. *Rigorous proof in mathematics education*. In: *Curriculum Series. The Ontario Institute for Studies in Education*, (1983).

Harrison, John. *Formalized Mathematics*, 1996.

Heim, Irene. *The Semantics of Definitie and Indefinitio Noun Phrases in English.* Ph.D. thesis, University of Massachusetts, Amherst, Massachusetts, 1982.

Johnson, M. & Klein, E. *Discourse, anaphora and parsing.* In: *Proceedings of the 11th International Conference on Computational Linguistics*, pp. 669--675. Bonn, 1986.

Jones, Roger Bishop. *Philosophical Revolutions*, 2007.

Kamp, Hans. *A theory of truth and semantic representation.* In: M. Stokhof J. Groenendijk, T. Janssen, ed., *Formal Methods in the Study of Language*. Mathematical Center, 1981.

Kline, Morris. *Mathematics: The Loss of Certainty.* Oxford University Press US, 1982.

Krantz, S. G. *A Primer of Mathematical Writing.* American Mathematical Society, 1997.

Krantz, Steven. *The history and concept of mathematical proof*, 2007.

Mathias, Adrian R. D. *The Ignorance of Bourbaki.* In: *Mathematical Intelligencer*, 14 (1992)(3):4--13. URL `citeseer.ist.psu.edu/mathias90ignorance.html`.

Mendelson, Elliott. *Introduction to mathematical logic.* Chapman & Hall, London, 1997.

Millo, Richard A. De, Lipton, Richard J. & Perlis, Alan J. *Social processes and proofs of theorems and programs.* In: *Commun. ACM*, 22 (1979)(5):271--280.

Russell, Bertrand. *Introduction to Mathematical Philosophy.* George Allen and Unwin, London, 1919.

Russell, Bertrand. *The autobiography of Bertrand Russell.* Allen & Unwin, 1968.

Schwitter, Rolf & Tilbrook, Marc. *Dynamic Semantics at Work.* In: *New Frontiers in Artificial Intelligence*, volume 3609, pp. 416--426. Springer, Berlin / Heidelberg, 2007.

Steenrod, Norman E., Halmos, Paul R., Schiffer, Menahem M. & Dieudonne, Jean A. *How to Write Mathematics*, 1973.

Thurston, William P. *On proof and progress in mathematics.* In: *Bulletin of the American Mathematical Society*, 30 (1994):161.

Trybulec, Zinaida & Święczkowska, Halina. *The language of mathematical texts.* In: *Studies in Logic, Grammar and Rhetoric*, 10/11 (1991):103--124.

van der Hoeven, Joris. *GNU TeXmacs*, 1998. URL http://www.texmacs.org.

van Gasteren, A. J. M. *On the shape of mathematical arguments.* In: *Lecture Notes in Computer Science*, 445 (1990).

van Heijenoort, Jean. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931.* Harvard University Press, 1967.

Veltman, Frank. *Defaults in Update Semantics.* In: *Journal of Philosophical Logic*, 25 (1996)(3):221--261.

Whitehead, Alfred North. *Science and the Modern World.* Cambridge University Press, Cambridge, 1925.

Whitehead, Alfred North & Russell, Bertrand. *Principia Mathematica.* Cambridge University Press, Cambridge, 1910, 1912, 1913.

Wolska, Magdalena & Kruijff-Korbayová, Ivana. *Analysis of mixed natural and symbolic language input in mathematical dialogs.* In: *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, p. 25. Association for Computational Linguistics, Morristown, NJ, USA, 2004a.

Wolska, Magdalena & Kruijff-Korbayová, Ivana. *Building a Dependency-Based Grammar for Parsing Informal Mathematical Discourse.* In: Petr Sojka, Ivan Kopecek & Karel Pala, eds., *TSD*, volume 3206 of *Lecture Notes in Computer Science*, pp. 645--652. Springer, 2004b.

Zinn, Claus. *On the Use of Variables in Mathematical Discourse.* In: E.Buchberger, ed., *Proceedings of Konvens 2004 (7. Konferenz zur Verarbeitung natuerlicher Sprache)*, pp. 217--220. Wien, 2004a.

Zinn, Claus. *Understanding Informal Mathematical Discourse.* Ph.D. thesis, Arbeitsberichte des Instituts für Informatik Friedrich-Alexander-Universität Erlangen Nürnberg, 2004b.

Die Generierung von Beweisrepräsentationsstrukturen für das Projekt Naproche

Naproche ist ein Forschungsprojekt, das von Prof. Dr. Bernhard Schröder (ehemals Institut für Kommunikationsforschung der Universität Bonn, IfK) und Prof. Dr. Peter Koepke (Mathematisches Institut der Universität Bonn) ins Leben gerufen wurde. Das Projekt hat zwei zentrale Ziele—eine kontrollierte Sprache zu entwickeln, die das Aufschreiben mathematischer Beweise erlaubt, und ein System bereitzustellen, das die automatisierte Überprüfung der Korrektheit von Beweisen in dieser Sprache ermöglicht. Das System ist als Hilfe sowohl für Autoren gedacht, die ihre eigene Arbeit überprüfen wollen bevor sie eingereicht wird, als auch für Rezensenten, die ihnen zur Bewertung gereichte Arbeiten überprüfen müssen. Die prototypischen Benutzer des Systems sind Studenten der Mathematik und verwandte Disziplinen wie Physik und Informatik, die mathematische Beweise als Hausarbeiten einreichen müssen, und Professoren und Dozenten, die große Mengen solcher Beweise bewerten müssen.

Aus der Sicht des Benutzers ist das System einfach. Es kann in den Editor $\mathrm{TEX}_{\mathrm{MACS}}$ integriert werden und ermöglicht die Überprüfung des sich im Editorfenster befindenden mathematischen Text per Tastendruck. Die Benutzerinteraktion ist minimal—nach einem Aufruf des Systems bekommt der Benutzer die Meldung *"Proof accepted." ("Beweis angenommen.")* oder eine Fehlermeldung mit dem Hinweis auf den Fehler, der eine erfolgreiche Überprüfung verhindert hat. Unter der Haube ist das System jedoch komplex und mehrschichtig—der Text im Editorfenster wird mehreren Formattransformationen ausgesetzt, bevor er dann in eine formale Struktur überführt wird, die ihrerseits einem externen automatischen Beweisprüfer übergeben wird. Diese Arbeit beschreibt die Art

dieser Transformationen und insbesondere die Methoden der Erstellung formaler Repräsentationen aus (pseudo-)informellem mathematischen Diskurs.

Mathematical Proof (Mathematischer Beweis) Kapitel 2 ist dem mathematischen Beweis gewidmet. Wir verfolgen die Evolution der mathematischen Beweisführung—von der Entwicklung der axiomatischen Methode und den ersten logischen Inferenzregeln, den Syllogismen, von Aristoteles bis hin zur den Versuchen von Frege, Russel und Hilbert, einen einheitliches, rein logisches Fundament für die Mathematik zu finden. Die argumentativen und die formalen Methoden der Beweisführung führen zu zwei verschiedenen Auffassungen des mathematischen Beweises—der Beweis als überzeugendes Argument für andere Mathematiker (informaler Beweis) und der Beweis als eine in sich geschlossene Kette von logischen Schritten, die in einem formalen System ausgeführt wird (formaler Beweis).

Informale Beweise werden mit Hilfe einer nicht strickt festgehaltenen Sprache, einer Mischung aus natürlicher und symbolischer Sprache, ausgedrückt. Sie beinhalten Hinweise und Erklärungen, die Hintergrundwissen und Erfahrung seitens des Lesers voraussetzen. Somit ist es nur möglich, sie durch Interpretation zu auf ihre Korrektheit hin zu verifizieren. Die Verifizierung läuft als sozialer Prozess des Peer-Review ab, eine maschinelle Verifizierung ist nicht möglich.

Formale Beweise werden vollständig in einem formalen System durchgeführt, das eine wohldefinierte symbolische Sprache, eine Menge an grundlegenden Aussagen (Axiomen) und eine Menge an logischen Schlußfolgerungsregeln bereitstellt. Formale Beweise sind deshalb mechanisch, durch eine automatische Überprüfung ausschließlich auf syntaktischer Ebene, verifizierbar.

Naproche Beweise weisen eine Mischung aus Eigenschaften sowohl formaler als auch informaler Beweise auf. Sie werden in einer formalen Sprache geschrieben, die jedoch viele Ausdrücke und Sprachkonstruktionen beinhaltet, die der informalen mathematischen Praxis entliehen sind. Somit sind sie sowohl von Menschen gut lesbar, als auch automatisch verifizierbar.

Tabelle Z.1 bietet einen Vergleich der verschiedenen Beweistypen in Hinsicht auf ihre Sprache, Lesbarkeit und Methoden der Verifizierung.

The Language of Mathematics (Die Sprache der Mathematik) Kapitel 3 bietet einen Überblick über die Sprache der mathematischen Texte. Mathematische Texte sind immer informal, halbstarr strukturiert und sind in einer hybriden Sprache—einer Mischung aus ethnischer, natürlicher Sprache und einer universellen symbolischen Sprache—geschrieben. Man unterscheidet zwischen mehreren *Satztypen*—Aussagen, Annahmen, Definitionen u.s.w—die eigenen Einfluss auf den Verlauf des Beweises

	Formal	*Informal*	*Naproche*
Sprache	formal, symbolisch, festgehalten	Mischung aus natürlicher und symbolischer, nicht festgehalten	Mischung aus natürlicher und symbolischer, festgehalten
Lesbarkeit	kaum/nicht lesbar	lesbar	lesbar
Verifikation	mechanische, auf syntaktischer Ebene, Interpretation nicht erforderlich	Peer-Review, sozialer Prozess	mechanische und durch sozialea Prozesse

Table Z.1: Vergleichende Tabelle der Sprache, Lesbarkeit für Menschen und Methoden der Verifizierung für formale, informale und Naproche Beweise.

ausüben. Z.B. Annahmen definieren einen neuen Skopus für Variablen, Definitionen führen neue Sprachkonstrukte ein, etc.

The Naproche language (Die Naproche Sprache) Kapitel 4 definiert die Sprache, in der Naproche Beweise geschrieben werden. Sie ist eine kontrollierte Sprache mit festgehaltener Semantik, die aus einer ausgewählten Menge an natürlichsprachlichen Ausdrücken aus der mathematischen Praxis und einer symbolischen Sprache für mathematische Ausdrücke besteht. In der aktuellen Naproche Implementation ist die Sprache eine Template-Sprache, die Satztypen werden an Triggerphrasen erkannt, z.B. *"Assume that* $\boldsymbol{x}$ *..."* oder *"Let* $\boldsymbol{x}$ *..."* für Annahmen, *"Define* $\boldsymbol{Ord}(\boldsymbol{x})$ *if and only if ..."* für Definitionen. Die Templates sind auf der Idee von sub-Statements aufgebaut—einzelne bedeutungstragende Ausdrücke, die in den Templates eingebunden werden. Die Verarbeitungsregeln der Naproche Grammatik setzen die semantischen Beiträge der sub-Statements in Beziehung zu einander.

Discourse Representation Theory (Diskursrepräsentationstheorie) Im 5. Kapitel haben wir einen Überblick der Diskursrepräsentationstheorie (DRT) geschafft, als inspirierender Startpunkt für die Entwicklung des Naproche Systems. DRT stellt einen Ansatz für die Formalisierung von natürlichsprachlichen Diskursen dar und ist somit einer der angestrebten Grundfunktionalitäten des Naproche nicht unähnlich. Die Grunddatenstruktur der DRT ist die Discourse Representation Structure (DRS)—ein 2-Tupel, bestehend aus einer Menge *Diskursreferenten* und einer Menge *Diskursbedingungen*. DRSen werden durch die iterative Verarbeitung eines Diskurses, repräsentiert als endliche Liste von Sätzen, aufgebaut. Die DRS Definition ist vom Aufbauverfahren unabhängig—die zwei etablierten Verfahren sind Threading und Konstruktion durch $\boldsymbol{\lambda}$-Ausdrücke.

Proof Representation Structures (Beweisrepräsentationsstrukturen) Kapiteln 6 und 7 stellen den Kern dieser Arbeit dar. Die Proof Representation Structure (PRS) ist eine speziell für das NAPROCHE System entwickelte Datenstruktur. Sie basiert auf der DRS, besitzt jedoch eine reichere Semantik und Sprache. Die PRS ist definiert, um die Repräsentation wichtiger Eigenschaften von mathematischen Beweisen zu ermöglichen. Die wichtigsten davon sind:

Beweistruktur Mathematische Beweise sind hierarchisch geschachtelte Strukturen im Gegensatz zu erzählerischen Texten. Diese Hierarchie muss in der formalen Beweisrepräsentation wiederzufinden sein.

Reihenfolge Die Reihenfolge der Sätze muss in der Repräsentation beibehalten werden. Der semantische Beitrag einzelner Sätze muss auch einzeln festgehalten werden, Satzrepräsentationen dürfen nicht veinigt werden.

Wiederverwedung von Variablennamen Der hierarchische Aufbau mathematischer Beweise führt dazu dass Variablennamen, z.B. x, y, z in Laufe des Beweises auf verschiedenen Variablen mit verschiedenen Existenzbedingugen referieren.

Definitionen Definitionen verändern die Sprache in der ein Beweis geschrieben wird, indem sie neue Sprachkonstrukte einführen.

Intra- und Intertextuelle Referenzen Mathematiker verwenden in ihren Beweisen oft bereits bewiesene Sätze und weisen auf deren Beweis hin.

Lokalisation Jeder Satz muss eindeutig identifiziert werden können, damit das System dem Benutzer möglichst akkurate Fehlermeldung präsentieren kann.

Um diese Eigenschaften festhalten zu können erweitert die Definition der PRS die definition der DRS um 3 zusätzliche Merkmale—die *id* (die eine eindeutige Identifikation von Sätzrepräsentationen erlaubt), die *Liste der mathematischen Referenten* (eine Liste der mathematischen Objekte über die gesprochen wird) und die *Liste der Referenzen* (eine Liste der Referenzen auf andere Beweise). S.g. Container-PRSen (mit leeren Merkmalen, bis auf die *id*) dienen der Strukturierung der Beweise.

Der Aufbau von PRS ist ein iterativer Prozess, der auf zwei Ebenen verläuft. Die Mikrokonstruktion (auf Satzebene) ist der Prozess der Generierung von *Update Functionen* auf der Basis des Parsebaums der Satzes und des Kontext der bis dahin verarbeiteten Sätze. Die Makrokonstruktion (auf der Diskursebene) ist der Prozess der Anwendung der Update Funktion für jeden Satz auf die partielle PRS, um eine neue, reichere PRS zu ergeben. Diese neue PRS wird dann in der nächsten Iteration als Kontext für die Mikrokonstruktion verwendet usw.

In dem letzten Kapitel weisen wir auf möglich Richtungen für die zukünftige Entwiklung des NAPROCHE Projekts.