Proof-checking mathematical texts in controlled natural language

Dissertation
zur Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Marcos Cramer
aus
Buenos Aires

Bonn, 2013

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Peter Koepke
2. Gutachter: Prof. Dr. Bernhard Schröder
Tag der Promotion: 7. Oktober 2013
Erscheinungsjahr: 2013

Für Uljana

Acknowledgements

This dissertation would not have been possible without the support that I received from various people during my doctoral studies. I am truly indebted and thankful for their support.

I would like to thank my supervisor Prof. Dr. Peter Koepke for warmly accepting me into the interesting interdisciplinary Naproche project and for his academic support throughout my time as a PhD student, especially for the fruitful discussions about the vision of the Naproche project and for his advice both about subtleties in the realm of mathematical logic and about the potential pitfalls of interdisciplinary work. Furthermore, I am grateful for his support in surmounting organizational problems like those related to becoming and being a PhD student in my home country Germany with a foreign degree.

I would like to thank my co-supervisor Prof. Dr. Bernhard Schröder for introducing me into formal and computational linguistics and for calling my attention to interesting phenomena on the edge between linguistics and logic relevant to the study of the language of mathematics. The suggestions he provided as a co-author of academic papers were a great help to me for learning the art of scientific writing.

I am grateful to Daniel Kühlwein for our fruitful collaboration during my first two and a half years at the Naproche project, which laid the ground for most of the ideas developed in this thesis. Special thanks go to him as well as Merlin Carl, Bernhard Fisseni and Torsten Nahm for the useful suggestions and corrections of earlier versions of this thesis.

My wife Uljana has been my main emotional support while I prepared and wrote this thesis. By making me the gift of two children during this time, she ensured that I did not become too absorbed into academic work and thoughts, but had the pleasure to feel the joy of life in its purest form. I am very grateful to her. Finally, I warmly thank my parents for their emotional and financial support during my doctoral studies.

Abstract

The research conducted for this thesis has been guided by the vision of a computer program that could check the correctness of mathematical proofs written in the language found in mathematical textbooks. Given that reliable processing of unrestricted natural language input is out of the reach of current technology, we focused on the attainable goal of using a *controlled natural language* (a subset of a natural language defined through a formal grammar) as input language to such a program. We have developed a prototype of such a computer program, the *Naproche system*. This thesis is centered around the novel logical and linguistic theory needed for defining and motivating the controlled natural language and the proof checking algorithm of the Naproche system. This theory provides means for bridging the wide gap between natural and formal mathematical proofs.

We explain how our system makes use of and extends existing linguistic formalisms in order to analyse the peculiarities of the language of mathematics. In this regard, we describe a phenomenon of this language previously not described by other logicians or linguists, the *implicit dynamic function introduction*, exemplified by constructs of the form "for every x there is an $f(x)$ such that ...". We show how this function introduction can lead to a paradox analogous to Russell's paradox. To tackle this problem, we developed a novel foundational theory of functions called *Ackermann-like Function Theory*, which is equiconsistent to ZFC (Zermelo-Fraenkel set theory with the Axiom of Choice) and can be used for imposing limitations to implicit dynamic function introduction in order to avoid this paradox.

We give a formal account of implicit dynamic function introduction by extending *Dynamic Predicate Logic*, a formalism developed by linguists to account for the dynamic nature of natural language quantification, to a novel formalism called *Higher-Order Dynamic Predicate Logic*, whose semantics is based on Ackermann-like Function Theory. Higher-Order Dynamic Predicate Logic also includes a formal account of the linguistic theory of *presuppositions*, which we use for clarifying and formally modelling the usage of potentially undefined terms (e.g. $\frac{1}{x}$, which is undefined for $x = 0$) and of definite descriptions (e.g. "the even prime number") in the language of mathematics. The semantics of the controlled natural language is defined through a translation from the controlled natural language into an extension of Higher-Order Dynamic Predicate Logic called *Proof Text Logic*. Proof Text Logic extends Higher-Order Dynamic Predicate Logic in two respects, which make it suitable for representing the content of mathematical texts: It contains features for representing complete texts rather than single assertions, and instead of being based on Ackermann-like Function Theory, it is based on a richer foundational theory called *Class-*

Map-Tuple-Number Theory, which does not only have maps/functions, but also classes/sets, tuples, numbers and Booleans as primitives.

The proof checking algorithm checks the deductive correctness of proof texts written in the controlled natural language of the Naproche system. Since the semantics of the controlled natural language is defined through a translation into the Proof Text Logic formalism, the proof checking algorithm is defined on Proof Text Logic input. The algorithm makes use of *automated theorem provers* for checking the correctness of single proof steps. In this way, the proof steps in the input text do not need to be as fine-grained as in formal proof calculi, but may contain several reasoning steps at once, just as is usual in natural mathematical texts. The proof checking algorithm has to recognize implicit dynamic function introductions in the input text and has to take care of presuppositions of mathematical statements according to the principles of the formal account of presuppositions mentioned above. We prove two soundness and two completeness theorems for the proof checking algorithm: In each case one theorem compares the algorithm to the semantics of Proof Text Logic and one theorem compares it to the semantics of standard first-order predicate logic.

As a case study for the theory developed in the thesis, we illustrate the working of the Naproche system on a controlled natural language adaptation of the beginning of Edmund Landau's *Grundlagen der Analysis*.

Preface

This doctoral thesis presents interdisciplinary work about the language of mathematics that closely interlinks methods from mathematical logic and linguistics, and to a lesser extend from computer science. Since the work is likely to be of interest to readers with different backgrounds, we need to say some words about what prerequisites are needed in order to understand it fully, and which parts can still be read by people who lack some of these prerequisites.

A reader willing to read the complete thesis is assumed to be familiar with the basics of mathematical logic and set theory, i.e. with the material that is usually taught in two semesters worth of undergraduate lectures on these topics. Additionally, familiarity with formal semantics, especially with Dynamic Predicate Logic (Groenendijk & Stokhof, 1991), is helpful but not presupposed. A reader who lacks the mathematical prerequisites or who is primarily interested in the linguistic aspects of our work may read the thesis by dropping chapter 4 and sections 6.3 and 6.4. In order to still understand chapters 5 and 8 as well as possible, such a reader should know that in chapter 4 we define the following three mathematically consistent theories:

- *AFTB*, a theory for talking about maps/functions and Booleans
- *CMT*, a theory for talking about classes/sets, maps/functions, tuples and Booleans
- *CMTN*, a theory for talking about classes/sets, maps/functions, tuples, natural numbers and Booleans

Contents

Chapter 1

Introduction

Many mathematicians use computer programs to support their work: Computer algebra systems facilitate the algebraic manipulation of involved symbolic mathematics. Numerical analysis software provides efficient algorithms for finding numerical solutions to mathematical problems. And LaTeX is widely used by mathematicians for typesetting mathematical formulae and complete texts. But one of the central parts of mathematical work is that of providing mathematical proofs for establishing the truth of mathematical theorems, and for this work ordinary mathematicians hardly use the support of computer programs.

There do exist computer systems for checking the correctness of mathematical proofs, but these systems require the user to use a formal input language, close in nature to programming languages and not to the language mathematicians usually use for producing mathematical proofs. For this reason, these computer systems are used only by a comparatively small community of mathematicians, the formal mathematics community, but not by the mathematical community at large.

The research conducted for this doctoral thesis was guided by the vision of a future computer program which could support mathematicians as they write their mathematical proofs in the usual language employed by mathematicians for this purpose. More concretely, in the course of this research we have already developed a prototypical computer system, called the *Naproche system*, which can check the logical correctness of simple mathematical proofs written in a *controlled natural language*, i.e. in a strictly defined but expressively rich part of the natural language of mathematical proofs.

The development of such a computer system required novel theoretical work on the border between mathematical logic and formal linguistics. As a prerequisite to this work, a thorough understanding of the natural language of mathematical proofs was needed, including the parts which seem counter-logical to someone trained in mathematical logic. One phenomenon of this language previously not described by other logicians or linguists, which we termed the *implicit dynamic function introduction*, has motivated some interesting work in the foundations of mathematics. We have studied extensions of *Dynamic Predicate Logic*, a system used in formal linguistics, which formalize implicit dynamic function introduction and other linguistic and logical phenomena of the language of mathematics.

This thesis presents the multifaceted theoretical work that we developed in

the course of developing the Naproche system: The application of linguistic theory to the language of mathematics, the relation between implicit dynamic function introduction and the foundations of mathematics, and the soundness and completeness of the proof checking algorithm implemented in the Naproche system. Furthermore, it provides a detailed exposition of the controlled natural language that serves as the input language of the Naproche system, and illustrates the working of the Naproche system on an example text, an adaptation of the beginning of Edmund Landau's *Grundlagen der Analysis*.

For the rest of this introduction, we present different threads from the scientific endeavour which get intertwined to form the topic of this interdisciplinary thesis. Furthermore, we discuss related work by other researchers in this field, explain the institutional context of the research done for this thesis, motivate the choice of treated problems and provide an outline of the rest of the thesis.

1.1 The language of mathematics

Just as other sciences, mathematics has developed its own specialized language. This specialized language has a number of registers, i.e. varieties used in different social settings: There are purely written registers like the language of undergraduate textbooks, the language of graduate textbooks and the language of research journals. There are registers of spoken language accompanied by handwriting on a board or piece of paper, like the language of undergraduate lectures, the language of talks at scientific conferences and the language of informal communication between research mathematicians at a whiteboard or with pen and paper. Finally, there are purely spoken registers of informal communication without a whiteboard or pen and paper. Of course, the boundaries between these registers are fluid. In this thesis we will focus on the written registers of mathematical language, especially on the registers of undergraduate and graduate textbooks. So for the rest of this thesis, the term *the language of mathematics* will always refer to these written registers of the specialized language of mathematics.

There are of course separate languages of mathematics based on different natural languages: The English language of mathematics, the Russian language of mathematics, the French language of mathematics etc. In this thesis we concentrate on the English language of mathematics, even though much of what we will say about it applies equally or with small adaptations to the other languages.

As an example of the language of mathematics, we cite a text fragment from Wolfenstein (1969).

> **Definition 5.8.** *A linearly independent set (resp. sequence) whose elements generate a given vector space is called a basis (resp. ordered basis) of that space.*
>
> *Examples*
>
> 1. The empty set is a basis of the zero-space.
> 2. $(E_1, \ldots, E_m)$ is an ordered basis of $\mathbf{F}^m$. We call it the *canonical* basis.

3. The polynomials: 1, X, X^2, ... form a basis (or an ordered basis) of the space of polynomials.

In our three-dimensional geometric representation, any three non-coplanar vectors form a basis.

The following theorem gives a more useful characterization of bases.

Theorem 5.6. *Let $\mathcal{V}$ be a nontrivial vector space, $\mathcal{X}$ a subset of $\mathcal{V}$. Then $\mathcal{X}$ is a basis if, and only if, each vector of $\mathcal{V}$ has a unique representation as a linear combination of elements of $\mathcal{X}$.*

Proof. What we have to prove is that the linear independence of a set of generators $\mathcal{X}$ is equivalent to the uniqueness of the representation. If $\mathcal{X}$ is not linearly independent, the representation is, in general, surely not unique, since we have $a_1\mathbf{x}_1 + \cdots + a_n\mathbf{x}_n = \mathbf{0} = 0\mathbf{x}_1 + \cdots + 0\mathbf{x}_n$, where the $\mathbf{x}$'s are distinct elements of $\mathcal{X}$ and the a's are not all zero. Conversely, suppose that some vector $\mathbf{v}$ has two distinct representations as a linear combination of elements of $\mathcal{X}$. Then we have $\mathbf{v} = a_1\mathbf{x}_1 + \cdots + a_n\mathbf{x}_n = b_1\mathbf{x}_1 + \cdots + b_n\mathbf{x}_n$, where the $\mathbf{x}$'s are distinct elements of $\mathcal{X}$ and $a_i \neq b_i$ for at least one i. Consequently $(a_1 - b_1)\mathbf{x}_1 + \cdots + (a_n - b_n)\mathbf{x}_n = \mathbf{0}$, and the $\mathbf{x}$'s are linearly dependent.

As the example illustrates, the language of mathematics incorporates the syntax and semantics of the general natural language. Hence it takes over its complexity and some of its ambiguities. However, mathematical texts are distinguished from common language texts by several characteristics. Below we give a list of some of the most perspicuous characteristics of mathematical texts.[1] Some of the features mentioned are also found in general language, but are much more prevalent in the language of mathematics than in general language.

- Mathematical texts combine natural language expressions with mathematical symbols and formulae, which can syntactically function as noun phrases or sub-propositions.
- Constructions which are hard to disambiguate are generally avoided.
- Mathematical symbols can be used for disambiguation, e.g. by use of variables instead of anaphoric pronouns.
- Assumptions can be introduced and retracted. In the proof to theorem 5.6 in the above text fragment, the sentence beginning with "Conversely, suppose" introduces the assumption that some vector $\mathbf{v}$ has two distinct representations as a linear combination of elements of $\mathcal{X}$. The claims that follow are understood to be relativized to this assumption. When the assumption gets retracted at the end of the proof, it allows one to conclude one of the two implications needed for the bi-implicational claim of the theorem.

[1] This list of characteristics of the language of mathematics is an adapted and extended version of a similar list mentioned in Cramer, Fisseni, et al. (2010).

- Mathematical texts are highly structured, and their structure is often made explicit. At a global level, they are commonly divided into building blocks like definitions, lemmas, theorems and proofs. Inside a proof, assumptions can be nested into other assumptions, so that the scopes of assumptions define a hierarchical proof structure.

- The language is adaptive: Definitions add new symbols and expressions to the vocabulary and fix their meaning.

- On the pragmatic level, the expectation on assertions is reversed: Assertions have to be implied by the context rather than adding new information to it.

- Proof steps are commonly justified by referring to results in other texts, or previous passages in the same text. So there is a large amount of intertextual and intratextual references (often in a standardized form).

- Furthermore, mathematical texts often contain commentaries and hints which guide the reader through the process of the proof, e.g. by indicating the method of proof ("by contradiction", "by induction") or giving analogies.

A thorough linguistic analysis of the language of mathematics can be found in Ganesalingam (2009, pp. 25-48). Below we will discuss some of the above mentioned features of the language of mathematics in more detail. Some of the theoretically interesting features that the language of mathematics shares with natural language in general will be discussed in chapter 3.

1.1.1 Mathematical vs. metamathematical content

One can distinguish two kinds of content in a mathematical text:

- The *mathematical content*, which deals with mathematical objects (e.g. numbers, functions, vestors, sets, fields, groups, topological spaces) and their mathematical properties and relationships (e.g. being even/odd, being a derivative of, being a subset of).

- The *metamathematical content*, which consists of motivating, historical, meta-theoretical or didactic comments, for example explanations about the purpose of a definition or theorem, information about who first proved a theorem and clarifications about why a certain proof method is used in a certain situation.

In this thesis, we will only be concerned with the mathematical content of mathematical texts.

1.1.2 Symbolic mathematics[2]

One of the conspicuous features of the language of mathematics is the way it integrates mathematical symbols into natural language material. The mathematical symbols are combined to *mathematical expressions*, which are often referred

[2]This section is partly taken over from Cramer, Koepke, and Schröder (2011).

to as *mathematical formulae* or *mathematical terms* depending on whether they express propositions or whether they refer to mathematical objects. We will follow the terminology proposed by Ganesalingam (2009) and call the non-symbolic parts of mathematical texts that resemble natural language *textual* parts.

As pointed out by Ganesalingam, the use of symbolic material makes it possible to state mathematical facts more concisely: For example, the statement "The square root of 2 is irrational" can be abbreviated to "$\sqrt{2}$ is irrational", which can be further abbreviated to "$\sqrt{2} \notin \mathbb{Q}$". Most mathematical statements could be rephrased without the use of symbolic mathematics. But a special problem arises with variables: If the number of entities that we talk about is small, these can be replaced by anaphoric expressions common in natural language, for example anaphoric pronouns and anaphoric definite noun phrases (noun phrases starting with "the" and referring back to a previously mentioned entity). For example, the assertion of Theorem 5.6. from the above example text could be rephrased without variables as follows:

> *Given a nontrivial vector space, a subset of this vector space is a basis if, and only if, each vector of this vector space has a unique representation as a linear combination of elements of the subset.*

However, in mathematical texts we often need to talk about a larger number of entities, and doing so unambiguously without the usage of variables is often not viable. Thus variables do not only allow for a more concise formulation of mathematical statements, but also help avoiding ambiguities.

We will now have a look at the syntax of symbolic mathematics. Already at first sight, a whole variety of syntactic rules are encountered for forming complex terms and formulae out of simpler ones; a basic classification of these was provided by Ranta (1997b):

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

As noted by Ganesalingam (2009), "this simple classification is adequate for the fragment Ranta is considering, but does not come close to capturing the breadth of symbolic material in mathematics as a whole." It does not include notations like $[K : k]$ for the degree of a field extension, it does not allow infix operators to have an internal structure, like the $*_G$ in $a *_G b$ for denoting multiplication in a group G, nor does it account for the common way of expressing multiplication by concatenation, as in $a(b + c)$.

Another kind of prefix operator not mentioned by Ranta is the one that requires its argument(s) to be bracketed, e.g. f in $f(x)$. (Of course, the argument of a prefix operator like sin might also be bracketed, but generally this is done only if the argument is complex and the brackets are needed for making sure the

term is disambiguated correctly.) This is even the standard syntax for applying functions to their arguments, in the sense that a newly defined function would be used in this way unless its definition already specifies that it should be used in another way.

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

In section 7.4, we give a more detailed and more accurate syntactic description of symbolic mathematics and describe how we solve the problem of disambiguating symbolic expressions in the Naproche system.

1.1.3 Adaptivity through definitions

Another very conspicuous feature of the language of mathematics is its *adaptivity*[3] through definitions: The language is constantly expanded through the use of definitions, which introduce new textual or symbolic expressions and fully specify their meaning. This expansion of the language should not be confused with the change of language over time: What we mean is an expansion of the language used for one particular text and – related to this – an expansion of the language in the mind of a mathematician reading such a text. Of course, some definitionally introduced expressions become commonplace for the mathematicians of a given field, and in this case one can say that the language of mathematics itself has been expanded by that expression. But in this thesis we will focus on the local expansion of language for the purpose of a text, which might or might not become commonplace for the mathematicians of that field.

The introduction of new technical terms through definitions does, of course, also exist in other specialized languages. But, as Ganesalingam (2009) has pointed out, there are two important differences between definitions in mathematics and in other fields: Firstly, mathematical definitions contain no vagueness and hence perfectly specify the semantics of the defined expression. Secondly, in advanced mathematics all newly introduced terms are introduced through definitions, and mathematicians even go back to less advanced mathematics and rigorously define all terms used there.

We can distinguish expansions of the lexicon of the textual part of the language and extensions of the symbolic part. (1) is an example of a definition expanding only the textual lexicon:

(1) **Definition 1.1.5** A set D is *dense in the reals* if every open interval (a, b) contains a member of D. (Trench, 2003, p. 6)

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

[3]The use of the term *adaptivity* for this feature of the language of mathematics is due to Ganesalingam (2009).

(2) **Definition** Suppose R is a ring and A_1, A_2, ..., A_m are ideals of R. Then the *sum* $A_1 + A_2 + \cdots + A_m$ is the set of all $a_1 + a_2 + \cdots + a_m$ with $a_i \in A_i$. (Connell, 1999, p. 108)

Ganesalingam (2009) considers the expansion of the symbolic part of the language as an expansion of the *syntax* of this symbolic part. This is certainly a very sensible interpretation at a certain level of abstraction in the understanding of the term "syntax". We, however, prefer to take a more abstract view of syntax, under which this expansion of the symbolic part of the language can be viewed as an expansion of the lexicon, just as in the case of the expansion of the lexicon of the textual part of the language. For example, under this interpretation, the definition in (2) adds a lexical item of the form "$\bullet+\bullet+\cdots+\bullet$" to the lexicon of the symbolic part of the language. The syntax of the language under this interpretation must contain rules that specify what form definitions can take, what properties the symbolic lexical items have depending on the form of the definition, and in what way these properties influence how different items of the symbolic lexicon can be combined to symbolic expressions. In this way this abstract syntax indirectly specifies how definitions change what form symbolic expressions following the definition can take. Thus this abstract syntax specifies a more concrete syntax (i.e. a *syntax* in the way Ganesalingam used the term) for every position in a mathematical text, depending on which previously stated definitions are *accessible*, i.e. may be made use of.

1.1.4 Terminological conventions

We fix the following terminology for talking about certain elements of a mathematical text:

- We use the term *sentence* for any text unit delimited using typographic means like full stops or colons (dots and colons in mathematical formulae do not count as delimiters of sentences), capitalization and font. For example, the first sentence in the text fragment quoted above is "**Definition 5.8.**", and the second sentence is "*Let $\mathcal{V}$ be a nontrivial vector space, $\mathcal{X}$ a subset of $\mathcal{V}$.*".

- The term *statement* is used for content-full sentences. So the first sentence in the above quotation is not a statement, but the second one is.

- We use the term *assertion* to refer to any statement that is neither an assumption nor a definition. For example, in the text fragment quoted above, the sentence starting with "Conversely, suppose" is not an assertion, but the following sentence starting with "Then we have" is an assertion.

- We use the term *proof text* for a mathematical text that is directed towards proving various mathematical results. We assume proof texts to consist merely of mathematical (as opposed to metamathematical) content. Besides the actual proofs, proof texts may also contain axioms, definitions and statements of the results to be proven.

1.2 Formal linguistics and formal semantics

The linguistic aspects of this thesis can be considered to be a contribution to formal linguistics. *Formal linguistics* is a branch of linguistics that uses formal methods to explain aspects of the human language capacity and of particular languages. For example, it studies formal grammars that define formal languages, with the aim of modelling natural languages using such grammars. A sub-branch of formal linguistics is *formal semantics*, which makes use of mathematical models that are intended to describe how humans determine the meaning of complex expressions based on the meanings of their parts.

Applications of formal semantics often face the following two problems:

1. The meanings of lexical items are often vague or hard to determine precisely. So even if formal semantics has very precise tools for determining the meaning of complex expressions based on the meanings of their parts, the vagueness and indeterminateness of the meanings of lexical items usually gets inherited to the complex expressions that contain them.

2. Pragmatic aspects often influence the way people interpret a given utterance in a given context. However, from a theoretical perspective it is often difficult to determine which aspects of interpretation are due to pragmatic factors and which are purely semantic, i.e. parts of the literal meaning of the utterance. Furthermore, pragmatic influences on interpretation are usually much harder to model formally than purely semantic aspects of meaning composition.

In this thesis, we apply formal semantics to the language of mathematics, and in this application of formal semantics these two problems do not arise: The meaning of the lexical items of the language of mathematics is fixed precisely through definitions in the text or through axiomatic characterization (in case of the fundamental concepts of a mathematical theory), which leave no room for vagueness or semantic indeterminateness. Additionally, as Ganesalingam has noted, mathematical texts in general do not exhibit pragmatic phenomena that lead to interpretations of expressions that deviate from their literal meanings (Ganesalingam, 2009, p. 32-33).[4] Because of this, the application of formal linguistics to the language of mathematics is on the one hand a more promising undertaking than other applications of formal linguistics, and on the other hand an interesting test-bed for testing the models of formal linguistics.

Since the work presented in this thesis has been developed in parallel with a computer system implementing the ideas (see section 1.4 below), one can view the linguistic aspects of this thesis as a contribution to *computational linguistics*, too. Computational linguistics can make use of statistical methods, of rule-based methods, or of a combination thereof. A separate methodological division of computational linguistic is that between *deep natural language processing* and *shallow natural language processing*. Deep natural language processing aims at understanding texts in a human-like way, and is hence closely linked to the linguistic endeavour to model human language capacities. For

[4] The only exception that Ganesalingam acknowledges is conditional perfection of "if" in definitions to "if and only if". This isolated example can be treated separately in a purely formal way and thus does not cause any of the more serious problems that pragmatic reinterpretations can cause in applications of formal semantics.

this it uses rule-based methods or a combination of rule-based and statistical methods. Shallow natural language processing processes natural language texts mainly with statistical methods and without deep analysis.

The work presented in this thesis uses rule-based methods for deep natural language processing. Given the nature of the problem we want to tackle, namely to verify the deductive correctness of mathematical proofs, the usage of statistical methods would be highly problematic: In the case of mathematical proofs, one does not want a 99% verification of their correctness, but a 100% verification. In the outlook in chapter 9, we will discuss how statistical methods could be made use of in a limited way without departing from the goal of 100% verification; but the completed work that we present in this thesis lacks statistical methods altogether.

1.2.1 Controlled Natural Languages

Even the language of mathematics with its high precision and tendency to avoid ambiguities is still full of expressions that are very hard – if not impossible – to disambiguate in an automatic way. If, for the reasons just mentioned, one aims at a completely error-less disambiguation, this endeavour will be impossible. But there exists an approach which harmonizes with the goals of checking mathematical proofs, namely the approach of machine-oriented controlled natural language.

The term *controlled natural language* (CNL) is used for two rather distinct categories of languages (see Schwitter, 2010): Human-oriented CNLs, which aim at improving readability for humans, and machine-oriented CNLs, which enable reliable automatic semantic analysis. For the rest of this thesis, we will always mean machine-oriented CNL when we write "CNL".

A CNL is a subset of a natural language defined through a formal grammar and with a unique formal semantics fixed for each grammatical sentence. The existing fully developed general purpose CNLs are all based on English. Prominent examples are *Attempto Controlled English* (*ACE*) by Fuchs, Höfler, Kaljurand, Rinaldi, and Schneider (2005), *Processable English* (*PENG*) by White and Schwitter (2009) and *Computer Processable Language* (*CPL*) developed at Boeing Research and Technology (see Clark, Harrison, Murray, & Thomson, 2010). Furthermore, there are specialised CNLs for specific purposes, for example for legal contracts (see Pace & Rosner, 2010) and for querying ontologies (see Damljanović, 2010). In this vein, it makes sense to develop a specialized CNL for mathematical texts, and a reasonable application for such a CNL is to check the mathematical proofs written in it for deductive correctness.

1.3 Modelling mathematical reasoning – a historic overview

In this section we give a historic overview of accounts that aim at explaining and modelling mathematical reasoning, from developments in the 19th century to contemporary work closely related to the topic of this thesis. This historic overview aims to motivate the research conducted for this thesis, to put this research in the context of a general scientific endeavour, and to introduce some ideas needed for understanding this thesis.

We use the term "to model mathematical reasoning" in a rather broad way: It includes accounts that had a more prescriptive than descriptive goal, i.e. were aimed more at prescribing what mathematical reasoning should be like than at describing what mathematical reasoning is like. There is, at any rate, a continuum between prescriptive and descriptive attitudes; and since historically early prescriptive accounts have, to some extent, actually influenced mathematical practice, they may describe current mathematical practice better than they described the mathematical practice of their time. Furthermore, the idea of "modelling" something always includes some degree of idealization, which makes the model deviate from a purely descriptive account, and which allows for a prescriptive use of the model.

1.3.1 19th-century axiomatics

For more than 2000 years, Euclid's *Elements* was the prototype of rigorous mathematical reasoning based on a small set of postulates or axioms. During the early and mid-19th century, geometry made significant advances that reshaped the subject and changed its content: Non-euclidean geometries (i.e. hyperbolic and elliptic geometry) were developed and projective geometry advanced in importance up to the point that it came to be synonymous with modern geometry (see Torretti, 2010). These developments led to a reconsideration of the role of axiomatics in geometry, first by Moritz Pasch, who made explicit that deductive reasoning must be independent of the meaning of the terms involved:

> Es muss in der That, wenn anders die Geometrie wirklich deductiv sein soll, der Process des Folgerns überall unabhängig sein vom *Sinn* der geometrischen Begriffe, wie er unabhängig sein muss von den Figuren; nur die in den benutzten Sätzen, beziehungsweise Definitionen niedergelegten *Beziehungen* zwischen den geometrischen Begriffen dürfen in Betracht kommen. Während der Deduction ist es zwar statthaft und nützlich, aber *keineswegs nöthig*, an die Bedeutung der auftretenden geometrischen Begriffe zu denken; so dass geradezu, wenn dies nöthig wird, daraus die Lückenhaftigkeit der Deduction und (wenn sich die Lücke nicht durch Abänderung des Raisonnements beseitigen lässt) die Unzulänglichkeit der als Beweismittel vorausgeschickten Sätze hervorgeht. (Pasch, 1882, p. 98)[5]

Pasch realized that Euclid's text does not actually conform with this strict understanding of the axiomatic method: He noted hidden assumptions in Euclid's reasoning and formulated axioms aimed at filling these gaps, for example the axiom now termed *Pasch's axiom*, which – informally speaking – asserts that any line that meets one side of a given triangle and does not pass through

[5] "If geometry is to be truly deductive, the process of inference must be independent in all its parts from the meaning of the geometric concepts, just as it must be independent from the diagrams. All that need be considered are the relations between the geometric concepts, recorded in the statements and definitions. In the course of deduction it is both permitted and useful to bear in mind the meaning of the geometric concepts that occur in it, but it is not at all necessary. Indeed, when it actually becomes necessary, this shows that there is a gap in the proof, and – if the gap cannot be eliminated by modifying the argument – that the premises are too weak to support it." (Translation from Torretti (2010))

any vertex of the triangle meets another side of the triangle. He did not actually provide a complete axiomatization of Euclidean geometry, but only one for projective geometry (published in Pasch, 1882), and it was left to David Hilbert to provide the first complete axiomatization of Euclidean geometry (published in Hilbert, 1899).

This rigorous understanding of the axiomatic method allowed for a mathematically precise notion of what constitutes correct geometric reasoning. Hence one can say that it constitutes a mathematical model of geometric reasoning: This model is certainly an idealization of how humans in general or mathematicians in particular reason about geometry, but it can be understood as a first approximation at understanding this reasoning with mathematical precision.

A development similar to that in geometry occurred around the same time in the fields of analysis and arithmetic: After the independent inception of the infinitesimal calculus by Gottfried Leibniz and Isaac Newton in the late 17th century, this new branch of mathematics on the one hand led to very fruitful developments and applications, but on the other hand led to serious inconsistencies resulting from working with infinitely small quantities. These inconsistencies motivated a more rigorous approach to the infinitesimal calculus, which led to the development of modern analysis: This approach – first conceived by Augustin-Louis Cauchy in the 1820s and perfected by Karl Weierstrass in the 1870s – eliminated the talk about infinitely small quantities in favour of the ε/δ-method, which only required reference to real numbers (see Volkert, 1988, pp. 206, 218). This move towards a more rigorous foundation of analysis – later coined the *arithmetization of analysis* by Felix Klein – naturally led to a critical examination of the concept of a real number: In 1872, four independent works by Georg Cantor, Richard Dedekind, Charles Méray and Weierstrass[6] expounded constructions of the real numbers from the rational numbers (see Volkert, 1988, p. 214). For example, Dedekind (1872) defined certain sets of rational numbers to be *cuts* (nowadays termed *Dedekind cuts*), and for every cut not specified by a rational number he *created* an irrational number, thus extending the system of rational numbers to the system of real numbers, which he could now prove to have the desired completeness property lacking in the system of rational numbers. From the modern point of view, these constructions involve set theory, but the practitioners of the time considered set theory to be part of logic (see Ferreirós, 2001, pp. 443-444), and could hence announce to have constructed the real numbers from the rational numbers on purely logical grounds.

Similar constructions of the integers and rational numbers from the natural numbers were already known at that time (see Reck, 2011), but Dedekind aimed to give an ultimately logical foundation to all of analysis and arithmetic by providing a similar construction of the natural numbers on purely "logical" grounds. This construction was published in 1888 in his renowned monograph *Was sind und was sollen die Zahlen?*[7] (Dedekind, 1888). Four years earlier, but until then not known to Dedekind, Gottlob Frege had published an alternative construction of the natural numbers on purely "logical" grounds (Frege, 1884). Also in Frege's work, "logic" includes what would now be termed set theory. Unlike Dedekind, Frege made the logic he used precise: In his *Begriffsschrift* (Frege, 1879), he had devised a formal language and a formal calculus that con-

[6] Weierstrass' construction was published by his pupil Ernst Kossak.

[7] "What are numbers and what should they be?"

stituted the first formalism meeting the standards of rigour of modern logic and the first theory of quantifiers.[8] In Frege (1893), he extended this formalism by a notation for sets (called *extensions* (German "Umfang") by him), and added a "logical" law, Basic Law V, corresponding to the Axiom of Extensionality in modern set-theoretic terminology. As first noted by Bertrand Russell, this extension of his formalism resulted in a contradiction, now widely known as *Russell's paradox*.

1.3.2 Type theory, first-order logic and axiomatic set theory

In order to rescue parts of Frege's and Dedekind's logicism from his paradox, Russell devised a theory of types. This was a higher-order logical formalism, still with set-theory regarded as part of logic, in which every set is typed, i.e. contains only elements of a fixed type. In his co-authored monumental work *Principia Mathematica* (Whitehead & Russell, 1910, 1912, 1913), he used this type theory to give a detailed formal account of the foundations of mathematics, more precisely of set theory including the theories of ordinal and cardinal numbers on the one hand and the theory of real numbers on the other hand. An originally planned fourth volume of *Principia Mathematica* on geometry was never completed (see Russell, 1959).

The type-theoretic logic of the *Principia Mathematica* still enabled the construction of the integers, rational numbers and real numbers from the natural numbers; but the natural numbers could no longer be constructed using purely logical means, as in Dedekind's and Frege's accounts. In order to construct them, Russell had to make the extra-logical assumption that there are infinitely many objects of the base type.

The *Principia Mathematica* for the first time showed a serious drawback of the young field of formal mathematics: A colossal amount of work was necessary to formally develop only the basics of two mathematical theories. To develop more advanced mathematical theories in such a formalism was thus outside the reach of the humanly possible.

For more than two decades after the publication of the *Principia Mathematica*, type theory (first as *ramified* type theory as presented in the *Principia Mathematica*, but after a simplification proposed by Frank Ramsey in 1925 usually in the form of *simple type theory*) dominated as the formal system studied by logicians and used for describing the foundations of mathematics (see Ferreirós, 2001, p. 445). In the 1930s, a combination of different factors contributed to a shift in logic and the foundations of mathematics (see Ferreirós, 2001): Now first-order logic got to be viewed as the paradigmatic formal system to be studied by logicians, and axiomatic set theory formalized over first-over logic became the paradigmatic system for describing the foundations of mathematics. Since this account of the foundations of mathematics is still the prevailing account today, both in mathematical logic and in the philosophy of mathematics, we will say a bit more about it.

In first-order logic[9] there is a single domain of discourse, and all quantification is quantification over this domain: There is no quantification over sets

[8] Using modern terminology, his formalism was a higher-order propositional calculus.

[9] We discuss the standard one-sorted first-order logic here. For many-sorted first-order logic some assertions would have to be rephrased, but the relevant points would stay unchanged.

of elements of the discourse domain, nor is there quantification over properties, relations or functions on the discourse domain. One can define the semantics of first-order formulae model-theoretically: First one defines what it means for a formula to be true in a given structure (a set together with some relations and functions on that set); next one defines that a set Γ of formulae *logically implies* a formula φ if all structures that make all formulae in Γ true also make φ true. There are *sound* and *complete* formal calculi for proving logical inference in first-order logic: This means that such a calculus defines a notion of *formal proof* in such a way that there exists a formal proof for φ from the premises Γ if and only if Γ logically implies φ.

The first proof systems were developed by Frege and Hilbert. These systems are axiomatic systems (also called *Hilbert systems*), i.e. they are characterized by a set of axioms and a very minimal set of inference rules. In 1934, two independent works by Gerhard Gentzen (Gentzen, 1934/35) and Stanisław Jaśkowski (Jaśkowski, 1934) presented a new kind of proof system, *natural deduction*. Natural deduction is characterized by a complete lack of axioms and a rich set of inference rules: Usually, for every logical connective and quantifier there is a set of inference rules consisting of one or more *introduction rules* for introducing the connective or quantifier and one or more *elimination rules* for eliminating it. Additionally there is a special rule for proofs by contradiction or for double negation elimination.[10] As the name suggests, natural deduction systems come closer to modelling our natural reasoning than axiomatic systems.

The standard system for axiomatic set theory over first-order logic is called *ZFC* (*Zermelo-Fraenkel set theory with the Axiom of Choice*). This is an axiomatization of pure set theory, i.e. all the objects in the domain of discourse are presupposed to be sets, all elements of these sets are also presupposed to be sets, etc. One can view the intended domain of discourse as constructed hierarchically: At the first step, one constructs the empty set $\emptyset$. Next one can construct the set $\{\emptyset\}$ that contains the empty set and nothing else. Next one can construct the sets $\{\{\emptyset\}\}$ and $\{\emptyset, \{\emptyset\}\}$. This construction can be continued *ad infinitum*. But with the usual meaning of *ad infinitum*, this would just give us finite sets. We have to go further: In the first step of going further this means that we consider the infinite construction described so far as completed, and now construct all sets consisting of sets constructed so far. This can again be continued *ad infinitum*. For every infinite construction from a given start-point, we can repeat this procedure of transcending the infinite construction. The steps in the overall construction can be described using *ordinal numbers*, an extension of the natural numbers into the realm of the infinite, first introduced by Georg Cantor in 1883. The axioms of ZFC are intended to capture this view of the *cumulative hierarchy of sets*. Using these axioms, one can formalize ordinal numbers, formalize the construction we just sketched and show that every set appears in some step of this construction.

Ordinary mathematicians do not consider all objects they are talking about to be sets. For example, an ordinary mathematician would not consider the number 2 to be a set. But when *ZFC* is used as a foundation of mathematics, all mathematical objects have to be modelled by some sets. The usual modelling of the natural numbers are the so-called *finite von-Neumann ordinals* $\emptyset$, $\{\emptyset\}$,

[10] Without this special additional rule, the resulting system is not a proof system for classical first-order logic, but one for intuitionistic logic (see Moschovakis (2010)) instead.

$\{\emptyset, \{\emptyset\}\}$, $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$, etc. All the standard machinery of pure mathematics is similarly modelled using only sets. For example, the ordered pair (a, b) of two mathematical objects is modelled by $\{\{a, b\}, \{a\}\}$. One can prove this construction to have the desired properties of the ordered pair, namely that $(a_1, b_1) = (a_2, b_2)$ if and only if $a_1 = a_2$ and $b_1 = b_2$. n-tuples for $n \geq 2$ are modelled by iterating ordered pairs, n-ary relations are modelled as sets of n-ary tuples and n-ary functions as $n+1$-ary relations satisfying a certain property that makes them *functional*.

First-order logic together with *ZFC* can be viewed as a model of what is considered correct mathematical reasoning. First-order logic covers the purely logical parts of mathematical reasoning that do not need the recourse to set-theoretic constructions. Standard set-theoretic constructions like the construction of a set of equivalence classes of a Cartesian product of two sets are easily implemented on the basis of the axioms of *ZFC*. Mathematicians often use basic mathematical structures to argue about other mathematical entities: For example, in arguments in the algebraic field of group theory, the natural numbers and their properties may be used without qualms. This is modelled without problems by *ZFC*: Since the natural numbers can be shown to exist (in their modelled form of finite von-Neumann ordinals) and to have the properties usually needed in such arguments, they can be used in such a way in any argument modelled within *ZFC*. *ZFC* is also good at modelling ordinary mathematical arguments aimed at showing that a mathematical structure with certain properties exists or cannot exist.

There are also aspects of mathematical reasoning that are not satisfactorily modelled by first-order logic together with *ZFC*:

- Since an ordinary mathematician would not identify the number 2 with the set $\{\emptyset, \{\emptyset\}\}$, the ordinary reasoning about the identity and non-identity of arbitrary mathematical objects is not correctly modelled.

- In *ZFC*, every mathematical statement has to be phrased in the language of *ZFC*, i.e. using only the symbol $\in$ for membership in a set and the logical symbols of first-order logic. Even basic set-theoretic notations like the $\emptyset$ and $\{\emptyset\}$ used above have to be translated into this form: For example, $x = \emptyset$ and $y = \{\emptyset\}$ become $\neg\exists z\ z \in x$ and $\forall w\ (w \in y \leftrightarrow \neg\exists z\ z \in w)$ respectively. This contrasts massively with the usage of language in ordinary mathematics, where definitions can be used to specify concise notation for more complex expressions. If one actually formalises mathematics in pure *ZFC*, there is a massive blow-up in the length of formulae needed to express simple mathematical statements. So first-order logic with *ZFC* does not model the language of mathematics very well.

- As we will see in chapter 3, the language of mathematics also exhibits many traits of natural language that linguists have only begun to describe in the second half of the 20th century, like dynamic quantifiers and presuppositions, which – as we try to show in this thesis – intimately influence ordinary mathematical reasoning, but which are completely ignored in first-order logic and *ZFC*.

- Furthermore, there is also a massive blow-up in the number of proof steps needed: Standard calculi for first-order logic are very fine-grained, i.e.

require very small logical steps to be made explicit. This contrasts with the very flexible size of reasoning steps found in textbooks proofs, which depends *inter alia* on the mathematical sophistication that the author assumes on the side of the targeted readership. First-order logic with *ZFC* thus cannot serve to model the size of reasoning steps usually employed by mathematicians.

1.3.3 Computer-assisted formal mathematics

Formal mathematics is a branch of mathematics that aims at developing substantive parts of mathematics in a purely formal way. *Principia Mathematica* can be considered the first comprehensive work in formal mathematics. As mentioned above, it also showed that formal mathematics is a too extensive programme to be completed by humans without the assistance of computers. For this reason this programme was not pursued seriously by the scientific community before the advent of computers. The issues with this *manual* formal mathematics can be divided in two parts:

- **Manual formalization:** To formalize an existing piece of mathematics involves a huge amount of straining and largely monotonous intellectual work: All details of the proofs have to be filled in, and everything has to be expressed in a severely limited formal language which is highly dissimilar to the natural language that we usually use to think and communicate about mathematics.

- **Manual checking:** In order to be checked for correctness, the prepared texts have to be read by humans, who find it difficult to follow the reasoning of overly detailed proofs written in an unnatural formal language. Thus the goal of making mathematical results more secure through formalizing them was of a more theoretical than practical nature: In practice, errors could be overlooked more easily in these formal texts that humans find hard to read than in usual mathematical texts.

The advent of computers drastically changed the landscape for formal mathematics. The second one of these two problems faced by *manual* formal mathematics can easily be seen to be solvable using computers: After all, in a formal logical system, the checking of proofs is a purely syntactical procedure that can be described algorithmically and hence implemented on a computer. In this way, the motivation for formal mathematics that it could make mathematical results more secure actually became a practical motivation rather than a purely theoretical one.

Automath

The *Automath* system by Nicolas Govert de Bruijn (first described in de Bruijn, 1968) was the first computer system for formal mathematics with automated proof checking (see Kamareddine, Laan, & Nederpelt, 2004, p. 179). De Bruijn analysed the way mathematicians reason and use their specialized language. Based on this analysis, he developed the formal language and formal system of Automath, whose goal it was to represent the usual reasoning in mathematical texts:

> The way mathematical material is to be presented to the system should correspond to the usual way we write mathematics. The only thing to be added should be details that are usually omitted in standard mathematics. (de Bruijn, 1994, p. 210).

Automath had a very type-theoretic approach and thus differed substantially from the by then common usage of axiomatic set theory over first-order logic as a foundation of mathematics. The Automath formalism introduced new approaches and notions that have led to significant advances in type theory and that have been taken over by later type-theoretically based systems for formal mathematics:

- The usage of dependent types in the Automath formalism was the first systematic development of dependent type theory (see Abramsky, Artemov, Shore, & Troelstra, 1999, p. 582).
- De Bruijn discovered the Curry-Howard correspondence (also known as the *propositions-as-types* interpretation, or – as de Bruijn preferred – the *proof-classes-as-types* interpretation) independently of Haskell Curry and William Alvin Howard (see Kamareddine et al., 2004), and Automath was the first implemented system to employ this correspondence.
- At the core of the Automath formalism is a definition system, so that Automath captures much of the natural usage of definitions in mathematical texts. This contributes to the attainment of the goal mentioned in the above quotation of de Bruijn, and contrasts with the absence of any treatment of definitions within the standard formalisms for axiomatic set theory: There definitions usually have to be treated as something meta-theoretical, and within the theory all defined terms have to be considered as replaced by their respective definienses.

According to the philosophy of the Automath project, the system was "tied as little as possible to any particular set of rules for logic and foundation of mathematics" (de Bruijn, 1994, pp. 209-210): Even the logical connectives and their introduction and elimination rules had to be introduced axiomatically by the user of the system.

As a proof of concept, the book *Grundlagen der Analysis* by Edmund Landau (Landau, 1930) was completely formalized and proof checked in the Automath system. This book is characterized by a very pure mathematical style (clearly structured axioms, definitions, theorems and proofs and an almost complete lack of motivating, historical, meta-theoretical or didactic comments) and a high degree of logical self-containment. We also use it as a test-bed for our system (see chapter 8).

While the Automath formalism certainly captured many aspects of mathematical reasoning better than any previous formalism, its language was quite detached both from the usual language employed by mathematicians as well as from standard formal languages studied by logicians. As an example of the Automath language, we present a very simple *Automath book* in figure 1.1. This unusual language certainly contributed to Automath's very limited practical use.

Automath only solved the problem of *manual checking* mentioned above. De Bruijn certainly made an effort to define the Automath formalism in such a

∅	prop	PN	type
∅	x	–	prop
x	y	–	prop
x,y	and	PN	prop
x	proof	PN	type
x,y	px	–	proof(x)
x,y,px	py	–	proof(y)
x,y,px,py	and-I	PN	proof(and)
x,y	pxy	–	proof(and)
x,y,pxy	and-01	PN	proof(x)
x,y,pxy	and-02	PN	proof(y)
x	prx	–	proof(x)
x,prx	and-R	and-I(x,x,prx,prx)	proof(and(x,x))
x,y,pxy	and-S	and-I(y,x,and-02,and-01)	proof(and(x,y))

Figure 1.1: Example (taken over from Kamareddine et al., 2004, p. 187) of a very simple Automath book, in which logical conjunction (`and(x,y)`) is introduced and the logical entailments from `x` to `and(x,x)` and from `and(x,y)` to `and(y,x)` are proved.

way as to not make the problem of *manual formalization* bigger than necessary, but his system did not implement any computer assistance for alleviating this problem.

Mizar

The *Mizar* project is a project for computer-assisted formal mathematics initiated by Andrzej Trybulec in 1973 (see Matuszewski & Rudnicki, 2005, p. 3). Until 1989 the project was characterized by a perpetual development of new systems improving on previous ones based on practical experience with attempts at formalizations. Since 1989, the *Mizar system* has been a more or less stable system with occasional extensions and improvements, and the Mizar project has mainly advanced through the creation of the *Mizar Mathematical Library*, the largest library of formal mathematics of any single system (see Wiedijk, 2009, p. 194). It is this more or less stable version of the Mizar system that we will describe in this section.

The Mizar language is much closer to the language of informal mathematics than the Automath language: It uses a set of English words and phrases which frequently appear in informal mathematics as keywords, but its syntax is defined by a relatively small set of rules, in a similar vein as in modern programming languages. Here is an example (taken over from Wiedijk, 2008) of a Mizar text presenting a proof of the irrationality of $\sqrt{2}$:

```
theorem
  sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 then consider i being Integer, n being Nat such that
W1: n<>0 and
W2: sqrt 2=i/n and
```

```
W3: for i1 being Integer, n1 being Nat st n1<>0
 & sqrt 2=i1/n1 holds n<=n1 by RAT_1:25;
A5: i=sqrt 2*n by W1,XCMPLX_1:88,W2;
C: sqrt 2>=0 & n>0 by W1,NAT_1:19,SQUARE_1:93;
 then i>=0 by A5,REAL_2:121;
 then reconsider m = i as Nat by INT_1:16;
A6: m*m = n*n*(sqrt 2*sqrt 2) by A5
  .= n*n*(sqrt 2)^2 by SQUARE_1:def 3
  .= 2*(n*n) by SQUARE_1:def 4;
 then 2 divides m*m by NAT_1:def 3;
then 2 divides m by INT_2:44,NEWTON:98;
 then consider m1 being Nat such that
W4: m=2*m1 by NAT_1:def 3;
 m1*m1*2*2 = m1*(m1*2)*2
    .= 2*(n*n) by W4,A6,XCMPLX_1:4;imp
 then 2*(m1*m1) = n*n by XCMPLX_1:5;
 then 2 divides n*n by NAT_1:def 3;
 then 2 divides n by INT_2:44,NEWTON:98;
 then consider n1 being Nat such that
W5: n=2*n1 by NAT_1:def 3;
A10: m1/n1 = sqrt 2 by W4,W5,XCMPLX_1:92,W2;
A11: n1>0 by W5,C,REAL_2:123;
  then 2*n1>1*n1 by REAL_2:199;
 hence contradiction by A10,W5,A11,W3;
end;
```

Apart from the usage of English keywords, another important aspect to make the Mizar language more similar to the language of informal mathematics than standard formal languages is the possibility of operators to be used in prefix, postfix, infix and circumfix notation. But as the above example shows, despite significant improvements on Automath, the Mizar language is still much more similar to a programming language than to the language of informal mathematics. As a comparison, here is a natural language proof of the same theorem, taken from Hardy and Wright (1960, p. 40):

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

Mizar lacks the usual notation for square roots and exponentiation, and does not allow multiplication to be expressed in the usual way by concatenation. If one reads a Mizar text with its English keywords as if it were a natural language text, one finds that it is full of ungrammatical constructs.

The Mizar system has an integrated proof-checker that can check simple multi-step logical inferences. This alleviates the problem of *manual formalization* to some extent, since some intermediate reasoning steps may be dropped. However, the granularity of reasoning steps required by Mizar is still much more detailed than that usually found in mathematical textbooks.

The extensive Mizar Mathematical Library contains material from various branches of mathematics based on a single system of axioms, the Tarski-

Grothendieck axiomatization of set theory. This set theory is basically the standard *ZFC* set theory extended with Tarski's axiom, which ensures that there are arbitrarily large strongly inaccessible cardinals (see Naumowicz & Korniłowicz, 2009, p. 70).[11] As of August 2012 (version 4.181.1147), the library contains 1150 articles by 244 authors, containing 51,762 theorems and 10,158 definitions (see Bancerek, 2012). As pointed out by Naumowicz and Korniłowicz (2009, p. 70), most of these theorems are – from a point of view of a mathematician – actually simple lemmas. Nevertheless, they cover not only the basics of many branches of pure mathematics, but also some advanced theorems, especially in topology (see Naumowicz & Korniłowicz, 2009, p. 70). Results from the Mizar Mathematical Library can be cited and reused in other Mizar texts, so that authors of Mizar texts don't have to start their formalization from first principles.

HOL

HOL is an interactive proof assistant both for checking the correctness of hardware and software and for checking the correctness of mathematical proofs, developed in the 1980s by Mike Gordon (see Wiedijk, 2009, p. 194). *Interactive* means that the user and the system co-construct the proof incrementally. For example, the user may present a certain *goal* (a result to be proved) to the system, and specify some *tactics* to be used by the system to simplify that goal into simpler goals. The system then performs this simplification and shows the user which simplified goals he still needs to establish to establish his original goal. This is a form of *backward reasoning*; HOL can combine such backward reasoning with the classical *forward reasoning* found in the already described systems.

This interactive approach helps to alleviate the problem of manual formalization mentioned above: The formalization process is no longer completely manual, but supported by a computer system.

HOL stands for *Higher-Order Logic*: The logical foundation on which it is based are those of simply typed higher-order logic, i.e. basically the logic that evolved out of Principia Mathematica's logic through the de-ramification proposed by Frank Ramsey in 1925 (see section 1.3.2 above).

There are various implementations of HOL. The one with the largest library of formalized mathematics among the HOL implementations is called *HOL Light* (see Wiedijk, 2009, p. 195). There is currently an ongoing project to formalize Thomas Hales's proof of Kepler's conjecture in HOL Light (see Hales, 2005). This shows that modern proof assistants are mature enough to tackle the formalization of current research mathematics.

HOL is used widely for proving the correctness of hardware and software. In this vein, the system is adapted to the expertise and needs of computer scientists, and does not intend to imitate the reasoning and language of informal mathematics closely.

[11] This strengthening of *ZFC* was motivated by certain constructions done in category theory (see Matuszewski & Rudnicki, 2005, p. 22).

Isabelle

Isabelle is an interactive proof assistant developed by Larry Paulson, Tobias Nipkow and Makarius Wenzel as a successor of HOL. The two main differences between Isabelle and HOL are succinctly stated by Wiedijk (2009):

> An important difference between Isabelle and HOL is that Isabelle did not hardwire the mathematical foundations into the system, but keeps it as a parameter of the system. (However, the HOL implementation on top of Isabelle, called Isabelle/HOL, is the only variant of the system that is significantly used.) Another difference between Isabelle and HOL is that Isabelle has a readable proof language inspired by the Mizar language called Isar.

Just like HOL, Isabelle is developed and used both for checking the correctness of hardware and software and for checking the correctness of mathematical proofs.

Coq

Similarly to HOL and Isabelle, *Coq* is an interactive proof assistant for both software correctness proofs and mathematical proofs. It is based on a rich type theory called *the calculus of inductive constructions*. The logic of Coq is modular: The core logic is intuitionistic, i.e. lacks the principle of excluded middle, but this principle can be added as an axiom. Another principle that is often needed when using Coq for mathematical proofs is the Axiom of Choice (in a functional variant), which can also be added as an axiom.

In 2004, Georges Gonthier completed a formalization of the Four Colour Theorem in Coq, which Wiedijk (2009) calls "the most impressive formalization thus far". For this formalization, Gonthier developed an extension library for Coq called *SSReflect*, which provides for an improved language for tactics and for effective automation of small proof steps.

The type-theoretic language of Coq is even more removed from the language of informal mathematics than the languages of HOL and Isabelle.

Automated theorem provers

In this subsection we describe computer programs of a somewhat different nature than the formal mathematics systems described in the previous subsections. *Automated theorem provers* (*ATPs*) are programs aimed at automatically finding proofs, rather than checking proofs provided by humans. Because of their unguided proof search, the problems they can solve are of a much simpler nature than the proofs that can be checked by formal mathematics systems. But ATPs can actually be a support tool for formal mathematics systems in that they can be used to fill in the logical gaps left by the human author in a proof provided to a formal mathematics system.

We use the term *proof obligation* for a problem given to an ATP. It usually consists of a finite list of *axioms* and a single *conjecture* which has to be shown to follow from the axioms. In this thesis, we will deviate from the standard terminology and speak of *premises* instead of *axioms* in order to avoid terminological clash with the axioms that are found in the mathematical texts that we study. The premises and the conjecture are all formulae of a fixed logical

formalism. The most powerful ATPs are those for standard first-order predicate logic. These are the only ones we will be considering in this thesis.

Many ATPs can not only prove the conjecture from the premises, but also discover counterexamples in case that the conjecture does not follow from the premises. Normally, an ATP is given a time limit for solving a given proof obligation. Hence there are three possible outputs from an ATP: *conjecture proven*, *counterexample found* and *time-out*.

1.3.4 Modelling the natural language of mathematics

One of the deficits that the systems for computer-assisted formal mathematics described in the previous section have in common, although to different extents, is the unnaturality of their input language. Users of such systems are forced to learn a formal input language that resembles programming languages. Also for reading the proofs written for such systems one needs a good acquaintance with this formal language.

One way to put the problem is to say that even though the input languages of these systems model the expressive power of the language of mathematics fairly well, they model the structure of this language very badly. We will now discuss the work of various researchers that have attempted to model the structure of the language of mathematics. Some of these have analysed the language of mathematics linguistically, while others have developed systems for checking proofs in more natural input languages.

Simon's Nthchecker

The first system for checking mathematical proofs written in a natural input language was Donald Simon's *Nthchecker*, described in Simon's (1990) Ph.D. thesis. It was geared towards the text of William LeVeque's *Elementary Theory of Numbers* (LeVeque, 1962). Nthchecker could parse 15 of the 65 proofs in this book and mechanically check two of them. Given that the input to Nthchecker was the actual text from LeVeque's book and not some CNL adaptation thereof, this might seem like an astonishing success. However, this is only due to the system's being geared towards that particular text. Simon did not carry out a proper linguistic analysis of the language of mathematics that would enable his system to be capable of parsing other input texts. As Claus Zinn has put it:

> From the linguistic point of view, Simon falls short of studying, first of all, the language of mathematics and second, the linguistic problems one encounters in textbook proofs. The main critique, however, is that Simon did not use or develop an adequate theory for the construction of semantic representations. It remains unclear how Simon handles anaphoric resolution and ellipsis reconstruction systematically. (Zinn, 2004, p. 25)

Ranta

The first thorough analysis of the language of mathematics using techniques from formal linguistics was provided by Aarne Ranta in a number of papers from

the mid-1990s (Ranta, 1994, 1995, 1996, 1997a, 1997b). Ranta analysed syntactic categories of both symbolic and textual mathematics within the framework of Constructive Type Theory (Martin-Löf, 1984). In the practical application of his analysis, he put a larger emphasis on the conversion of logical representation into the natural language of mathematics than on the conversion in the other direction that is needed in proof-checking mathematical texts.

Ranta was also the first researcher to express the idea that mathematical language is a rewarding test-bed for linguistics:

> Linguistically, the study of mathematical language rather than everyday language is rewarding because it offers examples that have complicated grammatical structure but are free from ambiguities. We always know exactly what a sentence means, and there is a determinate structure to be revealed. The informal language of mathematics thus provides a kind of grammatical laboratory. (Ranta, 1994, p. 354)

This idea has also been one of the motivations behind the Naproche project.

Zinn' Vip

For his doctoral project, Claus Zinn developed the system *Vip*, a prototypical proof-checker for natural language mathematics (see Zinn, 2004). He analysed the language of mathematics more thoroughly than Simon, and made use of more advanced linguistic techniques, especially Hans Kamp's Discourse Representation Theory. Just like Simon, he did not define a CNL, but attempted to parse any input written in the language of mathematics. As a consequence, he also had to gear his system towards a single text, for which he chose *An Introduction to the Theory of Numbers* by Hardy and Wright (1960). Coincidentally, the number of proofs that Vip could both parse and check successfully was two, just as with Simon's Nthchecker. Ganesalingam (2009, p. 20) has argued that not only Zinn's system, but also his linguistic analysis is "heavily tailored for his two proofs", and that it is "of a comparably shallow kind".

Zinn's thesis also provided a detailed analysis of reasoning patterns in mathematical proofs. In his system, the proof-checking was heavily reliant on Alan Bundy's concept of *proof plans* (see Bundy, 1988). The idea is that one analyses families of related proofs in order to identify common reasoning patterns in them, which are formally represented in *proof plan schemata*. These guide future proofs of the same family, and enable an automated system to fill in the details that the proof author has omitted. The heavy reliance on proof plans, however, has increased the extent to which the system is tailored towards particular proofs.

Evidence Algorithm (SAD/ForTheL)

In the early 1960s, the pioneering Soviet computer scientist Victor Glushkov initiated a research project called *Evidence Algorithm* in Kiev. The goal of the project was to develop a computer system that could check mathematical proofs written in a powerful input language that is close to the natural mathematical language and easy to use (see Lyaletski & Verchinine, 2010, p. 412). By 1978 the group had produced a first prototype for such a system, called

System for Automated Deduction (*SAD*), with a Russian-based input language. After improvements to the system in the early 1980s, the project came to a halt for over a decade, before it was reactivated in 1998 (see Lyaletski & Verchinine, 2010, p. 412). The SAD system was reimplemented by Andrei Paskevich, then a master student at Kiev and from 2001 a doctoral student at Kiev and Paris. In the course of the reimplementation, Paskevich developed an alternative English-based input language called *ForTheL*, an acronym for "FORmal THEory Language". During the course of his doctoral studies, the system was constantly improved. Below we describe his final version of the SAD system, as presented in Paskevych (2007).

We start our discussion of the input language ForTheL with an example text, namely the simplest example text provided on the web interface of SAD (Lyaletski, Verchinine, & Paskevich, 2008b):

```
[set/-s] [element/-s] [belong/-s] [subset/-s]

Signature SetSort.  A set is a notion.
Let S,T denote sets.

Signature ElmSort.  An element of S is a notion.
Let x belongs to X stand for x is an element of X.

Definition DefSubset.   A subset of S is a set T
    such that every element of T belongs to S.

Definition DefEmpty.    S is empty iff S has no elements.

Axiom ExEmpty.  There exists an empty set.

Proposition.
    S is a subset of every set iff S is empty.
Proof.
    Case S is empty. Obvious.

    Case S is a subset of every set.
        Take an empty set E.
        Let z be an element of S.
        Then z is an element of E.
        We have a contradiction.
    end.
qed.
```

It is immediately evident that this input language is much more natural than the input language of Mizar, which was the most natural input language for a formal mathematics system discussed so far. In the example text, all lines but the first read more or less like the natural language of mathematics. Hence one could be inclined to conclude that ForTheL is basically a CNL, only with some unnatural elements as in the first line. However, it should be noted that the system ignores a lot of grammatical features of natural language, and hence accepts input that is ungrammatical from a natural language

point of view. For example, the assertion of the proposition can be replaced by "`S are subset of every sets iff S be empty.`", which the system still accepts in the same way as the above grammatically sound variant.[12] Thus, even ignoring the unnatural parts like the first line, ForTheL is not really a subset of the natural language of mathematics, but rather a superset of such a subset. But a human user writing in ForTheL would normally stay inside the subset of the natural language of mathematics. We think that this kind of input language should be terminologically distinguished from *controlled natural languages* as defined before; for this we propose the term *potentially natural controlled language*.

Furthermore, there are expression which – though natural in character – are very unusual for the language of mathematics: "`A set is a notion.`" and "`An element of S is a notion.`". These expressions are used to extend the signature, i.e. to introduce new relation, function and constant symbols to the language, which are not defined but which can be axiomatically specified. Without knowledge of this special role of these expressions in ForTheL, one can hardly make sense of these sentences based on ones ability to understand the language of mathematics. So these are expressions which semantically speaking are not part of the natural language of mathematics, and hence must be counted amongst the unnatural part of ForTheL.

Having looked at the input language of SAD, let us now turn our attention to the system as a whole: The SAD system is up to now the most successful system for producing automatically checkable formal mathematics that can be read by humans almost like natural mathematical texts. Examples of more advanced results formalised in SAD include the Chinese Remainder Theorem, the result that square roots of prime numbers are irrational, the Cauchy-Bouniakowsky-Schwartz inequality for real vectors and Furstenberg's topological proof of the infinitude of primes (see Lyaletski, Verchinine, & Paskevich, 2008a). Each of these is proved in a text about eight times as long as the above example text. One of the drawbacks of the SAD system is that beyond this text length the system becomes very inefficient due to quadratic runtime complexity.

Given an input text, the SAD system parses the individual sentences in it, transforming them into first-order formulae. Furthermore, the system keeps track of the position and function of these first-order formulae in the overall proof structure, defined by the keywords that structure the proof text, like "`Proposition`", "`Proof`", "`Case`", "`end`" and "`qed`". Next, a module called the *reasoner* works linearly through the proof, trying to verify proof steps and keeping track of what is still required to finish a proof. For this, the reasoner combines a number of methods:

- It keeps track of a large list of *evidences*, which are literals (atomic formulae or their negations) which can be derived from earlier propositions or proof steps according to certain rules. Simple proof steps can often be verified using only this list of evidences.
- If this method fails, the reasoner will call an Automated Theorem Prover to verify the proof step. The ATP attempts to prove that this proof step follows from previous propositions and proof steps.

[12] Actually, the unnatural first line provides rules for identifying morphological variants of certain words. These words are identified without any further checking of natural-language grammaticality.

- When definitions as well as propositions whose logical form are that of an implication or bi-implication are used for proving subsequent statements, a special heuristic called *definitional expansion* is applied: In order to avoid an increase of the proof search tree, these definitions and implications are often not given to the ATP as axioms, but instead are used to modify the problem given to the ATP, in the case of definitions by *expanding* the definiendum to the definiens. The ATP is given subsequent reformulations of a problem which are expanded in this way, until it succeeds in verifying the proof step (or until it gives up due to time-out).
- Given a proposition followed by its proof, the reasoner attempts to keep track of a *goal-oriented provingthesis*, i.e. keep track of what is still needed in order to finish the proof of the proposition. At the beginning of the proof, the thesis coincides with the proposition. But the introduction of assumptions, the opening of case distinctions and the verification of intermediate proof steps can modify and simplify the thesis up to the point where it becomes trivial.[13]

Using this combination of methods, SAD can handle logical gaps between proof steps of a similar size as found in textbooks for undergraduate mathematics students. Note that some of these methods make use of natural patterns of structuring proof texts. Thus, notwithstanding the lack of proper grammatical analysis in SAD, the Evidence Algorithm project has certainly contributed significantly to a better understanding of the natural reasoning in mathematical texts.

Ganesalingam

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

One of the aspects of the language of mathematics that Ganesalingam has studied in detail and has shed light on is the aspect of adaptivity through definitions discussed in section 1.1.3 above. He has also linked this aspect to philosophical inquiries about the foundations of mathematics. In his theoretical description of the language of mathematics he intentionally stretches the adaptivity to its extremes. For this purpose, he deviates from a purely descriptive account of the language of mathematics, and speaks of the *projected language*, which can be viewed as an idealised version of the actual language of mathematics with full adaptivity down to the language used for the foundational building blocks of mathematics.

The issue that Ganesalingam studies most thoroughly is that of disambiguating mathematical language, both its textual and its symbolic parts. To handle

[13] We call this approach *goal-oriented proving* in later chapters.

potential ambiguities, he develops an ingenious novel type system for typing the objects that a given mathematical text refers to. The potential types available at any point in a text are extracted from preceding definitions. We will come back to his type system when we compare it to our approach in section 7.4.7.

Ganesalingam has been working on an implementation of the theoretical developments from his thesis in a computer system, which he finished in late 2012.

Humayoun

In January 2012, Muhammad Humayoun completed a Ph.D. thesis (Humayoun, 2012) with very similar goals to our thesis: It is part of a project that aims at creating a computer program that can validate mathematical proof texts written in a mathematical controlled natural language. However, his thesis leaves out the validation aspect and concentrates on the specification of the CNL and its translation into a formal representation language. Humayoun's linguistic analysis is linguistically not very sophisticated. For example, he intentionally refrains from making use of linguistic theories for modelling the dynamic nature of natural language quantification (see Humayoun, 2012, p. 26). Large parts of his thesis are on implementationary details of the program he developed.

1.4 The Naproche project

The research for this thesis was conducted as part of the interdisciplinary research project *Naproche* (*Natural language Proof Checking* or *Natural Proof Checking*), which has previously been described in Cramer, Fisseni, et al. (2010) and Cramer (2011). In this section, we give a chronological overview over the development of this project.

The project started at the University of Bonn in 2002 as a collaboration between the mathematical logician Peter Koepke and the linguist Bernhard Schröder. It aims at combining methods from linguistics and mathematical logic in order to improve our understanding of natural mathematical texts and the proofs contained in them. From the beginning, one major driving force for the research conducted in the Naproche project was the vision of a computer system that supports the development of formal mathematics in a much more natural language than current state-of-the-art formal mathematics computer systems. The *Naproche system* can be seen as a prototypical version of this envisioned computer system.

The first version of the Naproche system – retrospectively termed Naproche 0.1 – was developed by Peter Koepke in the years 2002-2006. It implemented a natural deduction calculus with natural language quantifiers and connectives as well as natural language keywords for structuring the proof text (see Koepke, 2006). The proof text could be structured by theorem-proof-blocks and by the introduction and retraction of assumptions, which gives rise to a hierarchical text structure as described in section 1.1. Naproche 0.1 could be used as a plug-in to the WYSIWYG mathematical editor *TeXmacs* (see van der Hoeven, 2011). The core of Naproche 0.1, like that of all later versions of the Naproche system, was programmed in *Prolog*, a declarative programming language for logic programming (see Blackburn, Bos, & Striegnitz, 2006).

A further focus in the early days of the Naproche project was the development of an annotation language for mathematical texts, called ProofML. The *Magister* thesis of Bernhard Fisseni, a student of linguistics, describes this annotation language (Fisseni, 2003).

In 2007, Peter Koepke, his Ph.D. students Merlin Carl and Jip Veldman, Bernhard Schröder, his *Magister* student Nickolay Kolev and Bernhard Fisseni developed *Proof Representation Structures* (*PRSs*), a semantic representation for mathematical proof texts.[14] The PRS format is based on Discourse Representation Theory, a theory developed by Hans Kamp in order to model the dynamic nature of natural language quantification[15] in formal semantics (see Kamp & Reyle, 1993). The structures that Discourse Representation Theory uses as semantic representation of multi-sentence natural language discourses are called *Discourse Representation Structures* (*DRSs*). PRSs are DRSs which are enriched in such a way as to represent the distinguishing characteristics of the language of mathematics discussed in Section 1.1.

Nickolay Kolev implemented an algorithm to generate PRSs from input texts written in a rudimentary mathematical CNL, which was basically the input language of Naproche 0.1 extended by a construct for definitions (see Kolev, 2008).

In 2008, the first version of the Naproche system that implemented Proof Representation Structures (Naproche 0.2) was developed by two diploma students of Peter Koepke, Daniel Kühlwein and Doerthe Arndt, with the technical support of two internees, Bhoomija Ranjan and Shruti Gupta. The generation of PRSs from input text was a modified version of Kolev's algorithm. The proof-checking was now performed on the PRSs. Naproche 0.2 was also the first version of Naproche to make use of an automated theorem prover (ATP) in the proof checking: The ATP helps to fill in gaps between subsequent reasoning steps in a proof (see section 6.1.1). Naproche 0.2 did not have a GUI (graphical user interface), i.e. could only be run on the command line. Kühlwein's diploma thesis (Kühlwein, 2009) describes the proof checking implemented in Naproche 0.2. Arndt's diploma thesis (Arndt, 2009) describes software verification methods suitable for software implemented in Prolog and their applicability to the code of Naproche 0.2.

In September 2008, this thesis's author joined the Naproche project as a doctoral student. After finishing his diploma, Daniel Kühlwein also became a doctoral student of the Naproche project in January 2009. Starting in March 2009, Kühlwein and I[16] implemented Naproche 0.3, with the technical support of the internees Mona Rahn and John Schmid. Naproche 0.3 was the first version of Naproche to implement sophisticated portions of textual mathematics, i.e. of natural language in mathematical texts, into the Naproche CNL. Attempto Controlled English served as a model for choosing principles of natural language disambiguation, and to some extent also as a model for implementing the module that parses Naproche CNL input and builds a semantic representation from it.

[14] Even though the terms "Proof Representation Structure" and "PRS" were already used by Zinn (2004), his definition of "PRS" is substantially different from the one developed within the Naproche project.

[15] We will describe this dynamic nature of natural language quantification in section 3.1.

[16] In this section I avoid the *pluralis auctoris* used in the rest of the thesis, in order to make clear which contributions were my own and which ones were the collective work of several members of the Naproche group.

After the first significant improvements to Naproche 0.3, we changed the version number to 0.4 in October 2009. Further improvements were made constantly until June 2010, raising the version number to 0.47. The theoretically most interesting and technically most involved improvement during that time was the introduction of definite descriptions starting with "the", which trigger existence and uniqueness *presuppositions* (see section 3.2), which in the proof checking have to be treated in a different way than existence and uniqueness *assertions* (see section 6.1.3).

Since the TeXmacs interface of Naproche 0.1 was hard to maintain with advances in the core of the system, we decided to develop a new interface for Naproche 0.3, namely a web interface that could be run in a web browser. In this way, we could also make Naproche available as a web application, sparing users the need to install it on their own computers. The web interface was originally developed by Daniel Kühlwein for Naproche 0.3, and significantly improved by Sebastian Zittermann, a computer science master student of Gregor Büchel from the Cologne University of Applied Sciences, in Naproche 0.4 up to 0.45.

In the development of Naproche 0.3 and 0.4x, Kühlwein and I had the following division of labour: I developed the Naproche CNL and implemented the linguistic module that built PRSs from input CNL text, and Kühlwein implemented the *logic module* that checked whether a PRS represents a logically sound proof text.

The main text on which the Naproche system has been tested since version number 0.3 is the beginning of Landau's *Grundlagen der Analysis*. More on this test-bed for the Naproche system can be found in chapter 8. Starting in January 2009, Richard Schüller, a diploma student of Peter Koepke, started to work on a Naproche CNL version of Euclid's *Elements*, with an axiomatization of Euclidean geometry based on the system E by Avigad, Dean, and Mumma (2009). This second test-bed for the Naproche system contained much more varied usage of textual mathematics, which motivated two improved semantic interpretation principles for the textual part of the Naproche CNL:

- I introduced a disambiguation between distributive and collective readings of plurals based on the common usage of plurals in mathematical texts; see Cramer and Schröder (2012) or section 7.6 of this thesis for further details.

- I improved the treatment of quantifiers in bi-implications and reversed implications; see section 7.5.9 for further details.

In August 2010, we considered three partly interdependent possible improvements of the Naproche systems:

- I planned to make the *formula grammar*, i.e. the grammar of the symbolic part of the Naproche CNL, more flexible. Before then, the system predefined for each symbol whether it should be parsed as a variable, as a constant symbol, as a function symbol or as a relation symbol. In mathematical texts, however, the author can decide in which of these syntactical roles to use a symbol by introducing the symbol – for example through a definition – in a way which explains the syntactical role of the symbol to the reader.

- Related to this, I planned to implement in the Naproche CNL the phenomenon of *implicit dynamic introduction of function symbols* prevalent in the language of mathematics and described in section 3.3.

- Furthermore, motivated by our work with longer texts, especially with the Euclid text mentioned above, we planned to implement the *macro-grammatical parser*, i.e. the module that parses the text structure above the sentence level, as an *incremental parser*, so that an already parsed text that gets extended or modified does no need to be parsed and checked again from the beginning, but only from the sentence containing the first modification.

We realized that each of these possible improvements, especially the first and the third, involved the development of a significant amount of new code and significant modifications to existing code. Hence it made sense to tackle these problems at once, and we started to use the version number 0.5 for the envisioned system implementing these improvements. We further realized that it would be hard to combine the incremental macro-grammatical parser with the web interface, and hence decided to implement a new GUI for Naproche in Java. The development of the GUI was carried out by Sebastian Zittermann as part of his master thesis project (Zittermann, 2011).

In December 2010, Daniel Kühlwein left the Naproche project in order to pursue a Ph.D. on automated reasoning in Nijmegen. This meant that in the development of Naproche 0.5, I had to take over the logic module previously developed by Kühlwein. Apart from Zittermann's support for the GUI, I had the technical support of Julian Schlöder and Johannes Seela for the development of Naproche 0.5. Naproche 0.5 got released in January 2012. In March 2012 we implemented some further improvements, especially improving the runtime, released as Naproche 0.51. In January 2013 I fixed a few bugs, giving rise to Naproche 0.52, the current version of the Naproche system.

In April 2011, Torsten Nahm became an external doctoral student of Peter Koepke in the Naproche project. He has been investigating the SAD system by Paskevich with the goal of combining achievements of Naproche and SAD in a single system.

1.5 Modularity of the developed theory

In this thesis, we will develop and intertwine several independent ideas. For example, we will develop a certain foundational theory and certain techniques for treating phenomena of the natural language of mathematics. When presenting these ideas in this thesis, we will give a special focus to the way they can be linked. On the other hand, the reader should be aware of the fact that nevertheless the theory is modular in the sense that most ideas could work just as well if completely different solutions for other discussed problems are chosen. For example, the linguistic techniques developed could be built on a different foundation theory than the one developed in this thesis, and the foundation theory could be used for other purposes as well.

1.6 Thesis outline

In this thesis, we define a CNL for mathematics and a proof checking algorithm for checking the deductive correctness of proof texts written in this CNL. The CNL and proof checking algorithm defined agree largely with what is implemented in Naproche 0.52, with some significant differences which are discussed in appendix C. The most significant difference should already be mentioned at this point: In our theoretical description in this thesis we do not refer to the PRSs (Proof Representation Structures) used in the implementation. Instead of PRSs, which are an extended versions of DRSs from Discourse Representation Theory, our theoretical description will employ an extended version of *Dynamic Predicate Logic* (*DPL*). Just like Discourse Representation Theory, Dynamic Predicate Logic is a formal system aimed at capturing the dynamic nature of natural language quantification. But unlike Discourse Representation Theory, it has a close syntactical resemblence to standard systems of first-order predicate logic.

The focus of the thesis is on the interaction between natural language phenomena and proof-checking. A special emphasis is put on one particular logico-linguistic phenomenon peculiar to the language of mathematics but, to our knowledge, previously not described by other logicians or linguists, which we termed *implicit dynamic function introduction*. In the outline that follows, we say a bit more about how this phenomenon connects to the different themes discussed in this thesis.

In chapter 2, we fix some notational and terminological conventions used in this thesis. Chapter 3 presents methods from formal semantics that are required in this thesis, with a special focus of their application to the language of mathematics. In the final section of chapter 3, section 3.3, we discuss implicit dynamic function introduction. We show how this function introduction can lead to a paradox analogous to Russell's paradox. Chapter 4 describes a foundational theory of functions equiconsistent to ZFC, which can be used for imposing limitations to implicit dynamic function introduction in order to avoid this paradox. Furthermore, section 4.3 describes a related but richer foundational theory, which does not only have functions, but also sets, tuples, numbers and Booleans as primitives.

In chapter 5, we define two formalisms, *Higher-Order Dynamic Predicate Logic* (*HODPL*) and *Proof Text Logic* (*PTL*), that extend *DPL* in order to capture implicit dynamic function introduction and to serve as a formal counterpart to the CNL to be defined in chapter 7. The definitions of their semantics require the foundational theories from chapter 4. In chapter 6, we motivate and define the proof checking algorithm as an algorithm for checking the correctness of *PTL* texts, and prove soundness and completeness theorems for this proof checking algorithm. In chapter 7, we finally define the Naproche CNL and specify its semantics by defining a translation from the CNL to *PTL*. This translation together with the proof checking algorithm from chapter 6 defines a proof checking algorithm for CNL texts.

Chapter 8 presents a case study, namely the application of the theory developed in the previous chapters to the beginnings of Landau's *Grundlagen der Analysis*. Chapter 9 concludes the thesis and provides an outlook to further research that could extend the research conducted for this thesis.

There are four appendices: Appendix A provides a complete formal grammar

of the Naproche CNL. Appendix B contains the Naproche CNL adaptation of the first chapter of Landau's *Grundlagen der Analysis*. In appendix C, we discuss the differences between the theory presented in this thesis and what is implemented in Naproche 0.52. Appendix D is a concise manual for Naproche 0.52.

Chapter 2

Notation and terminology

In this short chapter we fix some notational and terminological conventions used in this thesis. These are mainly needed for the most mathematical parts of the thesis, i.e. for chapter 4 and sections 6.3 and 6.4.

We assume that the reader is familiar with the standard notation used in basic mathematical logic and set theory, as found for example in Ebbinghaus, Flum, and Thomas (2007) and Hrbacek and Jech (1999).

Following the convention from the literature on Dynamic Predicate Logic (*DPL*), we use *PL* (for *Predicate Logic*) as an abbreviation for the standard system of classical first-order predicate logic with equality. *PL* is assumed to have the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$, the logical constants $\top$ and $\bot$ and the quantifiers $\exists$ and $\forall$. We follow the usual convention of operator priorities in order to drop superfluous brackets: $\wedge$ and $\vee$ bind stronger than $\rightarrow$ and $\leftrightarrow$. For example, $\varphi_1 \wedge \varphi_2 \rightarrow \psi_1 \vee \psi_2$ is shorthand for $((\varphi_1 \wedge \varphi_2) \rightarrow (\psi_1 \vee \psi_2))$. Also in the logics that we define in the thesis we follow these conventions for dropping superfluous brackets and enhance readability.

Both in *PL* and in the logics we define in the thesis, we – for the sake of readability – sometimes write relation symbols in infix notation (e.g. $x \in y$ instead of $\in (x, y)$), even when this notation has not been formally defined.

As usual in logic texts, we use $\bar{t}$ as an abbreviation for $t_1, \ldots, t_n$ when the length n of this term list is either clear from the context or not relevant. Additionally, $\bar{t}$ can also be an abbreviation for the tuple $(t_1, \ldots, t_n)$. We use the symbol $^\frown$ for the adjoinment of an element to a tuple: $(t_1, \ldots, t_n)^\frown t_{n+1} := (t_1, \ldots, t_n, t_{n+1})$.

We assume familiarity with the standard definition of $\varphi\frac{t}{x}$, the result of substituting the term t for all free occurrences of the variable x in φ, with renaming of bound variables to avoid binding of variables in t. We need an extension of this definition: Substitution has to be defined so as to allow term substitution too: $\varphi\frac{t_1}{t_0}$ means that all occurrences of t_0 are replaced by t_1.

In chapter 6, we extensively talk about finite sequences, also called *lists*. We use the notation $\langle x_1, \ldots, x_n \rangle$ for the finite sequence of elements $x_1, \ldots, x_n$ in this order. Finite sequences could be identified with tuples, but for clarity we prefer to use a separate notation for finite sequences. The idea is that we use tuples in contexts where the length of the tuple is fixed in advance, whereas finite sequences are used in contexts where their length can vary. In order to conveniently talk about finite sequences, we use some set-like notation for

them: For example, given an element x and a finite sequence Γ, we write $x \in \Gamma$ to mean that there is at least one occurrence of x in Γ. We use notations like $\langle x \in \Gamma \mid \varphi(x) \rangle$ and $\langle f(x) \mid f \in \Gamma \rangle$ to build new finite sequences from given ones. Noting that the original list Γ can be used to define the multiplicity and order of the elements in the newly built list, this notation explains itself based on the analogous notations for sets. Additionally, we use the notation $\Gamma_1 \oplus \Gamma_2$ for the concatenation of Γ_1 and Γ_2. If Γ_1 is of the form $\langle x_1, \ldots, x_n, y_1, \ldots, y_m \rangle$ and $\Gamma_2 = \langle x_1, \ldots, x_n \rangle$, then $\Gamma_1 - \Gamma_2$ denotes $\langle y_1, \ldots, y_m \rangle$. If each of $x_1, \ldots, x_n$ occurs only once in Γ, then $\Gamma \setminus \langle x_1, \ldots, x_n \rangle$ is defined to be Γ with the occurrences of $x_1, \ldots, x_n$ deleted and with its other elements kept in the original order.

In chapters 5 and 6, we also talk about (finite) multisets. These resemble finite sequences in that elements can occur more than once in them, and they resemble sets in that the order of the elements does not matter. We freely use either set or list notation for them.

As is usual in linguistic literature, we prefix example sentences that are considered ungrammatical with an asterisk (*). In case the grammaticality of the example sentence is dubious, we use a superscript question mark ($^?$) instead.

Chapter 3

Linguistic foundations of Naproche

In the first two sections of this chapter, we introduce the linguistic machinery required in this thesis. In the third and final section, we describe a logico-linguistic phenomenon peculiar to the language of mathematics, namely the implicit dymanic function introduction.

3.1 Dynamic Predicate Logic

Our formal treatment of mathematical reasoning and our controlled natural language for mathematical texts will be based on an extension of *Dynamic Predicate Logic* (*DPL*, see Groenendijk & Stokhof, 1991). *DPL* is a logical system for formalizing some of the dynamic features of natural language. Its syntax is identical to that of standard first-order predicate logic (*PL*), but its semantics is defined in such a way that the dynamic nature of natural language quantification is captured in the formalism. Consider the following example sentence and formulae:

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

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

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

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

[1] This example sentence is one of a number of standard examples from the linguistic literature about dynamic quantification, which are usually called *donkey sentences*. Donkey sentences were originally introduced by Geach (1962).

The natural language quantification used in mathematical texts also exhibits these dynamic features, as can be seen in the following quotation from Hatcher (2002, p. 36):

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

Since much of this thesis is about extensions of *DPL*, readers without any acquaintance of *DPL* are advised to read at least the first two sections of Groenendijk and Stokhof's paper. Below we will give a complete formal definition of the syntax and semantics of the variant of *DPL* used in this thesis. We will also say some words to motivate this formal definition; but these brief motivative comments cannot replace the much more detailed motivation presented in Groenendijk and Stokhof's paper.

DPL is based on the tenet that the meaning of a sentence is not determined by its truth conditions, but by its *information change potential*, i.e. by the way the sentence changes the information available to a person interpreting it (Groenendijk & Stokhof, 1991, p. 43). This is what makes it a *dynamic* rather than a *static* theory of meaning. However, in *DPL* only one aspect of information is treated dynamically, namely the information that determines which antecedents are available for subsequent anaphors.

PL can be considered a static theory of meaning: The meaning of a formula in *PL* is characterized by a set of variable assignments, namely the set of assignments that make the formula true. In *DPL*, on the other hand, the meaning of a formula determines how that formula can change variable assignments. Hence, the meaning of a formula does not just determine which assignments make that formula true, but also how these assignments are changed into other assignments by the formula. So in *DPL*, the meaning of a formula is its *assignment change potential*.

After these motivative comments, let us now define *DPL* formally. We define *DPL* syntax as follows: We fix a signature consisting of constant symbols, function symbols of fixed arity and relation symbols of fixed arity. As variables we may use any small Latin letters not reserved by the signature, possibly with a numerical subscript. *DPL* terms and formulae are defined recursively as follows:

A *DPL* term is either a variable, a constant symbol or of the form $f(t_1, \ldots, t_n)$ for *DPL* terms $t_1, \ldots, t_n$ and an n-ary function symbol f.

A *DPL* formula is of one of the following forms, where $t_1, \ldots, t_n$ are *DPL* terms, R is an n-ary relation symbol and φ and ψ are *DPL* formulae:

- $\top$
- $t_1 = t_2$
- $R(t_1, \ldots, t_n)$
- $\neg\varphi$
- $(\varphi \wedge \psi)$
- $(\varphi \vee \psi)$
- $(\varphi \to \psi)$

- $\exists x\ \varphi$
- $\Diamond\varphi$[2]

We present *DPL* semantics in a way slightly different but logically equivalent to its definition in Groenendijk and Stokhof (1991). Structures and assignments are defined as for *PL*:

Definition 3.1.1. A *structure* S is a pair (D, F), where D is a non-empty set of individuals, called the *domain* of S, and F is a map such that

- for every constant symbol c, $F(c) \in D$,
- for every n-ary function symbol f, $F(f)$ is a function from D^n to D,
- for every n-ary relation symbol R, $F(R) \subseteq D^n$.

Definition 3.1.2. Given a structure $S = (D, F)$, an *S-assignment* is a function from the set of variables to D. G_S is the set of S-assignments.

Remark. We usually use g, h, k and j to refer to assignments.

Definition 3.1.3. Given a *DPL* term t, a structure $S = (D, F)$ and an S-assignment g, we recursively define

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

Definition 3.1.4. Given two assignments g, h, we define $g[x_1, \ldots, x_n]h$ to mean that g differs from h at most in what it assigns to the variables x_1, …, x_n.

Groenendijk and Stokhof (1991) define an interpretation function $[\![\bullet]\!]_S$ from *DPL* formulae to subsets of $G_S \times G_S$. We instead define for every $g \in G_S$ an interpretation function $[\![\bullet]\!]^g_S$ from *DPL* formulae to subsets of G_S. [3]

Definition 3.1.5. Given a structure $S = (D, F)$ and an S-assignment g, we define the interpretation function $[\![\bullet]\!]^g_S \subseteq G_S$ recursively as follows:

1. $[\![\top]\!]^g_S := \{g\}$
2. $[\![t_1 = t_2]\!]^g_S := \begin{cases} \{g\} & \text{if } \frac{S}{g}(t_1) = \frac{S}{g}(t_2) \\ \emptyset & \text{otherwise} \end{cases}$
3. $[\![R(t_1, \ldots, t_n)]\!]^g_S := \begin{cases} \{g\} & \text{if } (\frac{S}{g}(t_1), \ldots, \frac{S}{g}(t_n)) \in F(R) \\ \emptyset & \text{otherwise} \end{cases}$
4. $[\![\neg\varphi]\!]^g_S := \begin{cases} \{g\} & \text{if there is no } h \text{ such that } h \in [\![\varphi]\!]^g_S \\ \emptyset & \text{otherwise} \end{cases}$

[2] The formula $\Diamond\varphi$ captures the truth-conditions of φ while blocking the binding power of existential quantifiers in φ for variables outside $\Diamond\varphi$ (see Groenendijk & Stokhof, 1991, p. 22). It is equivalent to $\neg\neg\varphi$.

[3] This can be viewed as a different currying of the uncurried version of Groenendijk and Stokhof's interpretation function.

5. $[\![\varphi \wedge \psi]\!]^g_S := \{h \mid \text{there is a } k \text{ such that } k \in [\![\varphi]\!]^g_S \text{ and } h \in [\![\psi]\!]^k_S\}$

6. $[\![\varphi \vee \psi]\!]^g_S := \begin{cases} \{g\} & \text{if there is there is an } h \text{ such that } h \in [\![\varphi]\!]^g_S \text{ or } h \in [\![\psi]\!]^g_S \\ \emptyset & \text{otherwise} \end{cases}$

7. $[\![\varphi \rightarrow \psi]\!]^g_S := \begin{cases} \{g\} & \text{if for all } k \text{ such that } k \in [\![\varphi]\!]^g_S, \text{ there is a } j \text{ such that } \\ & j \in [\![\psi]\!]^k_S \\ \emptyset & \text{otherwise} \end{cases}$

8. $[\![\exists x\ \varphi]\!]^g_S := \{h \mid \text{there is a } k \text{ such that } k[x]g \text{ and } h \in [\![\varphi]\!]^k_S\}$

9. $[\![\Diamond \varphi]\!]^g_S := \begin{cases} \{g\} & \text{if there is an } h \text{ such that } h \in [\![\varphi]\!]^g_S \\ \emptyset & \text{otherwise} \end{cases}$

The idea of Definition 3.1.5 is that the meaning of a formula φ is modelled as an assignment change potential: $[\![\varphi]\!]^g_S$ is the set of all assignments that can be the result of applying the assignment change potential of φ to g. We consider φ to be true if and only if $[\![\varphi]\!]^g_S$ is non-empty.

Let us now see how this definition of *DPL* semantics works in the case of formula (3) from the beginning of this section. Fix a structure $S = (D, F)$. By item 7 of Definition 3.1.5, $[\![\exists x\ (\mathit{farmer}(x) \wedge \exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y))) \rightarrow \mathit{beats}(x,y)]\!]^g_S$ can only be $\{g\}$ or $\emptyset$. Let us see under which conditions it is $\{g\}$, i.e. under which conditions (3) is true. This is precisely if for all k such that $k \in [\![\exists x\ (\mathit{farmer}(x) \wedge \exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y)))]\!]^g_S$, there is a j such that $j \in [\![\mathit{beats}(x,y)]\!]^k_S$. Now by item 3 of Definition 3.1.5, $j \in [\![\mathit{beats}(x,y)]\!]^k_S$ iff $(k(x), k(y)) \in F(\mathit{beats})$. Furthermore,

$k \in [\![\exists x\ (\mathit{farmer}(x) \wedge \exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y)))]\!]^g_S$
iff there is an h such that $h[x]g$ and $k \in [\![\mathit{farmer}(x) \wedge \exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y))]\!]^h_S$
iff there are h, h' such that $h[x]g$, $h' \in [\![\mathit{farmer}(x)]\!]^h_S$ and $k \in [\![\exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y))]\!]^{h'}_S$
iff there is an h such that $h[x]g$, $h(x) \in F(\mathit{farmer})$ and $k \in [\![\exists y\ (\mathit{donkey}(y) \wedge \mathit{owns}(x,y))]\!]^h_S$
iff there are h, h'' such that $h[x]g$, $h''[y]h$, $h(x) \in F(\mathit{farmer})$ and $k \in [\![\mathit{donkey}(y) \wedge \mathit{owns}(x,y)]\!]^{h''}_S$
iff there is an h such that $h[x]g$, $k[y]h$, $h(x) \in F(\mathit{farmer})$, $k(y) \in F(\mathit{donkey})$ and $(k(x), k(y)) \in F(\mathit{owns})$
iff $k[x,y]g$, $k(x) \in F(\mathit{farmer})$, $k(y) \in F(\mathit{donkey})$ and $(k(x), k(y)) \in F(\mathit{owns})$.

So according to *DPL* semantics, (3) is true if and only if for all k such that $k[x,y]g$, $k(x) \in F(\mathit{farmer})$, $k(y) \in F(\mathit{donkey})$ and $(k(x), k(y)) \in F(\mathit{owns})$, we have $(k(x), k(y)) \in F(\mathit{beats})$. These are precisely the intended truth conditions for (3).

We use $\forall x\ \varphi$ as shorthand for $(\exists x\ \top \rightarrow \varphi)$. Given this definition, the semantics of $\forall$ turns out to be exactly the semantics that it has by definition in Groenendijk and Stokhof (1991).[4]

[4]The reason for not having $\forall$ as primitive is that when we extend *DPL* to Higher-Order Dynamic Predicate Logic in chapter 5, the semantics of $\rightarrow$ is rather involved, and with a primitive $\forall$ we would have this involved definition repeated twice over.

We need two more definitions related to *DPL* semantics:

Definition 3.1.6. For *DPL* formulae $\varphi_1, \ldots, \varphi_n, \psi$, we define $\varphi_1, \ldots, \varphi_n \models \psi$ iff for all structures S and S-assignments $g_0, \ldots, g_n$ such that $g_i \in [\![\varphi_i]\!]_S^{g_{i-1}}$ for $1 \leq i \leq n$, there is an S-assignment h such that $h \in [\![\psi]\!]_S^{g_n}$.

Definition 3.1.7. A *DPL* formula φ is called a *tautology* iff $\models \varphi$.

3.1.1 Scope and binding

A distinctive feature of *DPL* is that it allows existential quantifiers to bind variables outside their scope. Groenendijk and Stokhof (1991, pp. 58-59) give a syntactic characterization of when an occurrence of a variable is bound by an occurrence of a quantifier. They do this by simultaneously defining three syntactical notions. We take over their definitions with only some minor notational modifications.

First we informally explain the three notions to be defined: The first one is that of the *set of binding pairs in* φ, denoted $\mathbf{bp}(\varphi)$. A binding pair consists of a quantifier occurrence and a variable occurrence such that the first binds the second. The second notion is that of the *set of active quantifier occurrences in* φ, denoted $\mathbf{aq}(\varphi)$. An occurrence of a quantifier is active if it has the potential to bind occurrences of the corresponding variable further on. The third notion is that of the *set of free occurrences of variables in* φ, denoted $\mathbf{fv}(\varphi)$. A variable occurrence is free if it not bound by any quantifier. Just as Groenendijk and Stokhof (1991), we refrain from explicitly introducing a notation for occurrences, which makes the formal definition a bit sloppy.

Definition 3.1.8. We define **bp**, **aq** and **fv** by simultaneous recursion as follows:

1. $\mathbf{bp}(R(t_1, \ldots, t_n)) := \emptyset$
   $\mathbf{aq}(R(t_1, \ldots, t_n)) := \emptyset$
   $\mathbf{fv}(R(t_1, \ldots, t_n)) := \{x \mid x \text{ is a variable occuring in } t_i \text{ for some } 1 \leq i \leq n\}$

2. $\mathbf{bp}(\neg\varphi) := \mathbf{bp}(\varphi)$
   $\mathbf{aq}(\neg\varphi) := \emptyset$
   $\mathbf{fv}(\neg\varphi) := \mathbf{fv}(\varphi)$

3. $\mathbf{bp}(\varphi \wedge \psi) := \mathbf{bp}(\varphi) \cup \mathbf{bp}(\psi) \cup \{(\exists x, x) \mid \exists x \in \mathbf{aq}(\varphi) \text{ and } x \in \mathbf{fv}(\psi)\}$
   $\mathbf{aq}(\varphi \wedge \psi) := \mathbf{aq}(\psi) \cup \{\exists x \in \mathbf{aq}(\varphi) \mid \exists x \notin \mathbf{aq}(\psi)\}$
   $\mathbf{fv}(\varphi \wedge \psi) := \mathbf{fv}(\varphi) \cup \{x \in \mathbf{fv}(\psi) \mid \exists x \notin \mathbf{aq}(\varphi)\}$

4. $\mathbf{bp}(\varphi \vee \psi) := \mathbf{bp}(\varphi) \cup \mathbf{bp}(\psi)$
   $\mathbf{aq}(\varphi \vee \psi) := \emptyset$
   $\mathbf{fv}(\varphi \vee \psi) := \mathbf{fv}(\varphi) \cup \mathbf{fv}(\psi)$

5. $\mathbf{bp}(\varphi \to \psi) := \mathbf{bp}(\varphi) \cup \mathbf{bp}(\psi) \cup \{(\exists x, x) \mid \exists x \in \mathbf{aq}(\varphi) \text{ and } x \in \mathbf{fv}(\psi)\}$
   $\mathbf{aq}(\varphi \to \psi) := \emptyset$
   $\mathbf{fv}(\varphi \to \psi) := \mathbf{fv}(\varphi) \cup \{x \in \mathbf{fv}(\psi) \mid \exists x \notin \mathbf{aq}(\varphi)\}$

6. $\mathbf{bp}(\exists x\ \varphi) := \mathbf{bp}(\varphi) \cup \{(\exists x, x) \mid x \in \mathbf{fv}(\varphi)\}$
   $\mathbf{aq}(\exists x\ \varphi) := \begin{cases} \mathbf{aq}(\varphi) \cup \{\exists x\} & \text{if } \exists x \notin \mathbf{aq}(\varphi) \\ \mathbf{aq}(\varphi) & \text{otherwise} \end{cases}$
   $\mathbf{fv}(\exists x\ \varphi) := \mathbf{fv}(\varphi)$ minus the occurrences of x in φ

7. $\mathbf{bp}(\Diamond\varphi) := \mathbf{bp}(\varphi)$
   $\mathbf{aq}(\Diamond\varphi) := \emptyset$
   $\mathbf{fv}(\Diamond\varphi) := \mathbf{fv}(\varphi)$

The **aq** function defined above only formalizes which quantifiers are active at the end of a given *DPL* formula. But it also makes sense to ask which quantifiers are active at a given position inside a formula. For this, we first need to formalize what we mean by a position in a formula:

Definition 3.1.9. Given a *DPL* formula φ, we call an occurrence of an atomic formula in φ a *position in* φ.

Definition 3.1.10. Given a *DPL* formula φ, an occurrence $\exists x$ of a quantifier in φ and a position p in φ, we say that $\exists x$ is an *active quantifier at position* p iff the formula φ' resulting from φ by placing x in an argument position at p has the binding pair $(\exists x, x)$, where the first element in this pair is the occurrence of $\exists x$ in question and the second element in this pair is the occurrence of x that we have added at position p.

3.2 Presuppositions

Loosely speaking, a *presupposition* of some utterance is an implicit assumption that is taken for granted when making the utterance. In the literature, presuppositions are generally accepted to be triggered by certain lexical items called *presupposition triggers*. Among them are definite noun phrases (in English marked by the definite article "the", possessive pronouns or genitives), factive verbs (like "regret", "realize" and "know"), change of state verbs ("stop" and "begin"), iteratives ("again") and some others.

In mathematical texts, most of the presupposition triggers discussed in the linguistic literature, e.g. factive verbs, change of state verbs and iteratives, are not very common or even completely absent. Definite noun phrases, however, do appear in mathematical texts as presupposition triggers (e.g. "the smallest natural number n such that $n^2 - 1$ is prime"). And there is another kind of presupposition trigger, which does not exist outside mathematical texts: Function symbols. For example, the division symbol "/" presupposes that its second (right hand) argument is non-zero; and in a context where one is working only with real and not with complex numbers, the square root symbol "$\sqrt{}$" presupposes that its argument is non-negative.

Presupposition projection is the way in which presuppositions triggered by expressions within the scope of some operator have to be evaluated outside this scope. Consider for example the following three sentences:

(4) The king has a son.

(5) The king's son is bald.

(6) If the king has a son, the king's son is bald.

If we restrict our attention to existential presuppositions triggered by definite noun phrases, we see that (4) and (6) presuppose that there is a king and (5) presupposes that there is a king and that the king has a son. So (6) inherits the existential presupposition of (4), which is identical to one of the two existential presuppositions of (5), but does not inherit the other existential presupposition of (5). The precise way in which presuppositions project under various operators has been disputed at great length in the literature (see for example Levinson (1983) and Kadmon (2001) for overviews of this dispute). Our formal treatment of presuppositions in mathematical texts turns out to have equivalent predictions about presupposition projection to Irene Heim's (1983) approach to presuppositions, which we briefly describe in section 3.2.3 below.

Presupposition accommodation is what we do if we find ourselves faced with a presupposition the truth of which we cannot establish in the given context: We add the presupposition to the context, in order to be able to process the sentence that presupposes it. For example, if I say "My wife is a philosopher" to someone who does not know that I have a wife, that person will accommodate the fact that I have a wife, i.e. add this presupposition to the context in which he interprets the sentence.

3.2.1 Definite descriptions[5]

Although terminology is not used in a fully uniform fashion among linguists, we will make the following distinctions suitable for our purposes. We analyse noun phrases syntactically into a determiner (here: "the") and a restricting property. We call definite noun phrases referring to a single object by a restricting property whose extension contains exactly one object *definite descriptions*. Definite noun phrases in the singular with restricting properties whose extension contains more than one object get their referential uniqueness usually by anaphoric reference to an object mentioned previously; they are called *anaphoric definite noun phrases*. A mathematical example of an anaphoric definite noun phrase is "the group" used to refer to a group mentioned recently in the text. The example above ("the smallest natural number n such that $n^2 - 1$ is prime") was an example of a definite description.

The presupposition of a singular definite description with the restricting property F is that there is a unique object with property F. This presupposition can be divided into two separate presuppositions: One existential presupposition, claiming that there is at least one F, and one uniqueness presupposition, claiming that there is at most one F.

3.2.2 Presuppositional information in definitions

As mentioned in section 1.1.3, definitions can be used to introduce new textual or symbolic lexical items and fix their meaning. Implicitly, definitions also fix which presuppositions are triggered by the newly introduced lexical items. Reconsider the following definition already cited in section 1.1.3:

(1) **Definition 1.1.5** A set D is *dense in the reals* if every open interval (a, b) contains a member of D. (Trench, 2003, p. 6)

[5]This section is largely taken over from Cramer, Kühlwein, and Schröder (2010).

Based on this definition, the lexical item "dense in the reals" triggers the presupposition that its subject must be a set. The fact that the setness of the subject is a presupposition and not part of what "dense in the reals" asserts can be seen from the fact that one cannot legitimately write "D is not dense in the real" when D is not even a set.

In the same vein, we find definitions of symbolic lexical items which make the newly introduced symbolic lexical item trigger presuppositions:

(7) **Definition 2.1.10** Suppose that f is bounded on $[a, x_0)$, where x_0 may be finite or ∞. For $a \leq x < x_0$, define

$$S_f(x; x_0) = \sup_{x \leq t < x_0} f(t)$$

and

$$I_f(x; x_0) = \inf_{x \leq t < x_0} f(t).$$

(Trench, 2003, p. 47)

Based on this definition, the symbolic expressions $S_f(x; x_0)$ and $I_f(x; x_0)$ trigger the presuppositions that f is bounded on $[a, x_0)$ and that $a \leq x < x_0$.

In general, definitions which define relations or functions introduce some variables which function as arguments of the defined relation or function. The conditions imposed on these variables before the actual definition is stated become presuppositions that can be triggered by the lexical item introduced for the defined relation or function.

3.2.3 Heim's approach to presuppositions

For the purpose of explaining the observable presupposition projection features of various operators in a unified way, Heim (1983) gives an account of the meanings of sentences in terms of their context change potential (CCP). More formally, the CCP of a sentence s is a function that maps a context c to a context $c + s$. As a first approximation to what contexts are, Heim identifies them with propositions, i.e. with sets of possible worlds, but later refines her account of contexts, identifying them with sets of pairs (g, w), where g is a sequence of individuals and w is a possible world. We can still extract a proposition from such a set of pairs: Given a set of sequence-world pairs c, the corresponding proposition is $\{w \mid \text{for some } g,\ (g, w) \in c\}$.

For comparing this account of contexts with our *DPL*-based approach to natural-language semantics, one can identify contexts with sets of pairs (g, S), where S is a structure and g is an S-assignment. The CCPs that Heim gives to sentences with operators like "if" and "every" correspond naturally to our above definition of the semantics of the corresponding *DPL* operators ($\rightarrow$ and $\forall$): Given a context c (in the sense of a set of pairs (g, S), where S is a structure and g is an S-assignment) and a *DPL* formula φ, $c + \varphi$ can be defined to be $\{(g', S) \mid \text{for some } (g, S) \in c,\ g' \in [\![\varphi]\!]^g_S\}$.

Heim reduces presupposition to a related notion, namely that of a given context *admitting* a given sentence. A sentence s presupposes a proposition p iff all contexts that admit s entail p. Heim characterizes admittance formally as follows: A context c admits a sentence s iff $c + s$ is defined. So the function $c \mapsto c + s$ must be viewed as a partial function not defined on all contexts.

Given our above characterization of $c \mapsto c + s$ in terms of $[\![\bullet]\!]^g_S$, we can give the following equivalent characterization of presupposition:[6] A formula φ presupposes a formula ψ iff for every structure S and every S-assignment g, if $[\![\varphi]\!]^g_S$ is defined, then $[\![\psi]\!]^g_S \neq \emptyset$. Of course $[\![\bullet]\!]^g_S$ now also has to be viewed as a partial function.

Given this formal apparatus, let us see how it helps to clarify the existential presuppositions of (6). By the definition of $[\![\varphi \to \psi]\!]^g_S$,

$$\left[\!\!\left[\begin{array}{l}\text{“If the king has a son,}\\ \text{the king's son is bald.”}\end{array}\right]\!\!\right]^g_S = \begin{cases}\{g\} & \text{if for all } k \text{ such that } k \in [\![\text{“The king has a son.”}]\!]^g_S, [\![\text{“The king's son is bald”}]\!]^k_S \neq \emptyset\\ \emptyset & \text{otherwise.}\end{cases}$$

So $[\![$“If the king has a son, the king's son is bald.”$]\!]^g_S$ is defined iff $[\![$“The king has a son.”$]\!]^g_S$ is defined and for all $k \in [\![$“The king has a son.”$]\!]^g_S$, $[\![$“The king's son is bald”$]\!]^k_S$ is defined. Restricting our attention to existential presuppositions, this means that $[\![$“If the king has a son, the king's son is bald.”$]\!]^g_S$ is defined iff there is a king and if the king has a son, the king has a son. The second conjunct is trivial, so that we can say that $[\![$“If the king has a son, the king's son is bald.”$]\!]^g_S$ is defined iff there is a king. This way we have explained why (6) inherits one of the existential presuppositions of its immediate constituents but not the other.

Heim's approach to presuppositions allows for two different kinds of presupposition accommodation, *global* and *local* accommodation (see Heim, 1983, p. 401). Global accommodation is the process of altering the global context in such a way that the presupposition in question can be justified; local accommodation on the other hand involves only altering some local context, leaving the global context untouched. Consider for example the following sentence, uttered in a context compatible with France not having a king:

(8) Mary did not see the king of France.

The definite description here is within the scope of the negation. So if we add the existence to the local context within the scope of the negation, it gets negated too, resulting in the interpretation that either there is no king of France or Mary did not see the king of France. Alternatively, we can add the presupposition to the global context, i.e. assume that there is a king of France. Heim (1983, p. 401) postulates that *ceteris paribus* global accommodation is preferred over local accommodation.

In section 5.1 of chapter 5 we present a formalism that extends *DPL* and formalizes presuppositions in the way described here, but without accommodation. In section 3.2.4 below we describe the particularities of presupposition accommodation in mathematical texts, and in section 7.5.10 of chapter 7 we sketch a possible treatment of accommodation in our CNL.

3.2.4 Accommodation in mathematical texts[7]

For the sake of simplicity, we identify contexts with propositions, i.e. with sets of possible worlds, in this section. In section 1.1, we mentioned the pragmatic

[6] Note that the difference between talking about natural-language sentences and talking about formulae that represent the content of natural-language sentences is not relevant for the points being made here.

[7] This section is largely taken over from Cramer, Kühlwein, and Schröder (2010).

principle in mathematical texts that new assertions do not add new information (in the sense of logically not inferable information) to the context. When mathematicians state axioms, they limit the context, i.e. the set of possible worlds they consider, to the set where the axioms hold. Similarly, when they make local assumptions, they temporarily limit the context. But when making assertions, these assertions are thought be logically implied by what has been assumed and proved so far, so they do not further limit the context.

The modification of the context in the case of local assumptions is certainly a modification of the local context. For the sake of giving a unified treatment, it is useful to view the modification of the context in the case of axioms also as a modification of the local context, only that the mathemtician is planning to stay in this locally modified context for the rest of the text. With this understading of local as opposed to global contexts, one may succinctly state the pragmatic principle mentioned above in terms of contexts as follows: In a mathematical text, the global context may not be altered.

This pragmatic principle implies that global accommodation is not possible in mathematical texts, since global accommodation implies adding something new to the global context. Local accommodation, on the other hand, is allowed, and does occur in real mathematical texts:

> Suppose that f has n derivatives at x_0 and n is the smallest positive integer such that $f^{(n)}(x_0) \neq 0$.
>
> (Trench, 2003, p. 102)

This is a local assumption. The projected existential presupposition of the definite description "the smallest positive integer such that $f^{(n)}(x_0) \neq 0$" is that for any function f with some derivatives at some point x_0, there is a smallest positive integer n such that $f^{(n)}(x_0) \neq 0$. Now this is not valid in real analysis, and we cannot just assume that it holds using global accommodation. Instead, we make use of local accommodation, thus adding the accommodated fact that there is a smallest such integer for f to the assumptions that we make about f with this sentence.

The fact that one has to accommodate locally rather than globally does not, however, always fix which context we alter when accommodating. Consider for example sentence (9), used in a context where we have already defined a set A_x of real numbers for every real number x.

(9) For all $x \in \mathbb{R}$, if A_x does not contain $\frac{1}{x}$, then A_x is finite.

The question is whether we need to check the finiteness of A_0 in order to establish the truth of (9), or whether the finiteness of A_0 is irrelevant. Since the use of $\frac{1}{x}$ presupposes that $x \neq 0$, which does not hold for any arbitrary $x \in \mathbb{R}$, we have to locally accommodate that $x \neq 0$. But we can either accommodate this within the scope of the negation or outside the scope of the negation, but still locally within the conditional. In the first case, we have to establish that A_0 is finite, whereas in the second case we don't.

3.3 Implicit dynamic function introduction[8]

Functions are often dynamically introduced in an implicit way in mathematical texts. For example, Trench (2003, p. 1) introduces the additive inverse function on the reals as follows:

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

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

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

(11) Since f is continuous at t, there is an open interval I_t containing t such that $|f(x) - f(t)| < 1$ if $x \in I_t \cap [a, b]$. (Trench, 2003, p. 62)

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

(12) Hence for each $u \in \mathbf{R}^n$ there is a number $f(u) \in \mathbf{C}$ with $f(u) \neq 0$ such that
$(\sigma(\alpha(u))^3, \sigma(\alpha(u))\ \Sigma(\alpha(u)), T(\alpha(u))) = f(u)(x_1(u), x_2(u), x_3(u))$.
The function f is locally a quotient of continuous functions, so it is itself continuous. (Bonk, 1992, p. 489)

(13) Suppose that, for each vertex v of K, there is a vertex $g(v)$ of L such that $f(st_K(v)) \subset st_L(g(v))$. Then g is a simplicial map $V(K) \to V(L)$, and $|g| \backsimeq f$. (Lackenby, 2008, p. 19)

(14) Since the multi-map Φ^{-1} is surjective, for every $x \in X$ there is a point $f(x) \in Y$ with $x \in \Phi^{-1}(f(x))$, which is equivalent to $f(x) \in \Phi(x)$. It follows from the bornologity of Φ that the map $f : X \to Y$ is bornologous. (Banakh & Zarichnyy, 2008, p. 5)

When no uniqueness claims are made about the object locally introduced to the discourse, implicit function introduction presupposes the existence of a choice function, i.e. presupposes the Axiom of Choice. We hypothesize that the naturalness of such implicit function introduction in mathematical texts contributes to the wide-spread feeling that the Axiom of Choice must be true.

Implicitly introduced functions are generally *partial functions*, i.e. they have a restricted domain and are not defined on the whole universe of the discourse. For example in (13), g is only defined on vertices of K and not on vertices of L.

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

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

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

[8] This section is partly taken over from Cramer (2012)

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

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

> **Unrestricted function comprehension**
>
> For every formula $\varphi(x, y)$, the following is an axiom:
>
> $$\forall x\ \exists y\ \varphi(x, y) \rightarrow \exists f\ \forall x\ \varphi(x, f(x))$$

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

Russell's paradox led to the abandonment of unrestricted comprehension in set theory. Two radically different approaches have been undertaken for restricting set comprehension: Russell himself restricted it through his Ramified Theory of Types, which was later simplified to Simple Type Theory (STT), mainly known via Church's formalisation in his simply typed lambda calculus (Church, 1940). On the other hand, the risk of paradoxes like Russell's paradox also contributed to the development of *ZFC* (Zermelo-Fraenkel set theory with the Axiom of Choice), which allows for a much richer set theoretic universe than the universe of simply typed sets. Since all the axioms of *ZFC* apart from the Axiom of Extensionality, the Axiom of Foundation and the Axiom of Choice are special cases of comprehension, one can view *ZFC* as an alternative way to restrict set comprehension.

Similarly, the above paradox must lead to the abandonment of unrestricted function comprehension. The type-theoretic approach can easily adapted to functions; see Cramer (2012) for the details. But the type restrictions that such a type-theoretic approach imposes may be too strict for some applications: Mathematicians sometimes make use of functions that do not fit into the corset of strict typing, e.g. a function defined on both real numbers and real functions. To overcome this restriction, we want an untyped theory of functions that avoids the above paradox, in a similar way in which *ZFC* is an untyped theory of sets that avoids Russell's paradox.

But there is no clear way to transfer the limitations that *ZFC* puts onto set comprehension to the case of function comprehension. However, there is an axiomatization of set theory called *Ackermann set theory* that is a conservative extension of *ZFC*. It turns out that the limitations that Ackermann set theory poses on set comprehension can be transferred to the case of function comprehension. We will show how to do this in the following chapter.

Chapter 4

Mathematical foundations of Naproche

In this chapter we describe the mathematical foundations needed for the Naproche system. After describing some variants of Ackermann set theory and proving facts we need about them later, we introduce a system called *Ackermann-like Function Theory* (*AFT*) by transferring the limitations that Ackermann set theory poses on set comprehension to the case of function comprehension, as announced at the end of the previous chapter. For this, a dichotomy similar to that between sets and classes in Ackermann set theory has to be imposed on functions. We propose the terms *function* and *map* respectively for this dichotomy.

Next we describe a rich mathematical background theory that has classes, maps, tuples, natural numbers and Booleans as primitive objects, and show that working in a logic enriched by this rich background theory is in a certain sense conservative over working in a logic without any mathematical background theory.

4.1 Ackermann set theory

We first present the original version of Ackermann set theory (Ackermann, 1956), which does not allow for urelements (i.e. objects that are not classes) and is called A in the literature.

All objects that the theory talks about are classes, and some of these classes are considered sets. Intuitively, one can think of the sets as those classes that are in some sense limited in size and because of this limitation more easily grasped as single objects.

The language of A contains two predicates: A binary predicate $\in$ and a unary predicate M (from the German word "'Menge"' for "'set"'). The axioms of A are as follows:

- Extensionality Axiom: $\forall x, y\ (\forall z\ (z \in x \leftrightarrow z \in y) \rightarrow x = y)$
- Class Comprehension Axiom Schema: Given a formula $F(y)$ (possibly with

parameters[1]) that does not have x among its free variables, the following is an axiom:
$\forall y\ (F(y) \to M(y)) \to \exists x\ \forall y\ (y \in x \leftrightarrow F(y))$

- Set Comprehension Axiom Schema: Given a formula $F(y)$ (possibly with parameters that are sets) that does not have x among its free variables and does not contain the symbol M, the following is an axiom:
$\forall y\ (F(y) \to M(y)) \to \exists x\ (M(x) \wedge \forall y\ (y \in x \leftrightarrow F(y)))$

- Element Axiom: Elements of sets are sets:
$\forall x, y\ (M(y) \wedge x \in y \to M(x))$

- Subset Axiom: Subsets of sets are sets:
$\forall x, y\ (M(y) \wedge \forall z\ (z \in x \to z \in y) \to M(x))$

There are three axioms that we sometimes add to A:

- Axiom of Foundation (for sets): Every non-empty set has an $\in$-minimal element:
$\forall x\ (M(x) \wedge \exists y\ y \in x \to \exists y \in x\ \forall z \in x\ z \notin y)$

- Axiom of Choice (for sets): For every set x of pairwise disjoint non-empty classes, there is a class y containing precisely one element from every element of x:
$\forall x\ (M(x) \wedge \forall y, z \in x\ \nexists w\ (w \in y \wedge w \in z) \wedge \forall y \in x\ \exists z\ z \in y \to \exists y\ \forall z \in x\ \exists! w \in y\ w \in z)$.

- Axiom of Global Choice: For every class x of pairwise disjoint non-empty classes, there is a class y containing precisely one element from every element of x:
$\forall x\ (\forall y, z \in x\ \nexists w\ (w \in y \wedge w \in z) \wedge \forall y \in x\ \exists z\ z \in y \to \exists y\ \forall z \in x\ \exists! w \in y\ w \in z)$.

We add the symbols *, C and G to the name of a theory to indicate addition of Foundation, Choice or Global Choice respectively. For example, A^*G is Ackermann set theory with Foundation and Global Choice. Additionally, we use the abbreviations AC and AGC for the Axiom of Choice and the Axiom of Global Choice respectively.

Later on we will also need to work with a variant of Ackermann set theory that allows for the existence of urelements. Ackermann himself presented a version of his theory with urelements, which we call A_{LU}, but we will work with a variant of it which we call A_U. To motivate A_U, it is useful to conceive the sets in Ackermann set theory as classes of limited size. Here we just take "limited" as a primitive concept that is useful in avoiding the paradoxes of set theory and function theory. In A_{LU}, all urelements are treated as limited objects, whereas A_U is more relaxed in that it allows for urelements which are not limited. The motivation for considering the possibility of unlimited urelements will become clear when we interpret Ackermann set theory within a function theory, in which there can be unlimited maps which are not classes and hence are urelements from

[1]This means that F may actually be of the form $F(\bar{z}, y)$, and that these parameters are universally quantified in the axiom:
$\forall \bar{z}\ \forall y\ (F(\bar{z}, y) \to M(y)) \to \exists x\ \forall y\ (y \in x \leftrightarrow F(\bar{z}, y))$

the point of view of the set theory that we interpret. A_{LU} can be obtained from A_U by adding an axiom stating that all urelements are limited. The full strength of A can be obtained from either A_U or A_{LU} by adding an axiom that states that there are no urelements.

The language of A_U contains three predicates: A binary predicate $\in$, a unary predicate C for classes and a unary predicate L for limited elements. The axioms of A_U are as follows:

- Extensionality Axiom: $\forall x, y\ (C(x) \wedge C(y) \wedge \forall z\ (z \in x \leftrightarrow z \in y) \rightarrow x = y)$
- Class Comprehension Axiom Schema: Given a formula $F(y)$ (possibly with parameters) that does not have x among its free variables, the following is an axiom:
  $\forall y\ (F(y) \rightarrow L(y)) \rightarrow \exists x\ (C(x) \wedge \forall y\ (y \in x \leftrightarrow F(y)))$
- Set Comprehension Axiom Schema: Given a formula $F(y)$ (possibly with parameters that are sets) that does not have x among its free variables and does not contain the symbol L, the following is an axiom:
  $\forall y\ (F(y) \rightarrow L(y)) \rightarrow \exists x\ (C(x) \wedge L(x) \wedge \forall y\ (y \in x \leftrightarrow F(y)))$
- Element Axiom: $\forall x, y\ (L(y) \wedge x \in y \rightarrow L(x))$
- Subset Axiom: $\forall x, y\ (L(y) \wedge \forall z\ (z \in x \rightarrow z \in y) \rightarrow L(x))$
- Classness Axiom: $\forall x, y\ (x \in y \rightarrow C(y))$

When presenting arguments in A_U in plain English, we use "set" for limited classes, i.e. for objects x such that $L(x) \wedge C(x)$.

We will now discuss the relationship between *ZF*(C) and different versions of Ackermann set theory. It is a result from the literature (Reinhardt, 1970) that A^* is a conservative extension of *ZF*, which implies two things: A^* interprets *ZF* and has the same consistency strength as *ZF* and *ZFC*. We will additionally prove that even the weaker theory A_U interprets *ZF*. To make these statements more precise, we first need some definitions. Definitions 4.1.1, 4.1.2, 4.1.3 and 4.1.4 are taken over from Koepke and Koerwien (2008).

Definition 4.1.1. Let L_1 and L_2 be *PL* languages and T_1 an L_1-theory. Let A be the signature of L_2 together with the identity relation symbol $=$ and an additional symbol $\mathbb{U}$. A function $\mathcal{A}$ from A to L_1 (considered as a set of formulae) is called a T_1-*definable* L_2-*structure* iff

- $\mathcal{A}(\mathbb{U})$ has exactly one free variable x and $T_1 \vdash \exists x\ \mathcal{A}(\mathbb{U})(x)$. We write $x \in \mathbb{U}$ instead of $\mathcal{A}(\mathbb{U})(x)$.
- For all relation symbols $R \in A$ the free variables of $\mathcal{A}(R)$ are exactly $v_1, \ldots, v_n$ where n is the arity of R.
- For all function symbols $f \in A$ the free variables of $\mathcal{A}(f)$ are exactly $v_1, \ldots, v_{n+1}$ where n is the arity of f. Moreover,
  $T_1 \vdash \forall v_1, \ldots, v_{n+2} \in \mathbb{U}\ (\mathcal{A}(f)(v_1, \ldots, v_n, v_{n+1}) \wedge \mathcal{A}(f)(v_1, \ldots, v_n, v_{n+2}) \rightarrow \mathcal{A}(=)(v_{n+1}, v_{n+2}))$ and
  $T_1 \vdash \forall v_1, \ldots, v_n \in \mathbb{U}\ \exists v_{n+1} \in \mathbb{U}\ \mathcal{A}(f)(v_1, \ldots, v_n, v_{n+1})$.

- For all constant symbols $c \in A$, $\mathcal{A}(c)$ has exactly one free variable x and $T_1 \vdash \exists x \in \mathbb{U}\ (\mathcal{A}(c)(x)) \wedge \forall x, y \in \mathbb{U}\ (\mathcal{A}(c)(x) \wedge \mathcal{A}(c)(y) \rightarrow \mathcal{A}(=)(x, y))$.
- T_1 proves that $\mathcal{A}(=)$ defines a *congruence relation* for L_2, i.e. it has the properties of an equivalence relation and respects all functions and relations defined by the formulas of $\mathcal{A}$.

Definition 4.1.2. Let L_1 and L_2 be *PL* languages, T_1 an L_1-theory and $\mathcal{A}$ a T_1-definable L_2-structure. Then for an L_2-formula ψ the *relativization of ψ to $\mathcal{A}$* is an L_1-formula $\psi^{\mathcal{A}}$ defined by recursion on the structure of ψ:

- If $\psi \equiv (x = y)$, where x and y are variables, then $\psi^{\mathcal{A}} := \mathcal{A}(=)(x, y)$.
- If x is a variable, c is a constant symbol and $\psi \equiv (x = c)$ then $\psi^{\mathcal{A}} := \mathcal{A}(c)(x)$.
- If x is a variable, f is a function symbol, $t_1, \ldots, t_n$ are L_2-terms and $\psi \equiv (x = f(t_1, \ldots, t_n))$ then $\psi^{\mathcal{A}} := \exists x_1, \ldots, x_n \in \mathbb{U}\ ((x_1 = t_1)^{\mathcal{A}} \wedge \ldots \wedge (x_n = t_n)^{\mathcal{A}} \wedge \mathcal{A}(f)(x_1, \ldots, x_n, x))$.
- If R is a relation symbol (including the identity), then $R(t_1, \ldots, t_n)^{\mathcal{A}} := \exists x_1, \ldots, x_n \in \mathbb{U}\ ((x_1 = t_1)^{\mathcal{A}} \wedge \ldots \wedge (x_n = t_n)^{\mathcal{A}} \wedge \mathcal{A}(R)(x_1, \ldots, x_n))$.
- $\top^{\mathcal{A}} = \top$ and $\bot^{\mathcal{A}} = \bot$.
- $(\neg\psi)^{\mathcal{A}} := \neg\psi^{\mathcal{A}}$.
- For every binary connective $*$, $(\psi_1 * \psi_2)^{\mathcal{A}} := \psi_1^{\mathcal{A}} * \psi_2^{\mathcal{A}}$
- $(\forall x\ \psi)^{\mathcal{A}} := \forall x \in \mathbb{U}\ (\psi^{\mathcal{A}})$ and $(\exists x\ \psi)^{\mathcal{A}} := \exists x \in \mathbb{U}\ (\psi^{\mathcal{A}})$.

We also write $\mathcal{A} \models \varphi$ for $\varphi^{\mathcal{A}}$.

Definition 4.1.3. If Φ is a set of L_2-formulae we define $\Phi^{\mathcal{A}} := \{\varphi^{\mathcal{A}} \mid \varphi \in \Phi\}$.

Definition 4.1.4. Let L_1 and L_2 be *PL* languages, T_1 an L_1-theory and T_2 an L_2-theory. Then T_2 is *interpretable in* T_1 (or T_1 *interprets* T_2) iff there is a T_1-definable L_2-structure $\mathcal{A}$ such that $T_1 \vdash T_2^{\mathcal{A}}$.

Remark. If T_2 is interpretable in T_1 and T_1 is consistent then T_2 is consistent.

Definition 4.1.5. Let L_1 and L_2 be *PL* languages, T_1 an L_1-theory and T_2 an L_2-theory. Then T_1 is a *conservative extension* of T_2 iff there is a T_1-definable L_2-structure $\mathcal{A}$ such that $T_1 \vdash T_2^{\mathcal{A}}$ and $T_2^{\mathcal{A}}$ proves every theorem of T_1 that is of the form $\varphi^{\mathcal{A}}$ for some $\varphi \in L_2$.

Definition 4.1.6. For an $\in$-formula φ, let φ_M denote the formula obtained from φ by restricting all quantifiers by the predicate M.

Definition 4.1.7. For a set Φ of $\in$-formulae, define $\Phi_M := \{\varphi_M \mid \varphi \in \Phi\}$.

Now we can state Reinhardt's (1970) result that A^* interprets *ZF* as follows:

Theorem 4.1.8. $A^* \vdash ZF_M$.

Remark. For every axiom φ of *ZF* apart from Replacement, φ_M is easily established in A^*. Foundation and Extensionality are part of A^*, and for every other *ZF* axiom φ apart from Replacement, we will actually establish φ_M in the weaker system A_U in Lemma 4.1.12 below. The interesting part is therefore Replacement, which was established by Reinhardt (1970).

Corollary. $A^*C \vdash ZFC_M$.

For the proof that follows we will need the following standard result:

Reflection Theorem Schema (Montague, 1961). *For any finite set of $\in$-formulae (possibly with parameters), ZF proves that there is an ordinal α such that all the formulae in the set are absolute for V_α.*

Remark. By adding the Powerset Axiom to the finite set of formulae, one can ensure that α must be a limit ordinal.

Additionally, we need the following definition:

Definition 4.1.9. Given an A^*-formula φ and a constant symbol c, $\varphi^{\frac{c}{M}}$ denotes the formula resulting from φ by replacing all occurrences of $M(t)$ in φ by $t \in c$.

Theorem 4.1.10. *For any $\in$-formula φ, $ZF \vdash \varphi$ iff $A^* \vdash \varphi_M$.*

Proof. The left-to-right implication directly follows from Theorem 4.1.8. The right-to-left implication was proved by Lévy (1959). Since his proof uses old-fashioned notation and is more involved than needed for this result (because one lemma proves something stronger than needed for this implication), it is worthwhile for us to present his proof of this central result in modern notation.

Let φ be an $\in$-formula such that $A^* \vdash \varphi_M$. Let $\varphi_1, \ldots, \varphi_n$ be the formulae used for Set Comprehension in the proof. Let α be an ordinal such that φ as well as φ_i and $\exists z\ \forall u\ (u \in z \leftrightarrow \varphi_i(u))$ for $1 \leq i \leq n$ are absolute for V_α. We define ZF_α to be the theory that is axiomatized by the axioms of *ZF* and the statement that V_α is absolute for these formulae, where V_α is considered to be a new constant symbol in this theory.

Then for every axiom ψ used in the proof of φ_M, $\psi^{\frac{V_\alpha}{M}}$ is a theorem of ZF_α:

Extensionality and Foundation
Trivial.

Element and Subset Axioms
Elements of elements of V_α and subsets of elements of V_α are in V_α.

Class Comprehension
For any $\in$-formula $\chi(x, y)$ such that $\forall x\ (\chi(V_\alpha, x) \rightarrow x \in V_\alpha)$, ZF_α implies that $\{x \mid \chi(V_\alpha, x)\}$ exists.

Set Comprehension
Let φ_i be one of the formulae for which Set Comprehension was used in the proof of φ_M, and suppose

$$\forall x\ (\varphi_i(x) \rightarrow x \in V_\alpha). \tag{4.1}$$

ZF_α implies

$$\exists z\ \forall u\ (u \in z \leftrightarrow \varphi_i(u)). \tag{4.2}$$

By absoluteness of (4.2) for V_α, we have

$$\exists z \in V_\alpha \, \forall u \in V_\alpha \, (u \in z \leftrightarrow \varphi_i^{V_\alpha}(u)) \tag{4.3}$$

Using absoluteness of φ_i for V_α, (4.1) and transitivity of V_α, (4.3) simplifies to (4.4) as required:

$$\exists z \in V_\alpha \, \forall u \, (u \in z \leftrightarrow \varphi_i(u)). \tag{4.4}$$

So $ZF_\alpha \vdash \varphi_M{}^{\frac{V_\alpha}{M}}$. Since $\varphi_M{}^{\frac{V_\alpha}{M}} = \varphi^{V_\alpha}$, the absoluteness of φ for V_α implies that $ZF_\alpha \vdash \varphi$. Finally, since φ does not contain the constant symbol V_α, and since ZF proves the existence of a V_α with the required absoluteness property, we can conclude $ZF \vdash \varphi$, as required. □

Remark. Note that this proof can be simplified to a proof that if $A^* \vdash \varphi$, then $ZF \vdash \varphi$: Working on the assumption that $A^* \vdash \varphi$ and noting that φ does not contain M, we can – at the place where we concluded $ZF_\alpha \vdash \varphi_M{}^{\frac{V_\alpha}{M}}$ in the above proof – conclude that $ZF_\alpha \vdash \varphi$. Then $ZF \vdash \varphi$ follows as in the above proof.

Corollary. *For any $\in$-formula φ, $ZFC \vdash \varphi$ iff $A^*G \vdash \varphi_M$.*

Proof. Clearly AC_M follows from A^*G, which gives us the additional strength needed for the left-to-right implication. For the right-to-left implication, we just need to replace ZF and A^* by ZFC and A^* G in the above proof of the right-to-left implication, and add to this proof that $\mathrm{AGC}^{\frac{V_\alpha}{M}}$ is a theorem of ZFC_α, which is easily seen to be true. □

4.1.1 A_U interprets A^* and ZF

Even though A_U is weaker than A^*, one can still interpret A^* and hence ZF in A_U. For this we first have to develop some set theory within A_U. This development is analogous to the development of Ackermann (1956), Lévy (1959) and Lévy and Vaught (1961) in A, with some minor adaptations in order to make it work in the weaker theory A_U.

Definition 4.1.11. $x \subseteq y$ iff $\forall z \, (z \in x \to z \in y)$.

Lemma 4.1.12. *The axioms of ZF apart from Extensionality, Replacement and Foundation, with quantifiers restricted to sets, are theorems of A_U.*

Proof. Each of these axioms postulates under certain condition the existence of a set with a certain property. For each axiom, we construct a class that is trivially seen to witness the restriction to sets of the existence claim, and show that it is actually a set.

Empty Set

Apply Set Comprehension to $x \neq x$ to construct the set $\{x \mid x \neq x\}$ (which we call $\emptyset$ as usual).

Pairing

Given sets a and b, apply Set Comprehension to $x = a \vee x = b$.

Powerset

Let a be a set. Set Comprehension can be applied to $x \subseteq a$, because any x satisfying this is a subclass of a and hence a set.

Separation

Let a be a set and φ be an $\in$-formula. $x \in a \wedge \varphi_M(x)$ defines a class, because all x satisfying this are in a and hence sets. This class is a subclass of a and hence a set.

Union

$\exists z\ (x \in z \wedge z \in a)$ defines a set, because any x satisfying it is a set by two applications of Element.

Infinity

Applying Class Comprehension to $L(x) \wedge C(x)$, one can establish that there is a class V containing all sets and nothing else. Now we can apply Set Comprehension to $\varphi(x) := \forall I\ (\emptyset \in I \wedge \forall y\ (y \in I \rightarrow y \cup \{y\} \in I) \rightarrow x \in I)$, because $\emptyset \in V \wedge \forall y\ (y \in V \rightarrow y \cup \{y\} \in V)$, i.e. any x satisfying $\varphi(x)$ is in V and hence a set. □

Remark. The sets that we have shown to exist do not only satisfy their extensionality conditions for sets, but for any objects. For example, the doubleton $\{a, b\}$ established by pairing satisfies not only $\forall x\ (set(x) \rightarrow (x \in \{a, b\} \leftrightarrow x = a \vee x = b))$ but also the stronger statement $\forall x\ (x \in \{a, b\} \leftrightarrow x = a \vee x = b)$. This strong characterization of these sets is usually needed when we apply these axiom in what follows.

We use the following standard definitions:

Definition 4.1.13. The *ordered pair* (x, y) is defined to be $\{\{x\}, \{x, y\}\}$ if this class exists.

Remark. Given Pairing and the Extensionality Axiom of A_U, (x, y) always exists for limited x, y. If x or y is unlimited, it is possible that (x, y) does not exist (though it might turn out to exist even in that case). An atomic statement involving (x, y) should be considered false if (x, y) does not exist. The same convention holds for all other terms we define without proving that they exist in all cases.

Definition 4.1.14. A *relation* is a class of ordered pairs. Given a relation R, we write $R(x, y)$ for $(x, y) \in R$.

Definition 4.1.15. A *map* is a relation R such that $R(x, y_1)$ and $R(x, y_2)$ implies $y_1 = y_2$. Given a map f, we write $f(x)$ for the element y such that $(x, y) \in f$ if such a y exists.

Definition 4.1.16. For a relation R, the *domain of* R ($dom(R)$) is the class of x such that $\exists y\ R(x, y)$ if such a class exists.

Definition 4.1.17. A class x is *transitive* ($trans(x)$) iff for all $z \in y \in x$, we have $z \in x$.

Definition 4.1.18. A class x is *well-ordered by* $\in$ iff $\forall y, z\ (y \notin y \wedge (y \in z \vee z \in y \vee y = z) \wedge \forall u \subseteq x\ (u \neq \emptyset \rightarrow \exists t\ (t \in u \wedge \nexists s\ (s \in t \wedge s \in u))))$.

As noted by Lévy and Vaught (1961) (pages 1054-1055), the expected definition of ordinals as transitive classes well-ordered by $\in$ does not suffice to prove that any two ordinals are comparable. Hence they added the condition that for any two subclasses x, y of an ordinal, $x \setminus y$ should exist. Since we are now working in a theory with urelements, we have to add the additional condition that all elements of an ordinal are classes:

Definition 4.1.19. x is an *ordinal* ($Ord(x)$) iff x is a transitive class well-ordered by $\in$, $\forall y, z \subseteq x\ \exists u\ \forall t\ (t \in u \leftrightarrow t \in y \wedge t \notin z)$ and $\forall y \in x\ C(y)$.

Definition 4.1.20. x is an *ordinal number* ($ord(x)$) iff x is an ordinal and is limited.

Definition 4.1.21. For an ordinal x, x' denotes $x \cup \{x\}$ if this class exists.

Definition 4.1.22. When x and y are ordinals, $x < y$ is an alternative notation for $x \in y$.

Lemma 4.1.23. *An element of an ordinal is an ordinal.*

Proof. Let x be an ordinal and $y \in x$. The transitivity and $\in$-well-orderedness of y can be established using standard techniques. y is a class by the last condition in the definition of $Ord(x)$. Every element of y is an element of x and hence a class. Finally, let a and b be subclasses of y. These are subclasses of x, so by $Ord(x)$ $a \setminus b$ exists as required. □

Lemma 4.1.24. *For any A_U-formula $\varphi(x)$, the following is a theorem of A_U: If $\exists x\ (\varphi(x) \wedge ord(x))$, then there is a least x such that $\varphi(x) \wedge ord(x)$.*

Proof. Choose y such that $\varphi(y) \wedge ord(y)$. If y is minimal with this property, we are done, so assume it is not. Since y is a set, $\{x \in y \mid \varphi(x)\}$ defines a set. Since it is a non-empty subset of y and y is well-ordered, it has a minimal element x. If there is a $z < x$ such that $\varphi(z)$, then $z \in y$ by transitivity of y, contradicting the choice of x. So x is minimal such that $\varphi(x) \wedge ord(x)$. □

Remark. This allows us to give proofs by transfinite induction over the ordinal numbers (but not over all ordinals).

Lemma 4.1.25. *If α and β are ordinals, then precisely one of the following properties holds:*

- $\alpha < \beta$
- $\beta < \alpha$
- $\alpha = \beta$

Proof. The fact that at most one of these properties holds is easily proved.

Now suppose for a contradiction that none of these three properties holds. $\alpha \setminus \beta$ and $\beta \setminus \alpha$ exist by the additional condition imposed on ordinals.

Suppose for a contradiction that $\alpha \setminus \beta$ is empty, i.e. $\alpha \subseteq \beta$. Since $\alpha \neq \beta$, $\beta \setminus \alpha$ is non-empty. Let x be the minimal element of $\beta \setminus \alpha$.

Suppose $y \in x$. Then by the transitivity of β, $y \in \beta$. If y were not in α, it would be in $\beta \setminus \alpha$, contradicting the minimality of x. So $y \in \alpha$. Thus $x \subseteq \alpha$.

Conversely, suppose $y \in \alpha$. Then $y \in \beta$. Furthermore $y \neq x$, since $x \notin \alpha$. Additionally $x \notin y$, for otherwise we would have $x \in \alpha$ by the transitivity of α. Since x and y are both in β, which is totally ordered by $\in$, we may conclude that $y \in x$. Thus $\alpha \subseteq x$.

So $x = \alpha$. But since $x \in \beta$, we can now conclude that $\alpha \in \beta$, contrary to our assumption.

Thus we have that $\alpha \setminus \beta$ is non-empty. Similarly, $\beta \setminus \alpha$ is non-empty. Let a be the minimal element of $\alpha \setminus \beta$ and let b be the minimal element of $\beta \setminus \alpha$.

Let $x \in a$. By minimality of a, $x \in \beta$. If $x = b$ or $b \in x$, then the transitivity of α implies that $b \in \alpha$, contrary to the choice of b. Since $\in$ totally orders β, $x \in b$. Thus $a \subseteq b$.

Similarly, $b \subseteq a$, i.e. $a = b$. But then $b \in \alpha$, contradicting the choice of b. □

Lemma 4.1.26. *The union of a set of ordinal numbers is an ordinal number.*

Proof. The existence of this union follows from the Axiom of Union that we have proved. Its transitivity and $\in$-well-orderedness can be established using standard techniques. It clearly contains only classes. And since subclasses of it are sets, their subtraction certainly exists by Separation. □

Since we have not proved Replacement, we cannot use transfinite recursion. Nevertheless, the von-Neumann hierarchy of V_α's can be shown to exist for ordinal numbers α (but not for any ordinal α, though for some it does exist). We introduce a notation for speaking about the V_α's and about restrictions of the map $\alpha \mapsto V_\alpha$ without ontological commitment:

Definition 4.1.27. Given an ordinal α, $V_\bullet|_\alpha$ denotes the map such that $dom(V_\bullet|_\alpha) = \alpha$ and $\forall x \in \alpha\ (y \in V_x|_\alpha \leftrightarrow \exists z \in x\ y \subseteq V_z|_\alpha)$, if such a map exists and is unique. (Here $V_x|_\alpha$ is a convenient notation for $V_\bullet|_\alpha(x)$.)

Definition 4.1.28. Given an ordinal α, V_α denotes the class $V_\alpha|_{\alpha'}$ if this class exists.

Lemma 4.1.29. *For every ordinal number α, V_α exists and is a set.*

Proof. If for all ordinal numbers β, $V_\bullet|_{\beta'}$ were a set, then the lemma would clearly hold. So assume for a contradiction that β is the smallest ordinal number such that $V_\bullet|_{\beta'}$ is not a set. Then the formula $\exists x < \beta\ (z = V_\bullet|_{x'})$ holds only for sets z, so by Set Comprehension $u := \{z \mid \exists x < \beta\ z = V_\bullet|_{x'}\}$ is a set. So $V_\bullet|_\beta = \bigcup u$ is a set. But then $V_\beta = \{x \mid$ there is a pair $(y, z) \in V_\bullet|_\beta$ such that $x \subseteq z\}$ is a set by Set Comprehension. Hence $V_\bullet|_{\beta'} = V_\bullet|_\beta \cup (\beta, V_\beta)$ is a set, contrary to our assumption. □

Definition 4.1.30. x is a *pure set* ($pset(x)$) iff $\exists\alpha\ (ord(\alpha) \wedge x \in V_\alpha)$.

Definition 4.1.31. x is a *pure class* ($PC(x)$) iff $C(x) \wedge \forall y \in x\ \exists\alpha\ (Ord(\alpha) \wedge trans(V_\alpha) \wedge \forall z \in V_\alpha\ C(z) \wedge y \in V_\alpha)$.

Remark. Even though we can show by transfinite induction that for all ordinal numbers α, V_α is transitive and contains only classes, this cannot be shown for ordinals, since we cannot carry out transfinite induction over the ordinals. So the conditions $trans(V_\alpha)$ and $\forall z \in V_\alpha\ C(z)$ do make a difference.

Definition 4.1.32. For a pure set x, the *rank* of x (*rank*(x)) is the smallest ordinal number α such that $x \in V_\alpha$.

Remark. By Lemma 4.1.24, *rank*(x) is well-defined for all pure sets x.

Definition 4.1.33. For an A^*-formula φ, we define the translation φ_p to be the A_U-formula obtained by replacing all occurrences of M by *pset* and restricting all quantifiers by *PC*.

Definition 4.1.34. For a set Φ of A^*-formulae, define $\Phi_p := \{\varphi_p \mid \varphi \in \Phi\}$.

Now the following theorem establishes that A_U interprets A^*:

Theorem 4.1.35. $A_U \vdash A^*_p$.

Proof. We have to prove that for every axiom φ of A^*, φ_p can be proved in A_U:

Extensionality

We have to prove that $\forall x, y\ (PC(x) \wedge PC(y) \wedge \forall z\ (PC(z) \rightarrow (z \in x \leftrightarrow z \in y)) \rightarrow x = y)$. Since all pure classes are classes, A_U-Extensionality implies that it is enough to show that any element of a pure class is a pure class. Let x be a pure class and let $y \in x$. Then there is an ordinal α such that V_α is transitive, all elements of V_α are classes and $y \in V_\alpha$. Then y is a class, and if $z \in y$, then $z \in V_\alpha$ by transitivity of V_α, so y is a pure class as required.

Class Comprehension

Given any A^*-formula φ such that for all x, $\varphi_p(x)$ implies that x is a pure set, we have to prove that there is a pure class y such that a pure class z is in y iff $\varphi_p(z)$. We apply A_U's Class Comprehension to $\varphi_p(x)$. The resulting class is a pure class, because all of its elements are pure sets, i.e. in a V_α for an ordinal number α (and as remarked above, for ordinal numbers α, the constraints *trans*(V_α) and $\forall z \in V_\alpha\ C(z)$ certainly hold).

Set Comprehension

Given any $\in$-formula φ such that for all x, $\varphi_p(x)$ implies that x is a pure set, we have to prove that there is a pure set y such that a pure class z is in y iff $\varphi_b(z)$. Since $\varphi_p(x)$ does not contain the symbol L, we may apply A_U's Set Comprehension to $\varphi_p(x)$ to show that $y := \{x \mid \varphi_p(x)\}$ is a set.

Now "z is the rank of an element of y" holds only for sets. So $v := \{z \mid z \text{ is the rank of an element of } y\}$ is a set. But then $\mu := \bigcup v$ is an ordinal number by Lemma 4.1.26. Now $y \subseteq V_\mu$, i.e. $y \in V_{\mu'}$, i.e. y is a pure set, as required.

Element Axiom

An element of a pure set is a pure set, since $\forall \alpha\ (ord(\alpha) \rightarrow trans(V_\alpha) \wedge \forall z \in V_\alpha\ C(z))$ by transfinite induction.

Subset Axiom

We need to show that a subclass of a pure set is a pure set. This directly follows from the fact that for an ordinal number α, V_α is closed under subclasses.

Foundation for sets

We need to show that every non-empty pure set contains a pure class disjoint from it. Let a be a non-empty pure set. Let b be an element of a of minimal rank. Then b is clearly a pure class and disjoint from a. □

Corollary. *A_U interprets ZF. (By Theorem 4.1.8.)*

Theorem 4.1.36. *A_UC interprets A^*C.*

Proof. Adding Choice for sets to A_U clearly results in Choice for pure sets in the above interpretation of A^*, as required. □

Corollary. *A_UC interprets ZFC. (By the corollary to Theorem 4.1.8.)*

4.2 Ackermann-like Function Theory

Now we present Ackermann-like Function Theory (*AFT*), which adapts the limitations that Ackermann set theory poses on set comprehension to function comprehension.

AFT is a theory of partial functions/maps in the sense that the functions/maps that it can talk about are not defined on the whole universe of discourse of *AFT*. But *AFT* is a theory over *PL* (standard first-order predicate logic), where all function symbols are considered to represent total functions on the domain of discourse. Hence the application function that *AFT* has for applying functions/maps from its domain to other objects of the domain must also be total. So we must assign a value even to the application of a function/map to an argument at which it is not defined. For this we introduce an undefinedness object into the domain of *AFT*, which we designate as u, and which is the value that we give to the application of any function/map to any argument where it is not defined.

The language of *AFT* contains a unary predicate F for functions, a unary predicate U for urelements[2], for every $n \geq 1$ a unary predicate symbol a_n ("to be an n-ary map"), a constant symbol u for undefinedness, and for every $n \geq 2$ an n+1-ary function symbol app_n for function application. Instead of $app_n(f, t_1, \ldots, t_n)$ we usually simply write $f(t_1, \ldots, t_n)$. Instead of $U(x) \vee F(x)$, we write $L(x)$ and say that x is *limited*.

The axioms of *AFT* are as follows:

- Extensionality Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f\ \forall g\ (a_n(f) \wedge a_n(g) \wedge \forall \bar{z}\ f(\bar{z}) = g(\bar{z}) \rightarrow f = g)$

- Map Comprehension Axiom Schema: Given formulae $P(\bar{z})$ and $R(\bar{z}, x)$ with parameters, the following is an axiom:
  $\forall \bar{z}\ \forall x\ (R(\bar{z}, x) \rightarrow L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \wedge\ \forall \bar{z}\ (P(\bar{z}) \rightarrow \exists x\ R(\bar{z}, x)) \rightarrow$
  $\exists f\ (a_n(f) \wedge \forall \bar{z}\ (P(\bar{z}) \rightarrow R(\bar{z}, f(\bar{z}))) \wedge \forall \bar{z}\ (\neg P(\bar{z}) \rightarrow f(\bar{z}) = u))$

[2]Now that we are describing a theory of functions/maps rather than a theory of sets/classes, *urelement* means an object of the domain that is not a map rather than an object of the domain that is not a class.

- Functionality Axiom Schema: Given formulae $P(\bar{z})$ and $R(\bar{z}, x)$ that have limited parameters and do not contain the symbol F, the following is an axiom:
  $\forall \bar{z}\ \forall x\ (R(\bar{z}, x) \to L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \to \forall f\ (a_n(f) \wedge \forall \bar{z}\ (P(\bar{z}) \to R(\bar{z}, f(\bar{z}))) \wedge \forall \bar{z}\ (\neg P(\bar{z}) \to f(\bar{z}) = u) \to F(f))$
- Element Axiom Schema ("elements of domain and range of functions are limited"): For $n \geq 1$ and $\bar{z}$ a variable list of length n:
  $\forall f\ \forall \bar{z}\ (F(f) \wedge f(\bar{z}) \neq u \to L(z_1) \wedge \ldots \wedge L(z_n) \wedge L(f(\bar{z})))$
- Subfunction Axiom Schema ("submaps of functions are functions"): For $n \geq 1$ and $\bar{z}$ a variable list of length n:
  $\forall f\ \forall g\ (F(g) \wedge a_n(g) \wedge a_n(f) \wedge \forall \bar{z}\ (f(\bar{z}) \neq u \to f(\bar{z}) = g(\bar{z})) \to F(f))$
- Undefinedness Axiom Schema: For $n, m \geq 0$ with $n + m \geq 1$:
  $\forall x_1, \ldots, x_n, y_1, \ldots, y_m\ app_{n+m}(x_1, \ldots, x_n, u, y_1, \ldots, y_m) = u$

Note that Map Comprehension and Functionality together imply the following Function Comprehension Theorem Schema:

- Given formulae $P(\bar{z})$ and $R(\bar{z}, x)$ that have limited parameters and do not contain the symbol F, the following is a theorem of AFT:
  $\forall \bar{z}\ \forall x\ (R(\bar{z}, x) \to L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \wedge\ \forall \bar{z}\ (P(\bar{z}) \to \exists x\ R(\bar{z}, x)) \to \exists f\ (a_n(f) \wedge F(f) \wedge \forall \bar{z}\ ((P(\bar{z}) \to R(\bar{z}, f(\bar{z}))) \wedge (\neg P(\bar{z}) \to f(\bar{z}) = u)))$

For our application of AFT in chapter 5, we will need an adapted version of AFT with two distinguished urelements representing Boolean values, for which we introduce special constant symbols $\top$ and $\bot$ and an additional axiom $\top \neq \bot$. We call this adaptation of AFT *AFTB*.

4.2.1 *AFT* equiconsistent with *ZFC*

In this subsection we will establish that AFT and *ZFC* are equiconsistent, but will actually prove two stronger results for achieving this: Firstly that AFT interprets *ZFC*, which we achieve by interpreting $A_U C$ in AFT. Secondly that $A^* G$ interprets AFT.

Proposition 4.2.1. *There exists a function $\emptyset$ such that for all x, $\emptyset(x) = u$.*

Proof. Apply Function Comprehension to the formula $\varphi(x, y) := x \neq x$. □

Definition 4.2.2. A *class* is a unary map f such that for all x, $f(x) = u$ or $f(x) = \emptyset$.

Definition 4.2.3. A *set* is a class that is a function.

Definition 4.2.4. Define $x\ \varepsilon\ y$ iff y is a class and $y(x) = \emptyset$.

Definition 4.2.5. For an A_U-formula φ, we define the translation φ_{AFT} to be the AFT-formula obtained by replacing all occurrences of C by *class*, all occurrences of $\in$ by ε and all occurrences of L by F.

Definition 4.2.6. For a set Φ of A_U-formulae, define $\Phi_{AFT} := \{\varphi_{AFT} \mid \varphi \in \Phi\}$.

Now the following theorem establishes that AFT interprets $A_U C$:

Theorem 4.2.7. $AFT \vdash A_U C_{AFT}$.

Proof. We have to prove that for every axiom φ of $A_U C$, φ_{AFT} can be proved in AFT:

Extensionality

Let a and b be classes such that $\forall x\ (x\ \varepsilon\ a \leftrightarrow x\ \varepsilon\ b)$. Then by map extensionality $a = b$, as required.

Class Comprehension

Let $\varphi(x)$ be an A_U-formula such that $\forall x\ (\varphi_{AFT}(x) \rightarrow F(x))$. Applying Map Comprehension to the formula $\varphi_{AFT}(x) \wedge y = \emptyset$, we can conclude that there is a unary map f defined precisely on those x such that $\varphi_{AFT}(x)$ and taking the value $\emptyset$ at all such x, as required.

Set Comprehension

Let $\varphi(x)$ be an A_U-formula not containing the symbol L such that $\forall x\ (\varphi_{AFT}(x) \rightarrow F(x))$. Since $\varphi_{AFT}(x)$ does not contain the symbol F, we may apply Function Comprehension to the formula $\varphi_{AFT}(x) \wedge y = \emptyset$. The resulting function clearly has the required properties.

Element Axiom

This follows directly from the Element Axiom of *AFT*.

Subset Axiom

This follows directly from the Subfunction Axiom of *AFT*.

Classness Axiom

This follows directly from the definition of ε.

Choice

Here we make use of the fact that *AFT* assert the existence of choice functions:

Let a be a set of pairwise disjoint non-empty sets. Any x, y satisfying $x\ \varepsilon\ a \wedge y\ \varepsilon\ x$ are limited by the Element Axiom, so we can apply Function Comprehension to $x\ \varepsilon\ a \wedge y\ \varepsilon\ x$ to construct a function f such that $\forall x\ \varepsilon\ a\ f(x)\ \varepsilon\ x$ and $\forall x \not\varepsilon\ a\ f(x) = u$. Now we apply Function Comprehension to $\exists z\ x = f(z) \wedge x \neq u \wedge y = \emptyset$ to construct a set b that satisfies $\forall x\ \varepsilon\ a\ \exists! y\ \varepsilon\ b\ y\ \varepsilon\ x$, as required. □

Theorem 4.2.8. *AFT interprets ZFC.*

Proof. By Theorem 4.2.7 and the corollary to Theorem 4.1.36. □

Now we still need to establish that A^*G interprets *AFT*. We present a proof that resembles the proof of Theorem 4.1.35 that A_U interprets A^*. We first need some definitions within A^*G:

Definition 4.2.9. A class a of n+1-tuples is called *functional* iff for all $y_1, \ldots, y_n$, z_1, z_2 such that $(y_1, \ldots, y_n, z_1) \in x$ and $(y_1, \ldots, y_n, z_2) \in x$, $z_1 = z_2$.

Now we will define a *cumulative hierarchy of functions*, in a similar way as we defined the cumulative hierarchy of sets of V_α's in Definitions 4.1.27 and 4.1.28. The intended definition is the following:

$$\Phi_0 := \emptyset$$
$$\Phi_{\alpha+1} := \{f \mid \text{For some } n \geq 2,\ f \text{ is a functional class of } n\text{-tuples of elements of } \Phi_\alpha\}$$
$$\Phi_\lambda := \bigcup_{\alpha<\lambda} \Phi_\alpha$$

As in the case of the V_α's in A_U, we cannot define the Φ_α's for all ordinals α, but the following definitions do ensure that it is defined for all ordinal numbers:

Definition 4.2.10. Given a set A, write $f :\to A$ iff for some $n \geq 2$, f is a functional class of n-tuples of elements of A.

Definition 4.2.11. Given an ordinal α, $\Phi_\bullet|_\alpha$ denotes the map such that $dom(\Phi_\bullet|_\alpha) = \alpha$ and $\forall x \in \alpha\ (y \in \Phi_x|_\alpha \leftrightarrow \exists z \in x\ y :\to \Phi_z|_\alpha)$, if such a map exists and is unique.

Definition 4.2.12. Given an ordinal α, Φ_α denotes the class $\Phi_\alpha|_{\alpha'}$ if this class exists.

Lemma 4.2.13. *For every ordinal number α, Φ_α exists and is a set.*

Proof. Analogous to the proof of Lemma 4.1.29. □

Definition 4.2.14. A class A is Φ-transitive iff for every $x \in A$, every element of a tuple in x is an element of A.

Definition 4.2.15. x is a Φ-*function* iff there is an ordinal number α such that $x \in \Phi_\alpha$.

Definition 4.2.16. x is a Φ-*map* iff x is a functional class and for every element y of a tuple in x, there is an ordinal α such that Φ_α is Φ-transitive, every element of Φ_α is a functional class and $y \in \Phi_\alpha$.

Lemma 4.2.17. *Every element of a tuple in a Φ-map is a Φ-map.*

Proof. Let x be a Φ-map and y be an element of a tuple in x. Then there is an ordinal α such that Φ_α is Φ-transitive, every element of Φ_α is a functional class and $y \in \Phi_\alpha$. Then y is a functional class, and if z is an element of a tuple of y, then $z \in \Phi_\alpha$ by the Φ-transitivity of Φ_α, so y is a Φ-map as required. □

Lemma 4.2.18. *Every Φ-map that is a set is a Φ-function.*

Proof. For every ordinal number α, let λ_α be the smallest limit ordinal greater than α. Now it is easily seen by transfinite induction that for every ordinal number α, every Φ-map in V_α is in Φ_{λ_α}. □

Fix a set $\bar{u}$ that is not a Φ-map (e.g. $\bar{u} := \{\emptyset\}$).

Definition 4.2.19. Let $\mathcal{F}$ be the A^*C-definable L_{AFT}-structure defined by

$$\begin{aligned}
\mathcal{F}(\mathbb{U}) &:= \text{“}x \text{ is a } \Phi\text{-map or } x = \bar{u}\text{”} \\
\mathcal{F}(=) &:= v_1 = v_2 \\
\mathcal{F}(F) &:= \text{“}v_1 \text{ is a } \Phi\text{-function”} \\
\mathcal{F}(U) &:= v_1 \neq v_1 \\
\mathcal{F}(a_n) &:= \text{“}v_1 \text{ is a } \Phi\text{-map consisting only of } n{+}1\text{-tuples” for } n \geq 2 \\
\mathcal{F}(app_n) &:= \text{“}(v_2, \ldots, v_{n+1}, v_{n+2}) \in v_1 \vee (\nexists x\ (v_2, \ldots, v_{n+1}, x) \in v_1 \wedge v_{n+2} = \bar{u})\text{”} \\
&\quad \text{for } n \geq 1 \\
\mathcal{F}(u) &:= \text{“}v_1 = \bar{u}\text{”}.
\end{aligned}$$

Now by Definition 4.1.2, the relativization $\varphi^{\mathcal{F}}$ of any AFT-formula φ to $\mathcal{F}$ and the relativization $\Phi^{\mathcal{F}}$ of any set Φ of AFT-formulae to $\mathcal{F}$ are defined. The following theorem now establishes that A^*G interprets AFT:

Theorem 4.2.20. $A^*G \vdash AFT^{\mathcal{F}}$.

Proof. We have to prove that for every axiom φ of AFT, $\varphi^{\mathcal{F}}$ can be proved in A^*G. In the proof we often informally describe the relativization $\varphi^{\mathcal{F}}$ of some AFT-formula φ. In these informal descriptions, we usually simplify the restriction on the quantifiers from “x is a Φ-map or $x = \bar{u}$” to “x is a Φ-map”. In every such case, it can easily be seen that the case “$x = \bar{u}$” does not pose any problems to the argument (bearing in mind that $\bar{u}$ is not a Φ-map and hence not a Φ-function nor an element of a tuple of a Φ-map).

Extensionality

We have to show that for any $n \geq 1$, any two Φ-maps of arity n that take the same values on the same n-tuples of Φ-maps are equal. It is easily seen that this follows from Lemma 4.2.17.

Map Comprehension

Assume that $P(\bar{z})$ and $R(\bar{z}, x)$ are AFT-formulae such that for all Φ-maps $\bar{z}$, x such that $R^{\mathcal{F}}(\bar{z}, x)$, x and all z_i's are Φ-functions, and such that for any Φ-maps $\bar{z}$ such that $P^{\mathcal{F}}(\bar{z})$, there is a Φ-map x such that $R^{\mathcal{F}}(\bar{z}, x)$. Then we have to show that there is a Φ-map f such that f is defined precisely on those $\bar{z}$ such that $P^{\mathcal{F}}(\bar{z})$ and such that if $\bar{z}$ are such that $P^{\mathcal{F}}(\bar{z})$, then there is an x such that $\bar{z}^\frown x \in f$ and $R^{\mathcal{F}}(\bar{z}, x)$.

For this we first apply A^*G's Class Comprehension to the statement “there are Φ-functions $\bar{z}$ such that x is the set of all tuples of the form $\bar{z}^\frown y$ where y is a Φ-function of minimal possible rank satisfying $P^{\mathcal{F}}(\bar{z})$ and $R^{\mathcal{F}}(\bar{z}, y)$” and call the resulting class A (the rank-restriction is necessary, since else one could not show that every such x must be a set, which is necessary for applying Class Comprehension). Next we apply Global Choice to A, in order to choose one tuple from each tuple set in this class, thus forming the desired f. f clearly has all the properties that we required (it is a Φ-map, because all elements of tuples in it are Φ-functions, i.e. in a Φ_α for some ordinal number α, and for an ordinal number α, Φ_α is always Φ-transitive and always contains only functional classes).

Functionality

Assume that $P(\bar{z})$ and $R(\bar{z}, x)$ are *AFT*-formulae with limited parameters, not containing the symbol F and such that for all Φ-maps $\bar{z}$, x such that $R^{\mathcal{F}}(\bar{z}, x)$, x and all z_i's are Φ-functions. Furthermore assume that f is a Φ-map defined precisely on those $\bar{z}$ such that $P^{\mathcal{F}}(\bar{z})$ and such that if $\bar{z}$ are such that $P^{\mathcal{F}}(\bar{z})$, then there is an x such that $\bar{z}^\frown x \in f$ and $R^{\mathcal{F}}(\bar{z}, x)$. We have to show that f is a Φ-function.

We define A as above, only that now we may use Set Comprehension rather than Class Comprehension for that definition, so that this time A is a set. So $f \subseteq \bigcup A$ is also a set and hence a Φ-function by 4.2.18.

Element Axiom

This directly follows from the fact that an element of a tuple of a Φ-function is a Φ-function.

Subfunction Axiom

We need to show that a subclass of a Φ-function is a Φ-function. This directly follows from the fact that for an ordinal number α, Φ_α is closed under subclasses.

Undefinedness

This directly follows from Lemma 4.2.17 and from the fact that $\bar{u}$ is not a Φ-map. □

Corollary. *AFT is equiconsistent with ZFC.*

4.3 Class-Map-Tuple-Number Theory

In this section we present two theories, *Class-Map-Tuple-Number Theory* (*CMTN*) and *Class-Map-Tuple Theory* (*CMT*), that formalize various kinds of foundational building blocks of mathematics as primitives, namely classes, maps, tuples, Booleans and in the case of *CMTN* also natural numbers. The restrictions on class and map comprehension imposed by these theories are taken from Ackermann set theory and *AFT*.

The language of *CMTN* consists of

- a unary relation symbol C for classes,
- a binary relation symbol $\in$ for membership in a class,
- a binary relation symbol M for maps of a specified arity
- a constant symbol u for undefinedness,
- for every $n \geq 1$, an n+1-ary function symbol app_n for application of an n-ary map to its arguments,
- a binary relation symbol T for tuples of a specified length,

- for every $n \geq 2$, an n-ary function symbol τ_n that maps n objects to the n-tuple consisting of them,
- a binary function symbol *nth* that maps a tuple and a natural number n to the n-th element of the tuple,
- a unary relation symbol N for natural numbers,
- a constant symbol 0 for the natural number 0,
- a constant symbol s for the successor function on the natural numbers,
- a unary relation symbol B for Booleans,
- a constant symbol $\top$ for truth,
- a constant symbol $\bot$ for falsity,
- a unary relation symbol U for urelements[3], and
- a unary relation symbol L for limited objects.

We use the following notational conventions when they do not cause problems:

- We write $f(\bar{x})$ instead of $app_n(f, \bar{x})$.
- We write $(x_1, \ldots, x_n)$ instead of $\tau_n(x_1, \ldots, x_n)$.
- We write n' instead of $app_1(s, n)$.

By recursion one can define a L_{CMTN}-term $\bar{n}$ for every natural number n in such a way that $\bar{0}$ is the constant symbol 0 and $\overline{n+1}$ is $\bar{n}'$. When this does not cause confusion, we usually simply write n for $\bar{n}$.

The axioms of *CMTN* are as follows:

Class axioms (a variation of the axioms of A_U)

- Class Extensionality Axiom:
  $\forall x, y\ (C(x) \wedge C(y) \wedge \forall z\ (z \in x \leftrightarrow z \in y) \rightarrow x = y)$
- Class Comprehension Axiom Schema: Given a formula $F(y)$ (possibly with parameters) that does not have x among its free variables, the following is an axiom:
  $\forall y\ (F(y) \rightarrow L(y)) \rightarrow \exists x\ (C(x) \wedge \forall y\ (y \in x \leftrightarrow F(y)))$
- Set Comprehension Axiom Schema: Given a formula $F(y)$ that has limited parameters, does not have x among its free variables and does not contain the symbol L, the following is an axiom:
  $\forall y\ (F(y) \rightarrow L(y)) \rightarrow \exists x\ (C(x) \wedge L(x) \wedge \forall y\ (y \in x \leftrightarrow F(y)))$
- Element Axiom: $\forall x, y\ (L(y) \wedge x \in y \rightarrow L(x))$
- Subset Axiom: $\forall x, y\ (L(y) \wedge \forall z\ (z \in x \rightarrow z \in y) \rightarrow L(x))$
- Classness Axiom: $\forall x, y\ (x \in y \rightarrow C(y))$
- Element Definedness Axiom: $\forall x, y\ (x \in y \rightarrow x \neq u)$

[3]In *CMTN*, urelements are elements of the domain that are neither classes, maps, tuples, natural numbers nor Booleans.

Map axioms (a variation of the axioms of *AFT*)

- Map Extensionality Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f\ \forall g\ (M(f,n) \wedge M(g,n) \wedge \forall \bar{z}\ f(\bar{z}) = g(\bar{z}) \rightarrow f = g)$

- Map Comprehension Axiom Schema: Given formulae $P(\bar{z})$ and $R(\bar{z},x)$ with parameters, the following is an axiom:
$\forall \bar{z}\ \forall x\ (R(\bar{z},x) \rightarrow L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \wedge\ \forall \bar{z}\ (P(\bar{z}) \rightarrow \exists x\ R(\bar{z},x)) \rightarrow$
$\exists f\ (M(f,n) \wedge \forall \bar{z}\ (P(\bar{z}) \rightarrow R(\bar{z}, f(\bar{z}))) \wedge \forall \bar{z}\ (\neg P(\bar{z}) \rightarrow f(\bar{z}) = u))$

- Functionality Axiom Schema: Given formulae $P(\bar{z})$ and $R(\bar{z},x)$ that have limited parameters and do not contain the symbol L, the following is an axiom:
$\forall \bar{z}\ \forall x\ (R(\bar{z},x) \rightarrow L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \rightarrow \forall f\ (M(f,n) \wedge \forall \bar{z}\ (P(\bar{z}) \rightarrow$
$R(\bar{z}, f(\bar{z}))) \wedge \forall \bar{z}\ (\neg P(\bar{z}) \rightarrow f(\bar{z}) = u) \rightarrow L(f))$

- Element Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f\ \forall \bar{z}\ (L(f) \wedge f(\bar{z}) \neq u \rightarrow L(z_1) \wedge \ldots \wedge L(z_n) \wedge L(f(\bar{z})))$

- Subfunction Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f\ \forall g\ (L(g) \wedge M(g,n) \wedge M(f,n) \wedge \forall \bar{z}\ (f(\bar{z}) \neq u \rightarrow f(\bar{z}) = g(\bar{z})) \rightarrow L(f))$

- Mapness Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f\ \forall \bar{z}\ (app_n(f,\bar{z}) \neq u \rightarrow M(f,n))$

- Arity Uniqueness Axiom: $\forall n,m\ (M(f,n) \wedge M(f,m) \rightarrow n = m)$

- Undefinedness Axiom Schema: For $n,m \geq 0$ with $n+m \geq 1$:
$\forall x_1,\ldots,x_n,y_1,\ldots,y_m\ app_{m+n}(x_1,\ldots,x_n,u,y_1,\ldots,y_m) = u$

Tuple axioms

- Tuple Element Axiom Schema: For $1 \leq m \leq n$ and $n \geq 2$:
$\forall x_1,\ldots,x_n \neq u\ nth(m,(x_1,\ldots,x_n)) = x_m$

- Tuple Identity Axiom Schema: For $n \geq 2$:
$\forall x\ (T(x,n) \rightarrow x = (nth(1,x),\ldots,nth(n,x)))$

- Tupleness Axiom Schema: For $n \geq 2$: $\forall x_1,\ldots,x_n \neq u\ T((x_1,\ldots,x_n),n)$

- Tuple Undefinedness Axiom Schema: For $n,m \geq 0$ with $n+m \geq 1$:
$\forall x_1,\ldots,x_n,y_1,\ldots,y_m\ (x_1,\ldots,x_n,u,y_1,\ldots,y_m) = u$

- Domain of *nth* Axiom: $\forall x,y\ (nth(x,y) \neq u \rightarrow N(x) \wedge \exists n\ (T(y,n) \wedge \forall c\ (x \in c \wedge \forall z\ (z \in c \rightarrow z' \in c) \rightarrow n \in c)))$[4]

- Limited Tuples Axiom Schema: For $n \geq 2$: $\forall x_1,\ldots,x_n\ (L((x_1,\ldots,x_n)) \leftrightarrow L(x_1) \wedge \ldots \wedge L(x_n))$

[4]This axiom really just says that if the x-th element of y is defined, then x is a natural number and y is a tuple of some length $n \geq x$. But since $\geq$ is not a primitive in the language of *CMTN*, $n \geq x$ has to be expressed in a non-primitive way as $\forall c\ (x \in c \wedge \forall z\ (z \in c \rightarrow z' \in c) \rightarrow n \in c)$.

Natural number axioms (Peano axioms)

- $N(0)$
- $\forall n\ (N(n) \to N(n'))$
- $\forall x\ (x' \neq 0)$
- $\forall n, m\ (N(n) \wedge N(m) \wedge n' = m' \to n = m)$
- Induction Axiom: $\forall x\ (C(x) \wedge 0 \in x \wedge \forall n\ (N(n) \wedge n \in x \to n' \in x) \to \forall n\ (N(n) \to n \in x))$
- Domain of s Axiom: $\forall x\ (x' \neq u \to N(x))$
- Limitedness of Numbers: $\forall x\ (N(x) \to L(x))$

Boolean axioms

- $\forall x\ (B(x) \leftrightarrow x = \top \vee x = \bot)$
- $\top \neq \bot$
- Limitedness of Booleans: $\forall x\ (B(x) \to L(x))$

General axioms

- Sort Disjointness Axiom: "For every x, at most one of $C(x)$, $\exists y\ M(x, y)$, $\exists y\ T(x, y)$, $N(x)$, $B(x)$, $U(x)$ and $x = u$ holds."
- Arity Axiom: $\forall x, y\ (M(x, y) \vee T(x, y) \to N(y))$
- Arity Uniqueness Axiom: $\forall x, y, z\ (M(x, y) \wedge M(x, z) \to y = z)$
- Tuple-Length Uniqueness Axiom: $\forall x, y, z\ (T(x, y) \wedge T(x, z) \to y = z)$
- Limitedness of Urelements: $\forall x\ (U(x) \to L(x))$
- Unlimitedness of Undefinedness: $\neg L(u)$

We will now show that *CMTN* is equiconsistent with *ZFC*. Clearly *CMTN* interprets A_U and hence *ZFC*. For the other direction, we will show that $A^*_U G$ interprets *CMTN* by adapting the proof of Theorem 4.2.20.

First we need some disjoint encodings of classes, maps, tuples, numbers and the undefinedness constant from *CMTN* in $A^*_U G$:

Definition 4.3.1. Define u^Ψ to be $(0, 0)$.

Definition 4.3.2. Given a class a, define a *Ψ-class over a* to be a pair of the form $(c, 1)$, where $c \subseteq a$.

Definition 4.3.3. Given a class a, define an *n-ary Ψ-map over a* to be a pair of the form $(f, 2)$, where f is a a functional class of n+1-tuples of elements of a.

Definition 4.3.4. Given a class a, define a *Ψ-n-tuple over a* to be a pair of the form $(t, 3)$, where t is an n-tuple of elements of a.

Definition 4.3.5. Define a Ψ-*number* to be a pair of the form $(n, 4)$, where $n \in \omega$.

Definition 4.3.6. Define a Ψ-*Boolean* to be a pair of the form $(b, 5)$, where $b = 0$ or $b = 1$.

Definition 4.3.7. Define a Ψ-*urelement* to be a pair of the form $(x, 6)$, where x is an urelement.

Now we want to define the closure of a class a under Ψ-tuples. This can only be ensured to exist if a is a set, but we do not make the restriction that a is a set in the definition, and for some classes it will be defined too.

Definition 4.3.8. Given a class a and $i \in \omega$, we recursively define $T_i(a)$ as follows:

$$\begin{aligned} T_0(a) &:= a \\ T_{n+1}(A) &:= \{t \mid \text{for some } m,\ t \text{ is a } \Psi\text{-}m\text{-tuple over } T_n(a)\} \end{aligned}$$

Definition 4.3.9. Given a class a, we define the closure of a under Ψ-tuples to be

$$T(a) := \bigcup_{i \in \omega} T_i(a).$$

In $A^*_U G$, we define a *cumulative hierarchy of CMTN-encodings*. The intended definition is the following:

$$\begin{aligned} \Psi_0 &:= \emptyset \\ \Psi_{\alpha+1} &:= T(\{x \mid x \text{ is a } \Psi\text{-number, } \Psi\text{-Boolean or } \Psi\text{-urelement, or a } \Psi\text{-class or } \Psi\text{-map over } \Psi_\alpha\}) \\ \Psi_\lambda &:= \bigcup_{\alpha < \lambda} \Psi_\alpha \end{aligned}$$

Just as we did with the hierarchies of V_α's and Φ_α's, it can be ensured that this is defined for all ordinal numbers without using the predicate M in the definition, which ensures that this hierarchy also extends to some extent into the proper class ordinals.

Definition 4.3.10. x is a Ψ-*element* of y iff either y is a Ψ-class $(c, 1)$ and $x \in c$, or y is a Ψ-map $(f, 2)$ and x is an element of a tuple in f, or y is a Ψ-tuple $(t, 3)$ and x is an element of the tuple t.

Definition 4.3.11. A class A is Ψ-*transitive* iff every Ψ-element of an element of A is an element of A.

Definition 4.3.12. x is Ψ-*limited* iff there is an ordinal number α such that $x \in \Psi_\alpha$.

Definition 4.3.13. x is a Ψ-*object* iff x is a Ψ-class, a Ψ-map, a Ψ-tuple, a Ψ-number, a Ψ-Boolean or a Ψ-urelement.

Definition 4.3.14. x is a *pure* Ψ-*object* iff x is a Ψ-object, and for every Ψ-element y of x, there is an ordinal α such that Ψ_α is Ψ-transitive, every element of Ψ_α is a Ψ-object and $y \in \Psi_\alpha$.

Lemma 4.3.15. *Every Ψ-element of a pure Ψ-object is a pure Ψ-object.*

Proof. Let x be a pure Ψ-object and y be a Ψ-element of x. Then there is an ordinal α such that Ψ_α is Ψ-transitive, every element of Ψ_α is a Ψ-object and $y \in \Phi_\alpha$. Then y is a Ψ-object, and if z is a Ψ-element of y, then $z \in \Psi_\alpha$ by the Ψ-transitivity of Ψ_α, so y is a pure Ψ-object as required. □

Lemma 4.3.16. *Every pure Ψ-object that is a set is Ψ-limited.*

Proof. Analogously to the proof of Lemma 4.2.18. □

Definition 4.3.17. Let $\mathcal{G}$ be the A_U^*G-definable L_{CMTN}-structure defined by

$\mathcal{G}(\mathbb{U}) :=$ "x is a pure Ψ-object or $x = u^\Psi$"
$\mathcal{G}(=) := v_1 = v_2$
$\mathcal{G}(C) :=$ "v_1 is a Ψ-class"
$\mathcal{G}(\in) :=$ "$\exists x\ (v_2 = (x, 1) \wedge v_1 \in x)$"
$\mathcal{G}(M) :=$ "v_1 is a v_2-ary Ψ-map"
$\mathcal{G}(u) :=$ "$v_1 = u^\Psi$"
$\mathcal{G}(app_n) :=$ "$\exists f\ (v_1 = (f, 2) \wedge (v_2, \ldots, v_{n+1}, v_{n+2}) \in f) \vee (\nexists f, x\ (v_1 = (f, 2) \wedge (v_2, \ldots, v_{n+1}, x) \in f) \wedge v_{n+2} = u^\Psi)$" for $n \geq 1$
$\mathcal{G}(T) :=$ "v_1 is a Ψ-v_2-tuple"
$\mathcal{G}(\tau_n) :=$ "$v_{n+1} = ((v_1, \ldots, v_n), 3)$" for $n \geq 2$
$\mathcal{G}(nth) :=$ "there is a tuple t of length at at least v_2 such that $v_1 = (t, 3)$ and v_3 is the v_2-th element of t, or there is no such tuple t and $v_3 = u^\Psi$"
$\mathcal{G}(N) :=$ "v_1 is a Ψ-number"
$\mathcal{G}(0) :=$ "$v_1 = (0, 4)$"
$\mathcal{G}(s) :=$ "$v_1 = (\{((n, 4), (n + 1, 4)) \mid n \in \omega\}, 2)$"
$\mathcal{G}(B) :=$ "$v_1 = (0, 5)$ or $v_1 = (1, 5)$"
$\mathcal{G}(\top) :=$ "$v_1 = (1, 5)$"
$\mathcal{G}(\bot) :=$ "$v_1 = (0, 5)$"
$\mathcal{G}(U) :=$ "$\exists x\ (U(x) \wedge v_1 = (x, 6))$"
$\mathcal{G}(L) :=$ "v_1 is Ψ-limited".

Theorem 4.3.18. $A_U^*G \vdash CMTN^\mathcal{G}$.

Proof sketch. We have to prove that for every axiom φ of *CMTN*, $\varphi^\mathcal{G}$ can be proved in A_U^*G. For Class Extensionality and Class and Set Comprehension the proofs are analogous to those for the corresponding axioms in the proof of Theorem 4.1.35. For Map Extensionality, Map Comprehension and Functionality, the proofs are analogous to those of the corresponding axioms in the proof of Theorem 4.2.20. All other axioms follow trivially from the definition of $\mathcal{G}$. □

Corollary. *CMTN is equiconsistent with ZFC.*

4.3.1 Class-Map-Tuple Theory

When removing natural numbers from *CMTN*, it is not enough to just remove the Peano axioms from *CMTN*'s axiom list, since natural numbers are also used to formalize arity and tuple length in *CMTN*. So instead of binary predicates M and T, we now have unary predicates M, T, M_n for $n \geq 1$, and T_n for $n \geq 2$. The axioms of *CMT* consist of the axioms of *CMTN* without the Peano axioms, with $\exists n\ M(x,n)$, $\exists n\ T(x,n)$, $M(x,n)$ and $T(x,n)$ replaced by $M(x)$, $T(x)$, $M_n(x)$ and $T_n(x)$ respectively, and with two axiom schemas added:

- For $n \geq 1$: $\forall x\ (M_n(x) \to M(x))$
- For $n \geq 2$: $\forall x\ (T_n(x) \to T(x))$

It is easily seen that *CMTN* interprets *CMT* and that *CMT* interprets A_U, so that *CMT* is equiconsistent with *ZFC*.

4.3.2 *CMTN*-based logic

When using *CMTN* as a background theory in a Naproche text, there are normally some mathematical objects that the text is about and that are not considered classes, maps, tuples or numbers. These objects correspond to the urelements of *CMTN*. But *CMTN* has no axioms about its urelements, whereas mathematical texts often do contain an axiomatization of the properties of the urelements that the text is about. Using *CMTN* to derive other properties of the urelements from these axioms should be conservative over doing this without *CMTN*. In other words, anything that can be proven about urelements using *CMTN* should also be provable directly from the axioms about the urelements. For making this precise, we first need some definitions:

Definition 4.3.19. Given two *PL* languages L_1 and L_2, the *L_2-expansion of L_1*, also called $L_1^{L_2}$, is the language whose signature contains all constants, function symbols and relation symbols of L_1, and additionally contains a constant symbol c_s for every constant, function symbol or relation symbol s of L_2.

Remark. For simplicity and convenience, we usually write s instead of c_s, when this does not lead to unclarity.

Definition 4.3.20. Given two *PL* languages L_1 and L_2 and an L_1-theory T, we write T_{L_2} for the theory T considered as a theory over the extended language $L_1^{L_2}$.

Definition 4.3.21. Let $L = (c_1, \ldots, c_l; f_1^{k_1}, \ldots, f_m^{k_m}; R_1^{k'_1}, \ldots, R_n^{k'_n})$ be a *PL* language (here the superscripts indicate the arities of the function and relation symbols). Define Γ_L to be the following set of L^L_{CMTN} formulae (which – intuitively speaking – formalize the statements that $c_1, \ldots, c_l$ are urelements, that for $1 \leq i \leq m$, $f_i^{k_i}$ is a k_i-ary map on the urelements and that for $1 \leq i \leq n$,

$R_i^{k_i'}$ is a k_i'-ary relation on the urelements):

$$\{U(c_i) \mid 1 \leq i \leq l\} \cup$$
$$\{M(f_i^{k_i}, k_i) \wedge \forall x_1, \ldots, x_{k_i}\ (f_i^{k_i}(x_1, \ldots, x_{k_i}) \neq u \leftrightarrow U(x_i) \wedge \ldots \wedge U(x_{k_i})) \wedge$$
$$\forall x_1, \ldots, x_{k_i}\ (U(x_i) \wedge \ldots \wedge U(x_{k_i}) \rightarrow U(f_i^{k_i}(x_1, \ldots, x_{k_i}))) \mid 1 \leq i \leq m\} \cup$$
$$\{M(R_i^{k_i}, k_i) \wedge \forall x_1, \ldots, x_{k_i}\ (R_i^{k_i}(x_1, \ldots, x_{k_i}) \neq u \leftrightarrow U(x_i) \wedge \ldots \wedge U(x_{k_i})) \wedge$$
$$\forall x_1, \ldots, x_{k_i}\ (U(x_i) \wedge \ldots \wedge U(x_{k_i}) \rightarrow B(R_i^{k_i}(x_1, \ldots, x_{k_i}))) \mid 1 \leq i \leq n\}.$$

Definition 4.3.22. Let $L = (c_1, \ldots, c_l; f_1^{k_1}, \ldots, f_m^{k_m}; R_1^{k_1'}, \ldots, R_n^{k_n'})$ be a *PL* language. Let $\mathcal{A}_L$ be the $CMTN_L$-definable L-structure defined by

$$\begin{aligned}
\mathcal{A}_L(\mathbb{U}) &:= U(x) \\
\mathcal{A}_L(=) &:= v_1 = v_2 \\
\mathcal{A}_L(c_i) &:= v_1 = c_i \text{ for } 1 \leq i \leq l \\
\mathcal{A}_L(f_i^{k_i}) &:= v_{k_i+1} = app_{k_i}(f_i^{k_i}, v_1, \ldots, v_{k_i}) \text{ for } 1 \leq i \leq m \\
\mathcal{A}_L(R_i^{k_i'}) &:= app_{k_i}(R_i^{k_i'}, v_1, \ldots, v_{k_i}) = \top \text{ for } 1 \leq i \leq n
\end{aligned}$$

Now we can make precise what we meant in the introduction of this section by the conservativity of working with *CMTN* as a background theory over working without *CMTN*:

> **Conservativity of *CMTN*:** Let L be a *PL* language. Let φ be an L-formula and Γ be a finite set of L-formulae. If $CMTN_L \cup \Gamma_L \cup \Gamma^{\mathcal{A}_L} \models \varphi^{\mathcal{A}_L}$, then $\Gamma \models \varphi$.

We will need this conservativity result for the proof of Theorem 6.3.24 in chapter 6, which is one of the two soundness theorems of the proof checking algorithm.

This conservativity statement implies the consistency of *CMTN* and hence the consistency of *ZFC*, so it certainly cannot be proved without some assumption at least as strong as *Con*(*ZFC*) ("*ZFC* is consistent"). Actually, the assumption we need for our proof of the statement is somewhat stronger than *Con*(*ZFC*) (which by Gödel's completeness theorem is equivalent to the existence of a model of *ZFC*), namely that *ZFC* has an ω-model:

Definition 4.3.23. A model M of *ZFC* is called an ω-model iff ω^M (i.e. the collection of natural numbers of M) is isomorphic to the actual natural numbers.

The assumption of the existence of an ω-model of *ZFC* is much weaker than the assumption that there is an inaccessible cardinal. Additionally, just like the assumption *Con*(*ZFC*) and unlike the assumption that there is an inaccessible cardinal, the assumption of the existence of an ω-model of *ZFC* does not presuppose our metatheory to be *ZFC*. Indeed, our result can be formulated in much weaker metatheories, e.g. in second-order arithmetic or sufficiently strong subsystems thereof.

Before we prove the theorem that the conservativity result follows from the existence of an ω-model of *ZFC*, we first need two lemmas and a definition:

Lemma 4.3.24. *Let M be an ω-model of ZFC. Let L be a PL language, let Γ be a finite set of L-formulae and let φ be a PL formula. Then $\Gamma \models \varphi$ iff $M \models \ulcorner\Gamma \models \varphi\urcorner$.*

Remark. For any informal mathematical statement P for which there is a canonical formalization in the language of set theory, $\ulcorner P\urcorner$ denotes this canonical formalization.

Proof. By Gödel's Completeness Theorem, there is a proof calculus p such that $\Gamma \models \varphi$ is equivalent to $\Gamma \vdash_p \varphi$. Viewing L-formulae and p-proofs as their Gödelizations, i.e. as natural numbers, $\Gamma \vdash_p \varphi$ can be considered a statement about natural numbers (of the form "there exists a natural number that is the Gödelization of the proof of φ from Γ"). Since the natural numbers are isomorphic to ω^M, $\Gamma \vdash_p \varphi$ is equivalent to $M \models \ulcorner\Gamma \vdash_p \varphi\urcorner$. But since Gödel's Completeness Theorem can be proved within M, $M \models \ulcorner\Gamma \vdash_p \varphi\urcorner$ is equivalent to $M \models \ulcorner\Gamma \models \varphi\urcorner$. □

Definition 4.3.25. Define *ZFCU* to be "*ZFC* with urelements", i.e. the theory whose language contains a binary predicate $\in$ and a unary predicate U, and whose axioms are as follows:

- The axioms of *ZFC* with some quantifiers restricted by $\neg U$:
  - Extensionality:
    $\forall x, y\ (\neg U(x) \wedge \neg U(y) \rightarrow (x = y \leftrightarrow \forall a\ (a \in x \leftrightarrow a \in y)))$
  - Empty set: $\exists x\ (\neg U(x) \wedge \forall a\ \neg a \in x)$
  - Separation and Replacement: These axiom schemes have to be extended to apply to all L_{ZFCU}-formulae and not just to $\in$-formulae. The quantifiers in these axioms do not get restricted by $\neg U$.
  - All other *ZFC* axioms are taken over to *ZFCU* without any changes.
- An axiom stating that the urelements do not have any elements:
  $\forall a\ (U(a) \rightarrow \neg\exists b\ b \in a)$

Remark. Note that with this definition of *ZFCU*, the Powerset Axiom states that given a set x, there is a set y containing all subsets of x and all urelements. One can use Separation on y to construct the usual powerset. Additionally, one can use Separation on y to establish that there is a set that contains precisely all the urelements.

The following lemma establishes that working with *ZFCU* as a background theory is conservative over working without *ZFCU*:

Lemma 4.3.26. *Suppose that ZFC has an ω-model. Let L be a PL language, let Γ be a finite set of L-formulae and let φ be an L-formula. If $ZFCU_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$, then $\Gamma \models \varphi$.*

Remark. Note that the formulae in Γ_L, the formulae in $\Gamma^{\mathcal{A}_L}$ and $\varphi^{\mathcal{A}_L}$ do not contain the symbol L, and that hence the formulae in $\Gamma_L^{\mathcal{G}}$, the formulae in $(\Gamma^{\mathcal{A}_L})^{\mathcal{G}}$ and $(\varphi^{\mathcal{A}_L})^{\mathcal{G}}$ do not contain the symbol M, i.e. are actually L^{ZFCU}-formulae.

Proof. Let M be an ω-model of *ZFC*. Suppose $ZFCU_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$. By compactness we may assume that there is a finite set $ZFCU'_L \subseteq ZFCU_L$ of L^{ZFCU}-formulae such that $ZFCU'_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$. Note that by Lemma 4.3.24, this means that $M \models \ulcorner ZFCU'_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}} \urcorner$.

We need to show that $\Gamma \models \varphi$. By Lemma 4.3.24, it is enough to show that $M \models \ulcorner \Gamma \models \varphi \urcorner$. For the rest of the proof, we argue within M; so the goal is to show $\Gamma \models \varphi$.

Suppose that $S \models \Gamma$. We need to show that $S \models \varphi$. Let $\mathcal{B}$ be the *ZFC*-definable L_{ZFCU}-structure defined as follows:

$$\begin{aligned}\mathcal{B}(\mathbb{U}) &:= x = x\\ \mathcal{B}(=) &:= v_1 = v_2\\ \mathcal{B}(\in) &:= v_1 \in v_2\\ \mathcal{B}(U) &:= v_1 \in S\end{aligned}$$

Note that given any L_{ZFCU}-formula ψ, $\psi^{\mathcal{B}}$ is a parametrized $\in$-formula, with S as parameter. Now by the reflection principle, there is an ordinal α such that the formulae in $(ZFCU'_L)^{\mathcal{B}}$ are absolute for V_α. We want to additionally ensure that $S \subseteq V_\alpha$, which can be attained by adding a formula with a second parameter x, namely the formula $x \in S$, to the finite set of formulae that we apply reflection to. Then $x \in S$, with both x and S considered as parameters, is absolute for V_α, which means that $\forall x\, ((x \in S)^{V_\alpha} \leftrightarrow x \in S)$, i.e. $\forall x\, (x \in S \cap V_\alpha \leftrightarrow x \in S)$, which implies that $S \subseteq V_\alpha$, as intended.

Note that for any $\varphi \in ZFCU_L$, $V \models \varphi^{\mathcal{B}}$. Since the formulae in $(ZFCU'_L)^{\mathcal{B}}$ are absolute for V_α, it follows that $V_\alpha \models (ZFCU'_L)^{\mathcal{B}}$.

We now build from S a larger L^{ZFCU}-structure S' that models $ZFCU'_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}}$. The domain of S' is V_α. The signature of L^{ZFCU} is interpreted as follows in S':

$$\begin{aligned}U^{S'} &:= S\\ \in^{S'} &:= \{(x, y) \in V_\alpha(S)^2 \mid x \in y\}\\ c^{S'} &:= c^S \text{ for every constant symbol } c \text{ of } L\\ f^{S'} &:= (f^S, 2) \text{ for every function symbol } f \text{ of } L\\ R^{S'} &:= (\{(x_1, \ldots, x_n, (1,5)) \mid (x_1, \ldots, x_n) \in R^S\} \cup \{(x_1, \ldots, x_n, (0,5)) \mid\\ &\quad (x_1, \ldots, x_n) \in S^n \setminus R^S\}) \text{ for every } n\text{-ary relation symbol } R \text{ of } L.\end{aligned}$$

Since $V_\alpha \models (ZFCU'_L)^{\mathcal{B}}$ and since S' and $\mathcal{B}$ agree about their interpretation of U, it follows that $S' \models ZFCU'_L$.

From the definition of $c^{S'}$, $f^{S'}$, $R^{S'}$, Γ_L and $\mathcal{G}$, it directly follows that $S' \models \Gamma_L^{\mathcal{G}}$.

One can easily see that $S' \models (\Gamma^{\mathcal{A}_L})^{\mathcal{G}}$ is just a more complicated way of formalizing $S \models \Gamma$ in *ZFC* (complicated for example by the fact that functions are not formalized simply as sets of tuples, but instead as tuples of the form $(f, 2)$, where f is a set of tuples; this kind of complication results from the use of $\mathcal{G}$ in this reinterpretation of Γ). Hence we can conclude that $S' \models (\Gamma^{\mathcal{A}_L})^{\mathcal{G}}$. Together with the previously established facts, this means that $S' \models ZFCU'_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}}$.

Since $ZFCU'_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$, it now follows that $S' \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$. But $S' \models (\varphi^{\mathcal{A}_L})^{\mathcal{G}}$ is just a more complicated formalization of $S \models \varphi$, so we may conclude $S \models \varphi$ as required. □

Remark. By doing the main work of this proof inside the ω-model M of *ZFC*, we did not only avoid committing ourselves to having *ZFC* as our metatheory, but also made it possible that this lemma does not take the form of a theorem schema (because of the application of the reflection theorem schema), but can actually be stated as a single theorem.

Theorem 4.3.27. *Suppose that ZFC has an ω-model. Let L be a PL language. Let φ be an L-formula and Γ be a finite set of L-formulae. If $CMTN_L \cup \Gamma_L \cup \Gamma^{\mathcal{A}_L} \models \varphi^{\mathcal{A}_L}$, then $\Gamma \models \varphi$.*

Proof. First we need an adaptation of the proof of Theorem 4.1.10 and its corollary to theories with urelements. For this, one needs to define a hierarchy of V_α^U's in *ZFCU*, analogous to the hierarchy of V_α's in *ZFC*, but with V_α^U containing not only the subsets of previous steps in the hierarchy, but also all urelements. Now one can formulate a reflection principle for *ZFCU* using this hierarchy. Adapting the proof of Theorem 4.1.10 (actually the simplified version mentioned in the remark to the theorem) and the proof of its corollary, one can prove that for any formula φ of *ZFCU*,

$$A_U^* G \models \varphi \text{ implies } ZFCU \models \varphi. \tag{4.5}$$

Now suppose $CMTN_L \cup \Gamma_L \cup \Gamma^{\mathcal{A}_L} \models \varphi^{\mathcal{A}_L}$. By compactness, there is a finite $\Gamma'^{\mathcal{A}_L} \subseteq \Gamma^{\mathcal{A}_L}$ such that $CMTN_L \cup \Gamma_L \cup \Gamma'^{\mathcal{A}_L} \models \varphi^{\mathcal{A}_L}$. Let ψ be the formula $\bigwedge \Gamma_L \wedge \bigwedge \Gamma'^{\mathcal{A}_L} \to \varphi$. Then $CMTN_L \models \psi$. Now by Theorem 4.3.18, $A_U^* G_L \models \psi^{\mathcal{G}}$. Then by (4.5), $ZFCU_L \models \psi^{\mathcal{G}}$, i.e. $ZFCU_L \cup \Gamma_L^{\mathcal{G}} \cup (\Gamma^{\mathcal{A}_L})^{\mathcal{G}} \models \varphi^{\mathcal{G}}$. Now Lemma 4.3.26 implies that $\Gamma \models \varphi$. □

Chapter 5

Dynamic formalisms for mathematics

In this chapter we describe two extensions of *DPL*, Higher-Order Dynamic Predicate Logic (*HODPL*) and Proof Text Logic (*PTL*). Both add to *DPL* features that make the formalisms more expedient at representing the content, the dynamic properties and the hierarchical structure of mathematical texts.

5.0.1 Currying and uncurrying

In order to explain the semantics of these extensions of *DPL*, we need the syntactic technique of *currying*. This is a technique of transforming a multi-argument function into a function of lower arity, returning a function as its value. To explain the details, we first need to fix some notation:

Given two sets A and B, we let $A \to B$ denote the set of functions from A to B. The usual notation $f : A \to B$ for a function f from A to B can now be considered an alternative notation for $f \in A \to B$.

Now given a binary function $f : A_1 \times A_2 \to B$, we can define a function $cur(f) : A_1 \to (A_2 \to B)$, called the curried form of f, as follows:

$$cur(f)(a_1)(a_2) := f(a_1, a_2).$$

In the case of a function $f : A_1 \times \cdots \times A_n \to B$ of a higher arity than 2, the usual definition of the curried form of f is the iteratively curried function of type $A_1 \to (A_2 \to \cdots \to (A_n \to B)\dots)$. In this thesis, however, we will need a different notion of currying of higher-arity functions: Given $n, m \geq 1$ and an $n+m$-ary function $f : A_1 \times \cdots \times A_{n+m} \to B$, we define $cur_{m,n}(f) : A_1 \times \cdots \times A_n \to (A_{n+1} \times \cdots \times A_{n+m} \to B)$ as follows:

$$cur_{m,n}(f)(a_1, \dots, a_n)(a_{n+1}, \dots, a_{n+m}) := f(a_1, \dots, a_{n+m}).$$

Given this definition, $cur_{1,1}$ is the same as the *cur* defined above.

The converse of currying is *uncurrying*: Given a function $f : A_1 \times \cdots \times A_n \to (A_{n+1} \times \cdots \times A_{n+m} \to B)$, we define $unc_{m,n}(f) : A_1 \times \cdots \times A_{n+m} \to B$ by

$$unc_{m,n}(f)(a_1, \dots, a_{n+m}) := f(a_1, \dots, a_n)(a_{n+1}, \dots, a_{n+m}).$$

5.1 Higher-Order Dynamic Predicate Logic

Higher-Order Dynamic Predicate Logic (*HODPL*) extends *DPL* to a higher-order system that formalizes the implicit dynamic introduction of functions discussed in section 3.3 of chapter 3. To our knowledge it is the first system that formalizes implicit dynamic function introduction.

Many systems for higher-order logic make use of types in order to syntactically restrict the possible arguments for a given function term. In *HODPL* we do not realize this restriction in a syntactic way using types, but by integrating a formal account of presuppositions into the system.[1] Presuppositions are also used to account for the difference between functions and relations. Additionally, *HODPL* contains an ι operator that formalizes definite descriptions, and whose semantics essentially depends on the treatment of presuppositions in *HODPL*.

HODPL has three logical relation symbols: the binary $=$ for equality, the unary U for urelements and the unary B for Booleans. *HODPL* syntax does not depend on a signature, as we do not allow for constant, function and relation symbols other than $=$, U and B. We will show in section 5.1.2 below how variables in *HODPL* can be used to mimic constant, function and relation symbols. We define *HODPL* syntax by defining *HODPL* terms and *HODPL* formulae via a simultaneous recursion:

Definition 5.1.1. An *HODPL term* is of the form x, $t_0(t_1, \ldots, t_n)$ or $\iota x\, \varphi$ for a variable x, *HODPL* terms t_0, $\ldots$, t_n and an *HODPL* formula φ. Terms not containing ι are called *ι-free terms*. We write T_{HODPL} for the set of ι-free terms.

An *HODPL formula* is of one of the following forms, where t_1, t_2 are *HODPL* terms, t is an ι-free *HODPL* term and φ and ψ are *HODPL* formulae:

- t_1
- $\top$
- $t_1 = t_2$
- $U(t_1)$
- $B(t_1)$
- $\neg\varphi$
- $(\varphi \wedge \psi)$
- $(\varphi \vee \psi)$
- $(\varphi \rightarrow \psi)$
- $\exists t\, \varphi$

[1] However, one can also develop Typed Higher-Order Dynamic Predicate Logic, which realizes this restrictions by adopting a type-theoretic approach (see Cramer, 2012).

- $\Diamond\varphi$
- $\mathrm{def}(t_1)$[2]

Note that *HODPL* terms can be *HODPL* formulae. Since terms cannot be formulae in *PL* or *DPL*, this needs some clarification: The functions that are used to form complex terms may take Booleans as values, in which case they may be considered relations. The application of such a function to an argument is still considered a term, but given this explanation, it is no longer surprising that it can also be considered a formula.

5.1.1 *HODPL* semantics

The most distinctive feature of *HODPL* syntax is that it allows not only variables but any well-formed terms to come after quantifiers. So (1) is an *HODPL* formula, and can be considered the *HODPL* formalization of (2):

(1) $\exists x\, U(x) \rightarrow \exists f(x)\, R(x, f(x))$

(2) For every urelement x, there is an $f(x)$ such that $R(x, f(x))$.

But what is the intended semantics of (1)? The truth conditions of (1) should turn out to be essentially equivalent to those of (3), but given what we have said about implicit dynamic function introduction in the language of mathematics in section 3.3, unlike (3), (1) dynamically introduces the function symbol f to the context, and should hence be essentially equivalent to (4).

(3) $\exists x\, U(x) \rightarrow \exists y\, R(x, y)$

(4) $\exists f\, (\exists x\, U(x) \rightarrow R(x, f(x)))$

We will come back to this example when clarifying the semantics of *HODPL* after its formal definition.

Since *HODPL* syntax does not depend on a signature, we will not need structures that give meaning to the constant, function and relation symbols of the signature. We only need a domain. But now a domain is no longer any set, but a model of *AFTB* (i.e. *AFT* with Booleans).[3] As in the case of DPL semantics, the interpretation of an *HODPL* formula will be a set of pairs of assignments. But the definition of *assignment* has to be modified:

[2]The intended meaning of $\mathrm{def}(t_1)$ is that the term t_1 is defined.

[3]For those readers who read this chapter without having read chapter 4, we give some explanations about the theory *AFTB* that we make use of here:

Informally speaking, *AFTB* is a theory of functions or – in other words – maps. In order to avoid the inconsistency of unrestricted function comprehension mentioned in section 3.3, it makes use of a distinction between limited and unlimited objects. We use the term *function* only for limited *maps*, so *map* is the more general term in the terminology of this theory. We use the symbol L for the property of being limited.

The theory talks about four kinds of objects:

- Maps: These can be of different arities and are usually not defined on the whole universe of discourse.
- The undefinedness object, designated u: This is the value of a map at arguments where it is not defined.
- The two Booleans $\top$ (*truth*) and $\bot$ (*falsity*). The Booleans are considered limited objects.
- Urelements: *AFTB* is used to model mathematicians talk about maps, which involves not only maps, but also objects which are not maps, but which can be the arguments or

Definition 5.1.2. Given an *AFTB* model M, an *M-assignment* g is a partial function from T_{HODPL} to $M \setminus \{u^M\}$. G_M is the set of M-assignments.

Definition 5.1.3. Given two assignments g and h, we define $g[t_1, \ldots, t_n]h$ to mean that $dom(g) = dom(h) \cup \{t_1, \ldots, t_n\}$ and for all $s \in dom(h) \setminus \{t_1, \ldots, t_n\}$, $g(s) = h(s)$.

Definition 5.1.4. Given elements $x_0, \ldots, x_n$ of an *AFTB*-model M, we write $x_0(x_1, \ldots, x_n)$ for $app_n^M(x_0, x_1, \ldots, x_n)$, and say that $x_0(x_1, \ldots, x_n)$ *is defined* iff $app_n^M(x_0, x_1, \ldots, x_n) \neq u^M$. (This notation is only used when it is clear which *AFTB*-model we are talking about.)

For the rest of this section, we refer to the notation introduced in the previous three definitions together with the usual mathematical notation that we employ in order to talk about *HODPL* terms and formulae, *AFTB*-models and their elements as *the metalanguage*, in order to distinguish it from the language of *HODPL* defined previously.

The most involved part of the definition of *HODPL* semantics is the definition of the semantics of $\varphi \to \psi$. This definition has to account for the fact that functions implicitly introduced in ψ have to be dynamically introduced to the context outside the scope of $\to$. The definition is significantly complicated by the fact that functions with dependencies on more than one variable can be introduced with an arbitrary ordering of the arguments and even with an optional currying over some arguments. We clarify what we mean by this by considering an example:

(5) If $x, y \in \mathbb{R}$, then there are real numbers $f_x(y)$ and $g(y, x)$ such that $R(x, y, f_x(y), g(y, x))$.

After this sentence one can use g as a binary function symbol. Additionally, $f_\bullet$ is now a unary function symbol, whose only argument is written in subscript notation. This $f_\bullet$ is a function from $\mathbb{R}$ to $\mathbb{R}^\mathbb{R}$, i.e. to unary functions from the reals to the reals. One can view $f_\bullet$ as the curried version of the function h that one would have introduced if one had written $h(x, y)$ instead of $f_x(y)$ in (5).

In *HODPL* we do not distinguish the different notations used in the language of mathematics for linking an argument to a function term. Hence we formalize (5) as follows:

(6) $\exists x\, \exists y\, (x \in \mathbb{R} \wedge y \in \mathbb{R}) \to \exists f(x)(y)\, \exists g(y, x)\, (f(x)(y) \in \mathbb{R} \wedge g(y, x) \in \mathbb{R} \wedge R(x, y, f(x)(y), g(y, x)))$

---

value of a map under consideration. The basic mathematical objects that are neither maps nor Booleans or the undefinedness object u are called *urelements*. We use the symbol U for the property of being a urelement. All urelements are considered limited objects.

The definition of the semantics of $\varphi \to \psi$ has to account for the introduction of functions f and g with the right argument structure: The two variables x and y are linked to these function symbols in different ways in the right hand side of (6). We need to introduce some notation that facilitates talking about the ways a fixed list of variables can be linked to a function symbol:

Definition 5.1.5. Let $n \geq 1$. An *n-place argument filler* σ is a pair $(p, (m_i)_{1 \leq i \leq l})$, where p is a permutation on $\{1, \ldots, n\}$ and $m_1, \ldots, m_l$ are natural numbers such that $0 = m_1 < m_2 < \cdots < m_l = n$.

Remark. We write $id_{\{1,\ldots,n\}}$ for the identity permutation on $\{1, \ldots, n\}$. Additionally, we sometimes use the standard cycle notation for permutations, with optional subscripts for indicating the domain. For example, $(2\ 3)_{\{1,2,3\}}$ is the permutation that maps 1 to 1, 2 to 3 and 3 to 2.

Definition 5.1.6. For an n-place argument filler $\sigma = (p, (m_i)_{1 \leq i \leq l})$, n terms $t_1, \ldots, t_n$ and a function term f, we write $f^\sigma(t_1, \ldots, t_n)$ for

$$f(t_{p(m_1+1)}, t_{p(m_1+2)}, \ldots, t_{p(m_2)})(t_{p(m_2+1)}, t_{p(m_2+2)}, \ldots, t_{p(m_3)}) \ldots \\ (t_{p(m_{l-1}+1)}, t_{p(m_{l-1}+2)}, \ldots, t_{p(m_l)}).$$

(Here the terms $f, t_1, \ldots, t_n$ can be either *HODPL* terms or terms of our metalanguage.)

Definition 5.1.7. For an n-place argument filler $\sigma = (p, (m_i)_{1 \leq i \leq l})$ and a function term f of our metalanguage, we say that f is *σ-defined at $a_1, \ldots, a_m$* iff there is an $i \leq l$ such that $m = m_i$ and

$$f(a_{m_1+1}, a_{m_1+2}, \ldots, a_{m_2})(a_{m_2+1}, a_{m_2+2}, \ldots, a_{m_3}) \ldots (a_{m_{i-1}+1}, a_{m_{i-1}+2}, \ldots, a_{m_i})$$

is defined.

Definition 5.1.8. For an n-place argument filler $\sigma = (p, (m_i)_{1 \leq i \leq l})$ and an element $m \in \{1, \ldots, n\}$, we define $\sigma(m) := p(m)$.

The possibility of presupposition failure is implemented in *HODPL* semantics by making the formula interpretation function partial rather than total. For conveniently talking about partial functions, we use the notation *def*$(f(x))$ to abbreviate that f is defined on x.[4]

We are now ready to present the definition of *HODPL* semantics. Note that the complicated definition of the semantics of $\varphi \to \psi$ (item 9 in the two lists below) will be motivated and clarified after the formal presentation of the definition.

Definition 5.1.9. Given an *AFTB* model M and an M-assignment g, we define the term interpretation function $\frac{M}{g}(\bullet) : T_{HODPL} \to M$ and the domain and values of the partial formula interpretation function $[\![\bullet]\!]^g_M \subseteq G_M \times G_M$ by a simultaneous recursion:

[4]Note that we distinguish typographically between the notation def(t) as part of the *HODPL* syntax and the notation *def*(t) which is part of our metalanguage.

$$\frac{M}{g}(t) = \begin{cases} g(t) & \text{if } g \text{ is defined on } t \\ app_n^M(\frac{M}{g}(t_0), \frac{M}{g}(t_1), \ldots, \frac{M}{g}(t_n)) & \text{if } g \text{ is undefined on } t \text{ and } t \text{ is of the form } t_0(t_1, \ldots, t_n) \\ u^M & \text{if } t \text{ is of the form } \iota x\ \varphi \text{ and either there is a } g'[x]g \text{ such that } [\![\varphi]\!]^{g'}_M \text{ is undefined or it is not the case that there is precisely one } h \text{ such that } h[x]g \text{ and } [\![\varphi]\!]^h_M \neq \emptyset \\ h(x) & \text{if } t \text{ is of the form } \iota x\ \varphi, \text{ for every } g'[x]g, [\![\varphi]\!]^{g'}_M \text{ is defined, and } h \text{ is the unique assignment such that } h[x]g \text{ and } [\![\varphi]\!]^h_M \neq \emptyset \end{cases}$$

- Domain of $[\![\bullet]\!]^g_M$:
  1. $def([\![t]\!]^g_M)$ iff $\frac{M}{g}(t) = \top^M$ or $\frac{M}{g}(t) = \bot^M$.
  2. $def([\![\top]\!]^g_M)$.
  3. $def([\![t_1 = t_2]\!]^g_M)$ iff $\frac{M}{g}(t_1) \neq u^M$ and $\frac{M}{g}(t_2) \neq u^M$.
  4. $def([\![U(t)]\!]^g_M)$ iff $\frac{M}{g}(t) \neq u^M$.
  5. $def([\![B(t)]\!]^g_M)$ iff $\frac{M}{g}(t) \neq u^M$.
  6. $def([\![\neg\varphi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$.
  7. $def([\![\varphi \wedge \psi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$ and for all $h \in [\![\varphi]\!]^g_M$, $def([\![\psi]\!]^h_M)$.
  8. $def([\![\varphi \vee \psi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$ and $def([\![\psi]\!]^g_M)$.
  9. $def([\![\varphi \rightarrow \psi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$ and for all $h \in [\![\varphi]\!]^g_M$, we have that $def([\![\psi]\!]^h_M)$ and that for every $k \in [\![\psi]\!]^h_M$, if there is a $t \in dom(k) \setminus dom(h)$ of the form $f^\sigma(t_1, \ldots, t_n)$, where $\{t_1, \ldots, t_n\} = dom(h) \setminus dom(g)$, f is an *HODPL* term and σ is an n-place argument filler, then $k(t) \in L^M$ and $h(t_i) \in L^M$ for $1 \leq i \leq n$.
  10. $def([\![\exists t\ \varphi]\!]^g_M)$ iff for all h such that $h[t]g$, $def([\![\varphi]\!]^h_M)$.
  11. $def([\![\Diamond\varphi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$.
  12. $def([\![\mathrm{def}(t)]\!]^g_M)$.
- Values of $[\![\bullet]\!]^g_M$:
  1. $[\![t]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) = \top^M \\ \emptyset & \text{if } \frac{M}{g}(t) = \bot^M \end{cases}$
  2. $[\![\top]\!]^g_M := \{g\}$
  3. $[\![t_1 = t_2]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t_1) = \frac{M}{g}(t_2) \\ \emptyset & \text{otherwise} \end{cases}$
  4. $[\![U(t)]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) \in U^M \\ \emptyset & \text{otherwise} \end{cases}$

5. $[\![B(t)]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) = \top^M \text{ or } \frac{M}{g}(t) = \bot^M \\ \emptyset & \text{otherwise} \end{cases}$

6. $[\![\neg\varphi]\!]^g_M := \begin{cases} \{g\} & \text{if there is no } h \text{ such that } h \in [\![\varphi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$

7. $[\![\varphi \wedge \psi]\!]^g_M := \{h \mid \text{there is a } k \text{ such that } k \in [\![\varphi]\!]^g_M \text{ and } h \in [\![\psi]\!]^k_M\}$

8. $[\![\varphi \vee \psi]\!]^g_M := \begin{cases} \{g\} & \text{if there is there is an } h \text{ such that } h \in [\![\varphi]\!]^g_M \text{ or } h \in [\![\psi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$

9. $[\![\varphi \rightarrow \psi]\!]^g_M := \{h \mid$ there are variables $f_1, \ldots, f_n$ (where $n \geq 0$ and the choice of n is maximal) such that $h[f_1, \ldots, f_n]g$ and such that there are variables $x_1, \ldots, x_m$ (where $m \geq 0$) and m-place argument fillers $\sigma_1, \ldots, \sigma_n$ such that for all $k \in [\![\varphi]\!]^g_M$, $k[x_1, \ldots, x_m]g$ and there is an assignment $j \in [\![\psi]\!]^k_M$ such that for $1 \leq i \leq n$, $j(f_i^{\sigma_i}(x_1, \ldots, x_m)) = h(f_i)^{\sigma_i}(k(x_1), \ldots, k(x_m))$, and for any $l > 0$ in the sequence that constitutes the second element of σ_i and any $a_1, \ldots, a_l \in M$, $h(f_i)$ is σ_i-defined at $a_1, \ldots, a_l$ iff there is a $k' \in [\![\varphi]\!]^g_M$ such that for all $s \leq l$, $k'(x_{\sigma_i(s)}) = a_s\}$

10. $[\![\exists t\ \varphi]\!]^g_M := \{h \mid \text{there is a } k \text{ such that } k[t]g \text{ and } h \in [\![\varphi]\!]^k_M\}$

11. $[\![\Diamond\varphi]\!]^g_M := \begin{cases} \{g\} & \text{if there is an } h \text{ such that } h \in [\![\varphi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$

12. $[\![\mathrm{def}(t)]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) \neq u^M \\ \emptyset & \text{otherwise} \end{cases}$

In order to make case 9 of the definition more comprehensible, let us first consider its role in determining the semantics of (1), i.e. of $\exists x\ U(x) \rightarrow \exists f(x)\ R(x, f(x))$:

- $[\![\exists f(x)\ R(x, f(x))]\!]^k_M$ is the set of assignments j satisfying $R(x, f(x))$ (i.e. for which $[\![R(x, f(x))]\!]^j_M$ is non-empty) such that $j[f(x)]k$.
- $[\![\exists x\ U(x)]\!]^g_M$ is the set of assignments k such that $k[x]g$ and $k(x) \in U^M$.

For the sake of simplicity, we first ignore the last part of the definition of $[\![\varphi \rightarrow \psi]\!]^g_M$, namely the part starting with "and for any $l > 0$". Under this simplification, the definition yields the following (with $n = m = 1$ and the only possible 1-place argument filler for σ_1):

$$\begin{aligned} [\![\exists x\ U(x) \rightarrow \exists f(x)\ R(x, f(x))]\!]^g_M &= \{h \mid h[f]g \text{ and there is a variable } x_1 \text{ such that for all } k \text{ such that } k[x]g \text{ and } k(x) \in U^M, \\ &\quad k[x_1]g \text{ and there is an assignment } j \text{ satisfying } R(x_1, f(x_1)) \text{ such that } j[f(x_1)]k \text{ and } \\ &\quad j(f(x_1)) = h(f)(k(x_1))\} \\ &= \{h \mid h[f]g \text{ and for all } k \text{ such that } k[x]g \\ &\quad \text{and } k(x) \in U^M, \text{ there is an assignment } j \\ &\quad \text{satisfying } R(x, f(x)) \text{ such that } j[f(x)]k \text{ and } \\ &\quad j(f(x)) = h(f)(k(x))\} \end{aligned}$$

$$= \{h \mid h[f]g \text{ and for all } k \text{ such that } k[x]h \text{ and } k(x) \in U^M, \ k \text{ satisfies } R(x, f(x))\}$$
$$= [\![\exists f \ \ (\exists x\, U(x) \to R(x, f(x)))]\!]^g_M$$

The last part of the definition of $[\![\varphi \to \psi]\!]^g_M$ imposes an additional restriction on the dynamically introduced function f, namely that f is only defined on arguments that are urelements. An additional difference between the semantics of (1) and of (4) is that $[\![\exists x\ U(x) \to \exists f(x)\ R(x, f(x))]\!]^g_M$ can be undefined, namely if $M \models \exists x, y\ (U(x) \wedge app_2(g(R), x, y) \wedge \neg L(y))$. This possibility of $[\![\exists x\, U(x) \to \exists f(x)\, R(x, f(x))]\!]^g_M$ being undefined is due to the third (and last) condition in the definition of $def([\![\varphi \to \psi]\!]^g_M)$ (starting from "for every $k \in [\![\psi]\!]^h_M$"). This condition has been added to the definition of $def([\![\varphi \to \psi]\!]^g_M)$ in order to ensure that if such an implication is defined and has the syntactic form of an implicit dynamic function introduction, then the Map Comprehension Axiom Schema of *AFTB* actually implies the existence of a map satisfying the properties that the implicitly introduced map must satisfy.

The truth condition of a formula φ under (M, g) is determined by $[\![\varphi]\!]^g_M$ being empty or non-empty (with emptiness corresponding to falsehood). The claim that the truth conditions of (1) and of (3) are essentially equal can now be made precise: If $[\![\exists x\, U(x) \to \exists f(x)\, R(x, f(x))]\!]^g_M$ is defined, then it is empty iff $[\![\exists x\ U(x) \to \exists y\ R(x, y)]\!]^g_M$ is empty. This will actually follow from Lemma 5.1.10 below, which characterizes the truth conditions of $[\![\varphi \to \psi]\!]^g_M$, but it helps understanding to see this example case proven directly from the definitions:

> Suppose $[\![\exists x\ U(x) \to \exists f(x)\ R(x, f(x))]\!]^g_M$ is defined. This implies two facts:
>
> (i) For any $x, y \in M$ with $x \in U^M$, $app_2^M(g(R), x, y)$ equals $\top^M$ or $\bot^M$.
>
> (ii) For any $x, y \in M$ with $x \in U^M$ and $app_2^M(g(R), x, y) = \top^M$, $y \in L^M$.
>
> (i) implies that $[\![\exists x\, U(x) \to \exists y\, R(x, y)]\!]^g_M$ is defined too. Now
>
> $[\![\exists x\, U(x) \to \exists f(x)\, R(x, f(x))]\!]^g_M \neq \emptyset$
>
> iff $\{h \mid h[f]g$ and for all k such that $k[x]g$ and $k(x) \in U^M$, there is an assignment j satisfying $R(x, f(x))$ such that $j[f(x)]k$ and $j(f(x)) = h(f)(k(x))$, and for $a \in M$, $h(f)$ is defined at a iff there is a k' such that $k'[x]g$, $k'(x) \in U^M$ and $k'(x) = a\} \neq \emptyset$
>
> iff $\{h \mid h[f]g$ and for all k such that $k[x]g$ and $k(x) \in U^M$, $app_2^M(R^M, k(x), \bar{h}(f)(k(x))) = \top^M$, and $h(f)$ is defined at $a \in M$ iff $a \in U^M\} \neq \emptyset$
>
> iff there is an $\bar{f} \in M$ defined precisely on the urelements of M such that for all k such that $k[x]g$ and $k(x) \in U^M$, $app_2^M(R^M, k(x), \bar{f}(k(x))) = \top^M$
>
> iff for all k such that $k[x]g$ and $k(x) \in U^M$, there is a $\bar{y} \in M$ such that $app_2^M(R^M, k(x), \bar{y}) = \top^M$ (the right-to-left implication follows from fact (ii) above and Function Comprehension of *AFTB*)
>
> iff $[\![\exists x\, U(x) \to \exists y\, R(x, y)]\!]^g_M \neq \emptyset$.

In order to clarify the usage of argument fillers in the definition of $[\![\varphi \to \psi]\!]^g_M$, we will consider their role in determining the semantics of the following simplification of (6):

(7) $\exists x\, \exists y\, (U(x) \wedge U(y)) \to \exists f(x)(y)\, \exists g(y,x)\, (R(x, y, f(x)(y), g(y,x)))$

In determining the semantics of this *HODPL* formula, the definition of $[\![\varphi \to \psi]\!]^g_M$ is used with $m = n = 2$, with f, g corresponding to f_1, f_2, with x, y corresponding to x_1, x_2 and with $(id_{\{1,2\}}, (0,1,2))$ and $((1\ 2)_{\{1,2\}}, (0,2))$ corresponding to σ_1 and σ_2. These two 2-place argument fillers encode the way x and y are fed as arguments to f and g respectively: g accepts them in the form $g(y,x)$, i.e. with their order interchanged; hence the permutation $(1\ 2)_{\{1,2\}}$. f accepts them in the form $f(x)(y)$; here their order is not interchanged, i.e. we use the permutation $id_{\{1,2\}}$, but f is curried, i.e. the arguments are split to different levels of function application. This is encoded by the tuple $(0,1,2)$, whereas the tuple $(0,2)$ of σ_2 encodes that the arguments are not split.

We now present the already mentioned lemma that characterizes the truth conditions of $[\![\varphi \to \psi]\!]^g_M$:

Lemma 5.1.10. *Let M be an AFTB model, g an M-assignment and φ and ψ HODPL formulae such that $[\![\varphi \to \psi]\!]^g_M$ is defined. Then $[\![\varphi \to \psi]\!]^g_M \neq \emptyset$ iff for every $k \in [\![\varphi]\!]^g_M$, $[\![\psi]\!]^k_M \neq \emptyset$.*

The proof of this lemma is analogous to the proof of a corresponding lemma for Proof Text Logic (Lemma 5.2.21), which we will present in section 5.2 below and prove in chapter 6.

5.1.2 Mimicking constants, function symbols and relation symbols in *HODPL*

As already explained above, *HODPL* syntax does not allow for constant, function and relation symbols other than $=$, U and B. We will now show by means of an example how variables in *HODPL* can be used to mimic the usage of constant, function and relation symbols in *PL* or *DPL*.

One common application of *PL* is to formally axiomatize some theory and then develop the theory from the axioms. The axioms may involve constants as well as function and relation symbols that get characterized through the axioms. For example, the following axioms of *partially ordered groups* involve the constant 1, the binary function symbol $\cdot$ (for readability written in infix notation) and the binary relation symbol $\leq$ (also written in infix notation):

$$\begin{aligned}
\varphi_1 &:= \forall x\, (x \cdot 1 = x \wedge 1 \cdot x = x)\\
\varphi_2 &:= \forall x, y, z\ x \cdot (y \cdot z) = (x \cdot y) \cdot z\\
\varphi_3 &:= \forall x\, \exists y\, (x \cdot y = 1 \wedge y \cdot x = 1)\\
\varphi_4 &:= \forall x, y, z, w\, (x \leq y \to z \cdot (x \cdot w) \leq z \cdot (y \cdot w))
\end{aligned}$$

If we prove that some formula ψ follows from these axioms, we can say that we have shown that $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \to \psi$ is a valid *PL* formula.

Now in case we want to formalize the axioms in *HODPL*, we cannot use the constant 1, the function symbol $\cdot$ and the relation symbol $\leq$. Instead, we can make use of the dynamic character of existential quantification and the fact that

we can implicitly dynamically introduce functions, which can also be declared as relations by asserting their value to be always a Boolean. The domain of quantification of the above *PL* formulae would be identified with the urelements in *HODPL*. The axiom conjunction $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4$ can now be mimicked by the following *HODPL* formula φ':

$$\begin{aligned}\varphi' := {} & (\exists x\ \exists y\ U(x) \wedge U(y) \rightarrow \exists x \cdot y\ U(x \cdot y)) \wedge \\ & \exists 1\ U(1) \wedge \\ & (\exists x\ \exists y\ U(x) \wedge U(y) \rightarrow \exists x \leq y\ B(x \leq y)) \wedge \\ & (\exists x\ U(x) \rightarrow x \cdot 1 = x \wedge 1 \cdot x = x)) \wedge \\ & (\exists x, y, z\ (U(x) \wedge U(y) \wedge U(z) \rightarrow x \cdot (y \cdot z) = (x \cdot y) \cdot z) \wedge \\ & (\exists x\ U(x) \rightarrow \exists y\ (x \cdot y = 1 \wedge y \cdot x = 1)) \wedge \\ & (\exists x, y, z, w\ (U(x) \wedge U(y) \wedge U(z) \wedge U(w) \wedge x \leq y) \rightarrow z \cdot (x \cdot w) \leq z \cdot (y \cdot w))\end{aligned}$$

Here the first three lines dynamically introduce the symbols $\cdot$, 1 and $\leq$ respectively. Note, however, that these are *HODPL* variables and not a function symbol, a constant and a relation symbol as in the *PL* variant. The remaining four lines correspond to the four axioms φ_1, φ_2, φ_3 and φ_4 above.

A *PL* formula ψ that follows from the *PL* variant of the axioms can be transformed in a similar way into an *HODPL* formula ψ', in which $\cdot$, 1 and $\leq$ are free variables. Then the valid *PL* formula $\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \rightarrow \psi$ corresponds to the valid *HODPL* formula $\varphi' \rightarrow \psi'$. In this *HODPL* text, the occurrences of $\cdot$, 1 and $\leq$ in ψ' are bound by the dynamic existential quantifiers in φ'. (The notion of binding in *HODPL* is similar to that in *DPL* described in section 3.1.1 and completely analogous to that in the *HODPL* extension *PTL* defined below, described in section 5.2.3. The notion of validness of an *HODPL* formula is also completely analogous to the corresponding notion for *PTL* defined in Definition 5.2.12 below.)

5.2 Proof Text Logic

In this section we present a formalism called *Proof Text Logic* (*PTL*), which has the same expressive power as the Naproche CNL, but is of a completely formal character, i.e. does not contain any natural language elements, but has a syntax resembling that of standard predicate logic. It can be viewed as an extension of *HODPL* with two distinguishing characteristics:

- The largest syntactic category of *HODPL* – just as of *DPL* and *PL* – are formulae. These correspond roughly to natural language sentences. In *PTL*, on the other hand, the largest syntactic category is that of a *PTL text*, corresponding to a mathematical proof text including axioms, definitions, theorems and lemmas with their proofs and the possibility of references to previous theorems or lemmas.

- *HODPL* is based on a pure function theory, whereas *PTL* is based on a theory of sets, functions, tuples and natural numbers, namely *CMTN*.

For the concatenation of assertions above the sentence level, we introduce a second conjunctional connective & besides the $\wedge$ already present in *HODPL*.

For theorems (and lemmas, propositions and corollaries), we introduce the construct *thm*$(\bullet, \bullet, \bullet)$: Its first argument marks the *theorem type* ("theorem", "lemma", "proposition" or "corollary"), its second argument contains the theorem assertion, and its third argument contains the proof of the theorem. For references we introduce two constructs:

- *label*$(\bullet, \bullet)$, whose first argument is an ID attached as a label to the second argument.
- *ref*$(\bullet, \bullet)$, whose first argument contains a list of reference IDs and whose second argument contains an assertion, whose proof is claimed to depend on the premises that are being referenced.

PTL does not have a special construct for definitions. Definitions can be modelled in *PTL* using the dynamic existential quantifier and implicit dynamic function introduction. More on this in section 7.5.4.

5.2.1 *PTL* syntax

5.2.1 The symbols in the language of *CMTN* apart from u and a become logical symbols in *PTL*. More precisely, we have logical constant symbols 0, s, $\top$ and $\bot$, we have unary logical relation symbols C, N, B and L, we have binary logical relation symbols $=$, $\in$, M and T, and we have a binary logical function *nth* and for every $n \geq 2$, an n-ary logical function symbol τ_n.[5] Additionally to the countably infinite supply of variables, we also need to assume a countably infinite supply of IDs for our definition of *PTL* syntax. Furthermore, we have four theorem type symbols *thm*, *lem*, *prop* and *cor*.

In *PTL* syntax we distinguish between *PTL texts* and *PTL formulae*: The first correspond to possibly multi-sentential mathematical texts, whereas the second correspond to single sentences in a mathematical texts. By this characterization, *PTL* formulae are always also *PTL* texts.

As in the case of *HODPL* syntax, we define *PTL* syntax by defining a number of syntactic concepts via a simultaneous recursion:

Definition 5.2.1.

- A *PTL term* is a variable, a logical constant symbol or of the form $f_n(t_0, \ldots, t_n)$, $t_0(t_1, \ldots, t_n)$ or $\iota x\ \varphi$ for an n-ary logical function symbol f, *PTL* terms $t_0, \ldots, t_n$, a variable x and a *PTL* formula φ. We write T_{PTL} for the set of *quantifiable PTL terms*, i.e. *PTL* terms that do not contain ι and do not contain logical function symbols or logical constant symbols.
- A *PTL text* is of one of the following forms, where $t_1, \ldots, t_n$ are *PTL* terms, t is a *PTL* term in T_{PTL}, t_0 is an ι-free *PTL* term, φ and ψ are *PTL* formulae, θ and ξ are *PTL* texts, ϑ is a theorem type symbol, α is an ID and S is a finite non-empty sequence of IDs:
  - t_1
  - $R_n(t_1, \ldots, t_n)$

[5] In order to avoid unnecessary case distinctions, we sometimes treat the logical constant symbols as 0-ary logical function symbols.

- $\neg\varphi$
- $(\varphi \wedge \psi)$
- $(\varphi \vee \psi)$
- $(\varphi \rightarrow \theta)$
- $(\theta \,\&\, \xi)$
- $\exists t\ \varphi$
- $\Diamond\varphi$
- $\mathit{label}(\alpha, \theta)$
- $\mathit{ref}(S, \varphi)$
- $\mathit{thm}(\vartheta, \varphi, \theta)$
- $\mathrm{def}(t_0)$

We usually write $t_1 = t_2$ instead of $=(t_1, t_2)$.

- A *PTL formula* is a *PTL* text not containing &, $\mathit{ref}(\bullet, \bullet)$ and $\mathit{thm}(\bullet, \bullet, \bullet)$.

We sometimes use $\forall t\ \varphi$ and $(\varphi \leftrightarrow \psi)$ as an abbreviations for $(\exists t\ \top \rightarrow \varphi)$ and $(\Diamond(\varphi \rightarrow \psi) \wedge \Diamond(\psi \rightarrow \varphi))$ respectively.

5.2.2 *PTL* semantics

In this section we present a formal semantics for *PTL* that closely resembles *HODPL* semantics. However, this semantics does not really capture the role of labels, references ($\mathit{ref}(\bullet, \bullet)$) and the theorem-proof blocks ($\mathit{thm}(\bullet, \bullet, \bullet)$). This is because references and the theorem-proof-blocks have a procedural role: They give hints on how the *PTL* text can be checked for logical validity. This procedural role is captured by the proof checking algorithm for *PTL* presented in the next chapter. The semantics presented in this section only captures the non-procedural aspects of the meaning of *PTL* texts.

The difference between $\varphi \wedge \psi$ and $\varphi \,\&\, \psi$ is also purely procedural, so that in this section they will be given the same semantics. The intended difference between the two is that when proof-checking $\exists x\ \varphi \,\&\, \psi$, one can check $\exists x\ \varphi$ first and then check ψ under the assumption that φ, whereas when proof-checking $\exists x\ \varphi \wedge \psi$ one has to check $\exists x\ (\varphi \wedge \psi)$. The natural language "and" will always be translated by $\wedge$, whereas & will in general be used for translating the concatenation of assertions above the sentence level.

PTL semantics is defined in a similar way to *HODPL* semantics. The preliminary notions, like assignments and n-place argument fillers, are defined just as for *HODPL* semantics, only replacing *AFTB* models by *CMTN* models in the definitions.

Definition 5.2.2. Given a *CMTN* model M and an M-assignment g, we define the term interpretation function $\frac{M}{g}(\bullet) : T_{PTL} \rightarrow M$ and the domain and values of the partial text interpretation function $[\![\bullet]\!]^g_M \subseteq G_M \times G_M$ by a simultaneous recursion:

$$\frac{M}{g}(t) = \begin{cases} c^M & \text{if } t \text{ is a logical constant symbol } c \text{ (in } c^M\text{, } c \text{ refers to the } CMTN \text{ constant symbol corresponding to the logical constant symbol } c\text{)} \\ g(t) & \text{if } g \text{ is defined on } t \\ app_n^M(\frac{M}{g}(t_0), \frac{M}{g}(t_1), \ldots, \frac{M}{g}(t_n)) & \text{if } g \text{ is undefined on } t \text{ an } t \text{ is of the form } t_0(t_1, \ldots, t_n) \\ f^M(\frac{M}{g}(t_1), \ldots, \frac{M}{g}(t_n)) & \text{if } t \text{ is of the form } f(t_1, \ldots, t_n) \text{ for a logical function symbol } f \text{ (in } f^M\text{, } f \text{ refers to the } CMTN \text{ function symbol corresponding to the logical function symbol } f\text{)} \\ u & \text{if } t \text{ is of the form } \iota x\ \varphi \text{ and either there is a } g'[x]g \text{ such that } [\![\varphi]\!]_M^{g'} \text{ is undefined or it is not the case that there is precisely one } h \text{ such that } h[x]g \text{ and } [\![\varphi]\!]_M^h \neq \emptyset \\ h(x) & \text{if } t \text{ is of the form } \iota x\ \varphi\text{, for every } g'[x]g\text{, } [\![\varphi]\!]_M^{g'} \text{ is defined, and } h \text{ is the unique assignment such that } h[x]g \text{ and } [\![\varphi]\!]_M^h \neq \emptyset \end{cases}$$

- Domain of $[\![\bullet]\!]_M^g$:
  1. $def([\![t]\!]_M^g)$ iff $\frac{M}{g}(t) \in B^M$.
  2. $def([\![R(t_1, \ldots, t_n)]\!]_M^g)$ iff $\frac{M}{g}(t_1) \neq u^M$, ..., $\frac{M}{g}(t_n) \neq u^M$.
  3. $def([\![\neg\varphi]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$.
  4. $def([\![\varphi \wedge \psi]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$ and for all $h \in [\![\varphi]\!]_M^g$, $def([\![\psi]\!]_M^h)$.
  5. $def([\![\varphi \vee \psi]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$ and $def([\![\psi]\!]_M^g)$.
  6. $def([\![\varphi \rightarrow \theta]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$ and for all $h \in [\![\varphi]\!]_M^g$, we have that $def([\![\theta]\!]_M^h)$ and that for every $k \in [\![\theta]\!]_M^h$, if there is a $t \in dom(k) \setminus dom(h)$ of the form $f^\sigma(t_1, \ldots, t_n)$, where $\{t_1, \ldots, t_n\} = dom(h) \setminus dom(g)$, f is a PTL term and σ is an n-place argument filler, then $k(t) \in L^M$ and $h(t_i) \in L^M$ for $1 \leq i \leq n$.
  7. $def([\![\theta\ \&\ \xi]\!]_M^g)$ iff $def([\![\theta]\!]_M^g)$ and for all $h \in [\![\theta]\!]_M^g$, $def([\![\xi]\!]_M^h)$.
  8. $def([\![\exists t\ \varphi]\!]_M^g)$ iff for all h such that $h[t]g$, $def([\![\varphi]\!]_M^h)$.
  9. $def([\![\Diamond\varphi]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$.
  10. $def([\![label(\alpha, \theta)]\!]_M^g)$ iff $def([\![\theta]\!]_M^g)$.
  11. $def([\![ref(S, \varphi)]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$.
  12. $def([\![thm(\vartheta, \varphi, \theta)]\!]_M^g)$ iff $def([\![\varphi]\!]_M^g)$.
  13. $def([\![\mathrm{def}(t)]\!]_M^g)$.

- Values of $[\![\bullet]\!]^g_M$:

  1. $[\![t]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) = \top^M \\ \emptyset & \text{if } \frac{M}{g}(t) = \bot^M \end{cases}$
  2. $[\![R(t_1, \ldots, t_n)]\!]^g_M := \begin{cases} \{g\} & \text{if } (\frac{M}{g}(t_1), \ldots, \frac{M}{g}(t_1)) \in R^M \\ \emptyset & \text{otherwise} \end{cases}$
  3. $[\![\neg\varphi]\!]^g_M := \begin{cases} \{g\} & \text{if there is no } h \text{ such that } h \in [\![\varphi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$
  4. $[\![\varphi \wedge \psi]\!]^g_M := \{h \mid \text{there is a } k \text{ such that } k \in [\![\varphi]\!]^g_M \text{ and } h \in [\![\psi]\!]^k_M\}$
  5. $[\![\varphi \vee \psi]\!]^g_M := \begin{cases} \{g\} & \text{if there is there is an } h \text{ such that } h \in [\![\varphi]\!]^g_M \text{ or } h \in [\![\psi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$
  6. $[\![\varphi \to \theta]\!]^g_M := \{h \mid$ there are PTL terms $f_1, \ldots, f_n$ (where $n \geq 0$ and the choice of n is maximal) such that $h[f_1, \ldots, f_n]g$ and such that there are PTL terms $t_1, \ldots, t_m$ (where $m \geq 0$) and m-place argument fillers $\sigma_1, \ldots, \sigma_n$ such that for all $k \in [\![\varphi]\!]^g_M$, $k[t_1, \ldots, t_m]g$ and there is an assignment $j \in [\![\theta]\!]^k_M$ such that for $1 \leq i \leq n$, $j(f_i^{\sigma_i}(t_1, \ldots, t_m)) = h(f_i)^{\sigma_i}(k(t_1), \ldots, k(t_m))$, and for any $l > 0$ in the sequence that constitutes the second element of σ_i and any $a_1, \ldots, a_l \in M$, $h(f_i)$ is σ_i-defined at $a_1, \ldots, a_l$ iff there is a $k' \in [\![\varphi]\!]^g_M$ such that for all $s \leq l$, $k'(t_{\sigma_i(s)}) = a_s\}$
  7. $[\![\theta \,\&\, \xi]\!]^g_M := \{h \mid \text{there is a } k \text{ such that } k \in [\![\theta]\!]^g_M \text{ and } h \in [\![\xi]\!]^k_M\}$
  8. $[\![\exists t\ \varphi]\!]^g_M := \{h \mid \text{there is a } k \text{ such that } k[t]g \text{ and } h \in [\![\varphi]\!]^k_M\}$
  9. $[\![\Diamond\varphi]\!]^g_M := \begin{cases} \{g\} & \text{if there is an } h \text{ such that } h \in [\![\varphi]\!]^g_M \\ \emptyset & \text{otherwise} \end{cases}$
  10. $[\![\mathit{label}(\alpha, \theta)]\!]^g_M := [\![\theta]\!]^g_M$
  11. $[\![\mathit{ref}(S, \varphi)]\!]^g_M := [\![\varphi]\!]^g_M$
  12. $[\![\mathit{thm}(\vartheta, \varphi, \theta)]\!]^g_M := [\![\varphi]\!]^g_M$
  13. $[\![\mathrm{def}(t)]\!]^g_M := \begin{cases} \{g\} & \text{if } \frac{M}{g}(t) \neq u^M \\ \emptyset & \text{otherwise} \end{cases}$

The following definitions give us a three-valued classification of PTL texts into meaningless, true and false ones:

Definition 5.2.3. We define a ternary *validity function* v as follows: Given a PTL text θ, a $CMTN$ model M and an M-assignment g, define

$$v(\theta, M, g) := \begin{cases} u & \text{if } [\![\theta]\!]^g_M \text{ is undefined} \\ \top & \text{if } [\![\theta]\!]^g_M \text{ is defined and non-empty} \\ \bot & \text{if } [\![\theta]\!]^g_M \text{ is defined and empty} \end{cases}$$

5.2.3 Scope and binding

Just as DPL allows for existential quantifiers to bind variables which are outside their syntactic scope, *PTL* allows for existential quantifiers to bind quantifiable terms which are outside their syntactic scope. We now give a syntactic definition of when an occurrence of a quantifiable term is bound by an occurrence of a quantifier, which is an adaptation of the definition by Groenendijk and Stokhof (1991) presented in section 3.1.1 of chapter 3.

Compared to the definition of Groenendijk and Stokhof (1991), we have to capture one additional difficulty: Groenendijk and Stokhof (1991) could assume that an occurrence of $\exists t$ can only bind occurrences of t and not occurrence of other terms. In *PTL* however, because of the phenomenon of implicit dynamic function introduction, an occurrence of $\exists t_0(t_1, \ldots, t_n)$ in the right hand side of an implication may bind t_0 outside the scope of the implication. In order to capture this, we will have to keep track of which term an active occurrence of a quantifier can currently bind.

We will define three functions on *PTL* texts by simultaneous recursion:

- $\mathbf{bp}(\theta)$, the set of *binding pairs* in θ.
- $\mathbf{aq}(\theta)$, the set of *active quantifier pairs* in θ.
- $\mathbf{ft}(\theta)$, the set of *free occurrences of terms* in θ.

An active quantifier pair consists of a quantifier occurrence and a term such that the quantifier occurence has the potential to bind occurrences of the term further on. The notions of a binding pair and of a free term correspond directly to the notions of a binding pair and of a free variable from section 3.1.1.

Definition 5.2.4. We define the functions **bp**, **aq** and **ft** on *PTL* terms and *PTL* texts by simultaneous recursion as follows:

1. $\mathbf{bp}(x) := \emptyset$
   $\mathbf{aq}(x) := \emptyset$
   $\mathbf{ft}(x) := \{x\}$

   $\mathbf{bp}(f(t_1, \ldots, t_n)) := \mathbf{bp}(t_1) \cup \cdots \cup \mathbf{bp}(t_n)$
   $\mathbf{aq}(f(t_1, \ldots, t_n)) := \emptyset$
   $\mathbf{ft}(f(t_1, \ldots, t_n)) := \mathbf{ft}(t_1) \cup \cdots \cup \mathbf{ft}(t_n)$

   $\mathbf{bp}(t_0(t_1, \ldots, t_n)) := \mathbf{bp}(t_0) \cup \cdots \cup \mathbf{bp}(t_n)$
   $\mathbf{aq}(t_0(t_1, \ldots, t_n)) := \emptyset$
   $\mathbf{ft}(t_0(t_1, \ldots, t_n)) := \mathbf{ft}(t_0) \cup \cdots \cup \mathbf{ft}(t_n) \cup \{t_0(t_1, \ldots, t_n)\}$

   $\mathbf{bp}(\iota x\ \varphi) := \mathbf{bp}(\varphi) \cup \{(\iota x, x) \mid x \in \mathbf{ft}(\varphi)\}$
   $\mathbf{aq}(\iota x\ \varphi) := \emptyset$
   $\mathbf{ft}(\iota x\ \varphi) := \mathbf{ft}(\varphi)$ minus the occurrences of x in φ

2. $\mathbf{bp}(R(t_1, \ldots, t_n)) := \mathbf{bp}(t_1) \cup \cdots \cup \mathbf{bp}(t_n)$
   $\mathbf{aq}(R(t_1, \ldots, t_n)) := \emptyset$
   $\mathbf{ft}(R(t_1, \ldots, t_n)) := \mathbf{ft}(t_1) \cup \cdots \cup \mathbf{ft}(t_n)$

3. $\mathbf{bp}(\neg\varphi) := \mathbf{bp}(\varphi)$
   $\mathbf{aq}(\neg\varphi) := \emptyset$
   $\mathbf{ft}(\neg\varphi) := \mathbf{ft}(\varphi)$

4. $\mathbf{bp}(\varphi\wedge\psi) := \mathbf{bp}(\varphi)\cup\mathbf{bp}(\psi)\cup\{(\exists t_1, t_2) \mid (\exists t_1, t_2) \in \mathbf{aq}(\varphi)$ and $t_2 \in \mathbf{ft}(\psi)\}$
   $\mathbf{aq}(\varphi\wedge\psi) := \mathbf{aq}(\psi)\cup\{(\exists t_1, t_2) \in \mathbf{aq}(\varphi) \mid$ there is no t_1' such that $(\exists t_1', t_2) \in \mathbf{aq}(\psi)\}$
   $\mathbf{ft}(\varphi \wedge \psi) := \mathbf{ft}(\varphi) \cup \{t \in \mathbf{ft}(\psi) \mid$ there is no t' such that $(\exists t', t) \in \mathbf{aq}(\varphi)\}$

5. $\mathbf{bp}(\varphi \vee \psi) := \mathbf{bp}(\varphi) \cup \mathbf{bp}(\psi)$
   $\mathbf{aq}(\varphi \vee \psi) := \emptyset$
   $\mathbf{ft}(\varphi \vee \psi) := \mathbf{ft}(\varphi) \cup \mathbf{ft}(\psi)$

6. $\mathbf{bp}(\varphi \to \theta) := \mathbf{bp}(\varphi)\cup\mathbf{bp}(\theta)\cup\{(\exists t_1, t_2) \mid (\exists t_1, t_2) \in \mathbf{aq}(\varphi)$ and $t_2 \in \mathbf{ft}(\theta)\}$
   $\mathbf{aq}(\varphi \to \theta) := \{(\exists t, t_0) \mid \mathbf{aq}(\varphi) = \{(\exists t_1', t_1), \ldots, (\exists t_n', t_n)\}$ for $n \geq 1$, $(\exists t, t') \in \mathbf{aq}(\theta)$ and there is an n-place argument filler σ such that $t' = t_0^{\sigma}(t_1, \ldots, t_n)\}$
   $\mathbf{ft}(\varphi \to \theta) := \mathbf{ft}(\varphi) \cup \{t \in \mathbf{ft}(\theta) \mid$ there is no t' such that $(\exists t', t) \in \mathbf{aq}(\varphi)\}$

7. $\mathbf{bp}(\theta \,\&\, \xi) := \mathbf{bp}(\theta) \cup \mathbf{bp}(\xi) \cup \{(\exists t_1, t_2) \mid (\exists t_1, t_2) \in \mathbf{aq}(\theta)$ and $t_2 \in \mathbf{ft}(\xi)\}$
   $\mathbf{aq}(\theta \,\&\, \xi) := \mathbf{aq}(\xi)\cup\{(\exists t_1, t_2) \in \mathbf{aq}(\theta) \mid$ there is no t_1' such that $(\exists t_1', t_2) \in \mathbf{aq}(\xi)\}$
   $\mathbf{ft}(\theta \,\&\, \xi) := \mathbf{ft}(\theta) \cup \{t \in \mathbf{ft}(\xi) \mid$ there is no t' such that $(\exists t', t) \in \mathbf{aq}(\theta)\}$

8. $\mathbf{bp}(\exists t\ \varphi) := \mathbf{bp}(\varphi) \cup \{(\exists t, t) \mid t \in \mathbf{ft}(\varphi)\}$
   $\mathbf{aq}(\exists t\ \varphi) := \begin{cases} \mathbf{aq}(\varphi) \cup \{(\exists t, t)\} & \text{if there is no } t' \text{ such that } (\exists t', t) \in \mathbf{aq}(\varphi) \\ \mathbf{aq}(\varphi) & \text{otherwise} \end{cases}$
   $\mathbf{ft}(\exists t\ \varphi) := \mathbf{ft}(\varphi)$ minus the occurrences of t in φ

9. $\mathbf{bp}(\Diamond\varphi) := \mathbf{bp}(\varphi)$
   $\mathbf{aq}(\Diamond\varphi) := \emptyset$
   $\mathbf{ft}(\Diamond\varphi) := \mathbf{ft}(\varphi)$

10. $\mathbf{bp}(\mathit{label}(\alpha, \theta)) := \mathbf{bp}(\theta)$
    $\mathbf{aq}(\mathit{label}(\alpha, \theta)) := \mathbf{aq}(\theta)$
    $\mathbf{ft}(\mathit{label}(\alpha, \theta)) := \mathbf{ft}(\theta)$

11. $\mathbf{bp}(\mathit{ref}(S, \varphi)) := \mathbf{bp}(\varphi)$
    $\mathbf{aq}(\mathit{ref}(S, \varphi)) := \mathbf{aq}\varphi)$
    $\mathbf{ft}(\mathit{ref}(S, \varphi)) := \mathbf{ft}(\varphi)$

12. $\mathbf{bp}(\mathit{thm}(\vartheta, \varphi, \theta)) := \mathbf{bp}(\varphi) \cup \mathbf{bp}(\theta)$
    $\mathbf{aq}(\mathit{thm}(\vartheta, \varphi, \theta)) := \mathbf{aq}(\varphi)$
    $\mathbf{ft}(\mathit{thm}(\vartheta, \varphi, \theta)) := \mathbf{ft}(\varphi) \cup \mathbf{ft}(\theta)$

13. $\mathbf{bp}(\mathrm{def}(t)) := \mathbf{bp}(t)$
    $\mathbf{aq}(\mathrm{def}(t)) := \emptyset$
    $\mathbf{ft}(\mathrm{def}(t)) := \mathbf{ft}(t)$

Just as in the case of *DPL*, we can define a notion of active quantifier at any position in a *PTL* text. Again, we first need to formalize the notion of position:

Definition 5.2.5. Given a *PTL* text θ, we call an occurrence of an atomic formula in θ a *position in* θ.

Definition 5.2.6. Given a *PTL* text θ, an occurrence $\exists t_1$ of a quantifier in θ, a *PTL* term t_2 and a position p in θ, we say that the pair $(\exists t_1\ , t_2)$ is an *active quantifier pair at position* p iff the *PTL* text θ' resulting from replacing position p by t_2 has the binding pair $(\exists t_1\ , t_2)$, where the second element in this pair is now considered to be the occurrence of t_2 that has replaced position p.

5.2.4 Further *PTL* notions

We now define some further syntactic notions dependent on those defined above:

Definition 5.2.7. Given a *PTL* text θ, we define the set of *terms with binding capability after* θ by

$$\mathbf{tbc}(\theta) := \{t \mid \text{there is a term } t' \text{ such that } (\exists t', t) \in \mathbf{aq}(\theta)\}.$$

Definition 5.2.8. Given a *PTL* text θ, a *hereditarily free term in* θ is a term $t \in \mathbf{ft}(\theta)$ such that for all subterms t' of t, $t' \in \mathbf{ft}(\theta)$.

Definition 5.2.9. Given a *PTL* text θ, a *maximal hereditarily free term in* θ, usually abbreviated to *MHF term in* θ, is a hereditarily free term in θ that is not a proper subterm of a hereditarily free term in θ.

Definition 5.2.10. A *PTL* text θ is called *ground* if it contains no hereditarily free terms.

Additionally to the ternary validity function whose values depend not only on a *PTL* text but also on a *CMTN* model M and an M-assignment g, we can now also define an absolute validity function on ground *PTL* terms whose value is independent of any particular model and assignment. For this we first introduce a convenient notation for empty assignments:

Definition 5.2.11. Given a *CMTN* model M, we call the empty M-assignment (i.e. the M-assignment that is undefined on all terms in T_{PTL}) e_M.

Definition 5.2.12. We define a unary *absolute validity function* v on ground *PTL* texts by

$$v(\theta) := \begin{cases} u & \text{if there is a } \mathit{CMTN} \text{ model } M \text{ such that } [\![\theta]\!]_M^{e_M} \text{ is undefined} \\ \top & \text{if for all } \mathit{CMTN} \text{ models } M,\ [\![\theta]\!]_M^{e_M} \text{ is defined and non-empty} \\ \bot & \text{otherwise, i.e. if } [\![\theta]\!]_M^{e_M} \text{ is defined for all } \mathit{CMTN} \text{ models } M, \text{ and is empty for some } M \end{cases}$$

When $v(\theta) = u$, we also say by abuse of language that v is undefined at θ. When $v(\theta) = \top$, we say that θ is a valid *PTL* text.

In chapter 7, we will specify a translation from Naproche CNL texts to *PTL* texts. This translation actually always results in *PTL* texts with certain nice syntactic properties, and these properties are presupposed by the proof checking algorithm described in chapter 6. Roughly speaking, these nice properties

amount to the avoidance of variable clashes and the avoidance of variables that were not dynamically introduced through an existential quantifier. We will formally define these properties in Definition 5.2.16, but first we need some auxiliary definitions:

Definition 5.2.13. Two *PTL* terms t and t' are called *independent* iff there are no *PTL* terms $t_1, \ldots, t_m$, $t'_1, \ldots, t'_n$ such that for some m-place argument filler σ_1 and some n-place argument filler σ_2, $t^{\sigma_1}(t_1, \ldots, t_m) = t'^{\sigma_2}(t'_1, \ldots, t'_n)$.

Definition 5.2.14. A multiset of *PTL* terms is called *pairwise independent* iff no term occurs more than once in it and any two distinct terms in it are independent.

Definition 5.2.15. A *PTL* text θ is called *semi-nice* iff the multiset of occurrences of terms after an ι or $\exists$ in θ is pairwise independent.

Definition 5.2.16. A *PTL* text θ is called *nice* iff it is semi-nice and ground.

Lemma 5.2.17. *For semi-nice PTL texts $\varphi \wedge \psi$, $\theta \,\&\, \xi$ and $\exists t\, \varphi$, the following simplified characterization of* **aq** *is correct:*

$$\begin{aligned}\mathbf{aq}(\varphi \wedge \psi) &= \mathbf{aq}(\varphi) \cup \mathbf{aq}(\psi)\\ \mathbf{aq}(\theta \,\&\, \xi) &= \mathbf{aq}(\theta) \cup \mathbf{aq}(\xi)\\ \mathbf{aq}(\exists t\, \varphi) &= \mathbf{aq}(\varphi) \cup \{(\exists t, t)\}\end{aligned}$$

Proof. Trivial. □

Lemma 5.2.18. *Let θ be a PTL text and $t \in \mathbf{tbc}(\theta)$. Then for some n, there is an n-place argument filler σ and terms $t_1, \ldots, t_n$ such that $\exists t^{\sigma}(t_1, \ldots, t_n)$ appears in θ.*

Proof sketch. This follows by an easy inductive proof from the definition of **aq**. □

Lemma 5.2.19. *If ξ_1 and ξ_2 are subtexts of a semi-nice PTL text θ, then $\mathbf{tbc}(\xi_1) \oplus \mathbf{tbc}(\xi_2)$ is pairwise independent.*[6]

Proof. Trivial by Lemma 5.2.18. □

The following lemma characterizes the relationship between the syntactic definition of **tbc** and the definition of *PTL* semantics:

Lemma 5.2.20. *Let θ be a semi-nice PTL text, and let M be a CMTN model. If g and h are M-assignments such that $h \in [\![\theta]\!]^g_M$ and such that the union of $dom(g)$ and the set of occurrences of terms after an ι or $\exists$ in θ is pairwise independent, then $\mathbf{tbc}(\theta) = dom(h) \setminus dom(g)$.*

We do not yet have the machinery required for proving this lemma: The proof of the lemma goes by induction over the complexity of θ, with an eleven-case case distinction covering the possible forms θ can have according to Definition

[6] Here the sets $\mathbf{tbc}(\xi_1)$ and $\mathbf{tbc}(\xi_2)$ are considered as multisets with each element appearing once in each of them. This allows us to apply $\oplus$ to them, and in the multiset $\mathbf{tbc}(\xi_1) \oplus \mathbf{tbc}(\xi_2)$ some elements may appear twice.

5.2.1. All cases apart from θ being of the form $\varphi \to \theta'$ are trivial, but this non-trivial case requires us to make use of the Map Extensionality Axiom Schema of *CMTN* to prove the existence of certain functions in M under certain conditions, which are expressed in terms of $[\![\varphi]\!]^g_M$ and $[\![\theta']\!]^k_M$. In order to make use of this first-order axiom schema, we first need to translate these conditions into first-order statements. For this we need a translation from *PTL* texts to first-order that conserves truth conditions. Since we develop such a translation as part of the proof checking algorithm treated in chapter 6, we postpone the proof of this lemma to section 6.3.1.

The following lemma characterizes the truth conditions of $[\![\varphi \to \theta]\!]^g_M$:

Lemma 5.2.21. *Let M be a CMTN model, g an M-assignment, φ a PTL formula and θ a PTL text such that $[\![\varphi \to \theta]\!]^g_M$ is defined. Then $[\![\varphi \to \theta]\!]^g_M \neq \emptyset$ iff for every $k \in [\![\varphi]\!]^g_M$, $[\![\theta]\!]^k_M \neq \emptyset$.*

Just as the proof of the previous lemma, the proof of this lemma depends on the machinery from the next chapter and will be proven in section 6.3.1.

Chapter 6

A proof checking algorithm for Proof Text Logic

In this chapter, we present the proof checking algorithm of the Naproche system in a formal way. The goal of the proof checking algorithm is to determine validity of a *PTL* text as defined by the absolute validity function v from Definition 5.2.12. Since the proof checking algorithm for *PTL* is rather involved, we first present an analogous proof checking algorithm for *DPL*, which is significantly simpler than the algorithm for *PTL*, and which helps to clarify its basic functioning.

In order to simplify the definition of the proof checking algorithm, we will assume that it works on a nice *PTL* text (as defined in Definition 5.2.16; roughly speaking, the niceness of a *PTL* text amounts to the avoidance of variable clashes and the avoidance of variables that were not dynamically introduced through an existential quantifier). Since all *PTL* texts that result from Naproche CNL texts are nice, this is not a significant limitation.

6.1 From *DPL* to *PTL* proof checking

6.1.1 A proof checking algorithm for *DPL*

In order to understand the purpose of this algorithm, one has to view the *DPL* formula that is given to the algorithm as representing the content of a mathematical proof: Assumptions and their consequences are represented by *DPL* formulae of the form $(\varphi \to \psi)$, and the simple linear concatenation of reasoning steps is represented by *DPL* conjunction with $\wedge$. We will assume that the algorithm works on a nice *DPL* formula, where the definition of niceness for *DPL* formulae is analogous to that for *PTL* texts in 5.2.16.

The checking algorithm keeps track of a list of *PL* formulae considered to be true, called *premises*, which gets continuously updated during the checking process. Each assertion is checked by an automated theorem prover (ATP) based on the currently active premises. In practice, the ATP has to be given a time limit. Our formalization of *provers* presented below corresponds to an actual ATP with a fixed time limit (and, of course, with other possible parameters of the ATP fixed as well).

In order to make clear the distinction between *DPL* or *PTL* formulae/texts on the one hand and *PL* formulae used as a translation of these for the proof checking algorithm on the other hand, we will use small Greek letters (like φ, ψ, χ, θ, ξ) for the first and capital Greek letters (like Φ, Ψ, X, Θ, Ξ) for the second. We will always use the corresponding small and capital Greek letters for the original formula and its translation. The distinction between φ, ψ and χ on the one hand and θ and ξ on the other hand is used, just as already practised in the previous chapter, for the distinction between *PTL* formulae and *PTL* texts. For clearly distinguishing *PTL* terms from *PL* terms we analogously use a small t and a capital T.

Definition 6.1.1. A *proof obligation* is of the form $\Gamma \vdash^? \Phi$, where Γ is a finite sequence of premises and Φ is a *PL* formula.

Definition 6.1.2. A *prover* P is a function from proof obligations to $\{-1, 0, 1\}$ such that if $P(\Gamma \vdash^? \Phi) = -1$ then $\Gamma \nvdash \Phi$, and if $P(\Gamma \vdash^? \Phi) = 1$ then $\Gamma \vdash \Phi$.

The intended meaning of this function is that given an ATP P and an obligation o of the form $\Gamma \vdash^? \Phi$, $P(o) = 1$ means that P can prove that $\Gamma \vdash \Phi$, $P(o) = -1$ means that P can prove that $\Gamma \nvdash \Phi$, and $P(o) = 0$ means that P cannot determine whether $\Gamma \vdash \Phi$ in the time it was given for the task.

Based on these three possible outputs of single calls of ATPs, there are also three possible outputs for the proof checking algorithm: The proof checking algorithm can determine that its input *DPL* formula is a tautology (i.e. represents a proof without erroneous proof steps), it can determine that it is not a tautology (i.e. represents a poof with erroneous proof steps), or it may fail to determine which of these two cases holds. The most important distinction for practical purposes is that between the first and the other two cases. For the sake of simplicity, we will present our formal definitions of the proof checking algorithms for *DPL* and *PTL* only with this distinction. In section 6.5, we will sketch how the proof checking algorithm can provide for the distinction between the latter two cases.

The final output of the proof checking algorithm is defined by keeping track of a *proof status value*, whose possible values are $\bot$ and $\top$, and which is set to $\top$ at the beginning and updated at every call of the prover with the update function *update* defined below. *update* takes two arguments: the previous proof status value, and the output of the prover. The output of *update* is considered the new proof status value.

Definition 6.1.3. We define an *update* function *update* from $\{\bot, \top\} \times \{-1, 0, 1\}$ to $\{\bot, \top\}$ by

$$update(\mu, i) := \begin{cases} \top & \text{if } \mu = \top \text{ and } i = 1 \\ \bot & \text{otherwise.} \end{cases}$$

We present the proof checking algorithms in pseudo-Prolog-code. This pseudo-code is based on Prolog syntax and standard indentation conventions for Prolog, as used for example in Blackburn et al. (2006), enriched by some self-explanatory natural language descriptions of the algorithm. In order to make the distinction between input and output values of a function defined using Prolog predicates more visible, we usually write $p(X_1, \ldots, X_n) = (Y_1, \ldots, Y_m)$ in pseudo-code, where the actual Prolog code would read $p(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$,

and where p is a Prolog predicate representing a function that takes $X_1, \ldots, X_n$ as input and $Y_1, \ldots, Y_m$ as output values.

Now we are ready for defining the function *check* that formalizes the proof checking algorithm:

Definition 6.1.4.

$check(\varphi) = (\nu)$:-[1]
  $check\_text(\varphi, \langle\rangle, \langle\rangle, \top) = (\_, \_, \nu)$.[2]

$check\_text(\top, \Gamma, \mathbb{V}, \mu) = (\Gamma, \mathbb{V}, \mu)$.[3]
$check\_text(R(t_1, \ldots, t_n), \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle R(t_1, \ldots, t_n)\rangle, \mathbb{V}, \nu)$:-
  $\nu = update(\mu, P(\Gamma \vdash^? R(t_1, \ldots, t_n)))$.[4]
$check\_text(\neg\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle\neg\exists\mathbb{V}_1\ \Phi\rangle, \mathbb{V}, \nu)$:-
  $read\_text(\varphi, \mathbb{V}) = (\mathbb{V}_1, \Phi)$,
  $\nu = update(\mu, P(\Gamma \vdash^? \neg\exists\mathbb{V}_1\ \Phi))$.
$check\_text((\varphi \wedge \psi), \Gamma, \mathbb{V}, \mu) = (\Gamma_2, \mathbb{V}_2, \nu)$:-
  $check\_text(\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma_1, \mathbb{V}_1, \mu_1)$,
  $check\_text(\psi, \Gamma_1, \mathbb{V}_1, \mu_1) = (\Gamma_2, \mathbb{V}_2, \nu)$.
$check\_text((\varphi \vee \psi), \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle(\exists\mathbb{V}_1\ \ \Phi \vee \exists\mathbb{V}_2\ \ \Psi)\rangle, \mathbb{V}, \nu)$:-
  $read\_text(\varphi, \mathbb{V}) = (\mathbb{V}_1, \Phi)$,
  $read\_text(\psi, \mathbb{V}) = (\mathbb{V}_2, \Psi)$,
  $\nu = update(\mu, P(\Gamma \vdash^? (\exists\mathbb{V}_1\ \Phi \vee \exists\mathbb{V}_2\ \Psi)))$.
$check\_text((\varphi \rightarrow \psi), \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle\forall\mathbb{V}_1\ \ (\Phi \rightarrow \Psi)\rangle, \mathbb{V}, \nu)$:-
  $read\_text(\varphi, \mathbb{V}) = (\mathbb{V}_1, \Phi)$,
  $check\_text(\psi, \Gamma \oplus \langle\Phi\rangle, \mathbb{V} \oplus \mathbb{V}_1, \mu) = (\Gamma_1, \mathbb{V}_2, \nu)$,
  $\mathbb{V}' = \mathbb{V}_2 - (\mathbb{V} \oplus \mathbb{V}_1)$,
  $\Psi = \exists\mathbb{V}\,' \bigwedge(\Gamma_1 - (\Gamma \oplus \langle\Phi\rangle))$.
$check\_text(\exists x\ \varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle\Phi\rangle, \mathbb{V} \oplus \mathbb{V}_0 \oplus \langle x\rangle, \nu)$:-
  $read\_text(\varphi, \mathbb{V} \oplus \langle x\rangle) = (\mathbb{V}_0, \Phi)$,
  $\nu = update(\mu, P(\Gamma \vdash^? \exists x\ \exists\mathbb{V}_0\ \Phi))$.
$check\_text(\Diamond\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma \oplus \langle\exists\mathbb{V}_1\ \ \Phi\rangle, \mathbb{V}, \nu)$:-
  $check\_text(\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma', \mathbb{V}', \nu)$,
  $\mathbb{V}_1 = \mathbb{V}' - \mathbb{V}$,
  $\Phi = \bigwedge(\Gamma' - \Gamma)$.

$read\_text(\varphi, \mathbb{V}) = (\mathbb{V}' - \mathbb{V}, \bigwedge\Gamma)$:-[5]

[1] The *check* function has one input argument and one output argument: its input is a *DPL* formula, and its output is a truth value that indicates whether the proof checking was successful or not.

[2] Since we use this algorithm to clarify the *PTL* proof checking algorithm, we already use *PTL*-like terminology for naming the functions: Hence *check_text* rather than *check_formula*.

[3] The *check_text* function has four input and three output arguments. The first input argument specifies the *DPL* formula to be checked. The remaining three input arguments as well and the three output arguments respectively keep track of the currently active premise list, the currently active list of accessible variables and the current proof status value. Since the value of these three items may change during the proof checking of the input *DPL* formula, we have an input and an output argument for each of these three values.

[4] This case also covers *DPL* formulae of the form $t_1 = t_2$.

[5] The *read_text* function *reads in* a *DPL* formula and translates it to *PL*. It has two input arguments and two output arguments: The first input argument is the *DPL* formula to be read in, and the second argument is the list of accessible variables that is active when the function is called. The first output argument lists all variables that the active quantifiers of the input *DPL* formula quantify over (see Definition 3.1.8 in chapter 3); the second output

$\mathit{check\_text}(\varphi, \langle\rangle, \mathbb{V}, \top) = (\Gamma, \mathbb{V}', \_)$.

If one keeps in mind that *DPL* formulae of the form $\varphi \wedge \psi$ represent concatenated parts of a proof, that *DPL* formulae of the form $\varphi \to \psi$ represent assumptions and their consequences and that all other *DPL* formulae represent single statements in a proof, one can easily see that the above defined proof checking algorithm formalizes the basic idea that the algorithm sequentially works through a proof, keeping track of a list of premises that represent the mathematical information gathered so far, adding information whenever a new assumption or assertion is made, and checking all new assertions for correctness based on the currently active premise list.

The *read_text* function serves the purpose of translating a *DPL* formula into a *PL* formula without checking the translated *DPL* formula for proof correctness. This is achieved by calling *check_text* but ignoring its third output value, which indicates the proof correctness of the checked formula. This means that all proof obligations sent to an ATP within a *check_text* procedure called by *read_text* can actually be ignored: In practice, one does not need to call the ATP at all in these cases.

Note how the dynamic character of the existential quantifier is realized in the proof checking algorithm: When an existential statement appears as an assertion, what has to be checked by the ATP is actually an existentially quantified *PL* formula. But what is added to the premise list does not have the existential quantifier, so that the previously existentially quantified variables are now free variables. If a later assertion in the proof mentions the dynamically introduced variable again, its translation to *PL* will also contain that variable as a free variable. We then have a proof obligation of the form $\ldots, \varphi(x), \cdots \vdash^? \psi(x)$, where x occurs freely on both sides of $\vdash^?$. Such a free variable behaves exactly like a constant symbol: $\ldots, \varphi(x), \ldots \models \psi(x)$ iff $\ldots, \varphi(c), \ldots \models \psi(c)$ for a new constant symbol c (i.e. a constant symbol that appears nowhere in "$\ldots, \varphi(x), \ldots \models \psi(x)$"). So what is added to the premise list can be considered the *Skolemized*[6] form of the checked existentially quantified *PL* formula. Without this Skolemization of existential statements, a later reuse of a dynamically introduced variable could not be modelled: If we had $\exists x\ \varphi(x)$ instead of $\varphi(x)$ in our premise list, the x in $\psi(x)$ after the $\vdash^?$ could not be made to refer to the same object of our domain as the x in $\varphi(x)$.

6.1.2 Soundness of the *DPL* proof checking algorithm

One can prove the following soundness theorem for this proof checking algorithm:

Theorem 6.1.5 (Soundness of the *DPL* proof checking algorithm)**.** *If φ is a nice DPL formula and check$(\varphi) = \top$, then φ is a tautology.*

Since we are really interested in the *PTL* proof checking algorithm, and use the *DPL* proof checking algorithm only as a simplified case in order to explain its basic functioning, we will not prove this theorem in detail, but only sketch

argument is the *PL* translation of the input *DPL* formula (without existential quantification over the active quantifiers of the input *DPL* formula).

[6] For a general introduction to the technique of Skolemization, see for example Brachman and Levesque (2004, p. 64).

a proof plan. The basic idea is to prove it by induction on the length of φ. But the recursive part of the definition of the proof checking algorithm does not lie in the function *check*, but in the function *check_text*. Hence the inductive hypothesis has to say something about the implications of $\mathit{check\_text}(\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma', \mathbb{V}', \nu)$, not just about the implications of $\mathit{check}(\varphi) = \nu$. Since these implications are somewhat involved, we write down a separate lemma for them, the *DPL* Detailed Soundness Lemma below.

This lemma does not only need to ensure the correctness of the output value ν, as the above soundness theorem, but also the correctness of the other two output values, Γ' and $\mathbb{V}'$. Additionally, the lemma needs to ensure that certain technical properties about the relation between $\mathbb{V}$, $\mathbb{V}'$ and the variables that are quantified in φ are conserved. To understand the purpose of these technical properties, note that the niceness of the input formula φ amounts to the avoidance of variable clashes and the avoidance of variables that were not dynamically introduced through an existential quantifier. In order to ensure that these two avoidances have their desired effect in the proof checking algorithm, we need to ensure that corresponding avoidances are conserved at the inductive step of the proof.

Lemma 6.1.6 (*DPL* Detailed Soundness Lemma)**.** *Let φ be a semi-nice DPL formula. Further assume the following properties:*

(i) $\mathbb{V}$ *is a set of variables that do not occur in* φ *after an* $\exists$*.*

(ii) All free variables of φ *are in* $\mathbb{V}$*.*

(iii) Γ *is a premise list such that all free variables in* Γ *are in* $\mathbb{V}$*.*

(iv) $\mathit{check\_text}(\varphi, \Gamma, \mathbb{V}, \mu) = (\Gamma', \mathbb{V}', \nu)$*.*

(v) M *is a structure and* g *an* M*-assignment such that* $M, g \models \Gamma$*.*

(vi) $\mathit{dom}(g) = \mathbb{V}$*.*

Then the following four properties hold:

1. $\mathbf{aq}(\varphi) = \mathbb{V}' - \mathbb{V}$.[7]
2. *All free variables in* $\Gamma' - \Gamma$ *are in* $\mathbb{V}'$*.*
3. *For all* M*-assignments* k*, the following two properties are equivalent:*
   (a) $k \in [\![\varphi]\!]^g_M$*.*
   (b) $k[\mathbb{V}' - \mathbb{V}]g$ *and* $M, k \models \Gamma' - \Gamma$*.*
4. *If* $\nu = 1$*, then* $[\![\varphi]\!]^g_M \neq \emptyset$ *and* $\mu = 1$*.*

The proof of this lemma now mainly consists of checking these four properties for all of the eight cases in the above definition of *check_text*. We refrain from presenting the proof here, but will present the proof for the corresponding Detailed Soundness Lemma for the *PTL* proof checking algorithm.

[7] We should actually say that the set whose elements are the elements of the sequence $\mathbb{V}' - \mathbb{V}$ is equal to $\mathbf{aq}(\varphi)$. For the sake of simplicity and since it does not cause problems, we use the simplified expression $\mathbf{aq}(\varphi) = \mathbb{V}' - \mathbb{V}$ instead.

6.1.3 Proof checking with presuppositions[8]

We will now describe what features have to be added to the above proof checking algorithm for *DPL* in order to transform it to a proof checking algorithm for *PTL*. First we consider how presuppositions have to be treated in such a proof checking algorithm.

In section 3.2 of chapter 3, we described the behaviour of presuppositions using the concept of the *context* in which an utterance is interpreted. We gave different accounts of what contexts could be. When working with *PTL* semantics, it is natural to identify contexts with pairs (g, M) consisting of a *CMTN* model M and an M-assignment g. When working with a proof checking algorithm like the one described above, on the other hand, it is natural to identify contexts with the premise lists that the proof checking algorithm keeps track of.

As noted in section 1.1, assertions in mathematical texts are expected to be logically implied by the available knowledge rather than adding something logically new to it. Because of this pragmatic peculiarity of mathematical texts, both presuppositions and assertions in proof texts have to follow logically from the context. For a sentence like "The largest element of M is finite" to be legitimately used in a mathematical text, both the unique existence of a largest element of M and its finiteness must be inferable from the context.

The remaining distinctive feature between assertions and presuppositions is that the failure of the latter ones makes the containing sentences meaningless, not only false. We have already accounted for the distinction between false and meaningless sentences in the definition the validity function v for *PTL* texts (Definition 5.2.12). So we will now work with three possible proof status values u, $\top$ and $\bot$, which are also the three elements of the codomain of v.

A proof is only considered correct if its proof status value is $\top$, so the distinction between $\top$ and the other two proof status values is more important than that between u and $\bot$. If we ignore the difference between u and $\bot$, what we said about the example sentence above results in treating presuppositions and assertions in the same way. This parallel treatment of presuppositions and assertions, however, does not necessarily hold for presupposition triggers that are subordinated by a logical operation like negation or implication. For example, in the sentence "A does not contain the empty set", the existence and uniqueness presuppositions do not get negated, whereas the containment assertion does. This is explained in the following way: In order to *make sense of* the negated sentence, we first need to make sense of what is inside the scope of the negation. In order to make sense of some expression, all presuppositions of that expression have to follow from the current context. The presuppositions triggered by "the empty set" are inside the scope of the negation, so they have to follow from the current context. The containment assertion, however, does not have to follow from the current context, since it is not a presupposition, and since it is negated rather than being asserted affirmatively.

In the proof checking algorithm, *making sense of* a *PTL* text corresponds to processing it in some way, whether using *read_text* or whether using *check_text* directly without it being called by *read_text*. In the *DPL* proof checking algorithm, all calls of ATPs within a call of *read_text* could be ignored. But according to the above explanation, ATP calls that check presuppositions also have to be checked when they are inside a call of *read_text*.

[8] This section is largely taken over from Cramer, Kühlwein, and Schröder (2010).

For example, the *PTL* text representing the sentence (1) is (2).

(1) A does not contain the empty set.

(2) $\neg \mathit{contains}(A, \iota x(\mathit{empty}(x) \wedge \mathit{set}(x)))$

When the checking algorithm encounters the negated *PTL* text, it needs to find the *PL* translation of the *PTL* text in the scope of the negation, for which it will call *read_text*. Now the ι triggers two presuppositions, which have to be checked despite being within a call of *read_text*. So we send the proof obligations (3) and (4) (for a new constant symbol c) to the ATP. Finally, the proof obligation that we want for the assertion of the sentence is (5).

(3) $\Gamma \vdash^? \exists x(\mathit{empty}(x) \wedge \mathit{set}(x))$

(4) $\Gamma \cup \{\mathit{empty}(c) \wedge \mathit{set}(c)\} \vdash^? \forall y(\mathit{empty}(y) \wedge \mathit{set}(y) \rightarrow y = c)$

(5) $\Gamma \cup \{\mathit{empty}(c) \wedge \mathit{set}(c), \forall y(\mathit{empty}(y) \wedge \mathit{set}(y) \rightarrow y = c)\} \vdash^? \neg \mathit{contain}(A, c)$

In order to get this, we need to have $\mathit{contain}(A, c)$ as our *PL* translation of the *PTL* text in the scope of the negation. In the *DPL* proof checking algorithm, we conjuncted all premises that had been added to the premise list in the course of processing a *DPL* formula in order to define the translation of that *DPL* formula to *PL*. This will no longer work, since we add $\mathit{empty}(c) \wedge \mathit{set}(c)$, $\forall y(\mathit{empty}(y) \wedge \mathit{set}(y) \rightarrow y = c)$ and $\mathit{contain}(A, c)$ to the premise list while processing $\mathit{contains}(A, \iota x(\mathit{empty}(x) \wedge \mathit{set}(x)))$, but we only want $\mathit{contain}(A, c)$ to be the *PL* translation. The solution is to conjunct only those premises that had been added to the premise list without originating from a presupposition. The premises originating from presuppositions have to be added to the list of premises that were active before calling *read_text*. This means that the new proof checking algorithm has to keep track of which premises originate from presuppositions and which do not.

This pulling out of presuppositional premises is not always as simple as in the above example. Consider for example sentence (6), whose (somewhat simplified) representation in *PTL* is (7).

(6) There is a finite non-empty set M of natural numbers such that the largest element of M is even.[9]

(7) $\exists M\ (\mathit{finite}(M) \wedge \mathit{non\text{-}empty}(M) \wedge \mathit{set\_of\_nats}(M) \wedge \mathit{even}(\iota x\ \mathit{largest\_elt}(x, M)))$

The Skolemized premise from the existential presupposition is $\mathit{largest\_elt}(c, M)$. According to the above account, it should be added to the set Γ of premises available before encountering this sentence, and this extended premise list should

[9] The definite noun phrase "The largest element of M" can be read like a function depending on M. When, like in our example, such functional definite descriptions are used as functions on a variable that we are quantifying over, the presuppositions of the functional definite description can restrict the domain of the quantifier to entities for which the presupposition is satisfied. Such a restriction of a quantifier is an instance of accommodation (local accommodation in our account), which will be treated in section 7.5.10. In this section we are interested in presupposition handling without accommodation, i.e. without restricting the domain of the quantifier in this example. So the presuppositions of "the largest element of M" have to be fulfilled for any finite non-empty set M of natural numbers.

be used for proving the existential statement asserting the existence of M. But *largest_elt*(c, M) contains a free occurrence of the variable M, so that this would result in this occurrence of M being pulled out of the scope of the quantifier introducing M, which makes the pulled out premise meaningless. Hence we need a more sophisticated approach to pulling out presuppositional premises:

According to the above account, we will check the existential presupposition in question using the proof obligation (8). Given that M does not appear in Γ (as it is a newly introduced variable), this is logically equivalent to having checked (9), whose Skolemized form (10) will be added to Γ (where sk_x is the new function symbol introduced for x when Skolemizing). This extended premise set is used to check the existential claim of the sentence in (11).

(8) $\Gamma \cup \{\mathit{finite}(M), \mathit{non\text{-}empty}(M), \mathit{set\_of\_\ nats}(M)\} \vdash^? \exists x\ \mathit{largest\_elt}(x, M)$

(9) $\Gamma \vdash^? \forall M\ (\mathit{finite}(M) \wedge \mathit{non\text{-}empty}(M) \wedge \mathit{set\_of\_nats}(M) \rightarrow \exists x\ \mathit{largest\_elt}(x, M))$

(10) $\forall M\ (\mathit{finite}(M) \wedge \mathit{non\text{-}empty}(M) \wedge \mathit{set\_\ of\_nats}(M) \rightarrow \mathit{largest\_elt}(sk_x(M), M))$

(11) $\Gamma \cup \{(10)\} \vdash^? \exists M\ (\mathit{finite}(M) \wedge \mathit{non\text{-}empty}\,(M) \wedge \mathit{set\_of\_nats}(M) \wedge \mathit{even}(sk_x(M)))$

This Skolemization of presuppositional premises that are pulled out when calling *read_text* will be realized through a separate predicate *pull_out_pres* in the definition of the *PTL* proof checking algorithm below (Definition 6.2.8).

For formalizing this Skolemization, we need for every $n \geq 0$ an infinite supply $\{sk_i^n \mid i \in \mathbb{N}\}$ of skolem function symbols of arity n (the 0-ary skolem function symbols could be considered constant symbols, but for avoiding unnecessary case distinctions, it is useful to consider them function symbols too). We usually omit the superscript indicating the arity of a skolem function symbol. In the definition of the *PTL* proof checking algorithm, we often have to ensure that we use a new skolem function symbol. In such cases, we use the notation sk^{new}; the intended meaning is sk_i for an $i \in \mathbb{N}$ that has not been used for any skolem function symbol used so far in the algorithm. If sk_{new} appears more than once in a clause of pseudo-Prolog-code, the intended meaning is that these occurrences refer to the same sk_i.

Since we now have a third proof status value, namely u, we need to adapt our definition of the update function. The update function now takes an additional argument that specifies whether the prover is called for a presupposition check (0) or for an assertion check (1). (This additional argument is in the second position, and the originally second argument is now the third argument.)

Definition 6.1.7. We define an update function *update* from $\{\bot, \top, u\} \times \{0, 1\} \times \{-1, 0, 1\}$ to $\{\bot, \top, u\}$ by

$$\mathit{update}(\mu, i, j) := \begin{cases} \mu & \text{if } j = 1 \\ \bot & \text{if } j \neq 1,\ i = 1 \text{ and } \mu \neq u \\ u & \text{otherwise (i.e. if } j \neq 1 \text{ and either } i = 0 \text{ or } \mu = u). \end{cases}$$

Just as the premises added to a premise list while processing a *DPL* formula φ also served the purpose of characterizing $[\![\varphi]\!]^g_M$ in statement 3 of the *DPL* Detailed Soundness Lemma, so the presuppositionally marked premises added to a premise list while processing a *PTL* text θ will also serve the purpose of

characterizing conditions for $def([\![\varphi]\!]^g_M)$ in the Detailed Soundness Lemma for the *PTL* proof checking algorithm below.

If we have partial functions that trigger presuppositions, we have a separate problem: No matter whether we translate the functions into *PL* directly using a separate function symbol for every function or whether we translate them using a separate function symbol *app* for function application, we have terms in the *PL* language we translate into that correspond to applications of functions to objects not in their domain. For this not to cause problems, we have to assume that the structure described by the *PL* formulae that we use as translations contains a separate object for undefinedness. As usual, we use the constant symbol u to refer to this object.

Quantifiers in formalisms like *HODPL* or *PTL* of course do not range over this undefinedness object. So when translating these quantifiers into *PL*, we now have to ensure that we do not quantify over the undefinedness object. So the translation of $\exists x\ \varphi$ has to be $\exists x\ (x \neq u \wedge \mathrm{tr}(\varphi))$, where $\mathrm{tr}(\varphi)$ is the translation of φ.

6.1.4 Proof checking with implicit dynamic function introduction

In order to include implicit dynamic function introduction into the proof checking algorithm, the algorithm has to work on formulae from some formalism that formalizes implicit dynamic function introduction, e.g. on *HODPL* formulae or *PTL* texts. Since we are working towards a proof checking algorithm for *PTL*, we will use *PTL* terminology in this section, but analogous statements could be made if one were to adapt the *DPL* proof checking algorithm to *HODPL*.

PTL has quantification over arbitrary terms from T_{PTL}, not just quantification over variables. But in the translation to *PL* we can only allow quantification over variables, so that we have to get rid of quantification over complex terms in the translation process. We solve this problem together with the above mentioned problem of avoiding quantification over the undefinedness object by translating *PTL* quantifiers using a special operation on *PL* formulae defined as follows:

Definition 6.1.8. Given a list $\mathbb{T} = \langle t_1, \ldots, t_n \rangle$ of *PL* terms and a *PL* formula φ, we define $\exists_{\mathbb{T}}\ \varphi$ to be the formula $\exists x_1\ \ldots \exists x_n\ (x_1 \neq u \wedge \ldots \wedge x_n \neq u \wedge \varphi \frac{x_1}{t_1} \ldots \frac{x_n}{t_n})$, where $x_1, \ldots, x_n$ are variables not occurring in φ.[10] Similarly, $\forall_{\mathbb{T}}\ \varphi$ is defined to be the formula $\forall x_1\ \ldots \forall x_n\ (x_1 \neq u \wedge \ldots \wedge x_n \neq u \rightarrow \varphi \frac{x_1}{t_1} \ldots \frac{x_n}{t_n})$, where $x_1, \ldots, x_n$ are variables not occurring in φ.[10]

Instead of keeping track of a list of variables that have been dynamically introduced up to a given point, the *PTL* proof checking algorithm keeps track of a list of terms that have been dynamically introduced.

Remember how we checked *DPL* formulae of the form $\varphi \rightarrow \psi$ in the *DPL* proof checking algorithm: First we translated φ to a *PL* formula Φ using *read_text* (keep in mind that Φ does not contain existential quantifiers for the variables that are introduced through dynamic existential quantification in φ). Having added Φ to our premise list, we checked ψ. To calculate a translation

[10] For this formula to be uniquely defined, one would have to specify a way of choosing n variables not appearing in a given formula φ.

Ψ of ψ, we took the conjunction of all premises added while checking ψ, and existentially quantified the result using all variables dynamically introduced in ψ. Finally we added $\forall\mathbb{V}_1\ (\Phi \to \Psi)$ to the original premise list, where $\mathbb{V}_1$ is the list of variables dynamically introduced in φ.

For checking a *PTL* text of the form $\varphi \to \theta$, we do precisely the same thing up to the point of checking θ. But before we go on to calculate the translation Θ of θ, we look out for implicitly introduced functions: Suppose that $\mathbb{T}_1$ is the list of terms dynamically introduced in φ. Then we check for every term dynamically introduced in θ whether it is of the form $T^\sigma(\mathbb{T}_1)$ for some term T and some *length*$(\mathbb{T}_1)$-place argument filler σ.[11] If it is, then T is an implicitly introduced function. Let $\mathbb{T}'$ be the list of terms introduced in θ that are not of this form, i.e. are not used for implicitly introducing functions. Now the translation Θ of θ is the conjunction of all non-presuppositionally marked premises that were added while checking θ, existentially quantified with $\exists_{\mathbb{T}'}$. The formula that we add to the original premise list to represent the content of the implication now is $\forall_{\mathbb{T}_1}\ (\Phi \to \Theta)$. In this way the terms referring to implicitly introduced functions remain unquantified in the premise list, as should be the case for dynamically introduced entities.

Additionally we need to add to our premise list information about the domain of implicitly introduced functions. According to our definition of the semantics of $\varphi \to \theta$, the domain of a function implicitly introduced in θ is the set of n-tuples satisfying φ, where n is the number of variables dynamically introduced by φ. This can be formalized in our *PL* translation as $\forall_{\mathbb{T}_1}\ (\Phi \leftrightarrow T^\sigma(\mathbb{T}_1) \neq u)$. But remember that because of our definition of n-place argument fillers, $T^\sigma(\mathbb{T}_1)$ could accept some arguments first, becoming a function that accepts the remaining arguments. For example, it could be of the form $f(x)(y)$, where f accepts x first to become a function $f(x)$ that can further be evaluated at y. In this case, we do not only need to store information about the domain of f, but also about the domain of $f(x)$. For being able to talk about all functions extractable in this way from $T^\sigma(\mathbb{T}_1)$, we need the following definition:

Definition 6.1.9. We recursively define a *PL* term T_1 to be a *function-head subterm* of a *PL* term T_2 iff $T_1 = T_2$ or T_2 is of the form $app_n(T, \mathbb{T})$ (for some term T and term list $\mathbb{T}$ of length n) and T_1 is a function-head subterm of T.

Now the domain information that the definition of the semantics of $\varphi \to \theta$ allows us to add to the premise list is the following: For every function-head subterm F of $T^\sigma(\mathbb{T}_1)$ with $F \neq T$, we add $\forall_{\mathbb{T}'}\ (\exists_{\mathbb{T}_1 - \mathbb{T}'}\ \Phi \leftrightarrow F_i \neq u)$ to the premise list, where $\mathbb{T}' = \langle T_0 \in \mathbb{T}_1 \mid T_0 \text{ occurs in } F\rangle$.

The Functionality Axiom Schema of *CMTN* gives a criterion for ensuring that maps proved to exist using Map Comprehension are actually limited (i.e. functions). In the proof checking algorithm, we need an analogous criterion that ensures that an implicitly introduced function is limited. The criterion in the Functionality Axiom Schema consists of two parts: The first is that the parametrized formulae $P(\bar{z})$ and $R(\bar{z}, x)$ used for Map Comprehension do not contain the symbol L; the second is that their parameters are limited.

[11] This use of the superscript σ notation is a slight abuse of notation: $T^\sigma(\mathbb{T}_1)$ really is of the form $app_{k_m}(\ldots app_{k_1}(T, T_{1,1}, \ldots, T_{1,k_1}), \ldots, T_{m,1}, \ldots, T_{m,k_m})$, but our convention about writing terms with app_n allows us to write $T^\sigma(\mathbb{T}_1)$ as $T(T_{1,1}, \ldots, T_{1,k_1}) \ldots (T_{m,1}, \ldots, T_{m,k_m})$, which according to Definition 5.1.6 is a possible value of $T^\sigma(\mathbb{T}_1)$. We will use this abuse of notation from now on without further comment.

Before we explain how we adapt these criteria to the proof checking algorithm, we want to discuss a special case, where the second criterion is not fulfilled, but can be made to be fulfilled: Suppose that $P(\bar{z})$ and $R(\bar{z}, x)$ do not contain the symbol L, but do contain instances of a single unlimited parameter p, which appears in $P(\bar{z})$ and $R(\bar{z}, x)$ only in terms of the form $app_1(p, t)$, where t is some term. Suppose furthermore that for some formula $\varphi(x)$ without unlimited parameters and not containing the symbol L, we can prove $\forall x\ (app_1(p, x) \leftrightarrow \varphi(x))$. In that case we can replace every occurrence of $app_1(p, t)$ in $P(\bar{z})$ and $R(\bar{z}, x)$ by $\varphi(t)$. The resulting formulae $P'(\bar{z})$ and $R'(\bar{z}, x)$ are equivalent to $P(\bar{z})$ and $R(\bar{z}, x)$ respectively, still do not contain the symbol L, and no longer contain any unlimited parameters. Hence Functionality may be applied with $P'(\bar{z})$ and $R'(\bar{z}, x)$ instead of $P(\bar{z})$ and $R(\bar{z}, x)$ in order to prove the map in question to be limited.

Now we discuss how to adapt the criteria imposed by the Functionality Axiom Schema of *CMTN* to the proof checking algorithm. The first criterion is easily adapted to the proof checking algorithm: The premises added to the premise list while processing φ and θ may not contain the symbol L.

The adaptation of the second criterion requires calling the ATP: For every term T that is either a skolem function symbol or from the list of terms that had been dynamically introduced before processing φ and that occurs in some premise added to the premise list while processing φ or θ, we have to let the ATP check a proof obligation of the form $\Gamma' \vdash^? L(T)$, where Γ' is the active list of premises after processing θ. The special case discussed above can be captured as follows in the proof checking algorithm: Suppose T is a term for which the above adaptation of the second criterion was not successful. Suppose furthermore that T appears in the added premises only within terms of the form $app_1(T, T')$ and that there is a formula of the form $\forall x\ (app_1(T, x) \leftrightarrow \varphi(x))$ in Γ', where $\varphi(x)$ does not contain L. Then for every term T^* in φ that is either a skolem function symbol or from the list of terms that had been dynamically introduced before processing φ, we check whether T^* is limited in the same way in which we have checked it above for T. In other words, we let the ATP check a proof obligation of the form $\Gamma' \vdash^? L(T^*)$, and if that fails, we recursively try out the special case for T^* again. This recursive check for limitedness of a term is formalized using the function *check_limitedness* in the formal definition of the proof checking algorithm presented in section 6.2 below.

If both of these adapted criteria for applying Functionality are fulfilled, we may add to the premise list that the dynamically introduced function is limited. Actually, just as with the domain information above, we may add a bit more, namely that all function-head-subterms of $T^\sigma(\mathbb{T}_1)$ are limited, whenever the terms $\mathbb{T}_1$ take a value satisfying Φ.

In the definition of the *PTL* proof checking algorithm below (Definition 6.2.8), we use the notation Γ_{func} for the list of all premises that encode domain information and information about the limitedness of dynamically introduced maps.

The treatment of presuppositional premises described in the previous section ensures that presuppositional premises corresponding to the first two conditions in the definition of $def([\![\varphi \to \theta]\!]^g_M)$ are added to the premise list. For the third condition of $def([\![\varphi \to \theta]\!]^g_M)$ (which, as described in section 5.1.1 of chapter 5, ensures that whenever $[\![\varphi \to \theta]\!]^g_M$ is defined and has the syntactic form of an implicit function introduction, the Map Comprehension Axiom Schema of *CMTN*

actually allows for this function introduction), we need to add additional presuppositional premises. These additional presuppositional premises are denoted Γ_{pres} in Definition 6.2.8 below.

6.1.5 References and theorem-proof blocks

One of the features distinguishing *PTL* from *HODPL* are the constructs of labels (*label*$(\bullet, \bullet)$), references (*ref*$(\bullet, \bullet)$) and the theorem-proof blocks (*thm*$(\bullet, \bullet, \bullet)$). As already mentioned there, the *PTL* semantics of the previous chapter did not capture the procedural role of these constructs, but the proof-checking does capture their procedural role.

To improve the performance of the proof checking algorithm, it is useful to restrict the number of premises given to the ATP. This is done by a *premise selection algorithm*, which takes into account explicit references (represented by the *label* and *ref* constructs in a *PTL* text) as well as other logical and text-structural criteria (for example information about the theorem types ("theorem", "lemma", "proposition" or "corollary") of the individual premises). An example of such a premise selection algorithm is described in Cramer, Koepke, Kühlwein, and Schröder (2010). For the purpose of this thesis, we will consider the ATP together with the premise selection algorithm as a black-box. We will adapt our above definition of *prover* (Definition 6.1.2) so that it will now formalize this combination of an ATP and a premise selection algorithm.

In the *DPL* proof checking algorithm, premises were just *PL* formulae. In section 6.1.3 we already mentioned that we have to mark whether a premise comes from a presupposition or not. For the sake of the premise selection algorithm, we now additionally have to mark the labelling of a premise with an ID, which makes it referenceable, as well as the theorem type of a premise. So our formal definition of a premise now is as follows:

Definition 6.1.10. A *premise* is a quadruple $\alpha : \Phi^p - \vartheta$, where Φ is a *PL* formula, α is either 0 or an ID, ϑ is either 0 or a theorem type and p is 0 or $\mathbb{P}$.

Remark. If any of α, θ or p is 0, we may omit it. In the case of α and θ, we then also omit the linking : or $-$ respectively. For example, we may write $0 : \Phi^0 - 0$ as Φ.

Here are the adaptations of the above definition of *proof obligation* and *prover* to the *PTL* proof checking algorithm:

Definition 6.1.11. A *proof obligation* is of the form $\Gamma \vdash^?_S \varphi$, where Γ is a finite sequence of premises, S is a finite sequence of IDs and φ is a *PL* formula. When we omit S in this notation, S is understood to be the empty sequence.

Definition 6.1.12. A *prover* P is a function from proof obligations to $\{-1, 0, 1\}$ such that if $P(\Gamma \vdash^?_S \varphi) = -1$ then $\Gamma \nvdash \varphi$, and if $P(\Gamma \vdash^?_S \varphi) = 1$ then $\Gamma \vdash \varphi$.

The proof in a theorem-proof block in a mathematical text serves the purpose of making the derivation of the theorem easier to find for a human reader. Similarly, the proof in a theorem-proof block in a *PTL* text can be viewed to serve the purpose of making the derivation of the theorem easier to find for the proof checking algorithm. This is achieved by letting the proof checking algorithm work on the proof first, and use the premise list that is active at the

end of the processing of the proof for checking the theorem. After the theorem is checked, the premises coming from the proof are no longer needed, and are hence removed from the premise list. The premises coming from the theorem are marked with the theorem type of the theorem.

6.1.6 *CMTN* axioms in the proof checking algorithm

We want our proof checking algorithm to account for the fact that *PTL* semantics is defined over *CMTN* models. We actually want the proof checking algorithm to be in a certain sense complete with respect to *CMTN*: Given a prover P with some minimal assumptions about its proving capacities and a theorem Φ of *CMTN*, there should be a *PTL* text that proves Φ and that can be successfully checked for correctness using the prover P (see Theorem 6.4.8 in section 6.4.1 below). For this we need to be able to use the axioms of *CMTN* in the proof checking process. Map Comprehension and Functionality are already implicitly included in the proof checking through the implicit dynamic function introduction as described in section 6.1.4. Class and Set Comprehension will also be treated implicitly: Whenever the algorithm has to prove the existence of a class or set, it first checks the conditions for an application of Class or Set Comprehension. (This is done in the *exist_check* function in Definition 6.2.8 below.) All other axioms (let us call them *non-comprehension axioms*) have to be added explicitly to the premise list.

But there is a problem with this: Given that some of the remaining axioms are actually infinite axiom schemata, we cannot add them all to the premise list. However, note that the remaining axiom schemata are actually all of the form

"For all $n \geq N$, Φ_n is an axiom."

for some natural number N and some recursively definable function $n \mapsto \Phi_n$ from $\mathbb{N}$ to *PL* formulae, or of the form

"For all $n, m \in \mathbb{N}$ such that $R(n, m)$, $\Phi_{n,m}$ is an axiom."

for some simple arithmetical property $R(n, m)$ and some recursively definable function $n, m \mapsto \Phi_{n,m}$ from $\mathbb{N}^2$ to *PL* formulae. The different values of n and m represent different arities of maps or lengths of tuples. For example, the Element Axiom schema is as follows:

- For $n \geq 1$ and $\bar{z}$ a variable list of length n:
  $\forall f\, \forall \bar{z}\, (L(f) \wedge f(\bar{z}) \neq u \rightarrow L(z_1) \wedge \ldots \wedge L(z_n) \wedge L(f(\bar{z})))$

In practice, we only need an instance of an axiom schema for a given arity or tuple length if we have to prove a formula containing a function application of that arity or a tuple of that length. Using this principle, we can always limit the set of axioms to be added to the premise list to a finite set.

As we will see when considering the application of the proof checking algorithm to Landau's *Grundlagen der Analysis* in chapter 8, it is useful to have – additionally to the non-comprehension axioms – certain axioms about currying and uncurrying functions. Some of these axioms are Skolemized versions of consequences from Map Comprehension, and others are consequences of these Skolemized axioms and map extensionality. As will be seen in section 8.3.2, in actual mathematical texts, the implicit introduction of functions may yield a

curried version of the multi-argument function that is actually intended. Since the implicit introduction of functions is the only way to apply Map Comprehension in the *PTL* proof checking algorithm, these currying and uncurrying axioms are needed to yield the intended form of the implicitly introduced function. The currying and uncurrying axioms also come in separate versions for separate arities:

(i) Currying Axiom Schema: For $m, n \geq 1$, the following is an axiom:

$\forall f\ (M(f, n+m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n, y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ f(x_1, \ldots, x_n, y_1, \ldots, y_m) = cur_{m,n}(f)(x_1, \ldots, x_n)(y_1, \ldots, y_m))$

(ii) Uncurrying Axiom Schema: For $m, n \geq 1$, the following is an axiom:

$\forall f\ (M(f, n) \wedge \forall x_1, \ldots, x_n\ M(f(x_1, \ldots, x_n), m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n)(y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ f(x_1, \ldots, x_n)(y_1, \ldots, y_m) = unc_{m,n}(f)(x_1, \ldots, x_n, y_1, \ldots, y_m))$

(iii) *cur-unc* Axiom Schema: For $m, n \geq 1$, the following is an axiom:

$\forall f\ (M(f, n+m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n, y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow cur_{m,n}(unc_{m,n}(f)) = f)$.

(iv) *unc-cur* Axiom Schema: For $m, n \geq 1$, the following is an axiom:

$\forall f\ (M(f, n) \wedge \forall x_1, \ldots, x_n\ M(f(x_1, \ldots, x_n), m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n)(y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow unc_{m,n}(cur_{m,n}(f)) = f)$.

Note that as in section 4.3, the function application notation here is actually shorthand for function application with app_n.

(i) and (ii) are Skolemized forms of the following consequences of *CMTN*'s Map Comprehension Axiom Schema:

- For $m, n \geq 1$, the following holds:

  $\forall f\ \exists g\ (M(f, n+m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n, y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ f(x_1, \ldots, x_n, y_1, \ldots, y_m) = g(x_1, \ldots, x_n)(y_1, \ldots, y_m))$

- For $m, n \geq 1$, the following holds:

  $\forall f\ \exists g\ (M(f, n) \wedge \forall x_1, \ldots, x_n\ M(f(x_1, \ldots, x_n), m) \wedge \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ (f(x_1, \ldots, x_n)(y_1, \ldots, y_m) \neq u \rightarrow L(x_1) \wedge \ldots \wedge L(x_n) \wedge L(y_1) \wedge \ldots \wedge L(y_m)) \rightarrow \forall x_1, \ldots, x_n, y_1, \ldots, y_m\ f(x_1, \ldots, x_n)(y_1, \ldots, y_m) = g(x_1, \ldots, x_n, y_1, \ldots, y_m))$

(iii) and (iv) follow from (i) and (ii) and *CMTN*'s map extensionality.

Just as with the infinite non-comprehension axiom schemas of *CMTN*, we will always limit the above axiom schemas to a finite list of axioms by using just those arities that occur in the consequence of the proof obligation in question.

For the sake of simplicity, when writing down the proof checking algorithm for *PTL* we will use the notation $P(\Gamma \vdash^?_S \varphi)$ for what should actually read $P(\Gamma \oplus \Delta \vdash^?_S \varphi)$ for an appropriate finite list Δ of non-comprehension *CMTN* axioms and currying-uncurrying axioms.

6.2 The proof checking algorithm for *PTL*

The differences between the proof checking algorithms for *DPL* and for *PTL* have already been described in sections 6.1.3, 6.1.4, 6.1.5 and 6.1.6.

Note that despite the higher-order nature of *PTL*, we still use first-order ATPs in the proof checking algorithm for *PTL*. The main reason for this is a very practical reason: The state-of-the-art automatic theorem provers for first-order logic are much stronger than any automatic theorem provers for higher-order logic.

Before we can present the formal definition of the proof checking algorithm for *PTL*, we will need some more definitions:

Definition 6.2.1. For a premise sequence Γ, $|\Gamma|$ is the sequence of all premises in Γ that are not marked as presuppositional premises, in the same order as they appear in Γ.

Definition 6.2.2. For a premise sequence Γ and an occurrence p of a premise in Γ, Γ_p is the subsequence of Γ preceding p, and Γ_{p+} is $\Gamma_p \oplus \langle p \rangle$ (i.e. the subsequence of Γ up to and including p).[12]

For the sake of readability, we do not always make it explicit whether we are talking about occurrences of premises from a premise sequence or just about the premises by themselves. The difference is important when it comes to equality claims: The same premise can occur more than once in a premise sequence. But when we have premise occurrences rather than premises in mind, an equality claim amounts to equality of premise occurrences, not just equality of premises. In most cases where we do talk about premise equality, we actually mean equality of premise occurrences. Only if there is a risk of misunderstanding, do we make explicit that we mean premise occurrences rather than premises.

Definition 6.2.3. Given $t \in T_{PTL}$, we recursively define a *PL* term $PL(t)$ by

$$PL(t) := \begin{cases} x & \text{if } t \text{ is the variable } x \\ app_n(PL(t_0), PL(t_1), \ldots, PL(t_n)) & \text{if } t \text{ is of the form } t_0(t_1, \ldots, t_n). \end{cases}$$

Remark. We treat *PL* as a normal function symbol, i.e. also use the notation PL^{-1} for its inverse and $PL^{-1}(\mathbb{T})$ for the list $\langle t \in T_{PTL} \mid PL(t) \in \mathbb{T} \rangle$.

Definition 6.2.4. A *PL* term is called a *PTL-PL* term iff it is of the form $PL(t)$ for some $t \in T_{PTL}$.

Definition 6.2.5. Given a *CMTN*-model M, M-assignments g and h and a list $\mathbb{T}$ of *PTL-PL* terms, we write $g[\mathbb{T}]h$ for $g[PL^{-1}(\mathbb{T})]h$.

Definition 6.2.6. A *substitution list* is a list of pairs of *PL* terms.

Definition 6.2.7. Given a substitution list $\mathbb{S} = \langle (T_1, T_1'), \ldots, (T_n, T_n') \rangle$ and a *PL* formula Φ, we define $\mathbb{S}(\Phi)$ to be $\Phi \frac{T_1'}{T_1} \ldots \frac{T_n'}{T_n}$.

Now we are ready for defining the function *check_text* that formalizes the proof checking algorithm:

[12] Note that as variables referring to premise lists we use besides Γ complex variables like Γ', Γ_0, Γ_1, Γ^+ and Γ^-. Applying the Γ_p notation to the complex variable Γ^+, we get Γ_p^+, which should not be confused with Γ_{p+}.

Definition 6.2.8.
$check(\theta) = (\nu)$:-[13]
$check\_text(\theta, \langle\rangle, \langle\rangle, \top) = (\_, \_, \nu)$.

$check\_text(t, \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle B(T)^{\mathbb{P}}, T = \top\rangle, \mathbb{T}, \nu)$:-[14]
t is a PTL term,
$read\_term(t, \Gamma, \mathbb{T}, \mu) = (\Gamma', T, \mu_0)$,
$\mu_1 = update(\mu_0, 0, P(\Gamma' \vdash^? B(T)))$,
$\nu = update(\mu_1, 1, P(\Gamma' \oplus \langle B(T)\rangle \vdash^? T = \top))$.
$check\_text(R(t_1, \ldots, t_n), \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle R(T_1, \ldots, T_n)\rangle, \mathbb{T}, \nu)$:-
$\Gamma_1 = \Gamma$,
$\mu_1 = \mu$,
for all $1 \leq i \leq n$, $read\_term(t_i, \Gamma_i, \mathbb{T}_i, \mu_i) = (\Gamma_{i+1}, T_i, \mu_{i+1})$,
$\Gamma' = \Gamma_{n+1}$,
$\nu = update(\mu_{n+1}, 1, P(\Gamma' \vdash^? R(T_1, \ldots, T_n)))$.
$check\_text(\neg\varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle\neg\exists_{\mathbb{T}_1} \Phi\rangle, \mathbb{T}, \nu)$:-
$read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}_1, \Phi, \mu')$,
$\nu = update(\mu', 1, P(\Gamma' \vdash^? \neg\exists_{\mathbb{T}_1} \Phi))$.
$check\_text((\varphi \wedge \psi), \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle\Phi, \Psi\rangle, \mathbb{T} \oplus \mathbb{T}_1 \oplus \mathbb{T}_2, \nu)$:-
$read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma'_0, \mathbb{T}_1, \Phi, \mu_0)$,
$read\_text(\psi, \langle\rangle, \Gamma'_0 \oplus \langle\Phi\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0) = (\Gamma'_1, \mathbb{T}_2, \Psi, \mu_1)$,
$\Gamma' = \Gamma'_1 \setminus \langle\Phi\rangle$,
$\nu = update(\mu_1, 1, P(\Gamma' \vdash^? \exists_{\mathbb{T}_1} (\Phi \wedge \exists_{\mathbb{T}_2} \Psi)))$.
$check\_text((\varphi \vee \psi), \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle(\exists_{\mathbb{T}_1} \Phi \vee \exists_{\mathbb{T}_2} \Psi)\rangle, \mathbb{T}, \nu)$:-
$read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma'_0, \mathbb{T}_1, \Phi, \mu_0)$,
$read\_text(\psi, \langle\rangle, \Gamma'_0, \mathbb{T}, \mu_0) = (\Gamma', \mathbb{T}_2, \Psi, \mu_1)$,
$\nu = update(\mu_1, 1, P(\Gamma' \vdash^? (\exists_{\mathbb{T}_1} \Phi \vee \exists_{\mathbb{T}_2} \Psi)))$.
$check\_text((\varphi \to \theta), \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \Gamma_{pres} \oplus \langle\forall_{\mathbb{T}_1} (\Phi \to \Theta)\rangle \oplus \Gamma_{func}, \mathbb{T} \oplus \mathbb{F}, \nu)$:-
$read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma_0, \mathbb{T}_1, \Phi, \mu_0)$,
$check\_text(\theta, \Gamma_0 \oplus \langle\Phi\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0) = (\Gamma_1, \mathbb{T}_2, \mu_1)$,
if the symbol L does not occur in $\Gamma_1 - \Gamma_0$ and for every term T
occurring in $\Gamma_1 - \Gamma_0$ that is either in $\mathbb{T}$ or a skolem function
symbol, $check\_limitedness(\Gamma_1, \Gamma_1 - \Gamma_0, \mathbb{T}, T)$:[15]
$\alpha = 1$,
else:
$\alpha = 0$,
$make\_functions(\mathbb{T}_1, \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1), \Gamma_0 \oplus \langle\Phi\rangle, \Gamma_1, \Phi, \alpha, \mu_1) = (\mathbb{F}, \mathbb{T}', \Gamma_{func}, \Gamma^-_{pres}, \nu)$,
$pull\_out\_pres(\langle\rangle, \mathbb{T}_2 - \mathbb{T}, \Gamma_0, \Gamma_1) = (\Gamma', \langle\_\rangle \oplus \Gamma_2, \_)$,
$\Theta = \exists_{\mathbb{T}'} \bigwedge \Gamma_2$,

[13] The *check* function has one input argument and one output argument: its input is a PTL text, and its output is a proof status value that indicates whether the proof checking has shown the input text to be valid or at least defined.

[14] The *check_text* function has four input and three output arguments, similarly to the *check_text* function in the *DPL* proof checking algorithm: The first input argument specifies the PTL text to be checked. The remaining three input arguments as well and the three output arguments keep track of the currently active premise list, the currently active list of accessible terms and the current proof status value.

[15] Here the proof checking algorithm checks whether it may apply the *CMTN* Functionality Axiom Schema to the maps which it will introduce in *make_functions* below based on the *CMTN* Map Comprehension Axiom Schema. $\alpha = 1$ indicates that Functionality may be applied, whereas $\alpha = 0$ indicates that Functionality may not be applied. Compare the discussion in section 6.1.4 above.

$\Gamma_{pres} = \Gamma^{-}_{pres} \oplus \langle \forall_{\mathbb{T}_2 - \mathbb{T}} \, (\Phi \wedge \Theta \rightarrow L(T))^{\mathbb{P}} \mid T \in \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1) - \mathbb{T}' \rangle$.

check_text$((\theta \,\&\, \xi), \Gamma, \mathbb{T}, \mu) = (\Gamma_2, \mathbb{T}_2, \nu)$:-
  check_text$(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}_1, \mu_1)$,
  check_text$(\xi, \Gamma_1, \mathbb{T}_1, \mu_1) = (\Gamma_2, \mathbb{T}_2, \nu)$.

check_text$(\exists t \; \varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle PL(t) \neq u^{\mathbb{P}}, \Phi \rangle, \mathbb{T} \oplus \mathbb{T}_0 \oplus \langle PL(t) \rangle, \nu)$:-
  read_text$(\varphi, \langle PL(t) \rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}_0, \Phi, \mu')$,
  exist_check$(1, \Gamma', \mathbb{T}, \exists_{\langle PL(t) \rangle} \; \exists_{\mathbb{T}_0} \; \Phi, \mu') = (\nu)$.

check_text$(\Diamond \varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma' \oplus \langle \exists_{\mathbb{T}_1} \; \Phi \rangle, \mathbb{T}, \nu)$:-
  read_text$(\varphi, \langle \rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma', \Phi, \mathbb{T}_1, \_)$,
  $\nu =$ *update*$(\mu, 1, P(\Gamma' \vdash^{?} \exists_{\mathbb{T}_1} \; \Phi))$.

check_text$(label(\alpha, \theta), \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$:-
  check_text$(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}', \nu)$,
  $\Gamma' = \Gamma \oplus \langle \alpha : \Phi^p - \vartheta \mid \_ : \Phi^p - \vartheta \in \Gamma_1 - \Gamma \rangle$.

check_text$(ref(S, \varphi), \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$:-
  call *check_text*$(\varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$, but for any proof obligation $\Gamma_0 \vdash^{?} \Phi$ called within this call of *check_text*, use $\Gamma_0 \vdash^{?}_{S} \Phi$ instead.

check_text$(thm(\vartheta, \varphi, \theta), \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$:-
  check_text$(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}_1, \mu_1)$,
  check_text$(\varphi, \Gamma_1, \mathbb{T}_1, \mu_1) = (\Gamma_2, \mathbb{T}_2, \nu)$,
  $\Gamma' = \Gamma \oplus \langle \alpha : \Phi^p - \vartheta \mid \alpha : \Phi^p - 0 \in \Gamma_2 - \Gamma_1 \rangle$,
  $\mathbb{T}' = \mathbb{T} \oplus (\mathbb{T}_2 - \mathbb{T}_1)$.

check_text$(\mathrm{def}(t), \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}, \nu)$:-
  read_term$(t, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \_, \mu')$,
  $\Gamma' = \Gamma \oplus \langle \alpha : \Phi^0 - \vartheta \mid \alpha : \Phi^{\mathbb{P}} - \vartheta \in \Gamma_1 - \Gamma \rangle$,
  if $\mu = u$:
    $\nu = u$,
  else:
    if $\mu' = u$:
      $\nu = \bot$,
    else:
      $\nu = \mu$.

read_text$(\theta, \mathbb{T}_0, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}_1, \Theta, \nu)$:-[16]
  check_text$(\theta, \Gamma, \mathbb{T} \oplus \mathbb{T}_0, \mu) = (\Gamma^{+}, \mathbb{T}', \mu_0)$,
  $\mathbb{T}_1 = \mathbb{T}' - (\mathbb{T} \oplus \mathbb{T}_0)$,
  pull_out_pres$(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^{+}) = (\Gamma', \Gamma_0, \_)$,
  $\Theta = \bigwedge \Gamma_0$,
  if $\mu_0 = u$:

[16]The *read_text* function *reads in* a *PTL* text, translates it to *PL*, and at the same time already checks the presuppositions of this *PTL* text. The function has five input arguments and four output arguments: The first input argument is the *PTL* text to be read in. The third and fifth input arguments and the first and fourth output arguments respectively keep track of the active premise list and the proof status value. Unlike in the *read_text* function of the *DPL* proof checking algorithm, we need to keep track of these values, since we need to check the presuppositions of the input *PTL* text and add the information of the presuppositions to the currently active premise list. The second input argument lists terms that appear in a quantifier whose scope contains the input *PTL* text. The fourth input argument is the list of the remaining terms that are accessible when the function is called. The second output argument lists the terms with binding capability after the input *PTL* text (see Definition 5.2.7 in chapter 5). The third output argument is the *PL* translation of the input *PTL* text (without existential quantification over the terms with binding capability after the input *PTL* text).

$\nu = u$,
else:
$\nu = \mu$.

pull_out_pres$(\_, \_, \Gamma, \Gamma) = (\Gamma, \langle\rangle, \langle\rangle)$.[17]
pull_out_pres$(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+ \oplus \langle \Phi^{\mathbb{P}} \rangle) = (\Gamma' \oplus \langle (\forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi} (\bigwedge \Gamma_0 \rightarrow \mathbb{S}'(\Phi)))^{\mathbb{P}} \rangle, \Gamma_0, \mathbb{S}')$:-
$pull\_out\_pres(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+) = (\Gamma', \Gamma_0, \mathbb{S})$,
$\mathbb{T}_\Phi = \langle t \in \mathbb{T}_1 \mid t \text{ occurs in } \Gamma^+ \oplus \langle \Phi \rangle \rangle$,
$\mathbb{S}' = \mathbb{S} \oplus \langle (sk_i(\mathbb{T}), sk_i(\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_\Phi)) \mid sk_i(\mathbb{T}) \text{ is introduced in } \Phi \rangle$.
pull_out_pres$(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+ \oplus \langle \Phi^0 \rangle) = (\Gamma', \Gamma_0 \oplus \langle \mathbb{S}(\Phi) \rangle, \mathbb{S})$:-
$pull\_out\_pres(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+) = (\Gamma', \Gamma_0, \mathbb{S})$.

read_term$(t, \Gamma, \mathbb{T}, \mu) = (\Gamma, PL(t), \mu)$:-[18]
$t \in T_{PTL}$,
$PL(t) \in \mathbb{T}$.
read_term$(f(t_1, \ldots, t_n), \Gamma, \mathbb{T}, \mu) = (\Gamma', f(T_1, \ldots, T_n), \nu)$:-
$f(t_1, \ldots, t_n) \notin T_{PTL}$ or $PL(f(t_1, \ldots, t_n)) \notin \mathbb{T}$,
f is a logical function symbol,
$\Gamma_1 = \Gamma$,
$\mu_1 = \mu$,
for all $1 \leq i \leq n$, *read_term*$(t_i, \Gamma_i, \mathbb{T}, \mu_i) = (\Gamma_{i+1}, \mu_{i+1})$,
$\Gamma' = \Gamma_{n+1} \oplus \langle (f(T_1, \ldots, T_n) \neq u)^{\mathbb{P}} \rangle$,
$\nu = update(\mu_{n+1}, 0, P(\Gamma_{n+1} \vdash^? f(T_1, \ldots, T_n) \neq u))$.
read_term$(t_0(t_1, \ldots, t_n), \Gamma, \mathbb{T}, \mu) = (\Gamma', app_n(T_0, T_1, \ldots, T_n), \nu)$:-
$t_0(t_1, \ldots, t_n) \notin T_{PTL}$ or $PL(t_0(t_1, \ldots, t_n)) \notin \mathbb{T}$,
$\Gamma_0 = \Gamma$,
$\mu_0 = \mu$,
for all $0 \leq i \leq n$, *read_term*$(t_i, \Gamma_i, \mathbb{T}_i, \mu_i) = (\Gamma_{i+1}, \mu_{i+1})$,
$\Gamma' = \Gamma_{n+1} \oplus \langle (app_n(T_0, T_1, \ldots, T_n) \neq u)^{\mathbb{P}} \rangle$,
$\nu = update(\mu_{n+1}, 0, P(\Gamma_{n+1} \vdash^? app_n(T_0, T_1, \ldots, T_n) \neq u))$.
read_term$(\iota x\ \varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, sk^{new}, \nu)$:-
read_text$(\varphi, \langle x \rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma_0, \mathbb{T}_0, \Phi, \mu_1)$,

[17] The *pull_out_pres* function *pulls out presuppositions* in the way explained in section 6.1.3. It has four input and three output arguments. The third and fourth input argument are two premise lists, of which the second one is an extension of the first. The function pulls out the presuppositional premises from the difference between these two premise lists. The second input argument lists terms that are quantificationally introduced in the *PTL* text represented by the difference between the two input premise lists. The first input argument lists terms that appear in a quantifier whose scope contains this *PTL* text. The first output argument lists the first input premise list together with premises that represent the projected version of the presuppositional premises pulled out from the difference between the two input premise lists. The second output argument lists the non-presuppositional premises of the difference between the two input premise lists, only with some modifications in the arguments of skolem functions. The third output argument is a substitution list that specifies the way arguments have to be added to skolem functions in the premises from the difference between the two input premise lists.

[18] The *read_term* function *reads in* a *PTL* term , translates it to a *PL* term, and at the same time already checks the presuppositions of this *PTL* term. The function has four input and three output arguments. The first input argument is the *PTL* term to be read in. The second and fourth input argument and the first and third output arguments keep track of the active premise list and the active proof status value respectively. The third input argument lists the terms that are accessible when the function is called. The second output argument is the *PL* translation of the input *PTL* term.

$\mathit{exist\_check}(0, \Gamma_0, \exists x\ \exists_{\mathbb{T}_0}\ \Phi, \mu_1) = (\mu_2)$,
$\nu = \mathit{update}(\mu_2, 0, P(\Gamma_0 \oplus \langle \exists_{\mathbb{T}_0}\ \Phi\frac{sk^{new}}{x}\rangle \vdash^?\ \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \to x = sk^{new})))$,
$\Gamma_1 = \Gamma_0 \oplus \langle(\exists_{\mathbb{T}_0}\ \Phi\frac{sk^{new}}{x})^{\mathbb{P}}, (\forall x\ (\exists_{\mathbb{T}_0}\ \Phi \to x = sk^{new}))^{\mathbb{P}}\rangle$.

$\mathit{check\_limitedness}(\Gamma, \_, \_, T)$:-[19]
$P(\Gamma \vdash^?\ L(T)) = 1$.
$\mathit{check\_limitedness}(\Gamma, \Gamma', \mathbb{T}, T)$:-
T appears in Γ' only in terms of the form $T(T')$,
for some variable x, Γ contains $\forall_x\ (T(x) \leftrightarrow \Phi(x))$,[20]
$\Phi(x)$ does not contain L,
for every term T^* occurring in $\Phi(x)$ that is either in $\mathbb{T}$ or a skolem function symbol, $\mathit{check\_limitedness}(\Gamma, \Gamma', \mathbb{T}, T^*)$.

$\mathit{make\_functions}(\mathbb{T}, \langle T'_1, \ldots, T'_n\rangle, \Gamma, \Gamma^+, \Phi, \alpha, \mu) = (\mathbb{F}, \mathbb{T}', \Gamma_{\mathit{func}}, \Gamma_{\mathit{pres}}, \nu)$:-[21]
$\mu_0 = \mu$,
for $1 \le i \le n$, $\mathit{make\_function}(\mathbb{T}, T'_i, \Gamma^+, \Phi, \alpha, \mu_i) = (\mathbb{F}_i, \mathbb{T}_i, \Gamma^i_{\mathit{func}}, \mu_{i+1})$,
$\Gamma_{\mathit{func}} = \bigoplus_{i=1}^n \Gamma^i_{\mathit{func}}$,
$\mathbb{F} = \bigoplus_{i=1}^n \mathbb{F}_i$,
$\mathbb{T}' = \bigoplus_{i=1}^n \mathbb{T}_i$,
$\nu = \mu_n$,
if $\Gamma_{\mathit{func}} = \langle\rangle$:
$\nu = \mu_n$,
$\Gamma_{\mathit{pres}} = \langle\rangle$,
else:
$\mathbb{T} = \langle T_1, \ldots, T_m\rangle$,

[19] The *check_limitedness* predicate checks whether a given *PL* term may be treated as a limited term for the purpose of the *CMTN* Functionality Axiom Schema; compare the discussion in section 6.1.4 above. The fourth argument is the *PL* term in question. The first argument is the list of premises that may be used for directly proving that this term is limited. The second argument is a further list of premises that needs to fulfil certain syntactic criteria in order for the special case discussed in section 6.1.4 above to be applicable. The third argument is a list of terms that needs to fulfil certain conditions for this special case to be applicable.

[20] Here, just as in the definition of *exist_check* below, $\Phi \leftrightarrow \Psi$ should be considered an abbreviation for $(\Phi \to \Psi) \wedge (\Psi \to \Phi)$.

[21] The *make_functions* function looks for dynamically implicitly introduced maps in the data received from checking an implication $\varphi \to \theta$. It has seven input and five output arguments.

The first input argument lists the terms that are dynamically existentially introduced in φ, and the second input argument lists the terms that are dynamically existentially introduced in θ. The third input argument lists the premise list that is active before checking θ, and the fourth input argument is the premise list that is active after checking θ. The fifth input argument is the *PL* translation of φ, and the sixth input argument specifies with a value 0 or 1 whether the *CMTN* Functionality Axiom Schema may be applied to the implicitly introduced maps.

For the dynamic implicit function introduction to be applicable, certain terms have to be shown to be limited. This involves sending proof obligation to the prover, which may change the proof status value. The seventh input argument and the fifth output argument keep track of the currently active proof status value.

The first output argument lists the terms that represent the implicitly introduced maps found by *make_functions*. The second output argument lists those terms that were dynamically existentially introduced in θ but that did not give rise to an implicitly introduced map. The third output argument lists premises that encode domain information and information about the limitedness of implicitly introduced maps. The forth output argument lists presuppositional premises that encode special presuppositions that need to be fulfilled in implications because of dynamic implicit function introduction (see the last paragraph of section 6.1.4 above).

for $1 \leq i \leq m$, $\mu_{n+i} = \mathit{update}(\mu_{n+i-1}, 0, P(\Gamma \vdash^? L(T_i)))$,
$\nu = \mu_{n+m}$,
$\Gamma_{pres} = \langle \forall_{\mathbb{T}}(\Phi \rightarrow \bigwedge_{i=1}^{m} L(T_i))^{\mathbb{P}} \rangle$.

make_function$(\mathbb{T}, T, \Gamma^+, \Phi, \alpha, \mu) = (\langle T' \rangle, \langle\rangle, \langle \Phi_1, \ldots, \Phi_k, \Psi_1, \ldots, \Psi_k \rangle \oplus \Gamma_L, \nu)$:-[22]
there is an n-place argument filler σ such that $T = T'^{\sigma}(\mathbb{T})$,
$F_1, \ldots, F_k$ are the function-head subterms of T that contain T' as a proper subterm,
$n_1, \ldots, n_k$ are the arities of $F_1, \ldots, F_k$ in T respectively,
for $1 \leq i \leq k$, $\mathbb{T}_i = \langle T_0 \in \mathbb{T} \mid T_0 \text{ occurs in } F_i \rangle$,
for $1 \leq i \leq k$, $\Phi_i = \forall_{\mathbb{T}_i} M(F_i, n_i)$,
for $1 \leq i \leq k$, $\Psi_i = \forall_{\mathbb{T}_i} (\exists_{\mathbb{T}-\mathbb{T}_i} \Phi \leftrightarrow F_i \neq u)$,
$\nu = \mathit{update}(\mu, 0, P(\Gamma^+ \vdash^? L(T)))$,
if $\alpha = 1$:
$\Gamma_L = \langle L(T') \rangle \oplus \langle \forall_{\mathbb{T}}(\Phi \rightarrow L(F_i)) \mid 1 \leq i \leq k \rangle$,
else:
$\Gamma_L = \langle\rangle$.
make_function$(\mathbb{T}, T, \_, \Phi, \_, \mu) = (\langle\rangle, \langle T \rangle, \langle\rangle, \langle\rangle, \mu)$:-
there is no n-place argument filler σ such that T is of the form $T'^{\sigma}(\mathbb{T})$.

exist_check$(\_, \Gamma, \_, \exists_{\langle T \rangle} \Phi, \mu) = (\mu)$:-[23]
Φ is of the form $(C(T) \wedge \forall_x (x \in T \leftrightarrow \Psi(x)))$,
$P(\Gamma \oplus \langle \Psi(x) \rangle \vdash^? L(x)) = 1$.
exist_check$(\_, \Gamma, \mathbb{T}, \exists_{\langle T \rangle} \Phi, \mu) = (\mu)$:-
Φ is of the form $(C(T) \wedge L(T) \wedge \forall_x (x \in T \leftrightarrow \Psi(x)))$,
$P(\Gamma \oplus \langle \Psi(x) \rangle \vdash^? L(x)) = 1$,

[22] The *make_function* function analyses a single term dynamically existentially introduced in the θ of an implication $\varphi \rightarrow \theta$ for determining whether it gives rise to an implicitly introduced map. It has six input and four output arguments.

The first input argument lists the terms that are dynamically existentially introduced in φ, and the second input argument lists the term from the list of terms dynamically existentially introduced in θ that is to be analysed now. The third input argument is the premise list that is active after checking θ. The fourth input argument is the *PL* translation of φ, and the fifth input argument specifies with a value 0 or 1 whether the *CMTN* Functionality Axiom Schema may be applied if there is an implicitly introduced map. The sixth input argument and the fourth output argument keep track of the currently active proof status value.

The first output argument lists the terms that represent the implicitly introduced maps found by *make_function*; it is either a list with a single term or the empty list. The second output argument is also either a list with a single term or the empty list: It contains the second input argument in case it did non give rise to an implicitly introduced map. The third output argument lists premises that encode domain information and information about the limitedness of the implicitly introduced map (if there is one; else it is the empty list).

[23] The *exist_check* function checks whether a certain existential statement can be established under certain conditions. Apart from letting the prover try to prove the existential statement from the currently active premise list, it also determines whether the *CMTN* Class or Set Comprehension Axiom Schemas may be applied in order to establish the existence of a class or set. The function has five input and one output argument. The first input argument specifies using the numbers 0 and 1 respectively whether the existential check is to be treated as a presupposition check or as an assertion check. The second input argument lists the currently active premises. The third input argument lists the terms that were accessible just before the existential *PTL* formula whose correctness is checked by *exist_check*; these terms may function as parameters to the formula to which we may need to apply the Set Comprehension Axiom Schema, in which case their limitedness has to be established. The fourth input argument is the existential formula to be checked. The fifth input argument and the sole output argument keep track of the currently active proof status value.

the symbol L does not occur in $\Psi(x)$,
for every term T_0 occurring in $\Psi(x)$ that is either in $\mathbb{T}$ or a skolem function symbol, *check_limitedness*$(\Gamma, \langle\Phi(x)\rangle, \mathbb{T}, T_0)$.

exist_check$(\alpha, \Gamma, \_, \exists_{\langle T\rangle}\ \Phi, \mu) = (\nu)$:-
none of the previous two clauses of *exist_check* is satisfied,
$\nu = update(\mu, \alpha, P(\Gamma \vdash^{?} \exists_{\langle T\rangle}\ \Phi))$.

6.3 Soundness of the proof checking algorithm

The soundness theorem that we want to prove takes a similar form as in the case of the *DPL* proof checking algorithm:

Theorem 6.3.1 (Soundness of the *PTL* proof checking algorithm)**.** *If θ is a nice PTL text and check$(\theta) = \top$, then $v(\theta) = \top$.*

Again we will need a Detailed Soundness Lemma that tells us something about the implication of *check_text*$(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$. For the proof of this Detailed Soundness Lemma, we will need to semantically interpret the premises that the algorithm keeps track of. In the *DPL* Detailed Soundness Lemma, we used a pair of a structure M and an M-assignment g for semantically interpreting the premises. The natural analogue for the *PTL* proof checking algorithm would be a pair of a *CMTN* model M and an M-assignment g. But this would not give any interpretation to the skolem function symbols. Hence we additionally use a *skolem-assignment*, which interprets the skolem function symbols.

Definition 6.3.2. Let Γ be a premise list and M be a *CMTN* model. A Γ-*skolem-assignment over* M is a function S whose domain is the set of all skolem function symbols appearing in Γ and such that for every $sk_i^n \in dom(S)$, $S(sk_i^n)$ is a function from M^n to $M \setminus \{u^M\}$.

Remark. When it is clear which M is intended, we usually omit the qualification "over M".

Definition 6.3.3. Let M be a *CMTN* model and let S be a Γ-skolem-assignment over M. Then $M + S$ is defined to be the structure over the language $L_{CMTN} \cup \{sk_i^n \mid sk_i^n \text{ occurs in } \Gamma\}$ that coincides with M on its interpretation of the symbols in L_{CMTN} and coincides with S on its interpretation of the skolem function symbols occurring in Γ.

Before we can state the Detailed Soundness Lemma for the *PTL* proof checking algorithm, we still need some more definitions:

Definition 6.3.4. For a *CMTN* model M, a skolem-assignment S over M, an M-assignment g and a *PL* term T over the language of $M + S$, we recursively define $\frac{M+S}{g}(T)$ as follows:

$$\frac{M+S}{g}(T) := \begin{cases} g(T) & \text{if } g(PL^{-1}(T)) \text{ is defined} \\ M + S(T) & \text{if } T \text{ is a constant symbol} \\ M + S(f)(\frac{M+S}{g}(T_1), \ldots, \frac{M+S}{g}(T_n)) & \text{if } g(PL^{-1}(T)) \text{ is undefined and } T \text{ is of the form } f(T_1, \ldots, T_n). \end{cases}$$

Remark. $\frac{M+S}{g}(T)$ is undefined if T contains an occurrence of a variable x such that $g(x)$ is undefined and $g(PL^{-1}(T'))$ is undefined for every subterm T' of T containing this occurrence of x.

Definition 6.3.5. For a *CMTN* model M, a skolem-assignment S over M, an M-assignment g and a *PL* formula Φ over the language of $M+S$, we define $M+S, g \models \Phi$ in a way analogous to the usual definition of $\mathcal{A}, g \models \Phi$ for a structure $\mathcal{A}$ and an $\mathcal{A}$-assignment g, but using $\frac{M+S}{g}(T)$ instead of $\frac{\mathcal{A}}{g}(T)$ for interpreting terms in Φ.

Remark. $M+S, g \models \Phi$ is undefined under similar circumstances as $\frac{M+S}{g}(T)$.

Definition 6.3.6. For a *CMTN* model M, a skolem-assignment S over M, an M-assignment g and a premise list Γ, we write $M+S, g \models \Gamma$ iff $M+S, g \models \Phi$ for every premise $\alpha : \Phi^p - \theta$ in Γ.

Definition 6.3.7. Given two skolem-assignments S and S' over M, we say that S' *extends* S and write $S' \succeq S$ iff $dom(S) \subseteq dom(S')$ and $S = S'|_{dom(S)}$.

Definition 6.3.8. A PTL_{sk} *symbol* is a symbol that is either a logical constant, function symbol or relation symbol of *PTL* or a skolem function symbol.

Definition 6.3.9. Given a *PL* formula Φ, a *free term in* Φ is a term occurring in Φ that is not a bound variable.

Remark. In this chapter, we use the expression *free term* both for free terms of *PL* formulae in the sense of this definition and for free terms of *PTL* texts in the sense of Definition 5.2.4. The same holds for the expressions *hereditarily free term* and *MHF term* (*maximal hereditarily free term*) defined in the next two definitions with respect to *PL* formulae, and already defined in Definitions 5.2.8 and 5.2.9 with respect to *PTL* texts.

Definition 6.3.10. Given a *PL* formula Φ, a *hereditarily free term in* Φ is a term T occurring in Φ such that all subterms of T are free terms in Φ.

Definition 6.3.11. Given a *PL* formula Φ, a *maximal hereditarily free term in* Φ, usually abbreviated to *MHF term in* Φ, is a hereditarily free term in Φ that is not a proper subterm of a hereditarily free term in Φ.

Definition 6.3.12. Given proof status values μ, ν (i.e. $\mu, \nu \in \{\top, \bot, u\}$), we define $\mu \geq \nu$ to mean that either $\mu = \nu$ or $\mu = \top$ or $\nu = u$. (In other words, the ordering on $\{\top, \bot, u\}$ is $\top > \bot > u$.)

Definition 6.3.13. Given proof status values μ, ν, we define $\mu + \nu$ to be the minimum of μ and ν according to the above ordering.

Definition 6.3.14. Given a *PTL* text θ, $\mathbf{qt}(\theta)$ is the multiset of all occurrences of terms in θ after an $\exists$ or after an ι.

For concisely expressing the criteria for $def([\![\theta]\!]^g_M)$ and for $k \in [\![\theta]\!]^g_M$ in the Detailed Soundness Lemma, we need the following two definitions:

Definition 6.3.15. Given premise lists Γ and Γ' such that Γ' extends Γ, a term list $\mathbb{T}$, a *CMTN* model M, a Γ-skolem-assignment S, an M-assignment g and a *PL* formula $\Phi \in \Gamma' - \Gamma$, we write $pres(\Gamma', \Gamma, \mathbb{T}, M, S, g, \Phi)$ iff for all Γ'_Φ-skolem-assignments S' extending S and all $k[\mathbb{T}]g$ such that $M+S', k \models (\Gamma' - \Gamma)_\Phi$, there is a $\Gamma'_{\Phi+}$-skolem-assignment S'' extending S' such that $M+S'', k \models \Phi$.

Definition 6.3.16. Given premise lists Γ' and Γ, a *CMTN* model M, a Γ-skolem-assignment S and an M-assignment g, we say that g *verifies* $\Gamma' - \Gamma$ *over* $M + S$ iff for every $\Phi \in |\Gamma' - \Gamma|$ and every Γ'_Φ-skolem-assignment S' extending S such that $M + S', g \models (\Gamma' - \Gamma)_\Phi$, we have $M + S', g \models \Phi$.

We are now ready to state the Detailed Soundness Lemma for the *PTL* proof checking algorithm. Its assumptions and assertions are almost perfectly analogous to those of the *DPL* Detailed Soundness Lemma. We only had to add one assumption, namely (vii), and two assertions, namely 3 and 6. All these additions are related to the treatment of presuppositions and undefinedness: Assertion 3 gives a criterion for $def([\![\theta]\!]^g_M)$, just as assertion 3 of the *DPL* Detailed Soundness Lemma and the analogous assertion 4 of this Detailed Soundness Lemma give a criterion for $k \in [\![\theta]\!]^g_M$. Assumption (vii) and assertion 6 accommodate for the fact that the terms in the term list that the *PTL* proof checking algorithm keeps track of are meant to be defined terms.

Lemma 6.3.17 (Detailed Soundness Lemma)**.** *Let θ be a semi-nice PTL text. Further assume the following properties:*

(i) $\mathbb{T}$ *is a list of PTL-PL terms such that* $PL^{-1}(\mathbb{T}) \oplus \mathbf{qt}(\theta)$ *is pairwise independent.*

(ii) All MHF terms of θ *are composed of terms in* $PL^{-1}(\mathbb{T})$.

(iii) Γ *is a premise list such that all MHF terms in* Γ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$.

(iv) $check\_text(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$.

(v) M *is a CMTN model,* S *a* Γ*-skolem-assignment and* g *an* M*-assignment such that* $M + S, g \models \Gamma$.

(vi) $dom(g) = PL^{-1}(\mathbb{T})$.

(vii) For all $T \in \mathbb{T}$, $\frac{M+S}{g}(T) \neq u$.

Then the following six properties hold:

1. $\mathbf{tbc}(\theta) = PL^{-1}(\mathbb{T}' - \mathbb{T})$.[24]
2. *All MHF terms in* $\Gamma' - \Gamma$ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}'$.
3. $def([\![\theta]\!]^g_M)$ *iff for all presuppositionally marked premises* Φ *in* $\Gamma' - \Gamma$, $pres(\Gamma', \Gamma, \mathbb{T}, M, S, g, \Phi)$.
4. *If* $def([\![\theta]\!]^g_M)$*, then for all* M*-assignments* k*, the following three properties are equivalent:*
   (a) $k \in [\![\theta]\!]^g_M$.
   (b) $k[\mathbb{T}' - \mathbb{T}]g$ *and* k *verifies* $\Gamma' - \Gamma$ *over* $M + S$.

[24] Analogously to what we said in footnote 7 on page 97, we should actually say that the set whose elements are the elements of the sequence $PL^{-1}(\mathbb{T}' - \mathbb{T})$ is equal to $\mathbf{tbc}(\theta)$. For the sake of simplicity and since it does not cause problems, we use the simplified expression $tbc(\theta) = PL^{-1}(\mathbb{T}' - \mathbb{T})$ instead.

(c) $k[\mathbb{T}' - \mathbb{T}]g$ *and there is a* Γ'*-skolem-assignment* $S' \succeq S$ *such that* $M + S', k \models \Gamma'$.

5. $\mu + v(\theta, M, g) \geq \nu$.

6. *If def*$([\![\theta]\!]^g_M)$ *and* $k \in [\![\theta]\!]^g_M$*, then for every* Γ'*-skolem-assignment* S' *extending* S *such that* $M + S', k \models \Gamma'$ *and for every* $T \in \mathbb{T}'$, $\frac{M+S'}{k}(T) \neq u^M$.

We will postpone the proof of this lemma to section 6.3.1 in order to first present some lemmas needed in the proof. In particular, we need soundness lemmas for some of the other predicates of the *PTL* proof checking algorithm, namely for *pull_out_pres*, *read_text*, *read_term* and *exist_check*.

The following *pull_out_pres* Soundness Lemma has two syntactic assertions that ensure that the way the active premise list is composed of terms from the active term list is not destroyed by the transformations preformed by *pull_out_pres*, and a semantic assertion that ensures that the conditions for the criterion for *def*$([\![\theta]\!]^g_M)$ are conserved by this transformation. The proof for the semantic assertion is rather involved, but it contains a useful method for transforming skolem-assignments that will be needed in later proofs too, but that is only spelled out in detail at this point.

Lemma 6.3.18 (*pull_out_pres* Soundness Lemma)**.** *Assume the following properties:*

(i) $\mathbb{T}$, $\mathbb{T}_0$ *and* $\mathbb{T}_1$ *are lists of PTL-PL terms.*

(ii) Γ *is a premise list that does not contain free terms from the term lists* $\mathbb{T}_0$ *and* $\mathbb{T}_1$.

(iii) *Every MHF term in* $\Gamma^+ - \Gamma$ *is composed of* PTL_{sk} *symbols and terms in* $\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_1$.

(iv) M *is a CMTN model,* S *a* Γ*-skolem-assignment and* g *an* M*-assignment such that* $M + S, g \models \Gamma$.

(v) *pull_out_pres*$(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+) = (\Gamma', \Gamma_0, \mathbb{S})$.

Then the following three statements hold:

1. *Every MHF term in* Γ_0 *is composed of* PTL_{sk} *symbols and terms in* $\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_1$.

2. *Every MHF term in* $\Gamma' - \Gamma$ *is composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$.

3. *The following two properties are equivalent:*

   (a) *For all presuppositionally marked premises* Φ *in* $\Gamma^+ - \Gamma$*, we have pres*$(\Gamma^+, \Gamma, \mathbb{T}_0 \oplus \mathbb{T}_1, M, S, g, \Phi)$.

   (b) *For every premise* Φ *in* $\Gamma' - \Gamma$ *and every* Γ'_Φ*-skolem-assignment* $S' \succeq S$ *such that* $M + S', g \models \Gamma'_\Phi$*, there is a* Γ'*-skolem-assignment* $S_0 \succeq S'$ *such that* $M + S_0, g \models \Gamma'$.

Proof.

1. It is easily seen from the definition of *pull_out_pres* that Γ_0 is of the form $\mathbb{S}(|\Gamma^+ - \Gamma|)$ for some substitution list $\mathbb{S}$ consisting of substitutions of the form $(sk_i(\mathbb{T}), sk_i(\mathbb{T} \oplus \mathbb{T}^*))$, where $\mathbb{T}^* \subseteq \mathbb{T}_0 \oplus \mathbb{T}_1$. This directly implies the desired result.

2. Every formula in $\Gamma' - \Gamma$ is of the form $\forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi}(\bigwedge \Gamma_2 \to \mathbb{S}(\Phi))$, where Φ is a formula in $\Gamma^+ - \Gamma$, $\mathbb{T}_\Phi$ consists of those terms in $\mathbb{T}_1$ that appear in $\Gamma^+ - \Gamma$ before or in Φ, $\mathbb{S}$ is of the same form as the $\mathbb{S}$ in case 1, and Γ_2 is characterized by $\mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+_\Phi) = (\_, \Gamma_2, \_)$. Clearly $\mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_\Phi, \Gamma, \Gamma^+_\Phi) = \mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+_\Phi)$. Now by applying case 1 of this lemma to $\mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_\Phi, \Gamma, \Gamma^+_\Phi)$, every MHF term in Γ_2 is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_\Phi$. By an argument analogous to that in case 1, the MHF terms in $\mathbb{S}(\Phi)$ are also composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_\Phi$. So the MHF terms in $\forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi}(\bigwedge \Gamma_2 \to \mathbb{S}(\Phi))$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$, as required.

3. We prove this by induction over the length of $\Gamma^+ - \Gamma$. (Note that $\mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+)$ is only defined if Γ^+ extends Γ.)

   In the base case, $\Gamma^+ = \Gamma$. Then both (a) and (b) are trivially true.

   Now suppose that the lemma holds for Γ^+. We need to show that is also holds for $\Gamma^+ \oplus \langle \Phi \rangle$ in place of Γ^+. If Φ is not presuppositionally marked, this is trivial, so we now assume Φ to be presuppositionally marked. Then $\mathit{pull\_out\_pres}(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+ \oplus \langle \Phi^{\mathbb{P}} \rangle)$ is of the form

   $$(\Gamma' \oplus \langle X \rangle, \Gamma_0, \mathbb{S}'),$$

   where

   $$\begin{aligned}
   (\Gamma', \Gamma_0, \mathbb{S}) &= \mathit{pull_out_pres}(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+), \\
   \mathbb{T}_\Phi &= \langle t \in \mathbb{T}_1 \mid t \text{ occurs in } \Gamma^+ \oplus \langle \Phi \rangle \rangle, \\
   \mathbb{S}' &= \mathbb{S} \oplus \langle (sk_i(\mathbb{T}), sk_i(\mathbb{T} \oplus \mathbb{T}_0 \oplus \mathbb{T}_\Phi)) \mid sk_i(\mathbb{T}) \text{ occurs in } \Phi \rangle, \text{ and} \\
   X &= \forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi}(\bigwedge \Gamma_0 \to \mathbb{S}'(\Phi)).
   \end{aligned}$$

   (a) ⇒ (b)

   Assume that $M + S, g \models \Gamma$, and that for all presuppositionally marked premises Ψ in $(\Gamma^+ \oplus \langle \Phi^p \rangle) - \Gamma$,

   $$\mathit{pres}(\Gamma^+ \oplus \langle \Phi^p \rangle, \Gamma, \mathbb{T}_0 \oplus \mathbb{T}_1, M, S, g, \Psi). \tag{6.1}$$

   Now fix a premise Ψ_0 in $(\Gamma' \oplus \langle X \rangle) - \Gamma$ and a $(\Gamma' \oplus \langle X \rangle)_{\Psi_0}$-skolem-assignment $S' \succeq S$ such that $M + S', g \models (\Gamma' \oplus \langle X \rangle)_{\Psi_0}$. We need to show that there is a $(\Gamma' \oplus \langle X \rangle)$-skolem-assignment $S_1 \succeq S'$ such that $M + S_1, g \models \Gamma' \oplus \langle X \rangle$.

   Note that from the definition of *pull_out_pres*, it follows that every replacement pair in $\mathbb{S}$ is of the form $(sk_i(\mathbb{T}^i), sk_i(\mathbb{T}^i + \mathbb{T}^i_0))$, where $\mathbb{T}^i_0$ is some

initial segment of $\mathbb{T}_0 + \mathbb{T}_\Phi$, and that

$\Gamma_0 = \mathbb{S}(|\Gamma^+ - \Gamma|)$ and

$\Gamma' = \Gamma \oplus \langle \forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Psi} (\bigwedge \mathbb{S}(|(\Gamma^+ - \Gamma)_\Psi|) \to \mathbb{S}(\Psi)) \mid \Psi$ is a presuppositionally marked formula in $\Gamma^+ - \Gamma \rangle$.

We now need to choose a Γ'-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, g \models \Gamma'$. If $\Psi_0 \neq X$ (as an inequality of premise occurrences), then such a S_0 exists by the inductive hypothesis. If $\Psi_0 = X$ (as an equality of premise occurrences), then we can set S_0 to be S'.

For every $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$, we define a Γ^+-skolem-assignment $S_k \succeq S'$ as follows: For every sk_i introduced in $\Gamma^+ - \Gamma^+_{\Psi_0}$ and every tuple $\vec{x}$ of elements from M of the same length as $\mathbb{T}^i$, we set

$$S_k(sk_i)(\vec{x}) := S_0(sk_i)(\vec{x} \oplus k(\mathbb{T}^i_0)).$$

Now it follows from (6.1) applied to Φ that for every $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$ such that $M + S_k, k \models (\Gamma^+ - \Gamma)_\Phi$, there is a $(\Gamma^+ \oplus \langle \Phi \rangle)$-skolem-assignment $S'_k \succeq S_k$ such that $M + S'_k, k \models \Phi$. This gives rise to a partial function $k \mapsto S'_k$, mapping M-assignments k with $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$ to $(\Gamma^+ \oplus \langle \Phi \rangle)$-skolem-assignment with $S'_k \succeq S_k$, defined on all such k that additionally satisfy $M + S_k, k \models (\Gamma^+ - \Gamma)_\Phi$. We make this function total, i.e. extend it to a total function $k \mapsto S'_k$ from M-assignments k with $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$ to $(\Gamma^+ \oplus \langle \Phi \rangle)$-skolem-assignment with $S'_k \succeq S_k$: The additional values for S'_k needed to make this function total may be chosen arbitrarily from the codomain (i.e. from the set of $(\Gamma^+ \oplus \langle \Phi \rangle)$-skolem-assignments with $S'_k \succeq S_k$).

We now define the required $(\Gamma' \oplus \langle X \rangle)$-skolem-assignment S_1 by extending S_0 as follows: For every sk_i introduced in X, every tuple $\vec{x}$ of elements from M of the same length as $\mathbb{T}^i$ and every tuple $\vec{y}$ of elements from M of the same length as $\mathbb{T}^i_0$, we set

$$S_1(sk_i)(\vec{x} \oplus \vec{y}) := S'_{g\frac{\mathbb{T}^i_0}{\vec{y}}}(sk_i)(\vec{x}),$$

where $g\frac{\mathbb{T}^i_0}{\vec{y}}$ is the M-assignment that coincides with g outside $\mathbb{T}^i_0$ and maps the n-th element of $\mathbb{T}^i_0$ to the n-th element of $\vec{y}$. Since S_1 coincides with S_0 on all skolem function symbols in Γ', $M + S_1, g \models \Gamma'$. So we now only have to show that $M + S_1, g \models X$, i.e. that

$$M + S_1, g \models \forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi} (\bigwedge \Gamma_0 \to \mathbb{S}'(\Phi)).$$

Let $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$. Suppose $M + S_1, k \models \Gamma_0$. We have to show that $M + S_1, k \models \mathbb{S}'(\Phi)$. Since Γ' does not contain free terms from $\mathbb{T}_0 \oplus \mathbb{T}_1$, $M + S_1, k \models \Gamma'$. Now it follows from the characterizations of Γ_0 and Γ' presented above that $M + S_1, k \models \mathbb{S}(\Gamma^+ - \Gamma)$. From this and the definition of S_k, it follows that $M + S_k, k \models \Gamma^+ - \Gamma$. By the choice of S'_k, this implies $M + S'_k, k' \models \Phi$, i.e. $M + S_1, k \models \mathbb{S}'(\Phi)$, as required.

(b) $\Rightarrow$ (a)

Assume that $M + S, g \models \Gamma$, and that the following property holds:

$$\begin{array}{l}\text{For every premise } \Psi \text{ in } (\Gamma' \oplus \langle X\rangle) - \Gamma \text{ and every } (\Gamma' \oplus \\ \langle X\rangle)_\Psi\text{-skolem-assignment } S' \succeq S \text{ such that } M + S', g \models \\ (\Gamma' \oplus \langle X\rangle)_\Psi, \text{ there is a } \Gamma' \oplus \langle X\rangle\text{-skolem-assignment } S_0 \succeq S' \\ \text{such that } M + S_0, g \models \Gamma' \oplus \langle X\rangle.\end{array} \tag{6.2}$$

Now for a presuppositionally marked Ψ in $\Gamma^+ - \Gamma$, (a) follows from the inductive hypothesis. So we only need to show (a) for X. So suppose that $S' \succeq S$ is a Γ^+-skolem-assignment and that $k[\mathbb{T}_0 \oplus \mathbb{T}_1]g$ is an M-assignment such that $M + S', k \models \Gamma^+$. Without loss of generality, we may assume that $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$. We need to show that there is a $\Gamma^+ \oplus \langle\Phi\rangle$-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Gamma^+ \oplus \langle\Phi\rangle$.

In order to be able to apply (6.2) for $\Psi = X$, we need to extend S to a Γ'-skolem-assignment $S^* \succeq S$ such that $M + S^*, g \models \Gamma'$. For this we recursively define for every Ψ in $\Gamma' - \Gamma$ a Γ'_Ψ-skolem-assignment $S_{\Gamma'_\Psi} \succeq S$ such that $M + S_{\Gamma'_\Psi}, g \models \Gamma'_\Psi$, and finally set $S^* := S_{\Gamma'}$. The base case of the recursive definition is $S_\Gamma := S$. Now suppose that $S_{\Gamma'_\Psi}$ has been defined in such a way that $S_{\Gamma'_\Psi} \succeq S$ and

$$M + S_{\Gamma'_\Psi}, g \models \Gamma'_\Psi. \tag{6.3}$$

We need to define $S_{\Gamma_{\Psi+}} \succeq S$ such that $M + S_{\Gamma'_{\Psi+}}, g \models \Gamma'_{\Psi+}$.

By the above characterization of Γ', Ψ is of the form $\forall_{\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}}(\bigwedge \mathbb{S}(|(\Gamma^+ - \Gamma)_{\Psi'}|) \to \mathbb{S}(\Psi'))$ for some presuppositionally marked Ψ' in Γ^+. Let $h[\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}]g$. Define a $\Gamma^+_{\Psi'}$-skolem-assignment $S^h_{\Psi'}$ as follows: For every sk_i introduced in $\Gamma^+_{\Psi'} - \Gamma$ and every tuple $\vec{x}$ of elements from M of the same length as $\mathbb{T}^i$, we set

$$S^h_{\Psi'}(sk_i)(\vec{x}) := S_{\Gamma'_\Psi}(sk_i)(\vec{x} \oplus h(\mathbb{T}^i_0)).$$

Now suppose $M + S_{\Gamma'_\Psi}, h \models \mathbb{S}(|(\Gamma^+ - \Gamma)_{\Psi'}|)$. Then $M + S_{\Gamma'_\Psi}, h \models \mathbb{S}((\Gamma^+ - \Gamma)_{\Psi'})$ by (6.3) and the characterization of Γ'. This implies that $M + S^h_{\Gamma'_\Psi}, h \models (\Gamma^+ - \Gamma)_{\Psi'}$. So by assertion (a) from the inductive hypothesis, there is a $\Gamma^+_{\Psi'+}$-skolem-assignment $S^h_{\Psi'+} \succeq S^h_{\Psi'}$ such that $M + S^h_{\Psi'+}, h \models \Psi'$.

Just as we made the partial function $k \mapsto S'_k$ total in the (a) $\Rightarrow$ (b) part of the proof, we now extend the partial function $h \mapsto S^h_{\Psi'}$ to a total function, i.e. we assume that some $\Gamma^+_{\Psi'+}$-skolem-assignment $S^h_{\Psi'+} \succeq S^h_{\Psi'}$ has been chosen for every $h[\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}]g$. Now we complete the recursive definition by defining the $\Gamma'_{\Psi'+}$-skolem-assignment $S_{\Gamma'_{\Psi'+}} \succeq S_{\Gamma'_{\Psi'}}$ as follows: For every sk_i introduced in Ψ', every tuple $\vec{x}$ of elements from M of the same length as T_i, and every tuple $\vec{y}$ of elements from M of the same length as $\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}$, we set

$$S_{\Gamma'_{\Psi'+}}(sk_i)(\vec{x} \oplus \vec{y}) := \begin{cases} S^{g\frac{\mathbb{T}_0, \mathbb{T}_{\Psi'}}{\vec{y}}}_{\Psi+}(sk_i)(\vec{x}) & \text{if } g\frac{\mathbb{T}_0, \mathbb{T}_{\Psi'}}{\vec{y}} \neq k \\ S'(sk_i)(\vec{x}) & \text{if } g\frac{\mathbb{T}_0, \mathbb{T}_{\Psi'}}{\vec{y}} = k. \end{cases}$$

We now need to check that our definition of $S_{\Gamma'_{\Psi'+}}$ actually ensures that $M+S_{\Gamma'_{\Psi+}}, g \models \Gamma'_{\Psi+}$. Since $S_{\Gamma'_{\Psi'+}} \succeq S_{\Gamma'_{\Psi'}}$, certainly $M+S_{\Gamma'_{\Psi+}}, g \models \Gamma'_{\Psi}$. So what remains to be shown is that $M+S_{\Gamma'_{\Psi+}}, g \models \Psi$. Recall that Ψ is of the form $\forall_{\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}}(\bigwedge \mathbb{S}(|(\Gamma^+ - \Gamma)_{\Psi'}|) \to \mathbb{S}(\Psi'))$. So assume that $h[\mathbb{T}_0 \oplus \mathbb{T}_{\Psi'}]g$ is such that $M + S_{\Gamma'_{\Psi+}}, h \models \mathbb{S}(|(\Gamma^+ - \Gamma)_{\Psi'}|)$. Then by the choice of $S^h_{\Psi'+}$, $M + S^h_{\Psi'+}, h \models \Psi'$. Additionally, note that since $M + S', k \models \Gamma^+$, $M+S', k \models \Psi'$. These two facts together with the definition of $S_{\Gamma'_{\Psi'+}}$ now imply that $M + S_{\Gamma'_{\Psi+}}, h \models \mathbb{S}(\Psi')$.

Now we can finally apply (6.2) with the Ψ and S' in (6.2) instantiated to X and S^*. This allows us to conclude that there is a $\Gamma' \oplus \langle X \rangle$-skolem-assignment $S_0 \succeq S^*$ such that $M + S_0, g \models \Gamma' \oplus \langle X \rangle$. Now we define $S'' \succeq S'$ as follows: For every sk_i introduced in Φ and every tuple $\vec{x}$ of elements from M of length T_i, we set

$$S''(sk_i)(\vec{x}) := S_0(sk_i)(\vec{x} \oplus k(\mathbb{T}_0 \oplus \mathbb{T}_1)).$$

In order to conclude the proof, we now only need to show that $M+S'', k \models \Gamma^+ \oplus \langle \Phi \rangle$. Since $S'' \succeq S'$, $M + S'', k \models \Gamma^+$, so it is enough to show that $M+S'', k \models \Phi$. For this we first conclude from $M+S'', k \models |\Gamma^+ - \Gamma|$ and the definition of S'' that $M+S_0, k \models \mathbb{S}(|\Gamma^+ - \Gamma|)$, i.e. that $M+S_0, k \models \Gamma_0$. Now $k[\mathbb{T}_0 \oplus \mathbb{T}_\Phi]g$, $M + S_0, g \models X$ and X is $\forall_{\mathbb{T}_0 \oplus \mathbb{T}_\Phi}(\bigwedge \Gamma_0 \to \mathbb{S}'(\Phi))$, so $M + S_0, k \models \mathbb{S}'(\Phi)$. Now the definition of S'' implies $M + S'', k \models \Phi$, as required. □

The following *read_text* Soundness Lemma will be needed in the inductive proof of the Detailed Soundness Lemma. It has a structure very analogous to that of the Detailed Soundness Lemma. The only structural differences are that it needs an additional assumption, namely that the Detailed Soundness Lemma holds for the *PTL* text θ to which we want to apply this lemma, and that the assertion 2 of the Detailed Soundness Lemma is split into two assertions, 2 and 3.

Lemma 6.3.19 (*read_text* Soundness Lemma)**.** *Assume the following properties:*

(i) *θ is a semi-nice PTL text such that the Detailed Soundness Lemma holds for θ.*

(ii) *$\mathbb{T}$ and $\mathbb{T}_0$ are PTL-PL term lists such that $PL^{-1}(\mathbb{T} \oplus \mathbb{T}_0) \oplus \mathbf{qt}(\theta)$ is pairwise independent.*

(iii) *All MHF terms of θ are composed of terms in $PL^{-1}(\mathbb{T} \oplus \mathbb{T}_0)$.*

(iv) *Γ is a premise list such that all MHF terms in Γ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$.*

(v) *$read\_text(\theta, \mathbb{T}_0, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}_1, \Theta, \nu)$.*

(vi) *M is a CMTN model, S a Γ-skolem-assignment and g an M-assignment such that $M + S, g \models \Gamma$.*

(vii) *$dom(g) = PL^{-1}(\mathbb{T})$.*

(viii) *For all $T \in \mathbb{T}$, $\frac{M+S}{g}(T) \neq u^M$.*

Then the following six properties hold:

1. $\mathbf{tbc}(\theta) = PL^{-1}(\mathbb{T}_1)$.
2. *All MHF terms of* Θ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T} \cup \mathbb{T}_0 \cup \mathbb{T}_1$.
3. *All MHF terms in* $\Gamma' - \Gamma$ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$.
4. *Either* $\nu = u$ *or* $\nu = \mu$ *and for every* $g'[\mathbb{T}_0]g$, *def*$([\![\theta]\!]^{g'}_M)$.
5. *The following two properties are equivalent:*
   (a) *For every* $g'[\mathbb{T}_0]g$, *def*$([\![\theta]\!]^{g'}_M)$.
   (b) *For every* Φ *in* $\Gamma' - \Gamma$ *and every* Γ'_Φ*-skolem-assignment* $S' \succeq S$ *such that* $M + S', g \models \Gamma'_\Phi$, *there is a* Γ'*-skolem-assignment* $S_0 \succeq S'$ *such that* $M + S_0, g \models \Gamma'$.
6. *Suppose that for every* $g'[\mathbb{T}_0]g$, *def*$([\![\theta]\!]^{g'}_M)$. *Fix an* M*-assignment* $g'[\mathbb{T}_0]g$. *Then the following three properties are equivalent:*
   (a) $k \in [\![\theta]\!]^{g'}_M$.
   (b) $k[\mathbb{T}_1]g'$ *and for every* Γ'*-skolem-assignment* S' *extending* S *such that* $M + S', g' \models \Gamma'$, $M + S', k \models \Theta$.
   (c) $k[\mathbb{T}_1]g'$ *and there is a* Γ'*-skolem-assignment* S' *extending* S *such that* $M + S', k \models \Gamma' \oplus \langle\Theta\rangle$.
7. *If* $g'[\mathbb{T}_0]g$, *def*$([\![\theta]\!]^{g}_M)$ *and* $k \in [\![\theta]\!]^{g'}_M$, *then for every* Γ'*-skolem-assignment* S' *extending* S *such that* $M + S', g' \models \Gamma' \oplus \langle\Theta\rangle$ *and for every* $T \in \mathbb{T}_1$, $\frac{M+S'}{k}(T) \neq u$.

Proof. Since *read_text*$(\theta, \mathbb{T}_0, \Gamma, \mu) = (\Gamma', \mathbb{T}_1, \Theta, \nu)$, there are Γ^+, $\mathbb{T}'$, μ_0 such that

I. *check_text*$(\theta, \Gamma, \mathbb{T} \oplus \mathbb{T}_0, \mu) = (\Gamma^+, \mathbb{T}', \mu_0)$,

II. $T_1 = \mathbb{T}' - (\mathbb{T} \oplus \mathbb{T}_0)$,

III. *pull_out_pres*$(\mathbb{T}_0, \mathbb{T}_1, \Gamma, \Gamma^+) = (\Gamma', \Gamma_0, \mathbb{S})$,

IV. $\Theta = \bigwedge \Gamma_0$, and

V. $\nu = update(\mu, 0, \pi_1(\mu_0))$.

Now we prove each of the assertions of the lemma separately:

1. This follows directly from II and from assertion 1 of the Detailed Soundness Lemma.
2. By assertion 2 of the Detailed Soundness Lemma applied to I, $\Gamma^+ - \Gamma$ is composed of PTL_{sk} symbols and terms in $\mathbb{T}'$. $\Theta = \bigwedge \Gamma_0$. So an MHF term of Θ is an MHF term of Γ_0, and hence by assertion 1 of the *pull_out_pres* Soundness Lemma composed of PTL_{sk} symbols and terms in $\mathbb{T}'$.
3. This directly follows from assertion 2 of the *pull_out_pres* Soundness Lemma.

4. There are two cases:

   Case A: $\mu_0 = u$. Then $\nu = u$ by V.

   Case B: $\mu_0 \neq u$. Then by V, $\nu = \mu$. By assertion 5 of the Detailed Soundness Lemma applied to I, $\mu + v(\theta, M, g) \geq \mu_0$, so $\mu + v(\theta, M, g) \neq u$, i.e. $def([\![\theta]\!]^g_M)$.

5. (a) $\Rightarrow$ (b):

   Assume that for every $g'[\mathbb{T}_0]g$, $def([\![\theta]\!]^{g'}_M)$.

   Let g' be an M-assignment such that $g'[\mathbb{T}_0]g$. Then $def([\![\theta]\!]^{g'}_M)$. Note that by (ii), no term in $\mathbb{T}_0$ is composed of terms in $\mathbb{T}$. This together with (vi), (iv) and (vii) implies that $M, g' \models \Gamma$. Now we can apply the Detailed Soundness Lemma to θ using g' instead of g. Assertion 3 of this application of the Detailed Soundness Lemma then implies that for all presuppositionally marked Φ from $\Gamma^+ - \Gamma$, for all Γ^+_Φ-skolem-assignments $S' \succeq S$ and for all $k[\mathbb{T}_1]g'$ such that $M, k \models (\Gamma^+ - \Gamma)_\Phi$, $M, k \models \Phi$.

   From this it follows that for all presuppositionally marked Φ from $\Gamma^+ - \Gamma$, for all Γ^+_Φ-skolem-assignments $S' \succeq S$ and for all $k[\mathbb{T}_0 \oplus \mathbb{T}_1]g$ such that $M, k \models (\Gamma^+ - \Gamma)_\pi$, $M, k \models \Phi$. Now it follows from assertion 3 of the *pull_out_pres* Soundness Lemma that for every Φ in $\Gamma' - \Gamma$ and every Γ'_Φ-skolem-assignment $S' \succeq S$ such that $M + S', g \models \Gamma'_\Phi$, there is a Γ'-skolem-assignment S_0 extending S' such that $M + S_0, g \models \Phi$.

   (b) $\Rightarrow$ (a):

   Assume that for every Φ in $\Gamma' - \Gamma$ and every Γ'_Φ-skolem-assignment $S' \succeq S$ such that $M + S', g \models \Gamma'_\Phi$, there is a Γ'-skolem-assignment S_0 extending S' such that $M + S_0, g \models \Phi$. Fix $g'[\mathbb{T}_0]g$. We have to show that $def([\![\theta]\!]^{g'}_M)$.

   By assertion 3 of the *pull_out_pres* Soundness Lemma, our assumption implies that for all presuppositionally marked premises Φ in $\Gamma^+ - \Gamma$, $pres(\Gamma^+, \Gamma, \mathbb{T}_0 \oplus \mathbb{T}_1, M, S, g, \Phi)$. In particular, this means that for all presuppositionally marked premises Φ in $\Gamma^+ - \Gamma$, $pres(\Gamma^+, \Gamma, \mathbb{T}_0, M, S, g', \Phi)$. Now by assertion 3 of the Detailed Soundness Lemma applied to θ, $def([\![\theta]\!]^{g'}_M)$.

6. (b) $\Rightarrow$ (c) follows from 5. By the definition of *pull_out_pres*, we have $\Gamma_0 = \mathbb{S}(|\Gamma^+ - \Gamma|)$, i.e. $\Theta = \bigwedge \mathbb{S}(|\Gamma^+ - \Gamma|)$. Using transformations between Γ'-skolem-assignments and Γ^+-skolem-assignments analogous to those used in the proof of assertion 3 of the *pull_out_pres* Soundness Lemma, (a) $\Rightarrow$ (b) and (c) $\Rightarrow$ (a) can now be derived from the corresponding implications in assertion 4 of the Detailed Soundness Lemma applied to I.

7. Suppose that $g'[\mathbb{T}_0]g$, $def([\![\theta]\!]^{g'}_M)$ and $k \in [\![\theta]\!]^{g'}_M$. Let $S' \succeq S$ be a Γ'-skolem-assignment such that $M + S', g' \models \Gamma' \oplus \langle\Theta\rangle$, and suppose $T \in \mathbb{T}_1$. By the already proven assertion 6 of this lemma, $k[\mathbb{T}_1]g'$. But since $T_1 = \mathbb{T}_1$, $T \in \mathbb{T}_1$, i.e. $T \in dom(k)$, i.e. $\frac{M+S'}{k}(T) = k(T) \neq u$ by the definition of *M-assignment*. □

Just as the *read_text* Soundness Lemma, the following *read_term* Soundness Lemma will be needed in the inductive proof of the Detailed Soundness Lemma

and has a structure very analogous to that of the *read_text* Detailed Soundness Lemma. The only structural difference is that there are no analogues to the assertions 1 and 7 of the *read_text* Detailed Soundness Lemma.

Lemma 6.3.20 (*read_term* Soundness Lemma)**.** *Assume the following properties:*

(i) t *is a PTL term such that the Detailed Soundness Lemma holds for all PTL texts that are subtexts of* t*.*

(ii) $\mathbb{T}$ *is a PTL-PL term list such that* $PL^{-1}(\mathbb{T}) \oplus \mathbf{qt}(t)$ *is pairwise independent.*

(iii) All MHF terms of t *are composed of terms in* $\mathbb{T}$*.*

(iv) Γ *is a premise list such that all MHF terms in* Γ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$*.*

(v) $read\_term(t, \Gamma, \mathbb{T}, \mu) = (\Gamma', T, \nu)$*.*

(vi) M *is a CMTN model,* S *a* Γ*-skolem-assignment and* g *an* M*-assignment such that* $M + S, g \models \Gamma$*.*

(vii) $dom(g) = PL^{-1}(\mathbb{T})$*.*

(viii) For all $T' \in \mathbb{T}$*,* $\frac{M+S}{g}(T') \neq u^M$*.*

Then the following five properties hold:

1. T *is composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$*.*

2. *All MHF terms in* $\Gamma' - \Gamma$ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T}$*.*

3. $\frac{M}{g}(t) \neq u^M$ *iff for every* Φ *in* $\Gamma' - \Gamma$ *and every* Γ'_Φ*-skolem-assignment* $S' \succeq S$ *such that* $M + S', g \models \Gamma'_\Phi$*, there is a* $\Gamma'_{\Phi+}$*-skolem-assignment* $S_0 \succeq S'$ *such that* $M + S_0, g \models \Phi$*.*

4. *If* $\frac{M}{g}(t) \neq u^M$ *and* S_0 *is a* Γ'*-skolem-assignment extending* S *such that* $M + S_0, g \models \Gamma'$*, then* $\frac{M}{g}(t) = \frac{M+S_0}{g}(T)$*.*

5. *Either* $\nu = u$ *or* $\nu = \mu$ *and* $\frac{M}{g}(t) \neq u^M$*.*

Proof. We prove this by induction over the length of t. So assume that this lemma holds for all *PTL* terms shorter than t. We have to distinguish four different cases:

1. $t \in T_{\boldsymbol{PTL}}$ **and** $\boldsymbol{PL}(t) \in \mathbb{T}$

In this case $\Gamma' = \Gamma$,$\nu = \mu$ and $T = PL(t)$. Since $T \in \mathbb{T}$, $\frac{M+S}{g}(T) \neq u^M$. $T = PL(t)$ cannot contain any skolem function symbols, so the definitions of $\frac{M}{g}(t)$, $\frac{M+S}{g}(T)$ and $PL(t)$ imply that $\frac{M}{g}(t) = \frac{M+S}{g}(T) \neq u^M$. Now the five assertions of the lemma follow trivially.

2. Case 1 does not hold and t is of the form $f(t_1, \ldots, t_n)$, where f is a logical function symbol and $t_1, \ldots, t_n$ are *PTL* terms

Then $\mathit{read\_term}(f(t_1, \ldots, t_n), \Gamma, \mathbb{T}, \mu) = (\Gamma', f(T_1, \ldots, T_n), \nu)$, where for all $1 \leq i \leq n$, $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i) = (\Gamma_{i+1}, T_i, \mu_{i+1})$, with

- $\Gamma_1 = \Gamma$,
- $\mu_1 = \mu$,
- $\Gamma' = \Gamma_{n+1} \oplus \langle (f(T_1, \ldots, T_n) \neq u)^{\mathbb{P}} \rangle$,
- $\nu = \mathit{update}(\mu_{n+1}, 0, P(\Gamma_{n+1} \vdash^? f(T_1, \ldots, T_n) \neq u))$.

Now we prove the five assertions of the lemma for this case:

1. This follows directly from assertion 1 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ for $1 \leq i \leq n$.
2. This follows directly from assertion 2 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ for $1 \leq i \leq n$.
3. This follows directly from assertions 3 and 4 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ for $1 \leq i \leq n$.
4. This follows directly from assertion 4 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ for $1 \leq i \leq n$.
5. Assume $\nu \neq u$. Since $\nu = \mathit{update}(\mu_{n+1}, 0, P(\Gamma_{n+1} \vdash^? f(T_1, \ldots, T_n) \neq u))$, this implies that $\mu_{n+1} \neq u$ and $\mathit{CMTN} \cup \Gamma_{n+1} \models f(T_1, \ldots, T_n) \neq u$. Then for any $1 \leq i \leq n$, we inductively get from assertion 3 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ that $\mu_i \neq u$ and $\frac{M}{g}(t) \neq u^M$. In particular we have $\mu = \mu_1 \neq u$. Additionally, by assertions 3 and 4 of this lemma applied to $\mathit{read\_term}(t_i, \Gamma_i, \mathbb{T}_i, \mu_i)$ for $1 \leq i \leq n$, there is a Γ_{n+1}-skolem-assignment extending $S_0 \succeq S$ such that $M + S_0, g \models \Gamma_{n+1}$, and $\frac{M}{g}(t_i) = \frac{M+S_0}{g}(T_i)$ for $1 \leq i \leq n$. Since $\mathit{CMTN} \cup \Gamma_{n+1} \models f(T_1, \ldots, T_n) \neq u$, $M + S_0, g \models f(T_1, \ldots, T_n) \neq u$, i.e. $\frac{M}{g}(f(t_1, \ldots, t_n)) = \frac{M+S_0}{g}(f(T_1, \ldots, T_n)) \neq u^M$, as required.

3. Case 1 does not hold and t is of the form $t_0(t_1, \ldots, t_n)$, where $t_0, t_1, \ldots, t_n$ are *PTL* terms

This case can be treated in a way completely analogous to the previous case.

4. t is of the form $\iota x\ \varphi$

Then $\mathit{read\_term}(\iota x\ \varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma', sk_i, \nu)$ for a new 0-ary skolem function symbol sk_i, where

I. $\mathit{read\_text}(\varphi, \langle x \rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma_0, \mathbb{T}_0, \Phi, \mu_1)$,

II. $\mu_2 = \mathit{update}(\mu_1, 0, P(\Gamma_0 \vdash^? \exists x\ \exists_{\mathbb{T}_0}\ \Phi))$,

III. $\nu = \mathit{update}(\mu_2, 0, P(\Gamma_0 \oplus \langle \exists_{\mathbb{T}_0}\ \Phi \frac{sk_i}{x} \rangle \vdash^? \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)))$,

IV. $\Gamma' = \Gamma_0 \oplus \langle (\exists_{\mathbb{T}_0}\ \Phi \frac{sk_i}{x})^{\mathbb{P}}, (\forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i))^{\mathbb{P}} \rangle$.

φ is a subtext of t, so by the first assumption of this lemma, the Detailed Soundness Lemma holds for φ. This means that we can apply the *read_text* Soundness Lemma to I (with the g of lemma the *read_text* Soundness Lemma instantiated to any M-assignment $g'[x]g$). Now we prove the five assertions of the lemma for this case:

1. Trivially, sk_i is composed of PTL_{sk} symbols.

2. For any MHF term T' in $\Gamma_0 - \Gamma$, it follows from assertion 3 of the *read_text* Soundness Lemma applied to I that T' is composed of PTL_{sk} symbols and terms in $\mathbb{T}$. From assertion 2 of this application of the *read_text* Soundness Lemma, it follows that all MHF terms of Φ are composed of PTL_{sk} symbols and terms in $\mathbb{T} \cup \langle x \rangle \cup \mathbb{T}_0$. But then all MHF terms of $\exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x}$ and of $\forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$, as required.

3. Left-to-right implication:

   Suppose $\frac{M}{g}(\iota x\ \varphi) \neq u^M$. Then by the definition of $\frac{M}{g}(t)$, $[\![\varphi]\!]^{g'}_M$ is defined for every $g'[x]g$. So by assertion 5 of the *read_text* Soundness Lemma, the required property holds for all Φ in $\Gamma_0 - \Gamma$. Now we only have to show that it also holds for $\exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x}$ and $\forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$.

   Since $\frac{M}{g}(\iota x\ \varphi) \neq u^M$, there is a unique $h[x]g$ such that $[\![\varphi]\!]^h_M \neq \emptyset$, say $k \in [\![\varphi]\!]^h_M$.

   Let $S' \succeq S$ be a Γ_0-skolem-assignment such that $M + S', g \models \Gamma_0$. By assumptions iv and ii, x does not occur freely in Γ. By the definition of *read_text*, it also can't occur freely in $\Gamma_0 - \Gamma$. From this we can conclude that $M + S', h \models \Gamma_0$. Now by assertion 6 of the *read_text* Soundness Lemma, $k[\mathbb{T}_0]h$ and $M + S', k \models \Phi$. Define S_0 to be the extension of S' to sk_i that maps sk_i to $h(x)$. Then $M + S_0, g \models \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x}$, as required.

   Now let $S' \succeq S$ be a $\Gamma_0 + \langle \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x} \rangle$-skolem-assignment such that $M + S', g \models \Gamma_0 + \langle \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x} \rangle$. We construct S_0 as above. By similar reasoning as above, the uniqueness of h implies that $M + S_0, g \models \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$, as required.

   Right-to-left implication:

   Assume the following:

   > For every Ψ in $\Gamma' - \Gamma$ and every Γ'_Ψ-skolem-assignment $S' \succeq S$ such that $M + S', g \models \Gamma'_\Psi$, there is a $\Gamma'_{\Psi+}$-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, g \models \Psi$. (6.4)

   From assertion 5 of the *read_text* Soundness Lemma, we can conclude that for all $g'[x]g$, $def([\![\varphi]\!]^{g'}_M)$.

   By recursive application of (6.4), there is a $\Gamma_0 + \langle \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x} \rangle$-skolem-assignment $S_1 \succeq S$ such that $M + S_1, g \models \Gamma_0 + \langle \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x} \rangle$. By assumptions (iv) and (ii) and by the definition of *read_text*, the terms in $\mathbb{T}_0$ do not occur freely in Γ_0. This allows us to conclude that there is a $k[\mathbb{T}_0]g$ such that $M + S_1, k \models \Gamma_0 + \langle \Phi\frac{sk_i}{x} \rangle$. Now define $h[x]g$ by $h(x) := S_1(sk_i)$ and $k'[x]k$ by $k'(x) := S_1(sk_i)$. Define S_0 to be the Γ_0-skolem-assignment such that $S_1 \succeq S_0$ (so $S_0 = S_1|_{dom(S_0)\setminus\{sk_i\}}$). Then $k'[\mathbb{T}_0]h$

and $M + S_0, k' \models \Gamma_0 + \langle\Phi\rangle$. Now by assertion 6 of the *read_text* Soundness Lemma, $k' \in [\![\varphi]\!]^h_M$, i.e. $[\![\varphi]\!]^h_M \neq \emptyset$.

In order to conclude that $\frac{M}{g}(\iota x\ \varphi) \neq u^M$, it is now enough to show that h is the only M-assignment such that $h[x]g$ and $[\![\varphi]\!]^h_M \neq \emptyset$. So suppose $h'[x]g$ and $[\![\varphi]\!]^{h'}_M \neq \emptyset$. We have to show that $h' = h$.

Choose $j' \in [\![\varphi]\!]^{h'}_M$. Since $M + S_1, g \models \Gamma_0$, assertion 6 of the *read_text* Soundness Lemma implies that $j'[\mathbb{T}_0]h'$ and $M + S_1, j' \models \Phi$. This means that $M + S_1, h' \models \exists_{\mathbb{T}_0}\ \Phi$. But by (6.4) with $\Psi = \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$, $M + S_1, g \models \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$, i.e. $M + S_1, h' \models \exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i$, i.e. $M + S_1, h' \models x = sk_i$. So by the definition of h, $h(x) = S_1(sk_i) = h'(x)$, i.e. $h = h'$, as required.

4. Suppose $\frac{M}{g}(\iota x\ \varphi) \neq u$, and let S_0 be a Γ'-skolem-assignment extending S such that $M + S_1, g \models \Gamma'$. We have to show that $\frac{M}{g}(\iota x\ \varphi) = \frac{M+S_1}{g}(sk_i)$.

   Using the same argument as in the right-to-left part of 3 above, we can conclude that for the unique $h[x]g$ such that $[\![\varphi]\!]^h_M \neq \emptyset$, $h(x) = S_1(sk_i)$. Now $\frac{M}{g}(\iota x\ \varphi) = h(x) = S_1(sk_i) = \frac{M+S_1}{g}(sk_i)$, as required.

5. Suppose $\nu \neq u$. Then by III and II, $\nu = \mu_2 = \mu_1 \neq u$, $\Gamma_0 \oplus \langle\exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x}\rangle \models \forall x\ (\exists\vec{y}\ \Phi \rightarrow x = sk_i)$ and $CMTN \cup \Gamma_0 \models \exists x\ \exists_{\mathbb{T}_0}\ \Phi$. Fix an M-assignment $g'[x]g$. By assertion 5 of the *read_text* Soundness Lemma, there is a Γ_0-skolem-assignment $S_0 \succeq S$ such that $M + S_0, g' \models \Gamma_0$. Then $M + S_0, g' \models \exists x\ \exists_{\mathbb{T}_0}\ \Phi$. Additionally, for any extension S_1 of S_0 to sk_i such that $M + S_1, g' \models \exists_{\mathbb{T}_0}\ \Phi\frac{sk_i}{x}$, $M + S_1, g' \models \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \rightarrow x = sk_i)$. These two facts together imply that there is a unique M-assignment $h[x]g$ such that $M + S_0, h \models \exists_{\mathbb{T}_0}\Phi$. Now it follows from assertion 6 of the *read_text* Soundness Lemma that there is a unique M-assignment $h[x]g$ such that $[\![\varphi]\!]^h_M \neq \emptyset$. The definition of $\frac{M}{g}(t)$ now implies that $\frac{M}{g}(t) \neq u^M$, as required.

   Additionally, by assertion 4 of the *read_text* Soundness Lemma, $\mu = \mu_1 = \nu$, as required. □

The following *exist_check* Soundness Lemma ensures that the predicate *exist_check* actually does check the correctness of the existential statement that is given to it.

Lemma 6.3.21 (*exist_check* Soundness Lemma)**.** *Assume that the following properties hold:*

(i) $\mathbb{T}$ *is term list,* T *a PL term and* Φ *a PL formula such that all MHF terms of* Φ *are composed of* PTL_{sk} *symbols and terms in* $\mathbb{T} \cup \{T\}$*.*

(ii) Γ *is a premise list,* M *a CMTN model,* S *a* Γ*-skolem-assignment and* g *and* M*-assignment such that* $M + S, g \models \Gamma$*.*

(iii) All skolem functions occurring Φ *also occur in* Γ*.*

(iv) μ *is a proof status value such that exist_check*$(\alpha, \Gamma, \mathbb{T}, \exists_{\langle T\rangle}\ \Phi, \mu) = (\nu)$*.*

If $\alpha = 1$*, then* $\nu = \bot$ *or* $\nu = \mu$*. If* $\alpha = 0$*, then* $\nu = u$ *or* $\nu = \mu$*. If either* $\alpha = 1$ *and* $\nu = \top$ *or* $\alpha = 0$ *and* $\nu \neq u$*, then* $M + S, g \models \exists_{\langle T\rangle}\ \Phi$*.*

Proof. The first two implications trivially follow from the definition of *exist_check*. Now suppose that $\alpha = 1$ and $\nu = \top$ or $\alpha = 0$ and $\nu \neq u$. We have to distinguish three cases:

Case 1: Φ is of the form $(C(T) \wedge \forall_x\ (x \in T \leftrightarrow \Psi(x)))$ and $P(\Gamma \oplus \langle \Psi(x) \rangle \vdash^? L(x)) = 1$

Since $P(\Gamma \oplus \langle \Psi(x) \rangle \vdash^? L(x)) = 1$, $M + S, g \models \forall x\ (\Psi(x) \rightarrow L(x))$. Now by the Class Extensionality Axiom of *CMTN* for $\Psi(x)$, $M+S, g \models \exists_{\langle T \rangle}\ (C(T) \wedge \forall_x\ (x \in T \leftrightarrow \Psi(x)))$, i.e. $M + S, g \models \exists_{\langle T \rangle}\ \Phi$, as required.

Case 2: Φ is of the form $(C(T) \wedge L(T) \wedge \forall_x\ (x \in T \leftrightarrow \Psi(x)))$, $P(\Gamma \oplus \langle \Psi(x) \rangle \vdash^? L(x)) = 1$, the symbol L does not occur in $\Psi(x)$, and for every term T_0 occurring in $\Psi(x)$ that is either in $\mathbb{T}$ or a skolem function symbol, $P(\Gamma \vdash^? L(T_0)) = 1$.

For simplifying the exposition, we first assume that all calls of *check_limitedness* in this call of *exist_check* succeeded in the first of the two possible ways that a *check_limitedness* call can succeed. In other words, we assume that for every T_0 that is either in $\mathbb{T}$ or a skolem function symbol and that occurs in $\Psi(x)$, $P(\Gamma \vdash^? L(T_0)) = 1$. Since all MHF terms of $\Psi(x)$ are composed of PTL_{sk} symbols and terms in $\mathbb{T} \cup \{T\}$, we can view $\Psi(x)$ as a parametrized formula all of whose parameters are limited (for formalizing this, we would have to replace all terms from $\mathbb{T}$ and all skolem function symbols occurring in $\Psi(x)$ by new variables). This allows us to apply Set Comprehension where we applied Class Comprehension in case 1.

Now in case some calls of *check_limitedness* succeeded in the second possible way, we have to proceed in a way analogous to the explanation of the special case of the second criterion in section 6.1.4. For the sake of simplicity, we make this explicit only for the case that a single call of *check_limitedness* succeeded in this special way (i.e. in the second possible way). In case that multiple *check_limitedness* calls succeed in this special way (even when they are nested because of the recursive definition of *check_limitedness*), we just need to proceed inductively with the exposed case as the inductive step.

So suppose *check_limitedness*$(\Gamma, \mathbb{T}, \langle \Psi(x) \rangle, T_0)$ succeeded in the second possible way. Then T_0 appears in $\Psi(x)$ only in terms of the form $T(T')$, and for some formula $\varphi(y)$ not containing L, Γ contains $\forall_y\ (T_0(y) \leftrightarrow \varphi(y))$ and for every term T^* occurring in $\varphi(y)$ that is either in $\mathbb{T}$ or a skolem function symbol, $P(\Gamma \vdash^? L(T^*)) = 1$. Let $\Psi'(x)$ be the formula resulting from $\Psi(x)$ by replacing all occurrences of terms of the form $T(T')$ in $\Psi(x)$ by $\varphi(T')$. Since Γ contains $\forall_y\ (T_0(y) \leftrightarrow \varphi(y))$, $M + S, g \models \forall_y\ (T_0(y) \leftrightarrow \varphi(y))$, i.e. $M + S, g \models \forall_x\ (\Psi(x) \leftrightarrow \Psi'(x))$. Hence we may use $\Psi'(x)$ instead of $\Psi(x)$ for applying Set Comprehension.

Case 3: Neither case 1 nor case 2 holds

In this case $\nu = \mathit{update}(\mu, \alpha, P(\Gamma \vdash^? \exists_{\langle T \rangle}\ \Phi))$. Since $\alpha = 1$ and $\nu = \top$ or $\alpha = 0$ and $\nu \neq u$, $CMTN \cup \Gamma \models \exists_{\langle T \rangle}\ \Phi$, i.e. $M + S, g \models \exists_{\langle T \rangle}\ \Phi$, as required. □

For the proof of the Detailed Soundness Lemma, we also need Lemma 5.2.20 from chapter 5 to hold for any θ with the property that the Detailed Soundness

Lemma holds for all subtexts of θ. The proof of Lemma 5.2.20 in its unrestricted form depends on the Detailed Soundness Lemma, so that we actually have to use a restricted form of 5.2.20:

Lemma 6.3.22. *Let θ be a semi-nice PTL text, and let M be a CMTN model. Suppose that the Detailed Soundness Lemma holds for all subtexts of θ. If g and h are M-assignments such that $h \in [\![\theta]\!]^g_M$ and such that the union of $dom(g)$ and the set of occurrences of terms after an ι or $\exists$ in θ is pairwise independent, then $\mathbf{tbc}(\theta) = dom(h) \setminus dom(g)$.*

Proof. We prove this by induction over the complexity of θ. So assume that the lemma holds for all subtexts of θ_0. To show that it holds for θ_0, one needs to separately check it for all 13 possible forms θ_0 can have according to Definition 5.2.1. All cases apart from θ_0 being of the form $\varphi \to \theta$ are trivial and do not even require the additional assumption that the Detailed Soundness Lemma holds for all subtexts of θ, so we only write out the proof for this case.

So suppose that g and h are M-assignments such that $h \in [\![\varphi \to \theta]\!]^g_M$ and such that the union of $dom(g)$ and the set of occurrences of terms after an ι or $\exists$ in θ is pairwise independent. First suppose that $t \in dom(h) \setminus dom(g)$. We have to show that $t \in \mathbf{tbc}(\varphi \to \theta)$. From the definition of $[\![\varphi \to \theta]\!]^g_M$, there are PTL terms $t_1, \ldots, t_m$ and an m-place argument filler σ such that for all $k \in [\![\varphi]\!]^g_M$, $k[t_1, \ldots, t_m]g$ and there is an assignment $j \in [\![\theta]\!]^k_M$ such that $j(t^\sigma(t_1, \ldots, t_m)) = h(t)(k(t_1), \ldots, k(t_m))$. By the inductive hypothesis, $\{t_1, \ldots, t_m\} = \mathbf{tbc}(\varphi)$ and $t^\sigma(t_1, \ldots, t_m) \in dom(j) \setminus dom(k) = \mathbf{tbc}(\theta)$. So by the $\varphi \to \theta$ case of the definition of $\mathbf{aq}$, $t \in \mathbf{tbc}(\varphi \to \theta)$, as required.

For the inverse direction, suppose $t \in \mathbf{tbc}(\varphi \to \theta)$. Write $\mathbf{tbc}(\varphi)$ as $t_1, \ldots, t_m$. Then there is an m-place argument filler σ and a $t_0 \in \mathbf{tbc}(\theta)$ such that $t_0 = t^\sigma(t_1, \ldots, t_m)$. Since $[\![\varphi \to \theta]\!]^g_M \neq \emptyset$, it follows from the definition of $[\![\varphi \to \theta]\!]^g_M$ that

$$\text{for every } k \in [\![\varphi]\!]^g_M, \text{ there is an } M\text{-assignment } j_k \in [\![\theta]\!]^k_M. \tag{6.5}$$

Now we make use of the fact that *read_text* gives us a way of translating PTL texts to PL formulae. Set $\mathbb{T} := PL(dom(g))$, and suppose that

$$read\_text(\varphi, \langle\rangle, \langle\rangle, \mathbb{T}, \top) = (\Gamma, \mathbb{T}_1, \Phi, \mu) \text{ and}$$
$$read\_text(\langle\rangle, \Gamma, \mathbb{T} \oplus \mathbb{T}_1, \mu) = (\Gamma', \mathbb{T}_2, \Theta, \nu).$$

By assertion 5 of the *read_text* Soundness Lemma applied to both of these uses of *read_text*, there is a Γ'-skolem-assignment S_1 such that $M + S_1, g \models \Gamma'$. Applying assertion 6 of the *read_text* Soundness Lemma to both uses of *read_text*, we can transform (6.5) into (6.6), which implies (6.7):

$$\begin{aligned}&\text{For every } k[\mathbb{T}_1]g \text{ such that } M + S_1, k \models \Phi, \text{ there is a } j[\mathbb{T}_2]k \\ &\text{such that } M + S_1, j \models \Theta.\end{aligned} \tag{6.6}$$

$$M + S_1, g \models \forall_{\mathbb{T}_1} (\Phi \to \exists_{\mathbb{T}_2} \Theta) \tag{6.7}$$

Since $def([\![\varphi \to \theta]\!]^g_M)$, we have by the definition of the domain of $[\![\bullet]\!]^g_M$ that for every $k \in [\![\varphi]\!]^g_M$ and every $j \in [\![\theta]\!]^k_M$, if there is a $t' \in dom(j) \setminus dom(k)$ of the form $f^\sigma(t_1, \ldots, t_m)$, then $j(t') \in L^M$ and $k(t_i) \in L^M$ for $1 \leq i \leq m$. Since $[\![\varphi \to \theta]\!]^g_M$ is non-empty, $[\![\theta]\!]^k_M$ is non-empty for all $k \in [\![\varphi]\!]^g_M$. Since t_0 is in $dom(j) \setminus dom(k)$ by the inductive hypothesis, and since t_0 is of the form

$t^\sigma(t_1, \ldots, t_m)$, we can now conclude that for every $k \in [\![\varphi]\!]^g_M$, $k(t_i) \in L^M$ for $1 \leq i \leq m$. Define

$$\begin{aligned}
\mathbb{T}'_2 &:= \langle T \in \mathbb{T}_2 \mid T \text{ is of the form } T'^\sigma(\mathbb{T}_1) \text{ for some } T', \sigma \rangle, \\
\mathbb{F} &:= \langle T' \mid \text{some } T \in \mathbb{T}'_2 \text{ is of the form } T'^\sigma(\mathbb{T}_1) \rangle, \text{ and} \\
\mathbb{T}^-_2 &:= \mathbb{T}_2 - \mathbb{T}'_2.
\end{aligned}$$

Applying again assertion 6 of the *read_text* Soundness Lemma to both uses of *read_text*, we can now derive the following two claims:

$$M + S_1, g \models \forall_{\mathbb{T}_1} (\Phi \to \bigwedge_{i=1}^{m} L(PL(t_i))). \tag{6.8}$$

$$M + S_1, g \models \forall_{\mathbb{T}_1 \oplus \mathbb{T}'_2} (\Phi \wedge \Theta \to \bigwedge_{T \in \mathbb{T}'_2} L(T)). \tag{6.9}$$

By recursive application of the Map Extensionality Axiom of *CMTN*, one can verify the following:

$$\begin{aligned}
&CMTN \vdash (\forall_{\mathbb{T}_1} (\Phi \to \exists_{\mathbb{T}_2} \Theta) \wedge \forall_{\mathbb{T}_1} (\Phi \to \bigwedge_{i=1}^{m} L(PL(t_i))) \\
&\wedge \forall_{\mathbb{T}_1 \oplus \mathbb{T}'_2} (\Phi \wedge \Theta \to \bigwedge_{T \in \mathbb{T}'_2} L(T))) \\
&\to \exists_{\mathbb{F}} \forall_{\mathbb{T}_1} ((\Phi \to \exists_{\mathbb{T}^-_2} \Theta) \wedge (\neg\Phi \to \bigwedge_{T \in \mathbb{T}'_2} T = u)).
\end{aligned}$$

This together with (6.7), (6.8) and (6.9) now implies that

$$M + S_1, g \models \exists_{\mathbb{F}} (\forall_{\mathbb{T}_1} (\Phi \to \exists_{\mathbb{T}^-_2} \Theta) \wedge (\neg\Phi \to \bigwedge_{T \in \mathbb{T}'_2} T = u)).$$

From this it follows that there is an $h'[\mathbb{F}]g$ such that

$$M + S_1, h' \models \forall_{\mathbb{T}_1} (\Phi \to \exists_{\mathbb{T}^-_2} \Theta) \wedge (\neg\Phi \to \bigwedge_{T \in \mathbb{T}'_2} T = u).$$

Again applying assertion 6 of the *read_text* Soundness Lemma to both uses of *read_text*, one can now verify that $h' \in [\![\varphi \to \theta]\!]^g_M$.

$t \in PL^{-1}(\mathbb{F})$, so $t \in dom(h')$. If $t \notin dom(h)$, then $dom(h) \setminus dom(g) \subsetneq PL^{-1}(\mathbb{F})$, contradicting the maximality of n in the definition of $[\![\varphi \to \theta]\!]^g_M$. Thus $t \in dom(h)$, as required. □

Similarly, we have to prove a restricted form of Lemma 5.2.21:

Lemma 6.3.23. *Let M be a CMTN model, g an M-assignment, φ a PTL formula and θ a PTL text such that $[\![\varphi \to \theta]\!]^g_M$ is defined. Suppose that the Detailed Soundness Lemma holds for all subtexts of $\varphi \to \theta$. Then $[\![\varphi \to \theta]\!]^g_M \neq \emptyset$ iff for every $k \in [\![\varphi]\!]^g_M$, $[\![\theta]\!]^k_M \neq \emptyset$.*

Proof. Suppose that $[\![\varphi \to \psi]\!]^g_M \neq \emptyset$, say $h \in [\![\varphi \to \psi]\!]^g_M \neq \emptyset$, and that $k \in [\![\varphi]\!]^g_M$. Then the existence of a $j \in [\![\psi]\!]^k_M \neq \emptyset$ follows directly from the definition of $[\![\varphi \to \psi]\!]^g_M \neq \emptyset$.

For the inverse direction, suppose that for every $k \in [\![\varphi]\!]^g_M$, $[\![\psi]\!]^k_M \neq \emptyset$. Now by the same construction as in the second part of the proof of the previous lemma, we can construct a $h' \in [\![\varphi \to \theta]\!]^g_M$, as required. □

6.3.1 Proof of the Detailed Soundness Lemma

We are now ready to present the proof of the Detailed Soundness Lemma. We prove this lemma by induction over the complexity of θ. So we fix a *PTL* text θ_0 and assume that the lemma holds for all subtexts of θ_0. Additionally, we assume the following properties:

[i] $\mathbb{T}$ is a list of *PTL-PL* terms such that $PL^{-1}(\mathbb{T}) \oplus \mathbf{qt}(\theta_0)$ is pairwise independent.

[ii] All MHF terms of θ_0 are composed of terms in $PL^{-1}(\mathbb{T})$.

[iii] Γ is a premise list such that all MHF terms in Γ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$.

[iv] $\mathit{check\_text}(\theta_0, \Gamma, \mathbb{T}, \mu) = (\Gamma', \mathbb{T}', \nu)$.

[v] M is a *CMTN* model, S a Γ-skolem-assignment and g an M-assignment such that $M + S, g \models \Gamma$.

[vi] $\mathit{dom}(g) = PL^{-1}(\mathbb{T})$.

[vii] For all $T \in \mathbb{T}$, $\frac{M+S}{g}(T) \neq u$.

Now we distinguish 13 different cases depending on the form of θ_0.

Case 1: θ_0 is a *PTL* term t

In this case, there is a premise list Γ^*, a *PL* term T and a proof status value μ_0 such that the following hold:

(i) $\mathit{read\_term}(t, \Gamma, \mathbb{T}, \mu) = (\Gamma^*, T, \mu_0)$.

(ii) $\mu_1 = \mathit{update}(\mu_0, 0, P(\Gamma^* \vdash^? B(T)))$.

(iii) $\nu = \mathit{update}(\mu_1, 1, P(\Gamma^* \oplus \langle B(T) \rangle \vdash^? T = \top))$.

(iv) $\Gamma' = \Gamma^* \oplus \langle B(T)^{\mathbb{P}}, T = \top \rangle$.

Now we prove each of the six assertions of the Detailed Soundness Lemma for t:

1. By the definition of *read_term*, $\mathbb{T}' - \mathbb{T}$ contains no $\mathbb{D}$-marked terms, i.e. $PL^{-1}(\mathbb{T}' - \mathbb{T}) = \emptyset = \mathbf{tbc}(t)$, as required.

2. This directly follows from assertions 1 and 2 of the *read_term* Soundness Lemma.

3. Left-to-right implication:

   Suppose that $\mathit{def}([\![t]\!]^g_M)$. Now by the definition of $\mathit{def}([\![t]\!]^g_M)$, $B^M(\frac{M}{g}(t))$, which by the Sort Disjointness Axiom of *CMTN* implies that $\frac{M}{g}(t) \neq u^M$. By assertion 3 of the *read_term* Soundness Lemma and using the fact that $\mathbb{T}' - \mathbb{T} = \emptyset$, $\mathit{pres}(\Gamma', \Gamma, \mathbb{T}' - \mathbb{T}, M, S, g, \Phi)$ holds for all presuppositionally marked Φ in $\Gamma^* - \Gamma$. Now we still have to show that $\mathit{pres}(\Gamma', \Gamma, \mathbb{T}' - \mathbb{T}, M, S, g, B(T))$. So let $S' \succeq S$ be a Γ'-skolem-assignment such that

$M + S', g \models \Gamma^*$. Now by assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(t) = \frac{M+S'}{g}(T)$, i.e. $B^M(\frac{M+S'}{g}(T))$, i.e. $M + S', g \models B(T)$, as required.

Right-to-left implication:

Suppose that for all presuppositionally marked premises Φ in $\Gamma' - \Gamma$, $pres(\Gamma', \Gamma, \mathbb{T}' - \mathbb{T}, M, S, g, \Phi)$. Note that by the definition of *read_term*, $\Gamma^* - \Gamma$ contains only presuppositionally marked premises. This allows us to conclude that there is a Γ^*-skolem-assignment $S_0 \succeq S$ such that $M+S_0, g \models \Gamma^*-\Gamma$. Now since $pres(\Gamma', \Gamma, \mathbb{T}'-\mathbb{T}, M, S, g, B(T))$, $M+S_0, g \models B(T)$, i.e. $B^M(\frac{M+S_0}{g}(T))$. Note that by assertion 3 of the *read_term* Soundness Lemma, $\frac{M}{g}(t) \neq u^M$. Now by assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(t) = \frac{M+S_0}{g}(T)$, i.e. $B^M(\frac{M}{g}(t))$, i.e. $def([\![t]\!]^g_M)$, as required.

4. Suppose that $def([\![t]\!]^g_M)$, and let k be an M-assignment.

   (a) ⇒ (b):

   Suppose $k \in [\![t]\!]^g_M$. Then $k = g$, i.e. $k[\mathbb{T}' - \mathbb{T}]g$. Now it is enough to show that g verifies $\Gamma' - \Gamma$ over $M + S$. For this suppose that $S' \succeq S$ is a Γ^*-skolem-assignment such that $M + S', g \models \Gamma^* - \Gamma$. We have to show that $M + S', g \models T = \top$.

   Since $[\![t]\!]^g_M \neq \emptyset$, $\frac{M}{g}(t) = \top^M$. But by assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(t) = \frac{M+S'}{g}(T)$, so $M + S', g \models T = \top$, as required.

   (b) ⇒ (c):

   This implication from (b) to (c) actually does not depend on which of the 13 cases of this proof we are in. So we only present the proof for this implication once here, and leave it out in all later cases.

   Suppose that $k[\mathbb{T}' - \mathbb{T}]g$ and that k verifies $\Gamma' - \Gamma$ over $M + S$. We prove inductively that for every initial segment Γ_0 of Γ', there is a Γ_0-skolem-assignment $S' \succeq S$ such that $M + S', k \models \Gamma_0$.

   For $\Gamma_0 = \Gamma$, we can deduce $M + S, k \models \Gamma$ from the fact that $M + S, g \models \Gamma$ and the fact that no free term in Γ is in $\mathbb{T}' - \mathbb{T}$ (which intuitively follows from the fact that the proof checking algorithm only keeps track of terms that have occurred in the already processed parts of a *PTL* text, and which can be proved formally from the fact that $\mathbf{tbc}(\theta_0) = \mathbb{T}' - \mathbb{T}$, from our basic assumptions [i] and [iii] and from the definition of **tbc**).

   For the inductive step, suppose that Φ is a premise in Γ and that $S' \succeq S$ is a Γ_Φ-skolem-assignment such that $M+S', k \models \Gamma_\Phi$. It now suffices to show that there is a $\Gamma_{\Phi+}$-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Phi$. If Φ is not presuppositionally marked, then $M + S', k \models \Phi$ follows directly from the fact that k verifies $\Gamma' - \Gamma$ over $M + S$. So suppose Φ is presuppositionally marked. Since $def([\![\theta_0]\!]^g_M)$, the already proved assertion 3 of the Detailed Soundness Lemma for θ_0 implies that $pres(\Gamma', \Gamma, \mathbb{T}'-\mathbb{T}, M, S, g, \Phi)$. This now implies the required result that there is a $\Gamma_{\Phi+}$-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Phi$.

   (c) ⇒ (a):

   Suppose that $k[\mathbb{T}' - \mathbb{T}]g$ and that there is a Γ'-skolem-assignment $S' \succeq S$ such that $M + S', k \models \Gamma'$. Since $\mathbb{T}' - \mathbb{T} = \emptyset$, $k = g$. Then $M + S', k \models$

$T = \top$, i.e. $\frac{M+S'}{k}(T) = \top^M$. Since $def(\llbracket t \rrbracket^g_M)$, $\frac{M}{k}(t) \neq u^M$, so by assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{k}(t) = \frac{M+S'}{k}(T) = \top^M$. Now by the definition of $\llbracket t \rrbracket^g_M$, $k = g \in \llbracket t \rrbracket^g_M$, as required.

5. It is enough to show that if $\nu \neq u$, then $\mu + v(t, M, g) \neq u$, and that if $\nu = \top$, then $\mu + v(t, M, g) = \top$.

   So suppose $\nu \neq u$. By (iii), $\mu_1 \neq u$, i.e. by (ii), $\mu_0 \neq u$ and $CMTN \cup \Gamma^* \models B(T)$, i.e. $pres(\Gamma', \Gamma, \mathbb{T}, M, S, g, B(T))$. Then by assertion 5 of the *read_term* Soundness Lemma, $\mu \neq u$ and $\frac{M}{g}(t) \neq u$. So by the above established assertion 3 of this lemma, $v(t, M, g) \neq u$. Hence $\mu + v(t, M, g) \neq u$.

   Now suppose $\nu = \top$. Then by (iii), $\mu_1 = \top$ and $CMTN \cup \Gamma^* \oplus \langle B(T) \rangle \models T = \top$. So by (ii), $\mu_0 = \top$. Now by assertion 5 of the *read_term* Soundness Lemma, $\mu = \top$ and $\frac{M}{g}(t) \neq u$, i.e. $def(\llbracket t \rrbracket^g_M)$. In order to conclude that $\mu + v(t, M, g) = \top$, it is now enough to show that $v(t, M, g) = \top$, i.e. that $\llbracket t \rrbracket^g_M \neq \emptyset$. By the above established assertion 4 of this lemma, it is therefore enough to show that g verifies $\Gamma' - \Gamma$ over $M + S$. So let $S' \succeq S$ be a $\Gamma^* \oplus \langle B(T) \rangle$-skolem-assignment such that $M + S', g \models \Gamma^* \oplus \langle B(T) \rangle$. Since $CMTN \cup \Gamma^* \oplus \langle B(T) \rangle \models T = \top$, $M + S', g \models T = \top$, as required.

6. Trivial (since $\mathbb{T}' = \mathbb{T}$).

Case 2: θ_0 is of the form $R(t_1, \ldots, t_n)$

In this case there are premise lists $\Gamma_1, \ldots, \Gamma_{n+1}$, term lists $\mathbb{T}_1, \ldots, \mathbb{T}_{n+1}$ and proof status values $\mu_1, \ldots, \mu_{n+1}$ such that the following properties hold:

(i) $\Gamma_1 = \Gamma$.

(ii) $\mathbb{T}_1 = \mathbb{T}$.

(iii) $\mu_1 = \mu$.

(iv) for all $1 \leq i \leq n$, $read\_term(t_i, \Gamma_i, \mathbb{T}_i, \mu_i) = (\Gamma_{i+1}, T_i, \mu_{i+1})$.

(v) $\Gamma' = \Gamma_{n+1} \oplus \langle R(T_1, \ldots, T_n) \rangle$.

(vi) $\mathbb{T}' = \mathbb{T}_{n+1}$.

(vii) $\nu = update(\mu_{n+1}, 1, P(\Gamma_{n+1} \vdash^? R(T_1, \ldots, T_n)))$.

Now we prove each of the six assertions of the Detailed Soundness Lemma for $R(t_1, \ldots, t_n)$:

1. By the definition of *read_term* and (iv), $PL^{-1}(\mathbb{T}' - \mathbb{T}) = \emptyset = \mathbf{tbc}(R(t_1, \ldots, t_n))$.

2. This follows directly from assertion 2 of the *read_term* Soundness Lemma applied to all applications of *read_term* in (iv).

3. This follows from the fact that $def(\llbracket R(t_1, \ldots, t_n) \rrbracket^g_M)$ iff for all $1 \leq i \leq n$, $\frac{M}{g}(t_i) \neq u^M$, and from assertion 3 of the *read_term* Soundness Lemma applied to all applications of *read_term* in (iv).

4. Assume $def([\![R(t_1,\ldots,t_n)]\!]^g_M)$. Then for all $1 \leq i \leq n$, $\frac{M}{g}(t_i) \neq u^M$.

   (a) $\Rightarrow$ (b):

   Assume $k \in [\![R(t_1,\ldots,t_n)]\!]^g_M$. Then $k = g$, i.e. $k[\mathbb{T} - \mathbb{T}]g$. It remains to be shown that k verifies $\Gamma' - \Gamma$ over $M + S$. For this, let $S' \succeq S$ be a Γ_{n+1}-skolem-assignment such that $M + S', g \models \Gamma_{n+1}$. Now we only need to show that $M + S', g \models R(T_1,\ldots,T_n)$.

   Since $g \in [\![R(t_1,\ldots,t_n)]\!]^g_M$, $(\frac{M}{g}(t_1),\ldots,\frac{M}{g}(t_n)) \in R^M$. By assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(t_i) = \frac{M+S'}{g}(T_i)$ for $1 \leq i \leq n$. Now $(\frac{M+S'}{g}(T_1),\ldots,\frac{M+S'}{g}(T_n)) \in R^M$, i.e. $M + S', g \models R(T_1,\ldots,T_n)$.

   (b) $\Rightarrow$ (c):

   As in case 1.

   (c) $\Rightarrow$ (a):

   Let $S' \succeq S$ be a Γ'-skolem-assignment such that $M + S', g \models \Gamma_{n+1} \oplus \langle R(T_1,\ldots,T_n)\rangle$. By assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(t_i) = \frac{M+S'}{g}(T_i)$ for $1 \leq i \leq n$, so $(\frac{M}{g}(t_1),\ldots,\frac{M}{g}(t_n)) \in R^M$, i.e. $g \in [\![R(t_1,\ldots,t_n)]\!]^g_M$, as required.

5. Suppose $\nu \neq u$. Then by (vii), $\mu_{n+1} \neq u$. So by assertion 5 of the *read_term* Soundness Lemma, $\mu_n = \cdots = \mu_1 \neq u$ and for all $1 \leq i \leq n$, $\frac{M}{g}(t_i) \neq u^M$. This implies that $\mu = \mu_1 \neq u$ and that $def([\![R(t_1,\ldots,t_n)]\!]^g_M)$, i.e. $\mu + v(R(t_1,\ldots,t_n), M, g) \neq u$.

   Now suppose $\nu = \top$. Then by (vii), $\mu_{n+1} = \top$ and $CMTN \cup \Gamma_{n+1} \models R(T_1,\ldots,T_n)$. So by assertion 5 of the *read_term* Soundness Lemma, $\mu_n = \cdots = \mu_1 = \top$ and for all $1 \leq i \leq n$, $\frac{M}{g}(t_i) = \top$. Now it follows that $\mu = \mu_1 = \top$ and $def([\![R(t_1,\ldots,t_n)]\!]^g_M)$. In order to conclude that $\mu + v(R(t_1,\ldots,t_n), M, g) = \top$, it is now enough to show that $v(R(t_1,\ldots,t_n), M, g) = \top$, i.e. that $[\![R(t_1,\ldots,t_n)]\!]^g_M \neq \emptyset$. By the above established assertion 4 of this lemma, it is therefore enough to show that g verifies $\Gamma' - \Gamma$ over $M + S$. So let $S' \succeq S$ be a Γ_{n+1}-skolem-assignment such that $M + S', g \models \Gamma_{n+1}$. Since $CMTN \cup \Gamma_{n+1} \models R(T_1,\ldots,T_n)$, $M + S', g \models R(T_1,\ldots,T_n)$, as required.

6. Trivial.

Case 3: θ_0 is of the form $\neg\varphi$

In this case there are Γ^*, $\mathbb{T}_0$, Φ and μ' such that the following hold:

(i) $read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma^*, \mathbb{T}_0, \Phi, \mu')$.

(ii) $\nu = update(\mu', 1, P(\Gamma^* \vdash^? \neg\exists_{\mathbb{T}_0} \Phi))$.

(iii) $\Gamma' = \Gamma^* \oplus \langle\neg\exists_{\mathbb{T}_0} \Phi\rangle$.

(iv) $\mathbb{T}' = \mathbb{T}$.

Now by the *read_text* Soundness Lemma, the following properties hold:

I. All MHF terms of Φ are composed of PTL_{sk} symbols and terms in $\mathbb{T} \cup \mathbb{T}_0$.

II. All MHF terms in $\Gamma^* - \Gamma$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$.

III. Either $\mu' = u$ or $\mu' = \mu$ and $def([\![\varphi]\!]^g_M)$.

IV. $def([\![\varphi]\!]^g_M)$ iff for every Ψ in $\Gamma^* - \Gamma$ and every Γ^*_Ψ-skolem-assignment $S' \succeq S$ such that $M + S', g \models \Gamma^*_\Psi$, there is a Γ^*-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, g \models \Gamma^*$.

V. If $def([\![\varphi]\!]^g_M)$, then the following three properties are equivalent:

(a) $k \in [\![\varphi]\!]^g_M$.

(b) $k[\mathbb{T}_0]g$ and for every Γ^*-skolem-assignment S' extending S such that $M + S', g \models \Gamma^*$, $M + S', k \models \Phi$.

(c) $k[\mathbb{T}_0]g$ and there is a Γ^*-skolem-assignment S' extending S such that $M + S', k \models \Gamma^* \oplus \langle\Phi\rangle$.

Now we prove the six assertions of the Detailed Soundness Lemma for $\neg\varphi$:

1. $\mathbf{tbc}(\neg\varphi) = \emptyset = PL^{-1}(\mathbb{T} - \mathbb{T})$ as required.

2. It easily follows from I and II that all MHF terms in $\Gamma' - \Gamma$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$, as required.

3. $def([\![\neg\varphi]\!]^g_M)$ iff $def([\![\varphi]\!]^g_M)$

   iff for every Ψ in $\Gamma^* - \Gamma$ and every Γ^*_Ψ-skolem-assignment $S' \succeq S$ such that $M + S', g \models \Gamma^*_\Psi$, there is a Γ^*-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, g \models \Gamma^*$ (by assertion IV)

   iff for every presuppositionally marked premise Ψ in $\Gamma' - \Gamma$, every Γ'_Φ-skolem-assignment $S' \succeq S$ and every $k[\mathbb{T} - \mathbb{T}]g$ such that $M + S', k \models \Gamma'_\Psi$, there is a Γ^*-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, g \models \Gamma^*$ (since $\neg\exists_{\mathbb{T}}\ \Phi$ is not presuppositionally marked in Γ'), as required.

4. Assume $def([\![\neg\varphi]\!]^g_M)$. Then $def([\![\varphi]\!]^g_M)$.

   (a) $\Leftrightarrow$ (c):

   $k \in [\![\neg\varphi]\!]^g_M$ iff $k = g$ and there is no $k' \in [\![\varphi]\!]^g_M$

   iff $k = g$ and there is no $k'[\mathbb{T}_0]g$ such that for every Γ^*-skolem-assignment S' extending S such that $M + S', g \models \Gamma^*$, $M + S', k' \models \Phi$ (by the equivalence of (a) and (b) in V)

   iff $k = g$ and there is a Γ^*-skolem-assignment S' extending S such that $M + S', g \models \Gamma^*$ and such that there is no $k'[\mathbb{T}_0]g$ such that $M + S', k' \models \Phi$

   iff $k = g$ and there is a Γ'-skolem-assignment S' extending S such that $M + S', g \models \Gamma'$ (note that a Γ'-skolem-assignment is just a Γ^*-skolem-assignment, since $\neg\exists_{\mathbb{T}_0}\ \Phi$ does not introduce skolem functions), as required.

   (a) $\Leftrightarrow$ (b):

   $k \in [\![\neg\varphi]\!]^g_M$ iff $k = g$ and there is no $k' \in [\![\varphi]\!]^g_M$

   iff $k = g$ and there is no $k'[\mathbb{T}_0]g$ and no Γ^*-skolem-assignment S' extending S such that $M + S', k' \models \Gamma^* \oplus \langle\Phi\rangle$ (by the equivalence of (a) and (c) in V)

iff $k = g$ and for every Γ^*_Φ-skolem-assignment $S' \succeq S$ such that $M+S', g \models \Gamma^* - \Gamma$, there is no $k'[\mathbb{T}_0]g$ such that $M + S', k' \models \Phi$ (by II and the fact that $M + S', g \models \Gamma$ for $S' \succeq S$)

iff $k = g$ and for every Γ^*_Φ-skolem-assignment $S' \succeq S$ such that $M+S', g \models \Gamma^* - \Gamma$, we have $M + S', g \models \neg\exists_{\mathbb{T}_0}\ \Phi$

iff $k = g$ and k verifies $\Gamma^* - \Gamma$ over $M + S$, as required.

5. Suppose $\nu \neq u$. Then $\mu' \neq u$, i.e. $\mu \neq u$ and $def([\![\varphi]\!]^g_M)$ by III. But then $def([\![\neg\varphi]\!]^g_M)$, i.e. $v(\neg\varphi, M, g) \neq u$, i.e. $\mu + v(\neg\varphi, M, g) \neq u$.

   Now suppose $\nu = \top$. Then $\mu' = \top$ and $P(\Gamma^* \vdash^? \neg\exists_{\mathbb{T}_0}\ \Phi) = \top$. This on the one hand implies $\mu = \top$ by III, and on the other hand implies that $CMTN \cup \Gamma^* \models \neg\exists_{\mathbb{T}_0}\ \Phi$. But then by the just proved assertion 4 of the Detailed Soundness Lemma for $\neg\varphi$, $g \in [\![\varphi]\!]^g_M$, i.e. $v(\neg\varphi, M, g) = \top$, i.e. $\mu + v(\neg\varphi, M, g) = \top$.

6. Trivial.

Case 4: θ_0 is of the form $\theta \wedge \psi$

This case can be verified in a way very similar to case 3.

Case 5: θ_0 is of the form $\varphi \vee \psi$

This case can be verified in a way very similar to case 3.

Case 6: θ_0 is of the form $\varphi \to \theta$

In this case, there are Γ_0, Γ_1, Γ_2, Γ_{func}, $\mathbb{T}_0$, $\mathbb{T}_1$, $\mathbb{T}_2$, $\mathbb{T}^*$, $\mathbb{F}$, Φ, α, μ_0 and μ_1 such that the following hold:

(i) $read\_text(\varphi, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma_0, \mathbb{T}_1, \Phi, \mu_0)$.

(ii) $check\_text(\theta, \Gamma_0 \oplus \langle\Phi\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0) = (\Gamma_1, \mathbb{T}_2, \mu_1)$.

(iii) $\alpha = \begin{cases} 1 & \text{if the symbol } L \text{ does not occur in } \Gamma_1 - \Gamma_0 \text{ and for every term } T \text{ occurring in } \Gamma_1 - \Gamma_0 \text{ that is either in } \mathbb{T} \text{ or a skolem function symbol, } check\_limitedness(\Gamma_1, \Gamma_1 - \Gamma_0, \mathbb{T}, T) \\ 0 & \text{otherwise.} \end{cases}$

(iv) $make\_functions(\mathbb{T}_1, \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1), \Gamma_0, \Gamma_1, \Phi, \alpha, \mu_1) = (\mathbb{F}, \mathbb{T}^*, \Gamma_{func}, \Gamma^-_{pres}, \nu)$.

(v) $pull\_out\_pres(\langle\rangle, \mathbb{T}_2 - \mathbb{T}, \Gamma_0, \Gamma_1) = (\Gamma^*, \langle\_\rangle \oplus \Gamma_2, \_)$.

(vi) $\Theta = \exists_{\mathbb{T}^*} \bigwedge \Gamma_2$.

(vii) $\Gamma_{pres} = \Gamma^-_{pres} \oplus \langle\forall_{\mathbb{T}_2 - \mathbb{T}}\ (\Phi \wedge \Theta \to L(T))^{\mathbb{P}} \mid T \in \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1) - \mathbb{T}^*\rangle$.

(viii) $\Gamma' = \Gamma^* \oplus \Gamma_{pres} \oplus \langle\forall_{\mathbb{T}_1}\ (\Phi \to \Theta)\rangle \oplus \Gamma_{func}$.

(ix) $\mathbb{T}' = \mathbb{T} \oplus \mathbb{F}$.

Note that the assertions (ii), (v) and (vi) closely resemble the definition of *read_text*. Using arguments analogous to those in the proof of the *read_text* Soundness Lemma, we can derive from them the following assertions:

I. $\mathbf{tbc}(\theta) = \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$.

II. All MHF terms of $\bigwedge \Gamma_2$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

III. All MHF terms in $\Gamma^* - \Gamma_0$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$.

IV. Suppose that $\mathit{def}([\![\varphi]\!]^g_M)$. Then the following two properties are equivalent:

(a) For all $h \in [\![\varphi]\!]^g_M$, $\mathit{def}([\![\theta]\!]^h_M)$.

(b) For every Ψ in $\Gamma^* - \Gamma_0$ and every Γ^*_Ψ-skolem-assignment $S' \succeq S$ such that $M + S', h \models \Gamma^*_\Psi$, there is a Γ^*-skolem-assignment $S_0 \succeq S'$ such that $M + S_0, h \models \Gamma^*$.

V. Suppose that $h \in [\![\varphi]\!]^g_M$, that $\mathit{def}([\![\theta]\!]^h_M)$ and that $S' \succeq S$ is a Γ_0-skolem-assignment such that $M + S', h \models \Gamma_0 \oplus \langle\Phi\rangle$. Then the following three properties are equivalent:

(a) $k \in [\![\theta]\!]^h_M$.

(b) $k[\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)]h$ and for every Γ^*-skolem-assignment S'' extending S' such that $M + S'', h \models \Gamma^*$, $M + S'', k \models \Gamma_2$.

(c) $k[\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)]h$ and there is a Γ^*-skolem-assignment S'' extending S' such that $M + S'', k \models \Gamma^* \oplus \Gamma_2$.

Now we prove the six assertions of the Detailed Soundness Lemma for $\varphi \to \theta$:

1. By the definition of *make_function*,

$$\begin{aligned} PL^{-1}(\mathbb{T}' - \mathbb{T}) &= PL^{-1}(\mathbb{F}) \\ &= PL^{-1}(\{T' \mid \text{for some } T \text{ in } \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1), \text{ there is a } \mathit{length}(\mathbb{T}_1)\text{-place argument filler } \sigma \text{ such that } T = T'^{\sigma}(\mathbb{T}_1)\}) \\ &= \mathbf{tbc}(\varphi \to \theta) \end{aligned}$$

by the definition of **aq** and **tbc** and the facts that $\mathbf{tbc}(\varphi) = PL^{-1}(\mathbb{T}_1)$ (assertion 1 of the *read_text* Soundness Lemma) and that $\mathbf{tbc}(\theta) = PL^{-1}(\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1))$ (assertion 1 of the Detailed Soundness Lemma).

2. Let T be an MHF term in $\Gamma' - \Gamma$. We have to show that T is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{F}$. For this we distinguish four cases:

Case 1: T is an MHF term in $\Gamma_0 - \Gamma$.

In this case, T is composed of PTL_{sk} symbols and terms in $\mathbb{T}$ by assertion 3 of the *read_text* Soundness Lemma.

Case 2: T is an MHF term in $\Gamma^* - \Gamma_0$.

In this case, T is composed of PTL_{sk} symbols and terms in $\mathbb{T}$ by assertion III above.

Case 3: T is an MHF term in Γ_{pres}.

This case is similar to case 4 below.

Case 4: T is an MHF term in $\forall_{\mathbb{T}_1}\ (\Phi \rightarrow \exists_{\mathbb{T}^*}\ \bigwedge \Gamma^*)$.

Note that by Definition 6.1.8, $\forall_{\mathbb{T}_1}\ (\Phi \rightarrow \exists_{\mathbb{T}^*}\ \bigwedge \Gamma^*)$ is actually of the form $\forall \vec{x}\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow \exists \vec{y}\ \bigwedge \Gamma^* \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1})$. Further, note that by assertion 2 of the *read_text* Soundness Lemma, all MHF terms of Φ are composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{T}_1$. Additionally, note that by assertion 2 of the Detailed Soundness Lemma, all MHF terms of Γ^* are composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

First assume that T occurs in $\Phi \frac{\vec{x}}{\mathbb{T}_1}$. Then T is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \vec{x}$. But if T contains variables from $\vec{x}$, then T is not hereditarily free in $\forall \vec{x}\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow \exists \vec{y}\ \bigwedge \Gamma^* \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1})$, contrary to our assumption. So T is composed of PTL_{sk} symbols and terms in $\mathbb{T}$, as required.

Now assume that T occurs in $\Gamma^* \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1}$. Then there is a term T_1 such that $T = T_1 \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1}$ and such that T_1 occurs in Γ^* at a position corresponding to the position of T in $\Gamma^* \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1}$. T_1 is composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

Let T_0 be a term from $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$ used for composing T_1. If T_0 is in $\mathbb{T}^*$, then T contains a variable from $\vec{y}$, contradicting our assumption about T. So T_0 is not in $\mathbb{T}^*$. Now the definition of $\mathbb{T}^*$ in the definition of *make_function* implies that T_0 is of the form $T'^{\sigma}(\mathbb{T}_1)$ for some *length*$(\mathbb{T}_1)$-place argument filler σ, where T' is in $\mathbb{F}$. So T_0 is composed of terms of terms in $\mathbb{F} \oplus \mathbb{T}_1$.

We can now conclude that T_1 is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{T}_1 \oplus \mathbb{F}$, i.e. that T is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{F} \oplus \vec{x}$. But if T contains variables from $\vec{x}$, then T is not hereditarily free in $\forall \vec{x}\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow \exists \vec{y}\ \bigwedge \Gamma^* \frac{\vec{y}}{\mathbb{T}^*} \frac{\vec{x}}{\mathbb{T}_1})$, contrary to our assumption. So T is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{F}$, as required.

Case 5: T is an MHF term in Γ_{func}.

Then by the definition of *make_function*, there is a term T^* in $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$, a term T' in $\mathbb{F}$, a *length*$(\mathbb{T}_1)$-place argument filler σ such that $T^* = T'^{\sigma}(\mathbb{T}_1)$, a natural number n and a function-head subterm F of T^* such that F contains T' as a proper subterm and such that T is an MHF term in $\forall_{\mathbb{T}_1} M(F, n)$, in $\forall_{\mathbb{T}_1}(\Phi \rightarrow F \neq u)$, in $L(T')$ or in $\forall_{\mathbb{T}_1}(\Phi \rightarrow L(F))$. By Definition 6.1.8, the first, second and fourth formulae are actually of the form $\forall \vec{x} \neq u\ M(F \frac{\vec{x}}{\mathbb{T}_1}, n)$, $\forall \vec{x} \neq u\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow F \frac{\vec{x}}{\mathbb{T}_1} \neq u)$ and $\forall \vec{x} \neq u\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow L(F \frac{\vec{x}}{\mathbb{T}_1}))$. Now there are four subcases:

Case 5a: T is an MHF term in $\Phi \frac{\vec{x}}{\mathbb{T}_1}$. Then by assertion 2 of the *read_text* Soundness Lemma, T is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \vec{x}$. But if T contains variables from $\vec{x}$, then T is not hereditarily free in $\forall \vec{x} \neq u\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow F \frac{\vec{x}}{\mathbb{T}_1} \neq u)$ or $\forall \vec{x} \neq u\ (\Phi \frac{\vec{x}}{\mathbb{T}_1} \rightarrow L(F \frac{\vec{x}}{\mathbb{T}_1}))$, contrary to our assumption. So T is composed of PTL_{sk} symbols and terms in $\mathbb{T}$, as required.

Case 5b: T is an MHF term in $F \frac{\vec{x}}{\mathbb{T}_1}$. Then T is composed of T' and terms in $\vec{x}$. By the same argument as above, T cannot contain terms in $\vec{x}$, so $T = T'$, i.e. T is in $\mathbb{F}$, thus satisfying the required property that it is composed of PTL_{sk} symbols and terms in $\mathbb{T} \oplus \mathbb{F}$.

Case 5c: T is u. Then T is a PTL_{sk} symbol and thus also satisfies the required property.

Case 5d: T is an MHF term in $L(T')$. Then T is T' and thus again satisfies the required property.

3. Left-to-right implication:

Suppose $def([\![\varphi \to \theta]\!]^g_M)$. Then by Definition 5.2.2, $def([\![\varphi]\!]^g_M)$ and for all $h \in [\![\varphi]\!]^g_M$, we have that $def([\![\theta]\!]^h_M)$ and that for every $k \in [\![\theta]\!]^h_M$,

$$\begin{array}{l}\text{if there is a } t \in dom(k) \setminus dom(h) \text{ of the form } f^\sigma(t_1, \ldots, t_n),\\ \text{where } \{t_1, \ldots, t_n\} = dom(h) \setminus dom(g),\ f \text{ is a } PTL \text{ term}\\ \text{and } \sigma \text{ is an } n\text{-place argument filler, then } k(t) \in L^M \text{ and}\\ h(t_i) \in L^M \text{ for } 1 \leq i \leq m.\end{array} \tag{6.10}$$

We have to show that for all presuppositionally marked Ψ in Γ', $pres(\Gamma', \Gamma, \mathbb{F}, M, S, g, \Psi)$. For Ψ in $\Gamma_0 - \Gamma$, this follows from assertion 5 of the *read_text* Soundness Lemma. For Ψ in $\Gamma^* - \Gamma_0$, it follows from assertion IV. Γ_{func} does not contain presuppositionally marked premises. So it is now enough to show $pres(\Gamma', \Gamma, \mathbb{F}, M, S, g, \Psi)$ for all $\Psi \in \Gamma_{pres}$.

So suppose $\Gamma_{pres} \neq \emptyset$. Then by the definition of *make_functions*, there is a term T_0 in $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$ of the form $T'^\sigma(\mathbb{T}_1)$ for some term T' and some $length(\mathbb{T}_1)$-place argument filler σ, and

$$\Gamma_{pres} = \langle \forall_{\mathbb{T}_1}(\Phi \to \bigwedge_{T \in \mathbb{T}_1} L(T))^{\mathbb{P}} \rangle \oplus \langle \forall_{\mathbb{T}_2 - \mathbb{T}}\ (\Phi \wedge \Theta \to L(T))^{\mathbb{P}} \mid T \in \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1) \text{ and } T \text{ is of the form } T'^\sigma(\mathbb{T}_1) \text{ for some term } T' \text{ and some } length(\mathbb{T}_1)\text{-place argument filler } \sigma \rangle.$$

Now let $S' \succeq S$ be a Γ^*-skolem-assignment such that $M + S', g \models \Gamma^*$. It is now enough to show that $M + S', g \models \Gamma_{pres}$. So let $h[\mathbb{T}_1]g$ be such that $M + S', h \models \Phi$, and let $k[\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)]h$ be such that $M + S', k \models \Theta$. Now it suffices to show that $M + S', h \models L(T)$ for all T in $\mathbb{T}_1$, and that $M + S', k \models L(T)$ for all T in $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$ of the form $T'^\sigma(\mathbb{T}_1)$. Assume that T is in $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$ and of the form $T'^\sigma(\mathbb{T}_1)$ (remember that there is at least one T with this property, namely T_0). It is now enough to show that $M + S', k \models L(T)$ and that for all T in $\mathbb{T}_1$, $M + S', h \models L(T)$.

Assertion 6 of the *read_text* Soundness Lemma now implies that $h \in [\![\varphi]\!]^g_M$, and assertion V implies that $k \in [\![\theta]\!]^h_M$. Since T is in $\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$, T is in $\mathbf{tbc}(\theta)$, and hence contains no skolem function symbols. So we can speak about $PL^{-1}(T)$. If we let t be $PL^{-1}(T)$, then all the assumptions of (6.10) are fulfilled, and we can conclude that $k(t) \in L^M$ and $h(t') \in L^M$ for $t' \in dom(h) \setminus dom(g)$, i.e. that $M + S', k \models L(T)$ and that for all T in $\mathbb{T}_1$, $M + S', h \models L(T)$.

Right-to-left implication:

Assume that for all presuppositionally marked premises Ψ in Γ', $pres(\Gamma', \Gamma, \mathbb{F}, M, S, g, \Psi)$. We have to show that $def([\![\varphi \to \theta]\!]^g_M)$. By assertion 5 of the *read_text* Soundness Lemma, $def([\![\varphi]\!]^g_M)$. By assertion IV, for all $h \in [\![\varphi]\!]^g_M$, $def([\![\theta]\!]^h_M)$. So it is now enough to show that for all h, k such that $h \in [\![\varphi]\!]^g_M$ and $k \in [\![\theta]\!]^h_M$, (6.10) holds. So suppose

$t \in dom(k) \setminus dom(h)$ is of the form $f^\sigma(t_1, \ldots, t_n)$, where $\{t_1, \ldots, t_n\} = dom(h) \setminus dom(g)$, f is a *PTL* term and σ is an n-place argument filler. By Lemma 6.3.22 and assertion 1 of the *read_text* Soundness Lemma, $dom(h) \setminus dom(g) = PL^{-1}(\mathbb{T}_1)$, and by Lemma 6.3.22 and assertion I, $dom(k) \setminus dom(h) = PL^{-1}(\mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1))$. Let T be $PL(t)$. Then T is of the form $T'^\sigma(\mathbb{T}_1)$, so by the definition of *make_functions*, Γ_{pres} has precisely the same form as in the left-to-right case above. Now by similar reasoning as above, it easily follows that $k(t) \in L^M$ and $h(t_i) \in L^M$ for $1 \leq i \leq m$, as required.

4. Suppose that $def([\![\varphi \to \theta]\!]^g_M)$.

   (a) $\Rightarrow$ (b):

   Assume $h \in [\![\varphi \to \theta]\!]^g_M$. Then $h[\mathbf{tbc}(\varphi \to \theta)]g$ by Lemma 6.3.22, so $h[\mathbb{T}' - \mathbb{T}]g$. Now it is enough to show that h verifies $\Gamma' - \Gamma$ over $M + S$. So let $\Psi \in |\Gamma' - \Gamma|$ and let $S' \succeq s$ be a Γ'_Ψ-skolem-assignment such that $M + S', h \models (\Gamma' - \Gamma)_\Psi$. We have to show that $M + S', h \models \Psi$. We distinguish three cases:

   Case 1: Ψ is $\forall_{\mathbb{T}_1}\ (\Phi \to \Theta)$.

   Suppose $k[\mathbb{T}_1]h$ is an M-assignment such that $M + S', k \models \Phi$. We have to show that $M + S', k \models \Theta$. Let $k' := k|_{\mathbb{T} \oplus \mathbb{T}_1}$. Then $k'[\mathbb{T}_1]g$, and by assertions 2 and 3 of the *read_text* Soundness Lemma and assertion III, $M + S', k' \models \Gamma^* \oplus \langle \Phi \rangle$. So by assertion 6 of the *read_text* Soundness Lemma, $k' \in [\![\varphi]\!]^g_M$. Write $\mathbb{T}_1 = \langle t_1, \ldots, t_m \rangle$ and $\mathbb{F} = \langle f_1, \ldots, f_n \rangle$. Now by the definition of $[\![\varphi \to \theta]\!]^g_M$, there are m-place argument fillers $\sigma_1, \ldots, \sigma_n$ and an assignment $j \in [\![\theta]\!]^{k'}_M$ such that for $1 \leq i \leq n$,

   $$j(f_i^{\sigma_i}(t_1, \ldots, t_m)) = h(f_i)(k'(t_1), \ldots, k'(t_m)). \tag{6.11}$$

   Define an M-assignment $j'[\mathbb{F}]j$ by

   $$j'(t) := \begin{cases} h(t) & \text{if } t \in \mathbb{F} \\ j(t) & \text{otherwise.} \end{cases}$$

   Now one can easily verify that $j'[\mathbb{T}^*]k$ and – using (6.11) – that $j' \in [\![\theta]\!]^{k'}_M$. Then by assertion V, $M + S', j' \models \Gamma_2$, i.e. $M + S', k \models \Theta$, as required.

   Case 2: Ψ is in Γ_{func} and of the form $\forall_{\mathbb{T}_F}\ (\exists_{\mathbb{T}_1 - \mathbb{T}_F}\ \Phi \leftrightarrow F \neq u)$.

   Then there is a $T \in \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1)$ of the form $T'^\sigma(\mathbb{T}_1)$ such that F is a function-head subterm of T containing T' as a proper subterm, and such that $\mathbb{T}_F = \langle T_0 \in \mathbb{T} \mid T_0 \text{ occurs in } F_i \rangle$.

   First we show that $M + S', h \models \forall_{\mathbb{T}_1}\ (\Phi \to F \neq u)$, which is equivalent to $M + S', h \models \forall_{\mathbb{T}_F}\ (\exists_{\mathbb{T}_1 - \mathbb{T}_F}\ \Phi \to F \neq u)$. Suppose $k[\mathbb{T}_1]h$ is an M-assignment such that $M + S', k \models \Phi$. We have to show that $M + S', k \models F \neq u$. Suppose for a contradiction that $M + S' \models F = u$. Then by recursive application of the Undefinedness Axiom Schema of *CMTN* with $n = 0$, $M + S', k \models T = u$, i.e. $\frac{M+S'}{k}(T) = u$. Now define k', $\sigma_1, \ldots, \sigma_n$ and j as in case 1 above. Then (6.11) implies that $\frac{M+S'}{j}(T) = u$. This together with the fact that $j \in [\![\theta]\!]^{k'}_M$ contradicts assertion 6 of the Detailed Soundness Lemma applied to (ii).

Now we still have to show that $M + S', h \models \forall_{\mathbb{T}_F}\ (F \neq u \rightarrow \exists_{\mathbb{T}_1 - \mathbb{T}_F}\ \Phi)$. Let $k[\mathbb{T}_F]h$ be such that $M + S', k \models F \neq u$. We have to show that $M + S', k \models \exists_{\mathbb{T}_1 - \mathbb{T}_F}\ \Phi$. By the definition of $[\![\varphi \rightarrow \theta]\!]^g_M$, there is an m-place argument filler σ such that

$$\begin{array}{l}\text{for } a_1, \ldots, a_l \in M,\ h(T') \text{ is } \sigma\text{-defined at } a_1, \ldots, a_l \text{ iff there} \\ \text{is a } k' \in [\![\varphi]\!]^g_M \text{ such that for all } s \leq l,\ k'(t_{\sigma(s)}) = a_s.\end{array} \tag{6.12}$$

Since $\frac{M+S'}{k}(F) \neq u$, $h(T')$ is σ-defined at $k(\mathbb{T}_F)$. So by (6.12), there is a $k' \in [\![\varphi]\!]^g_M$ such that $k'(\mathbb{T}_F) = k(\mathbb{T}_F)$. Now by assertion 6 of the *read_text* Soundness Lemma, $M + S', k' \models \Phi$. Define $k''[\mathbb{T}_1 - \mathbb{T}_F]k$ by

$$k''(t) := \begin{cases} k(t) & \text{if } t \in \mathbb{F} \\ k'(t) & \text{otherwise.} \end{cases}$$

Now $M + S', k'' \models \Phi$ by assertion 2 of the *read_text* Soundness Lemma. So $M + S', k \models \exists_{\mathbb{T}_1 - \mathbb{T}_F}\ \Phi$, as required.

Case 3: Ψ is in Γ_{func} and of the form $L(T')$ or $\forall_{\mathbb{T}_1}\ (\Phi \rightarrow L(F))$.

First assume that all calls of *check_limitedness* succeeded in the first of the two possible ways. Then, using similar reasoning as in the first two cases, $M + S', h \models \Psi$ follows from *CMTN*'s Functionality Axiom Schema applied to Φ and $\bigwedge \Gamma_2$.

Now if some call of *check_limitedness* succeeded in the second possible way, we have to proceed as in the explanation of the special case of the second criterion in section 6.1.4: Φ and $\bigwedge \Gamma_2$ are transformed into formulae Φ' and $\bigwedge \Gamma'_2$ that are equivalent in $M + S', h$ to their respective originals, and *CMTN*'s Functionality Axiom Schema is applied to Φ' and $\bigwedge \Gamma'_2$ instead of Φ and $\bigwedge \Gamma_2$. (Compare the more detailed exposition of the similar situation in case 2 of the proof of the *exist_check* Soundness Lemma.)

(b) $\Rightarrow$ (c):

As in case 1.

(c) $\Rightarrow$ (a):

This case can be verified using similar reasoning as in the (a) $\Rightarrow$ (b) case.

5. Assume $\nu \neq u$. Then (iv) and the definitions of *make_functions* and *make_function* imply that $\mu_1 \neq u$, that

$$\text{if } \Gamma_{func} \neq \langle\rangle, \text{ then for all } T \in \mathbb{T}_1,\ \Gamma \oplus \langle\Phi\rangle \vdash L(T), \tag{6.13}$$

and that

$$\text{for every } T \in \mathbb{T}_2 - (\mathbb{T} \oplus \mathbb{T}_1) \text{ of the form } T'^{\sigma}(\mathbb{T}_1),\ \Gamma_1 \vdash L(T). \tag{6.14}$$

By similar reasoning as above, (6.13) and (6.14) imply that

$$\begin{array}{l}\text{for every } h \in [\![\varphi]\!]^g_M \text{ and every } k \in [\![\theta]\!]^h_M, \text{ if there is a } t \in \\ dom(k) \setminus dom(h) \text{ of the form } f^{\sigma}(\mathbb{T}_1), \text{ then } k(t) \in L^M \text{ and} \\ h(t_i) \in L^M \text{ for } 1 \leq i \leq n.\end{array} \tag{6.15}$$

From $\mu_1 \neq u$ and assertion 5 of the Detailed Soundness Lemma applied to (ii), we can derive that $\mu_0 \neq u$ and that

$$\text{for every } h[\mathbb{T}_1]g \text{ and every } S' \succeq S \text{ such that } M + S', h \models \Gamma_0 \oplus \langle \Phi \rangle,\ v(\theta, M, h) \neq u. \tag{6.16}$$

By assertion 6 of the *read_text* Soundness Lemma, (6.16) implies that

$$\text{for every } h \in [\![\varphi]\!]^g_M,\ \mathit{def}([\![\theta]\!]^h_M). \tag{6.17}$$

From $\mu_0 \neq u$ and assertion 4 of the *read_text* Soundness Lemma, we can derive that $\mu \neq u$ and that $\mathit{def}([\![\varphi]\!]^g_M)$. Now (6.15) and (6.17) imply that $\mathit{def}([\![\varphi \to \theta]\!]^g_M)$, i.e. that $\mu + v(\varphi \to \theta, M, g) \neq u$, as required.

Now assume that $\nu = \top$. Then by (iv), $\mu_1 = \top$. Then by assertion 5 of the Detailed Soundness Lemma, we can derive that $\mu_0 = \top$ and that

$$\text{for every } h[\mathbb{T}_1]g \text{ and every } S' \succeq S \text{ such that } M + S', h \models \Gamma_0 \oplus \langle \Phi \rangle,\ v(\theta, M, h) = \top. \tag{6.18}$$

By assertion 6 of the *read_text* Soundness Lemma, (6.18) implies that

$$\text{for every } h \in [\![\varphi]\!]^g_M,\ [\![\theta]\!]^h_M \neq \emptyset. \tag{6.19}$$

Now by Lemma 6.3.23, $[\![\varphi \to \theta]\!]^g_M \neq \emptyset$, i.e. $v(\varphi \to \theta, M, g) = \top$. Since $\mu_0 = \top$, assertion 4 of the *read_text* Soundness Lemma implies that $\mu = \top$, i.e. that $\mu + v(\varphi \to \theta, M, g) = \top$, as required.

6. Suppose that $\mathit{def}([\![\varphi \to \theta]\!]^g_M)$, $h \in [\![\varphi \to \theta]\!]^g_M$, $M + S', h \models \Gamma'$ and $T \in \mathbb{F}$. Then by Lemma 6.3.22, $T \in \mathit{dom}(h)$, i.e. $\frac{M+S'}{h}(T) = h(T)$. But $h(T) \neq u^M$ by the definition of *M-assignment*.

Case 7: θ_0 is of the form $\theta \,\&\, \xi$

In this case, there are Γ_1, Γ_2, $\mathbb{T}_1$, $\mathbb{T}_2$ and μ_1 such that the following hold:

(i) $\mathit{check\_text}(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}_1, \mu_1)$.

(ii) $\mathit{check\_text}(\xi, \Gamma_1, \mathbb{T}_1, \mu_1) = (\Gamma_2, \mathbb{T}_2, \nu)$.

(iii) $\Gamma' = \Gamma_2$.

(iv) $\mathbb{T}' = \mathbb{T}_2$.

Now we apply the fact that the lemma holds for θ and ξ. We first concentrate on the first two assertions of the lemma in each case:

1.1. $\mathbf{tbc}(\theta) = PL^{-1}(\mathbb{T}_1 - \mathbb{T})$.

1.2. All MHF terms in $\Gamma_1 - \Gamma$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}_1$.

2.1. $\mathbf{tbc}(\xi) = PL^{-1}(\mathbb{T}_2 - \mathbb{T}_1)$.

2.2. All MHF terms in $\Gamma_2 - \Gamma_1$ are composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

Now we establish the first two assertions of the lemma for $\theta \,\&\, \xi$:

1. $\mathbf{tbc}(\theta \,\&\, \xi) = \mathbf{tbc}(\theta) \oplus \mathbf{tbc}(\xi) = PL^{-1}(\mathbb{T}_2 - \mathbb{T})$ by Lemma 5.2.17.

2. Let T be an MHF term in $\Gamma_2 - \Gamma$. There are two cases:

   Case 1: $T \in \Gamma_1 - \Gamma$. Then by 1.2, T is composed of PTL_{sk} symbols and terms in $\mathbb{T}_1$.

   Case 2: $T \in \Gamma_2 - \Gamma_1$. Then by 2.2, T is composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

   In both cases, T is composed of PTL_{sk} symbols and terms in $\mathbb{T}_2$.

The remaining four assertions of the lemma applied to θ are as follows:

1.3. $\mathit{def}(\llbracket\theta\rrbracket^g_M)$ iff for all presuppositionally marked premises Φ in $\Gamma_1 - \Gamma$, $\mathit{pres}(\Gamma_1, \Gamma, \mathbb{T}_1 - \mathbb{T}, M, S, g, \Phi)$.

1.4. If $\mathit{def}(\llbracket\theta\rrbracket^g_M)$, then for all M-assignments k, the following three properties are equivalent:

   (a) $k \in \llbracket\theta\rrbracket^g_M$.

   (b) $k[\mathbb{T}_1 - \mathbb{T}]g$ and k verifies $\Gamma_1 - \Gamma$ over $M + S$.

   (c) $k[\mathbb{T}_1 - \mathbb{T}]g$ and there is a Γ_1-skolem-assignment $S' \succeq S$ such that $M + S', k \models \Gamma_1$.

1.5. $\mu + v(\theta, M, g) \geq \mu_1$.

1.6. If $\mathit{def}(\llbracket\theta\rrbracket^g_M)$ and $k \in \llbracket\theta\rrbracket^g_M$, then for every Γ_1-skolem-assignment S' extending S such that $M+S', k \models \Gamma_1$ and for every $T \in \mathbb{T}_1$, $\frac{M+S'}{k}(T) \neq u^M$.

Additionally, if g' is an M-assignment and S' is a Γ_1-skolem-assignment such that $M + S', g' \models \Gamma_1$, $\mathit{dom}(g') = \mathbb{T}_1$ and for all $T \in \mathbb{T}_1$, $\frac{M+S'}{g'}(T) \neq u^M$, then by the lemma applied to ξ the following four properties hold:

2.3. $\mathit{def}(\llbracket\xi\rrbracket^{g'}_M)$ iff for all presuppositionally marked premises Φ in $\Gamma_2 - \Gamma_1$, $\mathit{pres}(\Gamma_2, \Gamma_1, \mathbb{T}_2 - \mathbb{T}_1, M, S', g', \Phi)$.

2.4. If $\mathit{def}(\llbracket\xi\rrbracket^g_M)$, then for all M-assignments k', the following three properties are equivalent:

   (a) $k' \in \llbracket\theta\rrbracket^{g'}_M$.

   (b) $k'[\mathbb{T}_2 - \mathbb{T}_1]g'$ and k' verifies $\Gamma_2 - \Gamma_1$ over $M + S'$.

   (c) $k'[\mathbb{T}_2 - \mathbb{T}_1]g'$ and there is a Γ_2-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Gamma_2$

2.5. $\mu_1 + v(\xi, M, g') \geq \nu$.

2.6. If $\mathit{def}(\llbracket\xi\rrbracket^{g'}_M)$ and $k' \in \llbracket\theta\rrbracket^{g'}_M$, then for every Γ_2-skolem-assignment S'' extending S' such that $M+S'', k' \models \Gamma_2$ and for every $T \in \mathbb{T}_2$, $\frac{M+S''}{k'}(T) \neq u^M$.

Note that by Lemma 6.3.22, 1.1, 1.4 and 1.6, for every $g' \in [\![\theta]\!]^g_M$, there is a Γ_1-skolem-assignment S' such that the requirements for concluding 2.3 up to 2.6 are fulfilled.

In what follows, we will several times make use of the following fact:

$$\text{If } M' \text{ is a } \mathit{CMTN} \text{ model, } S' \text{ is a } \Gamma_1 - \Gamma\text{-skolem-assignment and } k \text{ is an } M\text{-assignment such that } \mathit{dom}(k) \subseteq \mathbb{T}_1 \text{ and } M + S, k \models \Gamma_1 - \Gamma, \text{ then for every } M\text{-assignment } k'[\mathbb{T}_2 - \mathbb{T}_1]k, \text{ we have } M + S, k' \models \Gamma_1 - \Gamma. \tag{6.20}$$

One can easily see that (6.20) follows from the following syntactic fact:

$$\text{Every hereditarily free term in } \Gamma_1 - \Gamma \text{ that is in } \mathbb{T}_2 - \mathbb{T}_1 \text{ is a subterm of a term in } \mathbb{T}_1. \tag{6.21}$$

Intuitively (6.21) follows from the semi-niceness of $\theta \,\&\, \xi$ and the fact that the proof checking algorithm only keeps track of terms that have occurred in the already processed parts of a *PTL* text. Here is how (6.21) can be established formally: By 1.1, 2.1 and Lemma 5.2.19, $PL^{-1}(\mathbb{T}_2 - \mathbb{T})$ is pairwise independent. Together with the pairwise independence of $PL^{-1}(\mathbb{T}) \oplus \mathbf{qt}(\theta \& \xi)$ and the fact that $\mathbf{tbc}(\theta \& \xi) = PL^{-1}(\mathbb{T}_2 - \mathbb{T})$, this implies that $PL^{-1}(\mathbb{T}_2)$ is pairwise independent. Now let T be a hereditarily free term in $\Gamma_1 - \Gamma$ that is in $\mathbb{T}_2 - \mathbb{T}_1$. There there is an MHF term T' such that T is a subterm of T'. By 1.2, T' is composed of PTL_{sk} symbols and terms in $\mathbb{T}_1$. Suppose for a contradiction that T is not a subterm of a term in $\mathbb{T}_1$. Then T is composed of PTL_{sk} symbols and terms in $\mathbb{T}_1$, and is hence not independent of all terms in $PL^{-1}(\mathbb{T}_1)$, contradicting the pairwise independence of $PL^{-1}(\mathbb{T}_2)$.

We now need to prove the remaining four assertion of the lemma for $\theta \,\&\, \xi$:

3. Left-to-right implication:

   Assume $\mathit{def}([\![\theta \,\&\, \xi]\!]^g_M)$. Then $\mathit{def}([\![\theta]\!]^g_M)$ and for all $h \in [\![\theta]\!]^g_M$, $\mathit{def}([\![\xi]\!]^h_M)$. We have to show that for all presuppositionally marked $\Phi \in \Gamma_2 - \Gamma$, $\mathit{pres}(\Gamma_2, \Gamma, \mathbb{T}_2 - \mathbb{T}, M, S, g, \Phi)$.

   For a presuppositionally marked Φ in $\Gamma_1 - \Gamma$, $\mathit{pres}(\Gamma_2, \Gamma, \mathbb{T}_2 - \mathbb{T}, M, S, g, \Phi)$ follows from 1.3 and (6.20).

   Let $\Phi \in \Gamma_2 - \Gamma_1$, let S' be a $(\Gamma_2)_\Phi$-skolem-assignment and let $k[\mathbb{T}_2 - \mathbb{T}]g$ be an M-assignment such that $M + S', k \models (\Gamma_2)_\Phi - \Gamma$. Now we have to show that there is a $(\Gamma_2)_{\Phi+}$-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Phi$. Define an M-assignment k' by

$$k'(t) := \begin{cases} g(t) & \text{if } t \in \mathbb{T}_2 - \mathbb{T}_1 \\ k(t) & \text{otherwise.} \end{cases}$$

   Then $k[\mathbb{T}_2 - \mathbb{T}_1]k'$ and $k'[\mathbb{T}_1 - \mathbb{T}]g$. By (6.20), we can conclude that $M + S', k' \models \Gamma_1 - \Gamma$. By our basic assumption [iii], $M + S', k' \models \Gamma$, so by [1.6], we have that for every $T \in \mathbb{T}_1$, $\frac{M+S'}{k'}(T) \neq u^M$. This means that 2.3 up to 2.6 hold for k' in place of g'. Now by 1.4, $k' \in [\![\theta]\!]^g_M$, i.e. $\mathit{def}([\![\xi]\!]^{k'}_M)$. So by 2.3, there is a $(\Gamma_2)_\Phi$-skolem-assignment $S'' \succeq S'$ such that $M + S'', k \models \Phi$.

Right-to-left implication:

Assume that for all presuppositionally marked Φ in $\Gamma_2 - \Gamma$,

$$pres(\Gamma_2, \Gamma, \mathbb{T}_2 - \mathbb{T}, M, S, g, \Phi). \tag{6.22}$$

We need to show $def([\![\theta \,\&\, \xi]\!]^g_M)$, i.e. $def([\![\theta]\!]^g_M)$ and for all $h \in [\![\theta]\!]^g_M$, $def([\![\xi]\!]^h_M)$. $def([\![\theta]\!]^g_M)$ now follows from 1.3. Let $h \in [\![\theta]\!]^g_M$. Now by 1.4, there is a Γ_1-skolem-assignment $S' \succeq S$ such that $M + S', h \models \Gamma_1$. Now by 2.3 it is enough to show that for all Φ in $\Gamma_2 - \Gamma_1$, we have $pres(\Gamma_2, \Gamma_1, \mathbb{T}_2 - \mathbb{T}, M, S', h, \Phi)$.

Let Φ be in $\Gamma_2 - \Gamma_1$, let $S'' \succeq S'$ be a $(\Gamma_2)_\Phi$-skolem-assignment, let $k[\mathbb{T}_2 - \mathbb{T}]h$ be an M-assignment such that $M + S'', k \models (\Gamma_2)_\Phi - \Gamma_1$. By 1.4, $h[\mathbb{T}_1 - \mathbb{T}]g$, i.e. $k[\mathbb{T}_2 - \mathbb{T}]g$. By (6.20), $M + S', k \models \Gamma_1$, i.e. $M + S'', k \models \Gamma_1$, i.e. $M + S'', k \models (\Gamma_2)_\Phi - \Gamma$. Then by (6.22), there is a $(\Gamma_2)_\Phi$-skolem-assignment $S''' \succeq S''$ such that $M + S''', k \models \Phi$, as required.

4. Assume $def([\![\theta \,\&\, \xi]\!]^g_M)$, and let k be an M-assignment.

   (a) ⇒ (b):

   Assume $k \in [\![\theta \,\&\, \xi]\!]^g_M$. Then there is an h such that $h \in [\![\theta]\!]^g_M$ and $k \in [\![\xi]\!]^h_M$. Then by 1.4, $h[\mathbb{T}_1 - \mathbb{T}]g$, h verifies $\Gamma_1 - \Gamma$ over $M + S$. By 2.4, $k[\mathbb{T}_2 - \mathbb{T}_1]h$. Then $k[\mathbb{T}_2 - \mathbb{T}]g$ and by (6.20), k verifies $\Gamma_1 - \Gamma$ over $M + S$.

   Now for Γ_1-skolem-assignment $S' \succeq S$ such that $M + S', h \models \Gamma_1 - \Gamma$, k verifies $\Gamma_2 - \Gamma_1$ over $M + S'$ by 2.4. Hence k verifies $\Gamma_2 - \Gamma_1$ over $M + S$. This allows us to conclude that k verifies $\Gamma_2 - \Gamma$ over $M + S$, as required.

   (b) ⇒ (c):

   As in case 1.

   (c) ⇒ (a):

   Assume that $k[\mathbb{T}_2 - \mathbb{T}]g$ and that S'' is a Γ_2-skolem-assignment extending S such that $M + S'', k \models \Gamma_2 - \Gamma$. Define S' to be S'' restricted to the skolem function symbols appearing in Γ_1. Define an M-assignment k' by

$$k'(t) := \begin{cases} g(t) & \text{if } t \in \mathbb{T}_2 - \mathbb{T}_1 \\ k(t) & \text{otherwise.} \end{cases}$$

   Then $k[\mathbb{T}_2 - \mathbb{T}_1]k'$, $k'[\mathbb{T}_1 - \mathbb{T}]g$, $M + S'', k \models \Gamma_2 - \Gamma_1$ and by (6.20), $M + S', k' \models \Gamma_1 - \Gamma$. By our basic assumption [iii], $M + S', k' \models \Gamma$, so by [1.6], we have that for every $T \in \mathbb{T}_1$, $\frac{M+S'}{k'}(T) \neq u^M$. This means that 2.3 up to 2.6 hold for k' in place of g'. So by 1.4 and 2.4, $k' \in [\![\theta]\!]^g_M$ and $k \in [\![\xi]\!]^{k'}_M$, i.e. $k \in [\![\theta \,\&\, \xi]\!]^g_M$.

5. Assume $\nu \neq u$. Then by 2.5 and 1.5, $\mu_1 \neq u$, for all $g' \in [\![\theta]\!]^g_M$, $v(\xi, M, g') \neq u$ (i.e. $def([\![\xi]\!]^{g'}_M)$), $\mu \neq u$ and $v(\theta, M, g) \neq u$ (i.e. $def([\![\theta]\!]^g_M)$). Then $def([\![\theta \,\&\, \xi]\!]^g_M)$, i.e. $v(\theta \,\&\, \xi, M, g) \neq u$, i.e. $\mu + v(\theta \,\&\, \xi, M, g) \neq u$.

   Now assume $\nu = \top$. Then by 2.5 and 1.5, $\mu_1 = \top$, for all $g' \in [\![\theta]\!]^g_M$, $v(\xi, M, g') = \top$, $\mu = \top$ and $v(\theta, M, g) = \top$. $[\![\theta]\!]^g_M \neq \emptyset$, say $g' \in [\![\theta]\!]^g_M$. Then $[\![\xi]\!]^{g'}_M \neq \emptyset$, say $h \in [\![\xi]\!]^{g'}_M$. Then $h \in [\![\theta \,\&\, \xi]\!]^g_M$, so $v(\theta \,\&\, \xi, M, g) = \top$.

6. Suppose that $def([\![\theta \ \& \ \xi]\!]^g_M)$, $k \in [\![\theta \ \& \ \xi]\!]^g_M$, $S' \succeq S$, $M + S', k \models \Gamma_2$ and $T \in \mathbb{T}_2$. Then there is a $g' \in [\![\theta]\!]^g_M$ such that $k \in [\![\xi]\!]^{g'}_M$. Now by Lemma 6.3.22, 1.1 and 1.6, the requirements for concluding 2.3 up to 2.6 are fulfilled for g' and S'. This allows us to conclude by 2.6 that $\frac{M+S'}{k}(T) \neq u^M$.

Case 8: θ_0 is of the form $\exists t\ \varphi$

In this case, there is a premise list Γ^*, a term list $\mathbb{T}_0$, a *PL* formula Φ and a proof status value μ' such that the following hold:

(i) $\mathit{read\_text}(\varphi, \langle PL(t)\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma^*, \mathbb{T}_0, \Phi, \mu')$.

(ii) $\mathit{exist\_check}(\Gamma^*, \mathbb{T}, \exists_{\langle t\rangle}\ \exists_{\mathbb{T}_0}\ \Phi, \mu') = (\Gamma^+, \nu)$.

(iii) $\Gamma' = \Gamma^* \oplus \langle PL(t) \neq u^{\mathbb{P}}, \Phi\rangle$,

(iv) $\mathbb{T}' = \mathbb{T} \oplus \mathbb{T}_0 \oplus \langle PL(t)\rangle$

We now prove each of the six assertions of the Detailed Soundness Lemma for $\exists t\ \varphi$:

1. By assertion 1 of the *read_text* Soundness Lemma, $\mathbf{tbc}(\varphi) = PL^{-1}(\mathbb{T}_0)$. By Lemma 5.2.17, $\mathbf{tbc}(\exists t\ \varphi) = \mathbf{tbc}(\Phi) \cup \{t\} = PL^{-1}(\mathbb{T}')$, as required.

2. This follows from assertions 2 and 3 of the *read_text* Soundness Lemma.

3. This easily follows from assertion 5 of the *read_text* Soundness Lemma and the definition of Γ^* in the definition of *read_text*.

4. This easily follows from assertion 6 of the *read_text* Soundness Lemma.

5. This follows from the *exist_check* Soundness Lemma using similar reasoning as in the previous cases.

6. This easily follows from assertion 7 of the *read_text* Soundness Lemma.

Case 9: θ_0 is of the form $\Diamond\varphi$

This case can be verified in a way very similar to case 3.

Case 10: θ_0 is of the form $\boldsymbol{label}(\alpha, \theta)$

This case follows trivially from the application of this lemma to θ.

Case 11: θ_0 is of the form $\boldsymbol{ref}(S, \varphi)$

This case follows trivially from the application of this lemma to φ.

Case 12: θ_0 is of the form *thm*$(\vartheta, \varphi, \theta)$

In this case, there are premise lists Γ_1 and Γ_2, term lists $\mathbb{T}_1$ and $\mathbb{T}_2$ and a proof status value μ_0 such that the following hold:

(i) $\mathit{check\_text}(\theta, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}_1, \mu_1)$.

(ii) $\mathit{check\_text}(\varphi, \Gamma_1, \mathbb{T}_1, \mu_1) = (\Gamma_2, \mathbb{T}_2, \nu)$.

(iii) $\Gamma' = \Gamma \oplus \langle \alpha : \Phi^p - \vartheta \mid \alpha : \Phi^p - 0 \in \Gamma_2 - \Gamma_1 \rangle$.

(iv) $\mathbb{T}' = \mathbb{T} \oplus (\mathbb{T}_2 - \mathbb{T}_1)$.

We now prove each of the six assertions of the Detailed Soundness Lemma for $\mathit{thm}(\vartheta, \varphi, \theta)$:

1. $\mathbf{tbc}(\mathit{thm}(\vartheta, \varphi, \theta)) = \mathbf{tbc}(\varphi) = PL^{-1}(\mathbb{T}_2 - \mathbb{T}_1) = PL^{-1}(\mathbb{T}' - \mathbb{T})$.

2. It can be verified from the definition of *check_text* that if

$$\mathit{check\_text}(\varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma^*, \mathbb{T}^*, \nu'), \tag{6.23}$$

   then $\mathbb{T}^* = \mathbb{T}'$ and Γ^* differs from Γ' only in the theorem-type marking of its premises. Now assertion 2 of the Detailed Soundness Lemma applied to (6.23) implies that all MHF terms in $\Gamma^* - \Gamma$, and hence all MHF terms in $\Gamma' - \Gamma$, are composed of PTL_{sk} symbols and terms in $\mathbb{T}^* = \mathbb{T}'$.

3. This directly follows from assertion 3 of the Detailed Soundness Lemma applied to (ii) and the fact that $\mathit{def}(\llbracket \mathit{thm}(\vartheta, \varphi, \theta) \rrbracket^g_M)$ iff $\mathit{def}(\llbracket \varphi \rrbracket^g_M)$.

4. This directly follows from assertion 4 of the Detailed Soundness Lemma applied to (ii) and the fact that $\llbracket \mathit{thm}(\vartheta, \varphi, \theta) \rrbracket^g_M = \llbracket \varphi \rrbracket^g_M$.

5. By assertion 5 of the Detailed Soundness Lemma applied to (i), $\mu_1 \leq \mu + v(\theta, M, g) \leq \mu$. No by assertion 5 of the Detailed Soundness Lemma applied to (ii), we have $\nu \leq \mu_1 + v(\varphi, M, g) = \mu_1 + v(\mathit{thm}(\vartheta, \varphi, \theta), M, g) \leq \mu + v(\mathit{thm}(\vartheta, \varphi, \theta), M, g)$.

6. This easily follows from assertion 6 of the Detailed Soundness Lemma applied to (ii).

Case 13: θ_0 is of the form def(t)

In this case, there is a premise list Γ_1 and a proof status value μ' such that the following hold:

(i) $\mathit{read\_term}(t, \Gamma, \mathbb{T}, \mu) = \Gamma_1, \_, \mu')$.

(ii) $\Gamma' = \Gamma \oplus \langle \alpha : \Phi^0 - \vartheta \mid \alpha : \Phi^{\mathbb{P}} - \vartheta \in \Gamma_1 - \Gamma \rangle$.

(iii) $$\nu = \begin{cases} u & \text{if } \mu = u \\ \bot & \text{if } \mu \neq u \text{ but } \mu_1 = u \\ \mu & \text{otherwise} \end{cases}$$

(iv) $\mathbb{T}' = \mathbb{T}$.

We now prove each of the six assertions of the Detailed Soundness Lemma for $\mathrm{def}(t)$:

1. $\mathbf{tbc}(\mathrm{def}(t)) = \emptyset = \mathbb{T}' - \mathbb{T}$, since $\mathbb{T}' = \mathbb{T}$.

2. This directly follows from assertion 3 of the *read_text* Soundness Lemma.

3. Both sides of the required biimplication are necessarily true: The left hand side because of the definition of $\mathit{def}(\llbracket \mathrm{def}(t) \rrbracket^g_M)$, and the right hand side because $\Gamma' - \Gamma$ does not contain presuppositionally marked premises.

4. (a) $\Rightarrow$ (b):

   Assume $k \in \llbracket \mathrm{def}(t) \rrbracket^M_g$. Then $k = g$, i.e. $k[\mathbb{T}' - \mathbb{T}]g$. Now it is enough to show that g verifies $\Gamma' - \Gamma$ over $M + S$.

   Since $\llbracket \mathrm{def}(t) \rrbracket^g_M \neq \emptyset$, $\frac{M}{g}(t) \neq u$. Note that since t is an ι-free *PTL* term, $\Gamma_1 - \Gamma$ does not contain any skolem function symbols. Hence for every $\Phi \in \Gamma' - \Gamma$, every Γ'_Φ-skolem-assignment S' and every $\Gamma'_{\Phi+}$-skolem-assignment $S_0 \succeq S'$, $S_0 = S'$. Hence we can conclude from assertion 3 of the *read_term* Soundness Lemma that g verifies $\Gamma' - \Gamma$ over $M + S$.

   (b) $\Rightarrow$ (c):

   As in case 1.

   (c) $\Rightarrow$ (a):

   Suppose that $k[\mathbb{T}' - \mathbb{T}]g$ and that there is a Γ'-skolem-assignment $S' \succeq S$ such that $M + S', k \models \Gamma'$. Since $\mathbb{T}' - \mathbb{T} = \emptyset$, $k = g$. Since $\Gamma_1 - \Gamma$ does not contain any skolem function symbols, $S' = S$. So $M + S, g \models \Gamma'$. Now assertion 3 of the *read_term* Soundness Lemma implies that $\frac{M}{g}(t) \neq u$, i.e. $k = g \in \llbracket \mathrm{def}(t) \rrbracket^g_M$.

5. Similarly as in previous cases.

6. Trivial.

This completes the proof of the Detailed Soundness Lemma.

Note that with the proof of the Detailed Soundness Lemma being completed, the not yet proved lemmas 5.2.20 and 5.2.21 from chapter 5 now follow from lemmas 6.3.22 and 6.3.23 respectively.

6.3.2 Two soundness theorems

We proved the Detailed Soundness Lemma in order to be able to prove the Soundness theorem of the *PTL* proof checking algorithm. We restate this theorem before giving the proof:

Theorem 6.3.1 (Soundness of the *PTL* proof checking algorithm)**.** *If θ is a nice PTL text and $\mathit{check}(\theta) = \top$, then $v(\theta) = \top$.*

Proof. Suppose that $\mathit{check}(\theta) = \top$. By the definition of *check*,

$$\mathit{check\_text}(\theta, \langle\rangle, \langle\rangle, \top) = (\_, \_, \top). \tag{6.24}$$

We now apply the Detailed Soundness Lemma to (6.24). For this, we have to fix the values of the variables in the Detailed Soundness Lemma as follows:

- $\mathbb{T} = \langle\rangle$.
- $\Gamma = \langle\rangle$.
- M is an arbitrary *CMTN* model (so we actually apply the Detail Soundness Lemma once for every *CMTN* model).
- S is the empty skolem-assignment over M.
- g is e_M, i.e. the empty M-assignment.

The first assumption of the Detailed Soundness Lemma holds because θ is semi-nice. The second assumption holds because θ is ground, i.e. contains no hereditarily free terms. The other assumptions trivially hold, so we can actually apply the Detailed Soundness Lemma. By assertion 5 of the lemma we can conclude that $\mu + v(\theta, M, g) \geq \top$. This implies that $v(\theta, M, g) = \top$, i.e. that $[\![\theta]\!]_M^{e_M}$ is defined and non-empty. Since this holds for all *CMTN* models M, we can conclude that $v(\theta) = \top$. □

The above Theorem 6.3.1 links the proof checking algorithm to the semantics of *PTL*. But one can also use a *PTL* text to prove a *PL* entailment, so we will present a separate soundness theorem linking the proof checking algorithm to standard *PL* semantics. Let $L = (c_1, \ldots, c_l; f_1^{k_1}, \ldots, f_m^{k_m}; R_1^{k'_1}, \ldots, R_n^{k'_n})$ be a *PL* language (here the superscripts indicate the arities of the function and relation symbols). Suppose that $\Gamma = \{\Phi_1, \ldots, \Phi_k\}$ is a finite set of *PL* formulae, that Ψ is a further *PL* formula, and that we want to show that $\Gamma \models \Psi$. A systematized way of doing this using a natural language proof could be as follows:

> Let D be a domain of objects, and suppose that $c_1, \ldots, c_l \in D$, that $f_1^{k_1}$ is a k_1-ary function on D, ..., that $f_m^{k_m}$ is a k_m-ary function on D, that $R_1^{k'_1}$ is a k'_1-ary relation on D, ... and that $R_n^{k'_n}$ is a k'_n-ary relation symbol on D such that the following axioms hold:
>
> - Φ_1^D.
> - $\vdots$
> - Φ_k^D.
>
> (6.25)
>
> **Theorem.** Ψ^D.
>
> *Proof.* ... □

Here Φ^D denotes the relativization of Φ to D (or some natural language reformulation of this relativization), and the final "..." denotes some natural language proof of Ψ_D. We now formalize this kind of natural language argument in *PTL*, identifying the domain D with the domain of urelements in *PTL*. For this we need a way of expressing in *PTL* that $f_i^{k_i}$ is a k_i-ary function on the urelements, i.e. defined precisely on the urelements and returning an urelement

as its value. One convenient way of doing this in *PTL* is by using implicit function introduction to introduce the symbol $f_i^{k_i}$:

$$\begin{array}{l}\exists x_1 \ \ldots \exists x_{k_i} \ (U(x_1) \wedge \ldots \wedge U(x_{k_i})) \rightarrow \\ \exists f_i^{k_i}(x_1, \ldots, x_{k_i}) \ U(f_i^{k_i}(x_1, \ldots, x_{k_i})).\end{array} \tag{6.26}$$

We call the *PTL* formula in (6.26) $\mathbf{F}_{k_i}(f_i^{k_i})$. Similarly we need a *PTL* formula $\mathbf{R}_{k_i'}(R_i^{k_i'})$ that formalizes that $R_i^{k_i'}$ is a k_i-are relation on the urelements. Again we do this using implicit function introduction:

$$\begin{array}{rl}\mathbf{R}_{k_i'}(R_i^{k_i'}) := & \exists x_1 \ \ldots \exists x_{k_i'} \ (U(x_1) \wedge \ldots \wedge U(x_{k_i'})) \rightarrow \\ & \exists f_i^{k_i}(x_1, \ldots, x_{k_i'}) \ B(f_i^{k_i}(x_1, \ldots, x_{k_i})).\end{array}$$

The *PL* function and relation symbols $f_1^{k_1}$, ..., $f_m^{k_m}$, $R_1^{k_1'}$, ..., $R_n^{k_n'}$ are thus transformed to *PTL* variables in this *PTL* formalization. Similarly we will transform the *PL* constant symbols to *PTL* variables. Now any L-formula Φ can be transformed into an equivalent L-formula Φ^* by getting rid of all occurrences of $\leftrightarrow$ and $\forall$ (using equivalent expressions involving $\rightarrow$, $\wedge$, $\neg$ and $\exists$). We transform Φ further to a *PTL* text Φ' by relativizing all existential quantifiers in Φ^* to U and making them static using $\Diamond$ (i.e. recursively substituting $\exists x \ \Phi$ by $\Diamond \exists x \ (U(x) \wedge \Phi)$), and finally replacing the constants, function and relation symbols of L by the corresponding *PTL* variables.

Now given a *PTL* text θ that formalizes the natural language proof of the theorem in the above natural language text (6.25), the *PTL* text formalizing the total text (6.25) is as follows:

$$\begin{array}{l}\exists c_1 \ U(c_1) \wedge \ldots \wedge \exists c_l \ U(c_l) \wedge \mathbf{F}_{k_1}(f_1^{k_1}) \wedge \ldots \wedge \mathbf{F}_{k_m}(f_m^{k_m}) \wedge \\ \mathbf{R}_{k_1'}(R_1^{k_1'}) \wedge \ldots \wedge \mathbf{R}_{k_n'}(R_n^{k_n'}) \wedge \Phi_1' \wedge \ldots \wedge \Phi_k' \rightarrow \mathit{thm}(\mathit{thm}, \Psi', \theta).\end{array} \tag{6.27}$$

We abbreviate the *PTL* text in (6.27) to $L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$. Now we are able to state the soundness theorem that links the proof checking algorithm to *PL* semantics states. Since we will need Theorem 4.3.27 in the proof of this soundness theorem, we will have to take over the assumption of that theorem that *ZFC* has an ω-model.

Theorem 6.3.24. *Suppose that ZFC has an ω-model. Let L be a PL language and let $\Phi_1, \ldots, \Phi_k$ and Ψ be L-formulae. Suppose that there is a PTL text θ such that check$(L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi) = \top$. Then $\Phi_1, \ldots, \Phi_k \models \Psi$.*

Proof. By Theorem 6.3.1, $v(L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi) = \top$. This means that for all *CMTN* models M, $[\![L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi]\!]^M_{e_M}$ is defined and non-empty. From the *PTL* semantics of $\rightarrow$ it now follows that for k in

$$\begin{array}{l}[\![\exists c_1 \ U(c_1) \wedge \ldots \wedge \exists c_l \ U(c_l) \wedge \mathbf{F}_{k_1}(f_i^{k_1}) \wedge \ldots \wedge \mathbf{F}_{k_m}(f_i^{k_m}) \wedge \\ \mathbf{R}_{k_1'}(R_i^{k_1'}) \wedge \ldots \wedge \mathbf{R}_{k_n'}(R_i^{k_n'}) \wedge \Phi_1' \wedge \ldots \wedge \Phi_k']\!]^M_{e_M},\end{array}$$

$[\![\mathit{thm}(\mathit{thm}, \Psi', \theta)]\!]^M_k \neq \emptyset$, i.e. $[\![\Psi']\!]^M_k \neq \emptyset$.

Write Γ for $\{\Phi_1, \ldots, \Phi_k\}$. For the rest of the proof, we will make extensive usage of the notation from section 4.3.2 of chapter 4.

Let N be a $CMTN_L$ model such that $N \models \Gamma_L \cup \Gamma^{\mathcal{A}_L}$. By reconsidering the constant symbols in L^L_{CMTN} that result from symbols in the signature of L as

variable symbols, we can transform N into a pair consisting of a *CMTN* model M and an M-assignment k such that

$$k \in [\![\exists c_1\, U(c_1) \wedge \ldots \wedge \exists c_l\, U(c_l) \wedge \mathbf{F}_{k_1}(f_i^{k_1}) \wedge \ldots \wedge \mathbf{F}_{k_m}(f_i^{k_m}) \wedge \mathbf{R}_{k'_1}(R_i^{k'_1}) \wedge \ldots \wedge \mathbf{R}_{k'_n}(R_i^{k'_n}) \wedge \Phi'_1 \wedge \ldots \wedge \Phi'_k]\!]^M_{e_M}.$$

So by the above, we can conclude that $[\![\Psi']\!]^M_k \neq \emptyset$, i.e. that $N \models \Psi^{\mathcal{A}_L}$.

Hence we have established that $CMTN_L \cup \Gamma_L \cup \Gamma^{\mathcal{A}_L} \models \Psi^{\mathcal{A}_L}$. Now Theorem 4.3.27 implies that $\Gamma \models \Psi$, as required. □

6.4 Completeness of the proof checking algorithm

In this section we will present two completeness theorems about the proof checking algorithm, analogous to the two soundness theorems from section 6.3.2, i.e. one theorem linking the algorithm to *PL* semantics and one to *PTL* semantics. More precisely, the completeness theorems hold under the assumption that the prover in the proof checking algorithm has a certain minimal proving power. The first completeness theorem establishes that for every valid *PL* entailment there is a *PTL* text that proves this entailment and that will be successfully checked by the algorithm. The second completeness theorem establishes that for every valid *PTL* formula there is a *PTL* text that proves this formula and that will be successfully checked by the algorithm.

First we need to define the minimal proving power that the first completeness theorem requires from the prover. In this definition as well as in the proof of the first completeness theorem, we will refer to a restricted version of *PL*, in which the only connectives are $\rightarrow$ and $\neg$, the only logical constant is $\bot$ and the only quantifier is $\exists$, and which we call $PL_{\neg,\rightarrow,\bot,\exists}$.

Definition 6.4.1. A prover P is called *sufficiently strong* if it satisfies the following properties:

1. There is some translation function $\mathbf{t}$ from *PL* formulae to $PL_{\neg,\rightarrow,\bot,\exists}$ formulae with the following three properties:
   - For any *PL* formulae Φ, $\mathbf{t}(\Phi)$ is logically equivalent to Φ.
   - If Φ is a *PL* formula such that $\mathbf{t}(\Phi) \in \Gamma$, then $P(\Gamma \vdash^? \Phi) = 1$.
   - If Φ is a *PL* formula such that $\Phi \in \Gamma$, then $P(\Gamma \vdash^? \mathbf{t}(\Phi)) = 1$.
2. If Γ is a premise list and a Φ is a *PL* formula such that $(\neg\Phi \rightarrow \bot) \in \Gamma$, then $P(\Gamma \vdash^? \Phi) = 1$.
3. If $\Phi \in \Gamma$ and $\neg\Phi \in \Gamma$, then $P(\Gamma \vdash^? \bot) = 1$.
4. If $\Phi \rightarrow \exists_{\langle v_1,\ldots,v_n\rangle}\, (\Psi_1 \wedge \ldots \wedge \Psi_n) \in \Gamma$ and Ψ_n does not contain free occurrences of $v_1, \ldots, v_n$, then $P(\Gamma \vdash^? (\Phi \rightarrow \Psi_n)) = 1$.
5. If $(\Phi \rightarrow \Psi) \in \Gamma$ and $\Phi \in \Gamma$, then $P(\Gamma \vdash^? \Psi) = 1$.
6. If T is a *PL* term and $\Psi_1\frac{T}{x}, \Psi_2\frac{T}{x}, \Psi_3\frac{T}{x} \in \Gamma$, then $P(\Gamma \vdash^? \exists_{\langle x\rangle}\, (\Psi_1 \wedge \Psi_2 \wedge \Psi_3)) = 1$.

7. If $\exists_{\langle x\rangle}\ \Phi \in \Gamma$, then $P(\Gamma \vdash^? \exists_{\langle v\rangle}\ \Phi\frac{v}{x}) = 1$.

8. If T is a PL term, then $P(\Gamma \vdash^? T = T) = 1$.

9. If $T_1 = T_2 \in \Gamma$ and $\Phi\frac{T_1}{x} \in \Gamma$, then $P(\Gamma \vdash^? \Phi\frac{T_2}{x}) = 1$.

10. If $\forall_{\langle v_1,\ldots,v_n\rangle}(\Psi_1 \wedge \ldots \wedge \Psi_n \leftrightarrow \Phi) \in \Gamma$ and $T_1, \ldots, T_n$ are PL terms such that $T_1 \neq u$, $\ldots$, $T_n \neq u$, $\Psi_1\frac{T_1}{v_1}\ldots\frac{T_n}{v_n}$, $\ldots$, $\Psi_n\frac{T_1}{v_1}\ldots\frac{T_n}{v_n}$ are in Γ, then $P(\Gamma \vdash^? \Phi\frac{T_1}{v_1}\ldots\frac{T_n}{v_n}) = 1$.

Definition 6.4.2. Given a prover P and a PTL text θ, we write $check_P(\theta)$ for the result $check(\theta)$ of applying the proof checking algorithm with prover P to θ.

For the first completeness theorem, we would like to state that if $\Phi_1, \ldots, \Phi_k \models \Psi$ and P is a sufficiently strong prover, then there is a PTL text θ such that $check_P(L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi) = \top$. But there is a problem with this wording: In the definition of $L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$, we have $\Phi'_1 \wedge \ldots \wedge \Phi'_k$ among the assumptions on the left hand side of the main $\rightarrow$ operator. This basically means that we may assume Φ'_1, $\ldots$, Φ'_k to be true before looking at the $thm(thm, \Psi', \theta)$ on the right hand side of that $\rightarrow$. But we have to show that the presuppositions of Φ'_1, $\ldots$, Φ'_k are fulfilled before we may assume them to be true. There presuppositions amount from function applications involving the function and relation symbols of L, which are assumed to be defined on urelements and to return urelements. Based on these assumptions, the presuppositions of Φ'_1, $\ldots$, Φ'_k are actually always fulfilled. But to ensure that a prover with the above defined minimal proving power can prove that these presuppositions are fulfilled, we have to give to the prover some intermediate reasoning steps. In order to do this, we do not work with PTL texts of the form $L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$ as defined previously, but with PTL texts of a similar form, denoted $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$, with the following definition:

Definition 6.4.3. Let L be a PL language, let $\Phi_1, \ldots, \Phi_k$ and Ψ be L-formulae and let ξ and θ be PTL texts. Then we define $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$ to be the following PTL text:

$$\begin{array}{l}\exists c_1\ U(c_1) \wedge \ldots \wedge \exists c_l\ U(c_l) \wedge \mathbf{F}_{k_1}(f_1^{k_1}) \wedge \ldots \wedge \mathbf{F}_{k_m}(f_m^{k_m}) \wedge \\ \mathbf{R}_{k'_1}(R_1^{k'_1}) \wedge \ldots \wedge \mathbf{R}_{k'_n}(R_n^{k'_n}) \ \rightarrow\ (\xi \wedge (\Phi'_1 \wedge \ldots \wedge \Phi'_k \ \rightarrow \\ thm(thm, \Psi', \theta))).\end{array} \tag{6.28}$$

Here the ξ gives us the possibility to add intermediate reasoning steps needed for proving the presuppositions of Φ'_1, $\ldots$, Φ'_k. One can easily see that if $\xi = \top$, then $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$ is equivalent to $L; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$.

For proving the completeness theorems, we will make use of the completeness of a certain system of natural deduction for $PL_{\neg,\rightarrow,\bot,\exists}$. More precisely, it is a system of natural deduction with variable declaration (see Velleman, 2006). This means that it has special proof lines for declaring variables, and that a variable v may only appear freely in a formula φ if φ is inside the scope of a declaration of v.

This system of natural deduction has eight rules:

$$\frac{\neg\Phi \to \bot}{\Phi}\ \text{proof by contradiction} \qquad \frac{\Phi \quad \neg\Phi}{\bot}\ \neg\text{-elimination}$$

$$\frac{\begin{array}{c}\Phi\\ \vdots\\ \Psi\end{array}}{\Phi \to \Psi}\ \to\text{-introduction} \qquad \frac{\Phi \to \Psi \quad \Phi}{\Psi}\ \to\text{-elimination}$$

$$\frac{\Phi\frac{T}{x}}{\exists x\ \Phi}\ \exists\text{-introduction} \qquad \frac{\exists x\ \Phi}{\begin{array}{l}\text{Declare: } v\\ \Phi\frac{v}{x}\end{array}}\ \exists\text{-elimination}$$

$$\frac{}{T = T}\ =\text{-introduction} \qquad \frac{T_1 = T_2 \quad \Phi\frac{T_1}{x}}{\Phi\frac{T_2}{x}}\ =\text{-elimination}$$

Velleman (2006) sketches a completeness proof for such a system of natural deduction with variable declaration. He actually defines a system which has both existential and universal quantifiers, but the universal quantifiers play no role in his completeness proof, so that the proof goes through without them. Additionally, he is not precise about which connectives and which rules for the connectives he presupposes, but one can easily check that the above set is sufficient for his completeness proof to go through.

We still need two definitions and one lemma before presenting the first completeness theorem:

Definition 6.4.4. Given a *PTL* text φ, we let $PL(\varphi)$ denote the *PL* formula generated from φ by replacing every term t occurring in φ as an argument of a logical relation symbol by $PL(t)$, replacing every other term t occurring in φ without being a proper subterm of an occurrence of a term in φ by $PL(t) = \top$, dropping all occurrences of $\Diamond$ and relativizing all quantifiers to $\neq u$ (i.e. recursively replacing $\exists x\ \psi$ by $\exists x\ (x \neq u \wedge \psi)$).

Lemma 6.4.5. *Let L be a PL language. For every L-formula Φ, there is a list $\langle T_1, \ldots, T_n\rangle$ of L-terms that includes all terms occurring in Φ and such that every term T_i in this list is either a variable, a constant symbol or of the form $f(T_{i_1}, \ldots, T_{i_k})$ for some k-ary function symbol f of L and $i_1, \ldots, i_k < i$.*

Proof. Trivial from the recursive definition of *L-term*. □

Definition 6.4.6. Given a situation as in the above lemma, we call $\langle t_1, \ldots, t_n\rangle$ a *list of the terms in Φ ordered by term construction.*

We are now ready to present and prove the first completeness theorem, which links the proof checking algorithm to *PL* semantics:

Theorem 6.4.7. *Suppose that L is a PL language and that $\Phi_1, \ldots, \Phi_k, \Psi$ are L-formulae such that $\Phi_1, \ldots, \Phi_k \models \Psi$. Let P be a sufficiently strong prover. Then there are PTL texts ξ and θ such that $check_P(L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi) = \top$.*

Proof. Let $T_1, \ldots, T_n$ be a list of the terms in $\Phi'_1 \wedge \cdots \wedge \Phi'_k \wedge \Psi$ ordered by term construction. Now ξ is defined to be $U(T_1)\ \&\ \ldots\ \&\ U(T_n)$.

By the completeness of the above natural deduction calculus, there is a natural deduction derivation D of $\mathbf{t}(\Phi_1), \ldots, \mathbf{t}(\Phi_k) \vdash \mathbf{t}(\Psi)$. The idea is that we

transform this D into the required PTL text θ. For this we consider D to be written in Fitch's indentation notation (which is a common notation for natural deduction proofs, also used in Velleman (2006)).

For every proof line Φ in D that is not a variable declaration, we choose a list $T_1^\Phi, \ldots, T_n^\Phi$ of the terms in Φ ordered by term construction, and add the sequence $U(T_1^\Phi)$, $\ldots$, $U(T_n^\Phi)$ of proof lines in front of Φ and with the same indentation as Φ. Next, for every variable declaration "Declare: v" appearing in D, we replace "Declare: v" and the proof line Φ that follows it by a single proof line $\exists v\,(U(v) \wedge \Phi')$ of the same indentation. Afterwards, we replace every proof line Φ of D not touched in the previous step by Φ'. Next, every indented subproof of the form

$$\begin{array}{|l} \Phi \\ \hline \Psi_1 \\ \vdots \\ \Psi_n \end{array}$$

is transformed into a PTL text of the form $\Phi \rightarrow \Psi_1 \,\&\, \ldots \,\&\, \Psi_n$ (just as in section 6.3.2 above, we can consider L-formulae to be PTL texts by considering the constants, function symbols and relation symbols as PTL variables). This transformation is recursively applied to subproof including subproofs, and so on. On the highest level the subproofs and proof steps are conjuncted with & in the original order. Call the result of this transformation of D δ. Then θ is defined to be $\Diamond\mathbf{t}(\Phi_1)' \,\&\, \ldots \,\&\, \Diamond\mathbf{t}(\Phi_n)' \wedge \delta$.

We now still have to show that $check_P(L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi) = \top$.

First we ensure that all presuppositional calls to the prover are successful. Every such call results from a function application and is hence of the form $\Gamma \vdash^? PL(f^k(T_1', \ldots, T_n')) \neq u$ for the currently active premise list Γ, some function symbol f^k of L (considered as a PTL variable) and some PTL terms $T_1', \ldots, T_n'$. We need to show that $P(\Gamma \vdash^? PL(f^k(T_1', \ldots, T_n')) \neq u) = 1$ in every such case. First we show that for every i, Γ contains formulae of the form $T_i' \neq u$ and $U(PL(T_i'))$: If T_i' is complex, this follows from the definition of ξ and the first step in the transformation of D into δ. If T_i' is a constant symbol c of L, this follows from the fact that $\exists c_1\, U(c_1)$ appears among the assumptions in $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$. If T_i' is a variable, then it must be a declared variable, so $T_i' \neq u \in \Gamma$ and $U(T_i') \in \Gamma$ follow from the fact that for every variable declaration in D, we have added a PTL subtext of the form $\exists v\,(U(v) \wedge \Phi')$ to δ. From the definition of $\mathbf{F}_k(f^k)$ and the definition of Γ_{func} in the definition of the proof checking algorithm, it follows that Γ contains a formula of the form

$$\forall_{\langle x_1 \ldots x_k \rangle}(U(x_1) \wedge \ldots \wedge U(x_k) \leftrightarrow f^k(x_1, \ldots, x_k) \neq u).$$

Now $P(\Gamma \vdash^? PL(f^k(T_1', \ldots, T_n')) \neq u) = 1$ follows from property 10 in the definition of *sufficiently strong prover*.

We can now concentrate on non-presuppositional calls to the prover for the rest of this proof.

Keeping in mind the definitions of $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$, *check_text* and Φ', one can easily see that the proof checking algorithm has $PL(\Phi_1')$, $\ldots$, $PL(\Phi_n')$ in its active premise list when it starts processing the $thm(thm, \Psi', \theta)$ in $L; \xi; \Phi_1, \ldots, \Phi_k \vdash_\theta \Psi$.

For checking $thm(thm, \Psi', \theta)$, the algorithm will first check θ and then check Ψ' based on the premise list resulting from checking θ. For checking θ, it will first check $\Diamond\mathbf{t}(\Phi_1)$, ..., $\Diamond\mathbf{t}(\Phi_n)$ and then check δ. Let us first concentrate on the checking of $\Diamond\mathbf{t}(\Phi_1)$, ..., $\Diamond\mathbf{t}(\Phi_n)$.

We need the proof status value to stay $\top$ while checking $\Diamond\mathbf{t}(\Phi_1), \ldots, \Diamond\mathbf{t}(\Phi_n)$. For every $1 \leq i \leq n$, the proof checking algorithm will run the prover to calculate $P(\Gamma \vdash^? PL(\mathbf{t}(\Phi_i)'))$, where Γ is the currently active premise list, and we need $P(\Gamma \vdash^? PL(\mathbf{t}(\Phi_i)')) = 1$ for all $1 \leq i \leq n$. Since $PL(\Phi_i')$ is in Γ, and $PL(\mathbf{t}(\Phi_i)')$ is equal to $\mathbf{t}(PL(\Phi_i'))$, $P(\Gamma \vdash^? PL(\mathbf{t}(\Phi_i)')) = 1$ follows from property 1 in the definition of *sufficiently strong prover*.

Now we concentrate on the checking of δ. Note that while the algorithm checks δ, it processes the subformulae of δ corresponding to lines in D in the same order as they appear in D. Hence it makes sense to use a line in D for specifying a point in the checking process of δ.

We need the proof status value to stay $\top$ while checking δ. Now for every proof line Φ in D that is not a variable declaration and is not a hypothesis that starts a new subproof, the proof checking algorithm will run the prover to calculate $P(\Gamma \vdash^? PL(\Phi'))$ (or $P(\Gamma \vdash^? PL(\exists v\ (U(v) \wedge \Phi')))$ in the case that Φ is preceded by a variable declaration "Declare: v"), where Γ is the currently active premise list, and we need $P(\Gamma \vdash^? PL(\Phi')) = 1$ (or $P(\Gamma \vdash^? PL(\exists v\ (U(v) \wedge \Phi'))) = 1$) in every such case in order for the proof status value to stay $\top$. For establishing this, we first need to show that in such a case, Γ contains $PL(\Psi_0')$ for every proof line Ψ_0 preceding Φ in D and not contained in an already closed subproof (i.e. usable at line Φ as a premise for an application of a natural-deduction rule). For this we need to distinguish two cases:

Case 1: Ψ_0 is preceded by a line of the form "Declare: v"

Then $\exists v\ (U(v) \wedge \Psi_0')$ appears in the part of θ already processed by the algorithm, not embedded in any already closed scope of an implication. Hence the premise $PL(\Psi_0')$ that was added to the active premise list after processing $\exists v\ (U(v) \wedge \Psi_0')$ is still in the currently active premise list, as required.

Case 2: Ψ_0 is not preceded by a line of the form "Declare: v"

Then Ψ_0' appears in the part of θ already processed by the algorithm, not embedded in any already closed scope of an implication. Hence the premise $PL(\Psi_0')$ that was added to the active premise list after processing Ψ_0' is still in the currently active premise list, as required.

Now we are ready to show that $P(\Gamma \vdash^? PL(\Phi')) = 1$ (or $P(\Gamma \vdash^? PL(\exists v\ (U(v) \wedge \Phi'))) = 1$) for every proof line Φ in D. By the nature of the system of natural deduction that D is a proof of, we can distinguish eight cases:

Case 1: Φ follows from an earlier line in D by *proof by contradiction*

Then the earlier line from which Φ follows has the form $\neg\Phi \rightarrow \bot$. Then $PL((\neg\Phi \rightarrow \bot)')$, i.e. $\neg PL(\Phi') \rightarrow \bot$, is in the active premise list Γ. Now $P(\Gamma \vdash^? PL(\Phi')) = 1$ follows from property 2 in the definition of *sufficiently strong prover*.

Case 2: Φ follows from earlier lines in D by $\neg$-elimination

Then Φ is of the form $\bot$ and the lines from which it follows are of the form X and $\neg X$. Now $PL(X')$ and $PL((\neg X)')$, i.e. $\neg PL(X')$, are in Γ, so $P(\Gamma \vdash^? PL(\Phi')) = 1$ follows from property 3 in the definition of *sufficiently strong prover*.

Case 3: Φ follows from earlier lines in D by $\rightarrow$-introduction

Then Φ is of the form $\Psi_0 \rightarrow \Psi_n$, and the lines from which Φ follows are a subproof of the following form:

$$\begin{array}{|l} \Psi_0 \\ \hline \Psi_1 \\ \vdots \\ \Psi_n \end{array}$$

The translation of this subproof in θ is of the form $\Psi_0' \rightarrow (\ldots \,\&\, \Psi_n')$ or $\Psi_0' \rightarrow (\ldots \,\&\, \exists v\ (U(v) \wedge \Psi_n'))$. In either case, the active premise list is augmented by $PL(\Psi_0')$ when processing the left argument of $\rightarrow$ and augmented inter alia by $PL(\Psi_n')$ when processing the right argument of $\rightarrow$. Hence the active premise list Γ that is active when encountering Φ contains a premise of the form $PL(\Psi_0') \rightarrow \exists_{\langle v_1,\ldots,v_n\rangle}\ (PL(\Psi_1') \wedge \ldots \wedge PL(\Psi_n'))$, where $v_1, \ldots, v_n$ are the variables declared in the above subproof. Since the above subproof is used to conclude $\Psi_0 \rightarrow \Psi_n$, and since this line is not within the scope of the variable declarations inside the subproof, the limitation that all free variables of $\Psi_0 \rightarrow \Psi_n$ must be in the scope of some variable declaration declaring them implies that Ψ_n does not contain any free occurrences of $v_1, \ldots, v_n$. Now $P(\Gamma \vdash^? PL(\Phi')) = 1$ follows from property 4 in the definition of *sufficiently strong prover*.

Case 4: Φ follows from earlier lines in D by $\rightarrow$-elimination

Similar to cases 1 and 2, but using property 5 in the definition of *sufficiently strong prover*.

Case 5: Φ follows from an earlier line in D by $\exists$-introduction

Then Φ is of the form $\exists x\ X$ and follows from an earlier line of the form $X\frac{T}{x}$, so $PL(X\frac{T}{x})$ is in the active premise list Γ. By the same reasoning as in the case of the presuppositional calls treated above, $U(T)$ and $T \neq u$ are in Γ. Noting that $PL(\Phi')$ is of the form $\exists x\ (x \neq u \wedge U(x) \wedge X)$, we can deduce $P(\Gamma \vdash^? PL(\Phi')) = 1$ from property 6 in the definition of *sufficiently strong prover*.

Case 6: Φ follows from an earlier line in D by $\exists$-elimination

Then Φ is of the form $\Psi_0\frac{v}{x}$ and is preceded by a variable declaration of the form "Declare: v", and the line from which Φ follows is of the form $\exists x\ \Psi_0$. In this case the variable declaration and Φ together are translated as $\exists v\ (U(v) \wedge (\Psi_0\frac{v}{x})')$. Γ contains $PL((\exists_{\langle x\rangle}\ \Psi_0)')$, i.e. $PL(\exists_{\langle x\rangle}\ (U(x) \wedge \Psi_0'))$, so $P(\Gamma \vdash^? PL(\exists_{\langle v\rangle}\ (U(v) \wedge (\Psi_0\frac{v}{x})'))) = 1$ follows from property 7 in the definition of *sufficiently strong prover*.

Case 7: Φ follows from earlier lines in D by $=$-introduction

This easily follows from property 8 in the definition of *sufficiently strong prover*.

Case 8: Φ follows from an earlier line in D by $=$-elimination

This easily follows from property 9 in the definition of *sufficiently strong prover*.

Hence we have established that the proof status value stays $\top$ while checking θ. Now we still need to show that it stays $\top$ while checking Ψ'. Let Γ be the premise list resulting from checking θ. We need to show that $P(\Gamma \vdash^? PL(\Psi')) = 1$. Since D is a natural deduction derivation of $\mathbf{t}(\Phi_1), \ldots, \mathbf{t}(\Phi_k) \vdash \mathbf{t}(\Psi)$, it ends in $\mathbf{t}(\Psi)$. This means that $PL(\mathbf{t}(\Psi))$ is in Γ. Now $P(\Gamma \vdash^? PL(\Psi')) = 1$ follows from property 1 in the definition of *sufficiently strong prover*. □

6.4.1 Completeness with respect to *PTL* semantics

In this section we will prove the following completeness theorem that establishes completeness of the proof checking algorithm with respect to *PTL* semantics:

Theorem 6.4.8. *Let φ be a valid nice PTL formula and let P be a sufficiently strong prover.*[25] *Then there is a PTL text θ such that $check_P(\theta \,\&\, \varphi) = \top$.*

For simplifying the exposition, we will assume that φ does not contain any ι. At the end of this section we discuss the problems encountered when φ contains ι and sketch a solution to these problems.

The proof for this completeness theorem is significantly more involved than the proof for the previous completeness theorem (Theorem 6.4.7). Before going into the details of the proof, we will present a naive approach to proving the theorem, and explain which problems are encountered. In this way we motivate the actual proof that follows this discussion.

The basic idea is that a valid *PTL* formula φ corresponds to a *PL* formula Φ such that $CMTN \models \Phi$. Then there is a proof of Φ from *CMTN* in the above natural-deduction proof calculus, and as above we should be able to transform this proof into a proof of φ in *PTL*. There are, however, two related problems with this idea:

- When proof-checking φ, the proof checking algorithm will have to check not only non-presuppositional but also presuppositional proof obligations. This amounts to checking $T \neq u$ under the locally active premise list for every term t appearing in the proof. But in the natural deduction proof of Φ, there are no limitations for occurrences of terms: Terms that may turn out to equal u may appear in the proof and make it impossible to transform the proof into a proof of φ in which all terms can be shown to be defined.

- In the *PTL*-to-*PL* translation used in the proof checking algorithms, *PTL* quantifiers are always rendered in *PL* by quantifiers restricted to objects not equal to u. But the natural deduction proof of Φ may contain quantifiers not restricted in such a way. These can not be imitated in *PTL*.

[25] Note that the notion *sufficiently strong prover* will be defined differently in this section than in the previous one.

- *PTL* has no means to explicitly speak about the undefinedness object u. In the proof checking algorithm, presuppositional premises of the form $T \neq u$ get added to the premise list, and in the processing of a def(t)-construct, such presuppositional premises are changed to non-presuppositional premises, so that subformulae of the form $T \neq u$ can appear in all conceivable positions. But formulae containing u in another fashion cannot be imitated in *PTL*.

In order to combat these problems, we will define a modified natural deduction calculus, which adheres to three restrictions corresponding to these three problems:

- Terms may only appear in contexts in which they have been shown not to equal u.
- The only quantification allowed in the language of this modified calculus is restricted existential quantification of the form $\exists x \neq u\ \varphi$.
- The constant u may only occur at the end of formulae of the form $T \neq u$.

One central proposition needed for proving the above completeness theorem is that this restricted natural-deduction calculus is complete.

First we define a modification $PL_{\neq u}$ of the standard first-order predicate logic *PL*: In $PL_{\neq u}$, the quantifiers $\exists$ and $\forall$ are lacking, and we instead have two quantifiers $\exists_{\neq u}$ and $\forall_{\neq u}$. Every signature of a $PL_{\neq u}$-language must contain the constant symbol u, but this constant symbol may only occur at the end of subformulae of the form $\neg t = u$. Instead of $\exists_{\neq u} x\ \Phi$ and $\forall_{\neq u} x\ \Phi$ we usually write $\exists x \neq u\ \Phi$ and $\forall x \neq u\ \Phi$ respectively. This notation also explains the intended semantics of these new quantifiers.

Note that the premises in the active premise list of the proof checking algorithm as well as the conjectures of the proof obligations sent to the prover are *PL* formulae that correspond in a natural way to $PL_{\neq u}$ formulae: Quantifiers appear in them only with restriction to objects not equal to u (i.e. in subformulae of form $\exists x\ (x \neq u \wedge \Phi)$ and $\forall x\ (x \neq u \rightarrow \Phi)$), and the constant symbol u appears only at the end of subformulae of the form $\neg t = u$. A *PL* formula Ψ of this form can be translated in a canonical way into a $PL_{\neq u}$ formula $\mathbf{t}_{\neq \mathbf{u}}(\Psi)$ by replacing all occurrences of subformulae of the form $\exists x\ (x \neq u \wedge \Phi)$ and $\forall x\ (x \neq u \rightarrow \Phi)$ by $\exists_{\neq u}\ \Phi$ and $\forall_{\neq u}\ \Phi$ respectively.

Before defining the modified natural deduction calculus, we need to define the concept of *projected presuppositions* of a formula, which will be used to make precise what we mean by the restriction that terms may only appear in contexts in which they have been shown not to equal u. The idea is that any complex term T triggers the presupposition that $T \neq u$. If a complex term appears in a complex formula, this presupposition gets projected in the way explained in section 3.2. We now give a recursive formal definition of the set $p(\Phi)$ of projected presuppositions of a $PL_{\neq u}$-formula Φ:

Definition 6.4.9. Let Φ be a $PL_{\neq u}$-formula over the signature of *CMTN*. If Φ is atomic, then

$$p(\Phi) := \begin{cases} \{T \neq u \mid T \text{ is a proper subterm of } T_0\} & \text{if } \Phi \text{ is of the form } T_0 = u \\ \{T \neq u \mid T \text{ is a term appearing in } \Phi\} & \text{otherwise.} \end{cases}$$

Furthermore,

$$\begin{aligned}
p(\neg\Phi) &:= p(\Phi)\\
p(\Phi \wedge \Psi) &:= p(\Phi) \cup \{\Phi \to X \mid X \in p(\Psi)\}\\
p(\Phi \to \Psi) &:= p(\Phi) \cup \{\Phi \to X \mid X \in p(\Psi)\}\\
p(\Phi \vee \Psi) &:= p(\Phi) \cup p(\Psi)\\
p(\Phi \leftrightarrow \Psi) &:= \{\Psi \to X \mid X \in p(\Phi)\} \cup \{\Phi \to X \mid X \in p(\Psi)\}\\
p(\exists x \neq u\ \Phi) &:= \{\forall x \neq u\ X \mid X \in p(\Phi)\}\\
p(\forall x \neq u\ \Phi) &:= \{\forall x \neq u\ X \mid X \in p(\Phi)\}.
\end{aligned}$$

Example 6.4.10. Let Φ be the $PL_{\neq u}$-formula

$$\exists x \neq u\ (x \neq 0 \wedge (app_1(f,x) = 0 \vee app_1(g, app_1(g,x)) = 0)).$$

Then the projected presuppositions of Φ are

$$\begin{aligned}
&\forall x \neq u\ (x \neq 0 \to app_1(f,x) \neq u),\\
&\forall x \neq u\ (x \neq 0 \to app_1(g,x) \neq u), \text{ and}\\
&\forall x \neq u\ (x \neq 0 \to app_1(g, app_1(g,x)) \neq u).
\end{aligned}$$

Note that the relation of being a projected presupposition of a formula is transitive: A projected presupposition of a projected presupposition of Φ is already a projected presupposition of Φ. In the above example, the first two projected presuppositions of Φ do not have any projected presuppositions, whereas the third one has the second one as projected presupposition.

Just as we worked with the restricted language $PL_{\neg,\to,\bot,\exists}$ in the previous section in order to simplify the exposition and minimize the list of conditions that the prover has to satisfy, we will use a restriction $PL^-_{\neq u}$ of $PL_{\neq u}$ and a translation $\mathbf{t}$ from $PL_{\neq u}$-formulae to $PL^-_{\neq u}$-formulae in this section. This translation has to be faithful to the above definition of projected presuppositions, in the sense that $p(\mathbf{t}(\Phi))$ has to be $\mathbf{t}[p(\Phi)]$. For this purpose, we will have to keep the connective $\vee$ additionally to the connectives $\neg$ and $\to$ in our restricted language $PL^-_{\neq u}$. The quantifier $\forall_{\neq u}$ is dropped, so that we only retain the quantifier $\exists_{\neq u}$. Now $\mathbf{t}_{\neq \mathbf{u}}$ is defined recursively by

$$\begin{aligned}
\mathbf{t}_{\neq \mathbf{u}}(\Phi) &:= \Phi \text{ for atomic } \Phi\\
\mathbf{t}_{\neq \mathbf{u}}(\neg\Phi) &:= \neg\mathbf{t}_{\neq \mathbf{u}}(\Phi)\\
\mathbf{t}_{\neq \mathbf{u}}(\Phi \wedge \Psi) &:= \neg(\mathbf{t}_{\neq \mathbf{u}}(\Phi) \to \neg\mathbf{t}_{\neq \mathbf{u}}(\Psi))\\
\mathbf{t}_{\neq \mathbf{u}}(\Phi \to \Psi) &:= \mathbf{t}_{\neq \mathbf{u}}(\Phi) \to \mathbf{t}_{\neq \mathbf{u}}(\Psi)\\
\mathbf{t}_{\neq \mathbf{u}}(\Phi \vee \Psi) &:= \mathbf{t}_{\neq \mathbf{u}}(\Phi) \vee \mathbf{t}_{\neq \mathbf{u}}(\Psi)\\
\mathbf{t}_{\neq \mathbf{u}}(\Phi \leftrightarrow \Psi) &:= \neg(\neg(\mathbf{t}_{\neq \mathbf{u}}(\Phi) \to \mathbf{t}_{\neq \mathbf{u}}(\Psi)) \vee \neg(\mathbf{t}_{\neq \mathbf{u}}(\Psi) \to \mathbf{t}_{\neq \mathbf{u}}(\Phi)))\\
\mathbf{t}_{\neq \mathbf{u}}(\exists x \neq u\ \Phi) &:= \exists x \neq u\ \mathbf{t}_{\neq \mathbf{u}}(\Phi)\\
\mathbf{t}_{\neq \mathbf{u}}(\forall x \neq u\ \Phi) &:= \neg\exists x \neq u\ \neg\mathbf{t}_{\neq \mathbf{u}}(\Phi).
\end{aligned}$$

Now we define the proof calculus of *presuppositional natural deduction* over $PL^-_{\neq u}$ as follows: Proofs can be formed out of the ten rules of inference specified below in the usual fashion of natural deduction calculi, under the restriction

that a formula Φ may only appear in the proof in a position where all projected presuppositions of Φ have already been established. By this we mean that every projected presupposition Ψ of Φ must precede Φ in the proof, and that the open assumptions at the position where Ψ was deduced do not contain any assumption that is not open at the position where Φ is deduced. Here are the ten rules of inference for presuppositional natural deduction:

$$\frac{\neg\Phi \to \bot}{\Phi}\ \text{proof by contradiction} \qquad \frac{\Phi \quad \neg\Phi}{\bot}\ \neg\text{-elimination}$$

$$\frac{\begin{array}{c}\Phi\\ \vdots\\ \Psi\end{array}}{\Phi \to \Psi}\ \to\text{-introduction} \qquad \frac{\Phi \to \Psi \quad \Phi}{\Psi}\ \to\text{-elimination}$$

$$\frac{\neg\Phi \to \Psi}{\Phi \vee \Psi}\ \vee\text{-introduction} \qquad \frac{\Phi \vee \Psi}{\neg\Phi \to \Psi}\ \vee\text{-elimination}$$

$$\frac{\Phi\frac{T}{x} \quad T \neq u}{\exists x \neq u\ \Phi}\ \exists_{\neq u}\text{-introduction} \qquad \frac{\exists x \neq u\ \Phi}{\begin{array}{l}\text{Declare: } v\\ v \neq u\\ \Phi\frac{v}{x}\end{array}}\ \exists_{\neq u}\text{-elimination}$$

$$\frac{}{T = T}\ =\text{-introduction} \qquad \frac{T_1 = T_2 \quad \Phi\frac{T_1}{x}}{\Phi\frac{T_2}{x}}\ =\text{-elimination}$$

We write $\Gamma \vdash_p \Phi$ to mean that there is a proof of Φ from Γ in presuppositional natural deduction.

Note that the calculus of presuppositional natural deduction can be viewed as a way of modelling reasoning about partial functions and potentially undefined terms. In this thesis, we use this calculus only as a tool for proving the second completeness theorem of the proof checking algorithm. But at the same time we believe this calculus to be an interesting object of study in its own right.

Now one of the three central propositions needed for proving the second completeness theorem for the proof checking algorithm states that presuppositional natural deduction is complete:

Proposition 6.4.11. *Let Γ be a set of $PL^{-}_{\neq u}$-formulae such that for every formula $\Psi \in \Gamma$ and every formula $X \in p(\Psi)$, $\Gamma \vdash_p X$. Let Φ be a $PL^{-}_{\neq u}$-formula such that $\Gamma \models \Phi$ and $\Gamma \models \Psi$ for all $\Psi \in p(\Phi)$. Then $\Gamma \vdash_p \Phi$.*

Proof. We will prove the theorem by induction over the number of presuppositional proof obligations of Φ. (We choose this measure of formula complexity for our induction, because we need the property that the complexity of formulae in $p(\Phi)$ is always less than the complexity of Φ.)

Now by the inductive hypothesis and the transitivity of the projected presupposition relation, we may assume that $\Gamma \vdash_p \Psi$ for all $\Psi \in p(\Phi)$. Suppose for a contradiction that $\Gamma \nvdash_p \Phi$.

First we need to establish that $\Gamma \cup \{\neg\Phi\} \nvdash_p \bot$. Suppose for a contradiction that $\Gamma \cup \{\neg\Phi\} \vdash_p \bot$. The proof for this may serve as the antecedent for the $\to$-introduction rule for concluding $\neg\Phi \to \bot$ from assumptions Γ only. But in order

to ensure that the restrictions about projected presuppositions are fulfilled, we need to append the proofs for $\Gamma \vdash \Psi$ for all $\Psi \in p(\Phi)$ before the application of the $\rightarrow$-introduction rule. By an application of the proof by contradiction rule, we can then conclude Φ from the assumptions Γ, contrary to the assumption that $\Gamma \nvdash_p \Phi$.

We will now extend the set $\Gamma \cup \{\neg\Phi\}$ to a larger set Γ^+ such that $\Gamma^+ \nvdash_p \bot$ that will allow us to construct a structure in which Γ and $\neg\Phi$ hold, contrary to the assumption that $\Gamma \models \Phi$. The construction of Γ^+ is a modification of the completeness proof of normal natural deduction with variable declaration in Velleman (2006), which itself was a modification of a familiar proof due to Henkin (see Enderton (1972)).

While extending $\Gamma \cup \{\neg\Phi\}$ to Γ^+, we will add new constant symbols to the language L of $\Gamma \cup \{\neg\Phi\}$. We call the thus enriched language L^+.

We want to ensure that Γ^+ satisfies the following properties:

1. For every L^+-formula Ψ, $p(\Psi) \subseteq \Gamma^+$ iff either $\Psi \in \Gamma^+$ or $\neg\Psi \in \Gamma^+$.

2. For every formula of the form $\exists x \neq u\ \Psi(x)$ in Γ^+, there is a constant symbol c in the signature of L^+ such that $\Psi(c) \in \Gamma^+$ and $c \neq u \in \Gamma^+$.

We will now show how to construct Γ^+ such that it satisfies these properties and the properties already mentioned above.

Let $c_0, c_1, \ldots$ be infinitely many constant symbols not appearing in L. Let L_c be the language whose signature is the signature of L extended by $c_0, c_1, \ldots$. Let $\Phi_1, \Phi_2, \ldots$ be an enumeration of L_c-formulae in which every formula appears infinitely many times. We now recursively construct a sequence of pairs $(L_0, \Gamma_0), (L_1, \Gamma_1), \ldots$, where each L_i is a language and each Γ_i is a set of L_i-formulae satisfying the following two properties:

(a) For every formula $\Psi \in \Gamma_i$ and every formula $X \in p(\Psi)$, $\Gamma_i \vdash_p X$.

(b) $\Gamma_i \nvdash_p \bot$.

For the base case of the recursive construction, we let (L_0, Γ_0) be $(L, \Gamma \cup \{\neg\Phi\})$. For the recursive step we distinguish three cases:

1. If Φ_i is an L_i-formula such that $\Phi_i \notin \Gamma_i$ and $p(\Phi_i) \subseteq \Gamma_i$, we set $L_{i+1} := L_i$ and
$$\Gamma_{i+1} := \begin{cases} \Gamma_i \cup \{\Phi_i\} & \text{if } \Gamma_i \cup \{\Phi_i\} \nvdash_p \bot \\ \Gamma_i \cup \{\neg\Phi_i\} & \text{otherwise.} \end{cases}$$

2. If Φ_i is in Γ_i and has the form $\exists x \neq u\ \Psi(x)$, we let L_{i+1} be the extension of L_i by the constant symbol c_i and set $\Gamma_{i+1} := \Gamma_i \cup \{\Psi(c_i), c_i \neq u\}$.

3. In all other cases, we set $(L_{i+1}, \Gamma_{i+1}) := (L_i, \Gamma_i)$.

We now establish by induction that the two required properties hold for all Γ_i. The base case follows from the fact that $\Gamma \cup \{\neg\Phi\}$ satisfies the two required properties. For the inductive step, we again distinguish the three cases mentioned above.

1. Property (a) for Γ_{i+1} follows from the inductive hypothesis and the fact that $p(\Phi_i) = p(\neg\Phi_i) \subseteq \Gamma_i$. For property (b), note that if $\Gamma_i \cup \{\Phi_i\} \vdash_p \bot$, then $\Gamma_i \cup \{\neg\Phi_i\} \nvdash_p \bot$. For if $\Gamma_i \cup \{\neg\Phi_i\} \vdash_p \bot$, then the fact that $p(\neg\Phi_i \to \bot) = p(\Phi_i) \subseteq \Gamma_i$ ensures that we can use the proof by contradiction rule to conclude that $\Gamma_i \vdash_p \Phi_i$, i.e. $\Gamma_i \models \bot$, contrary to the inductive hypothesis.

2. For property (a), we need to establish that we can prove the projected presuppositions of $c_i \neq u$ and $\Psi(c_i)$ from Γ_{i+1}. $c_i \neq u$ does not have any projected presuppositions. The projected presuppositions of $\Psi(c_i)$ follow from $c_i \neq u$ together with the projected presuppositions of $\exists x \neq u\ \Psi(x)$, which are derivable from Γ_i by the inductive hypothesis: Let $X(x) \in p(\Psi(x))$. Then $X(c_i) \in p(\Psi(c_i))$ and $\neg\exists x \neq u\ \neg X(x) \in p(\exists x \neq u\ \Psi(x))$. We can derive $X(c_i)$ from $\neg\exists x \neq u\ \neg X(x)$ by deriving $\exists x \neq u\ \neg X(x)$ and hence $\bot$ from the assumption $\neg X(c_i)$. The requirements about projected presupposition in this derivation of $X(c_i)$ can be assumed to be fulfilled by an induction over the number of projected presuppositions of formulae in $p(\Psi(c_i))$.

   For establishing property (b) in this case, suppose for a contradiction that $\Gamma_{i+1} \vdash_p \bot$. We now establish a contradiction to the inductive hypothesis by transforming this proof into a proof for $\Gamma_i \vdash_p \bot$. For this we first apply $\exists_{\neq u}$-elimination to $\exists x \neq u\ \Psi(x)$, to infer the proof lines "Declare: v", $v \neq u$ and $\Psi(v)$. Now we transform the proof of $\bot$ from $\Gamma_i \cup \{\Psi(c_i), c_i \neq u\}$ into one from $\Gamma_i \cup \{\Psi(v), v \neq u\}$ be replacing c by v everywhere. Additionally we need to ensure that the syntactic condition about projected presuppositions is satisfied. The projected presuppositions of $v \neq u$ and $\Psi(v)$ follow from Γ_i in the same way as the projected presuppositions of $c_i \neq u$ and $\Psi(c_i)$ treated in the property (a) case above. Additionally, since Γ_i does not contain c_i, for every projected presupposition $X(c_i)$ on which the derivation of $\bot$ from $\Gamma_i \cup \{\Psi(c_i), c_i \neq u\}$ depends, $X(v)$ can be established from $\Gamma_i \cup \{\Psi(v), v \neq u\}$.

3. In this case, both required properties trivially follow from the inductive hypothesis.

Now we let $L^+ := \bigcup L_i$ and $\Gamma^+ := \bigcup \Gamma_i$. One can now easily see that Γ^+ has all the required properties.

Now we use the set Γ^+ for building a structure $\mathcal{A}$ in which Γ and $\neg\Phi$ hold. For this, we first define an equivalence relation on the ground terms of L^+ as follows: $T_1 \simeq T_2$ iff either $T_1 = T_2 \in \Gamma^+$ or $T_1 \neq u \notin \Gamma^+$ and $T_2 \neq u \notin \Gamma^+$. It can be easily seen that $\simeq$ actually is an equivalence relation.

The domain of $\mathcal{A}$ is defined to be the set of equivalence classes of $\simeq$ (we denote the $\simeq$-equivalence class of T as $[T]$). Function and relation symbols are interpreted as expected: $f^{\mathcal{A}}([T_1], \ldots, [T_n]) := [f(T_1, \ldots, T_n)]$, and $R^{\mathcal{A}} := \{([T_1], \ldots, [T_n]) \mid R(T_1, \ldots, T_n) \in \Gamma^+\}$. One can easily establish that with this definition the interpretation function is well-defined, and that the interpretation of a term T in $\mathcal{A}$ is always $[T]$.

Now we still need to show that $\mathcal{A} \models \Gamma \cup \{\neg\Phi\}$. For this we will show the stronger result that for every L^+-formula Φ such that $p(\Phi) \subseteq \Gamma^+$, $\mathcal{A} \models \Phi$ iff $\Phi \in \Gamma^+$. We show this by induction over the complexity of L^+-formulae. We always assume that $p(\Phi) \subseteq \Gamma^+$.

For the base case, suppose that Φ is an atomic formula. If Φ is of the form $T_1 = T_2$, then T_2 cannot be u, and $\mathcal{A} \models T_1 = T_2$ iff $T_1 \simeq T_2$ iff $T_1 = T_2 \in \Gamma^+$ (we cannot have $T_i \neq u \notin \Gamma^+$ for $i = 1, 2$, since $T_i \neq u \in p(\Phi) \subseteq \Gamma^+$). If Φ is of the form $R(T_1, \ldots, T_n)$, then $\mathcal{A} \models R(T_1, \ldots, T_n)$ iff $([T_1], \ldots, [T_n]) \in R^{\mathcal{A}}$ iff $R(T_1, \ldots, T_n) \in \Gamma^+$.

Now suppose Φ is of the form $\neg\Psi$. First suppose that $\mathcal{A} \models \Phi$. Then $\mathcal{A} \not\models \Psi$, i.e. by the inductive hypothesis (which we can apply since $p(\Psi) = p(\Phi) \subseteq \Gamma^+$), $\Psi \notin \Gamma^+$. But since $p(\Psi) \subseteq \Gamma^+$, we know that either $\Psi \in \Gamma^+$ or $\neg\Psi \in \Gamma^+$, so $\neg\Psi \in \Gamma^+$. Conversely, suppose $\neg\Psi \in \Gamma^+$. Then $\Psi \notin \Gamma^+$, for else we would have $\Gamma^+ \vdash_p \bot$ by $\neg$-elimination. So by the inductive hypothesis, $\mathcal{A} \not\models \Psi$, i.e. $\mathcal{A} \models \Phi$.

Next suppose that Φ is of the form $\Psi \to X$. First suppose that $\mathcal{A} \models \Phi$, i.e. either $\mathcal{A} \models \neg\Psi$ or $\mathcal{A} \models X$.

Case 1: $\mathcal{A} \models \neg\Psi$

In this case, we have $\neg\Psi \in \Gamma^+$ by the same reasoning as in the $\Phi = \neg\Psi$ case above. Now suppose for a contradiction that $\neg(\Psi \to X) \in \Gamma^+$. Let $p(\Psi)$ be $\{\Psi_1, \ldots, \Psi_n\}$, and let $p(X)$ be $\{X_1, \ldots, X_m\}$. Then $p(\Psi \to X) = \{\Psi_1, \ldots, \Psi_n, \Psi \to X_1, \ldots, \Psi \to X_m\} \subseteq \Gamma^+$. But then the following is a derivation of $\bot$ from Γ^+:

$$
\begin{array}{l}
\Psi_1 \\
\vdots \\
\Psi_n \\
\Psi \to X_1 \\
\vdots \\
\Psi \to X_m \\
\neg(\Psi \to X) \\
\neg\Psi \\
\left|\begin{array}{l}
\quad \Psi \\
\hline
\quad X_1 \\
\quad \vdots \\
\quad X_m \\
\quad \left|\begin{array}{l} \neg X \\ \hline \bot \end{array}\right. \\
\quad X
\end{array}\right. \\
\Psi \to X \\
\bot
\end{array}
$$

So $\neg(\Psi \to X) \notin \Gamma^+$, i.e. $\Psi \to X \in \Gamma^+$, as required.

Case 2: $\mathcal{A} \models X$

In this case, we have $X \in \Gamma^+$ by the inductive hypothesis. Again, assume for a contradiction that $\neg(\Psi \to X) \in \Gamma^+$. Then the following is a derivation of $\bot$ from Γ^+:

$$
\begin{array}{l}
\Psi_1 \\
\vdots \\
\Psi_n \\
\Psi \to X_1 \\
\vdots \\
\Psi \to X_n \\
\neg(\Psi \to X) \\
X \\
\left|\begin{array}{l}
\Psi \\
\hline
X_1 \\
\vdots \\
X_m \\
X
\end{array}\right. \\
\Psi \to X \\
\bot
\end{array}
$$

So in both cases $\Psi \to X \in \Gamma^+$.

For the converse direction, suppose that $\Psi \to X \in \Gamma^+$. We need to show that $\mathcal{A} \models \Phi \to X$. Suppose that $\mathcal{A} \models \Phi$. Now it is enough to show $\mathcal{A} \models X$. By the inductive hypothesis applied to Φ, $\Phi \in \Gamma^+$. Suppose for a contradiction that $X \notin \Gamma^+$.

Again we write $p(\Psi)$ as $\{\Psi_1, \ldots, \Psi_n\}$ and $p(X)$ as $\{X_1, \ldots, X_m\}$. For every $1 \leq i \leq m$, $\Psi \to X_i \in \Gamma^+$. By an induction over the number of projected presuppositions of the X_i's, one can show that for every $1 \leq i \leq m$, $X_i \in \Gamma^+$: The inductive hypothesis is that we already have all X_j's with less projected presuppositions than X_i in Γ^+, and we suppose for a contradiction that $\neg X_i \in \Gamma^+$. By $\to$-elimination applied to $\Psi \to X_i$ and Ψ, we can deduce X_i and hence $\bot$ from Γ^+. The syntactical restriction about projected presuppositions in this deduction is fulfilled by the inductive hypothesis, since the projected presuppositions of X_i are projected presuppositions of X that have less projected presuppositions than X_i. So $\neg X_i \notin \Gamma^+$. Again by the inductive hypothesis, we know that either X_i or $\neg X_i$ is in Γ^+. So $X_i \in \Gamma^+$.

Since all the projected presuppositions of X are in Γ^+, we can now conclude that $\neg X \in \Gamma^+$. But then we can derive $\bot$ from $\{\Psi \to X, \Psi, \neg X\} \subseteq \Gamma^+$, which is the required contradiction. So $X \in \Gamma^+$, i.e. $\mathcal{A} \models X$ by the inductive hypothesis applied to X.

The case that Φ is of the form $\Psi \vee X$ is similar to the case that Φ is of the form $\Psi \to X$, but actually simpler because of the simpler structure of projected

presuppositions of $\Psi \vee X$. We leave the details to the interested reader.

Finally, suppose Φ is of the form $\exists x \neq u\ \Psi(x)$.

First suppose that $\mathcal{A} \models \exists x \neq u\ \Psi(x)$. Then there is a $[T] \in \mathcal{A}$ such that $\mathcal{A} \models T \neq u$ and $\mathcal{A} \models \Psi(T)$ (since the interpretation of T in $\mathcal{A}$ is $[T]$). By the inductive hypothesis, we then have that $T \neq u$ and $\Psi(T)$ are in Γ^+. But since $p(\exists x \neq u\ \Psi(x)) \subseteq \Gamma^+$, we can now conclude that $\Gamma^+ \vdash_p \exists x \neq u\ \Psi(x)$, and hence that $\exists x \neq u\ \Psi(x) \in \Gamma^+$.

Conversely suppose that $\exists x \neq u\ \Psi(x) \in \Gamma^+$. By property (ii) of Γ^+, we know that there is a constant symbol c in the signature of L^+ such that $c \neq u$ and $\Psi(c)$ are in Γ^+. By the inductive hypothesis, $\mathcal{A} \models c \neq u$ and $\mathcal{A} \models \Psi(c)$, so $\mathcal{A} \models \exists x \neq u\ \Psi(x)$, as required. □

For concisely stating the second central proposition needed for the proof of the second completeness theorem, we will temporarily work with a somewhat weaker proof checking algorithm than the one defined in section 6.2. This weaker proof checking algorithm is obtained from the above proof checking algorithm by deleting the second clause in the definition of *check_limitedness* and the first two clauses in the definition of *exist_check*. In the proof of the second completeness theorem, we will see why a proposition about this weakened proof checking algorithm is relevant to proving a theorem about our original proof checking algorithm.

In the proof of the second completeness theorem, we will have to take *CMTN* axioms into account at a place where we need to use $PL_{\neq u}$ rather than *PL*. For this, we need to define a $PL_{\neq u}$ variant $CMTN_{\neq u}$ of the *CMTN* axioms. One problem is that the *CMTN* axioms express the fact that any *CMTN* relation yields a false statement when one of its arguments is the undefinedness object: We cannot possibly express this fact in $PL_{\neq u}$, because in $PL_{\neq u}$ we can only mention u in formulae of the form $T \neq u$ and not as an argument of a relation other than $=$. A related problem is that the *CMTN* axioms express that any *CMTN* function yields the undefinedness object when one of its arguments is the undefinedness problem. Again, we cannot express this in $PL_{\neq u}$.

So $CMTN_{\neq u}$ will be equivalent to the *CMTN* axioms in all respects apart from its lack of information about the value of relations and functions at u. To make this more precise, we define a set $\mathbb{U}$ of *PL*-formulae as follows:

$$
\begin{aligned}
\mathbb{U} :=& \mathbb{U}_1 \cup \mathbb{U}_2, \text{ where} \\
\mathbb{U}_1 :=& \{\neg C(u), \neg N(u), \neg B(u), \neg U(u), \neg L(u), \\
& \forall x\ \neg x \in u, \forall x\ \neg u \in x, \\
& \forall x\ \neg M(x,u), \forall x\ \neg M(u,x), \\
& \forall x\ \neg T(x,u), \forall x\ \neg T(u,x), \\
& \forall x\ nth(x,u) = u, \forall x\ nth(u,x) = u\}, \text{ and} \\
\mathbb{U}_2 :=& \{\Phi \mid \Phi \text{ is an instance of the } CMTN \text{ Undefinedness Axiom Schema} \\
& \quad \text{or the } CMTN \text{ Tuple Undefinedness Axiom Schema}\}
\end{aligned}
$$

$\mathbb{U}$ expresses the information that any *CMTN* relation yields a false statement when one of its arguments is the undefinedness object and that any function applies to the undefinedness object yields the undefinedness object. Now we will define a set $CMTN_{\neq u}$ of $PL_{\neq u}$ sentences corresponding in a natural way to the axioms of *CMTN* and satisfying the following three properties:

- Every sentence in $CMTN_{\neq u}$ corresponding to a non-comprehension $CMTN$ axiom logically follows from the non-comprehension $CMTN$ axioms.
- Every $CMTN$ axiom logically follows from $CMTN_{\neq u} \cup \mathbb{U}$.[26]
- For every sentence Φ in $CMTN_{\neq u}$, the sentences in $p(\Phi)$ are logically valid.

Definition 6.4.12. For every $CMTN$ axiom Φ not listed below, the $CMTN_{\neq u}$ correspondent of Φ is defined as a syntactical modification of Φ in the following way: Every quantifier $\forall x$ or $\exists x$ in Φ is replaced by $\forall x \neq u$ or $\exists x \neq u$ respectively, every atomic formula of the form $T = u$ is replaced by $\neg\neg T = u$, and every atomic formula Φ not of the form $T = u$ and containing complex terms $T_1, \ldots, T_n$ is replaced by $(T_1 \neq u \wedge \ldots \wedge T_n \neq u \wedge \Phi)$ (where the ordering of $T_1, \ldots, T_n$ is such that if T_i is a proper subterm of T_j, then $i < j$).

Below we give a list of $CMTN$ axioms that either do not have a correspondent in $CMTN_{\neq u}$ or whose correspondent in $CMTN_{\neq u}$ is defined in a special way:

- $CMTN_{\neq u}$ correspondent of the Map Extensionality Axiom Schema: For $n \geq 1$ and $\bar{z}$ a variable list of length n: $\forall f \neq u\ \forall g \neq u\ (M(f,n) \wedge M(g,n) \wedge \forall_{\neq u}\bar{z}\ ((f(\bar{z}) \neq u \leftrightarrow g(\bar{z}) \neq u) \wedge (f(\bar{z}) \neq u \wedge g(\bar{z}) \neq u \rightarrow f(\bar{z}) = g(\bar{z}))) \rightarrow f = g)$
- $CMTN_{\neq u}$ correspondent of the first Boolean axiom: $\forall x \neq u\ (B(x) \leftrightarrow x = \top \vee x = \bot) \wedge \top \neq u \wedge \bot \neq u$
- Unlimitedness of Undefinedness and the axioms in the $CMTN$ Undefinedness Axiom Schema and the $CMTN$ Tuple Undefinedness Axiom Schema have no correspondent in $CMTN_{\neq u}$ (as these are already in $\mathbb{U}$).

One can easily check that with this definition $CMTN_{\neq u}$ satisfies the above mentioned properties.

Before we can state the second central proposition needed for the proof of the second completeness theorem for the proof checking algorithm, we need to remind the reader that some of the proof obligations produced during the proof checking are not needed, in the sense that their result does not influence the final result of the proof checking. Those proof obligations that actually do have to be successfully checked for ensuring the proof checking is overall successful we will call the proof obligation *checked* by the algorithm.

Now we can state the second central proposition, which ensures that every proof obligation of a valid nice PTL formula can be encoded by a valid $PL_{\neq u}$ formula with valid projected presuppositions:

Proposition 6.4.13. *Let φ be a valid nice PTL formula. Then for every proof obligation p of the form $\Gamma_p \vdash^{?}_{S_p} \Psi_p$ checked by check(φ), there is a finite subset Δ_p of $CMTN_{\neq u}$ such that $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ is a valid $PL_{\neq u}$ formula all of whose projected presuppositions are valid $PL_{\neq u}$ formulae.*

Before proving this proposition, we need to introduce some further concepts. First recall that all proof obligations in the PTL proof checking algorithm are called using the *update* function, whose second argument determines whether the

[26] Since the notions of *structure* in PL and in $PL_{\neq u}$ are identical, it is easy to make sense of such assertions of logical entailment despite the fact that we are talking about formulae in these two syntactically distinct formalisms.

prover is called for a presupposition check (in which case the second argument is 0) or for an assertion check (in which case it is 1). In the first case we also say that the proof obligation checked by the algorithm at that point is a *presuppositional proof obligation*. In the second case we call it a *non-presuppositional proof obligation*.

We now still need the following three definitions:

Definition 6.4.14. Given a list Γ of $PL_{\neq u}$ formulae and a $PL_{\neq u}$ formula Φ, we recursively define the $PL_{\neq u}$ formula $\Gamma \Rightarrow \Phi$ as follows:

- $\langle\rangle \Rightarrow \Phi := \Phi$.
- $(\langle\Psi\rangle \oplus \Gamma) \Rightarrow \Phi := \Psi \rightarrow (\Gamma \Rightarrow \Phi)$.

One can easily see that $\Gamma \Rightarrow \Phi$ is logically equivalent to $\bigwedge \Gamma \rightarrow \Phi$. This motivates the following definition:

Definition 6.4.15. Given two sets A and B of $PL_{\neq u}$ formulae, we write $A \subsetneq\!\!\sim\, B$ iff every formula Φ in A can be obtained by a formula Ψ in B by replacing subformulae of Ψ of the form $\bigwedge \Gamma \rightarrow X$ by $\Gamma \Rightarrow X$.

By inspection of the definition of the proof checking algorithm, one can easily verify the following two lemmas:

Lemma 6.4.16. *If $\Gamma \vdash^?_S \Phi$ is a proof obligation produced by the proof checking algorithm, then $p[\mathbf{t}_{\neq\mathbf{u}}[\Gamma]] \cup p(\mathbf{t}_{\neq\mathbf{u}}(\Phi)) \subsetneq\!\!\sim\, \mathbf{t}_{\neq\mathbf{u}}[\Gamma]$.*

Lemma 6.4.17. *If p_1 is a proof obligation of the form $\Gamma_{p_1} \vdash^?_{S_{p_1}} \Phi_{p_1}$ produced by the proof checking algorithm and $\Phi \in p(\bigwedge \mathbf{t}_{\neq\mathbf{u}}[\Gamma_{p_1}] \rightarrow \mathbf{t}_{\neq\mathbf{u}}(\Psi_{p_1}))$, then there is a proof obligation p_2 of the form $\Gamma_{p_2} \vdash^?_{S_{p_2}} \Phi_{p_2}$ produced by the proof checking algorithm earlier than p_1 such that Φ is $\Gamma_{p_2} \Rightarrow \Psi_{p_2}$.*

For the proof of Proposition 6.4.13, we will make use of the following lemma:

Lemma 6.4.18. *Let φ be a valid nice PTL formula. Then for every proof obligation p of the form $\Gamma_p \vdash^?_{S_p} \Psi_p$ checked by check(φ), CMTN$\cup\,\Gamma_p \models \Psi_p$.*

In order to prove this lemma by an induction over the complexity of φ, one actually needs to prove the following stronger lemma, whose rather involved proof we will only sketch:

Lemma 6.4.19. *Assume the following properties:*

(i) *φ is a semi-nice PTL text.*

(ii) *$\mathbb{T}$ is a PTL-PL term lists such that $PL^{-1}(\mathbb{T}) \oplus \mathbf{qt}(\varphi)$ is pairwise independent.*

(iii) *All MHF terms of φ are composed of terms in $PL^{-1}(\mathbb{T})$.*

(iv) *Γ is a premise list such that all MHF terms in Γ are composed of PTL_{sk} symbols and terms in $\mathbb{T}$.*

(v) *For every CMTN model M, every Γ-skolem-assignment S and every M-assignment g such that $dom(g) = PL^{-1}(\mathbb{T})$ and $M + S, g \models \Gamma$, we have $def(\llbracket\varphi\rrbracket^M_g)$.*

(vi) p *is a proof obligation of the form* $\Gamma_p \vdash^?_{S_p} \Psi_p$ *checked by* $\mathit{check\_text}(\varphi, \Gamma, \mathbb{T}, \top)$.

(vii) At least one of the following two properties holds:

(a) p *is a presuppositional proof obligation.*

(b) For every CMTN model M*, every* Γ*-skolem-assignment* S *and every* M*-assignment* g *such that* $M + S, g \models \Gamma$*, we have* $[\![\varphi]\!]^M_g \neq \emptyset$.

Then $\mathit{CMTN} \cup \Gamma_p \models \Psi_p$.

Note. One can easily see that Lemma 6.4.18 follows from Lemma 6.4.19 by setting $\Gamma = \mathbb{T} = \langle\rangle$.

Proof sketch. We prove this lemma by an induction over the complexity of φ.

In the base case, φ can be either a *PTL* term t or of the form $R(t_1, \ldots, t_n)$ for a logical relation symbol R and *PTL* terms $t_1, \ldots, t_n$. Since the two cases are similar, we will only discuss the first case. In this case, the proof checking algorithm will calculate $\mathit{read\_term}(t, \Gamma, \mathbb{T}, \top) = (\Gamma', T, \nu)$ in the course of calculating $\mathit{check\_text}(t, \Gamma, \mathbb{T}, \top)$. We can divide the proof obligations checked by $\mathit{check\_text}(t, \Gamma, \mathbb{T}, \top)$ into three groups:

- The proof obligations checked by $\mathit{read\_term}(t, \Gamma, \mathbb{T}, \top)$:

  Informally speaking, these check that under the assumptions Γ, the term t is a defined term. This corresponds to our semantic assumption in (v) that in every model of Γ, φ (i.e. t) is defined.

  Formally, given one such proof obligation p of the form $\Gamma_p \vdash^?_{S_p} \Psi_p$, we need to show that $\mathit{CMTN} \cup \Gamma_p \models \Psi_p$. First we note by inspection of the definition of *read_term* that Γ' contains a formula Ψ'_p that is either identical to Ψ_p or a Skolemized version of Ψ_p, and Γ_p must be $\Gamma'_{\Psi'_p}$ (for this we need the fact that we are using the restricted proof checking algorithm in which only the first two clauses in the definition of *exist_check* have been deleted). Then by assertion 3 of the *read_term* Soundness Lemma and our semantic assumption in (v) mentioned above, we can conclude that for every *CMTN* model M, every $\Gamma'_{\Psi'_p}$-skolem-assignment S and every M-assignment g such that $M + S, g \models \Gamma'_{\Psi'_p}$, we have $M + S, g \models \Psi_p$. In other words, the axioms of *CMTN* together with $\Gamma_p = \Gamma'_{\Psi'_p}$ logically imply Ψ_p, as required.

- The proof obligation $\Gamma' \vdash^? B(T)$:

  Let M be a model of $\mathit{CMTN} \cup \Gamma'$. We need to show that $M \models B(T)$, i.e. that $\frac{M}{g}(T) \in B^M$. By assertion 4 of the *read_term* Soundness Lemma, $\frac{M}{g}(T) = \frac{M}{g}(t)$. By assumption (v), we know that $\mathit{def}([\![\varphi]\!]^M_g)$, which by the definition of *PTL* semantics (Definition 5.2.2) implies that $\frac{M}{g}(t) \in B^M$, as required.

- The proof obligation $\Gamma' \cup \langle B(T) \rangle \vdash^? T = \top$.

  Since this proof obligation is not presuppositional, we may assume that property (vii) (b) holds.

  Let M be a model of $\mathit{CMTN} \cup \Gamma' \cup \langle B(T) \rangle$. We need to show that $M \models T = \top$, i.e. that $\frac{M}{g}(T) = \top^M$. Again we have $\frac{M}{g}(T) = \frac{M}{g}(t)$

by assertion 4 of the *read_term* Soundness Lemma. By property (vii) (b), we have $[\![\varphi]\!]^M_g \neq \emptyset$, which by the definition of *PTL* semantics implies that $\frac{M}{g}(t) = \top^M$.

The many cases of the inductive step are also similar, and we only discuss the case where φ is of the form $\varphi_1 \wedge \varphi_2$. In this case, calculating *check_text*$(\varphi_1 \wedge \varphi_2, \Gamma, \mathbb{T}, \top)$ involves calculating

$$\textit{read\_text}(\varphi_1, \langle\rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma'_0, \mathbb{T}_1, \Phi_1, \mu_0), \text{ and} \tag{6.29}$$

$$\textit{read\_text}(\varphi_2, \langle\rangle, \Gamma'_0 \oplus \langle\Phi_1\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0) = (\Gamma'_1, \mathbb{T}_2, \Phi_2, \mu_1). \tag{6.30}$$

We can divide the proof obligations checked by *check_text*$(\varphi_1 \wedge \varphi_2, \Gamma, \mathbb{T}, \top)$ into three groups:

- The proof obligation checked by *read_text*$(\varphi_1, \langle\rangle, \Gamma, \mathbb{T}, \top)$:

  By the definition of *read_text*, these proof obligations are also checked by *check_text*$(\varphi_1, \langle\rangle, \Gamma, \mathbb{T}, \top)$. Hence for these proof obligations, the required result follows directly from the inductive hypothesis for φ_1.

- The proof obligation checked by *read_text*$(\varphi_2, \langle\rangle, \Gamma'_0 \oplus \langle\Phi_1\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0)$:

  Suppose that p is a proof obligation of the form $\Gamma_p \vdash^?_{S_p} \Psi_p$ checked by *read_text*$(\varphi_2, \langle\rangle, \Gamma'_0 \oplus \langle\Phi_1\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0)$. So p is a presuppositional proof obligation checked by *check_text*$(\varphi_2, \Gamma'_0 \oplus \langle\Phi_1\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0)$.

  The goal is to apply the inductive hypothesis to *check_text*$(\varphi_2, \Gamma'_0 \oplus \langle\Phi_1\rangle, \mathbb{T} \oplus \mathbb{T}_1, \mu_0)$, which will allow us to conclude the required result that there is a finite subset Γ_p^{CMTN} of the set of *CMTN* axioms such that $\bigwedge(\Gamma_p \cup \Gamma_p^{CMTN}) \rightarrow \Psi_p$ is a valid *PL* formula. Apart from some syntactical prerequisites, we need to ensure that the semantic assumption (v) is fulfilled for this application of the inductive hypothesis. In other words, we need to show that for every *CMTN* model M, every $\Gamma'_0 \oplus \langle\Phi_1\rangle$-skolem-assignment S and every M-assignment k such that $dom(k) = PL^{-1}(\mathbb{T} \oplus \mathbb{T}_1)$ and $M + S, k \models \Gamma'_0 \oplus \langle\Phi_1\rangle$, we have *def*$([\![\varphi_2]\!]^M_k)$.

  So let M be a *CMTN* model, S be a $\Gamma'_0 \oplus \langle\Phi_1\rangle$-skolem-assignment S and k be an M-assignment such that $dom(k) = PL^{-1}(\mathbb{T} \oplus \mathbb{T}_1)$ and $M + S, k \models \Gamma'_0 \oplus \langle\Phi_1\rangle$. Let g be $k|_{PL^{-1}(\mathbb{T})}$. Let S' be the Γ-skolem-assignment such that $S \succeq S'$. Then $M + S', g \models \Gamma$. So by assumption (v), we have *def*$([\![\varphi_1 \wedge \varphi_2]\!]^M_g)$. By the definition of *PTL* semantics, we have that *def*$([\![\varphi_1]\!]^M_g)$ and that for every $h \in [\![\varphi_1]\!]^M_g$, *def*$([\![\varphi_1]\!]^M_h)$. By the second fact, it is enough to show that $k \in [\![\varphi_1]\!]^M_g$. This now follows from the equivalence between (a) and (c) in assertion 6 of the *read_text* Soundness Lemma (with $\mathbb{T}_0 = \langle\rangle$ in this application of the *read_text* Soundness Lemma).

- The proof obligation $\Gamma' \vdash^? \exists_{\mathbb{T}_1} (\Phi \wedge \exists_{\mathbb{T}_2} \Psi)$.

  Since this proof obligation is not presuppositional, we may assume that property (vii) (b) holds.

  Assume that M is a model of $CMTN \cup \Gamma'$. We need to show that $M \models \exists_{\mathbb{T}_1} (\Phi \wedge \exists_{\mathbb{T}_2} \Psi)$. First we define the *CMTN* model M' to be the restriction of M to L_{CMTN}. Next we define a Γ'-skolem-assignment S over M' by setting $S(sk_i^n) := (sk_i^n)^M$. Furthermore we define an M'-assignment g

with $dom(g) = PL^{-1}(\mathbb{T})$ by $g(t) := t^M$. Then $M' + S, g \models \Gamma'$. It is now enough to show that $M' + S, g \models \exists_{\mathbb{T}_1} (\Phi \wedge \exists_{\mathbb{T}_2} \Psi)$.

By property (vii) (b), there is a $k \in [\![\varphi_i \wedge \varphi_2]\!]_g^{M'}$. By assertion 4 of the Detailed Soundness Lemma applied to $check\_text(\varphi_1 \wedge \varphi_2, \Gamma, \mathbb{T}, \top)$, we can conclude that $k[\mathbb{T}_1 \oplus \mathbb{T}_2]g$ and that k verifies $(\Gamma' \oplus \langle \Phi_1, \Phi_2 \rangle) - \Gamma$ over $M' + S$. This in turn implies that $M + S', k \models \Phi_1$ and $M + S', k \models \Phi_2$. Since Φ_1 does not contain any term from $\mathbb{T}_2$, we can now conclude that $M' + S, g \models \exists_{\mathbb{T}_1} (\Phi \wedge \exists_{\mathbb{T}_2} \Psi)$, as required. □

Proof of Proposition 6.4.13. Suppose that φ is a valid nice *PTL* formula. Let p be a proof obligation of the form $\Gamma_p \vdash^?_{S_p} \Psi_p$ checked by $check(\varphi)$. Then by Lemma 6.4.18, $CMTN \cup \Gamma_p \models \Psi_p$, i.e. $CMTN_{\neq u} \cup \mathbb{U} \cup \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \models \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$.

First we show that there is a finite $\Delta_p \subset CMTN_{\neq u}$ such that $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ is logically valid. By compactness it is enough to show that $CMTN_{\neq u} \cup \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \models \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$. So let M be a model of $CMTN_{\neq u} \cup \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$. We need to show that $M \models \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$. We modify M to a model M' of $CMTN_{\neq u} \cup \mathbb{U} \cup \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$ by changing the interpretations of the *CMTN* relation and function symbols in M as follows: For a *CMTN* relation symbol R, $R^{M'}$ is the result of deleting all tuples containing u^M from R^M. For an n-ary *CMTN* function symbol f,

$$f^{M'}(x_1, \ldots, x_n) := \begin{cases} u^M & \text{if } x_i = u^M \text{ for some } 1 \leq i \leq n \\ f^M(x_1, \ldots, x_n) & \text{otherwise.} \end{cases}$$

The fact that M' models $\mathbb{U}$ is trivial. Now we still need to show that M' models $CMTN_{\neq u} \cup \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$. For this we first note that if for some $PL_{\neq u}$ formula φ $M \models p(\varphi)$, then $M \models \varphi$ iff $M' \models \varphi$. Since the elements of $p[CMTN_{\neq u}]$ are logically valid, we can already conclude that M' models $CMTN_{\neq u}$. Additionally, by Lemma 6.4.16, $p[\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]] \subseteq \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$. This allows us to prove by induction that M' also models $\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$ and hence Γ_p. Hence the assumption that $CMTN \cup \Gamma_p \models \Psi_p$ implies that M' models Ψ_p and hence $\mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$. Again by Lemma 6.4.16, $p(\mathbf{t}_{\neq \mathbf{u}}(\Psi_p)) \subseteq \mathbf{t}_{\neq \mathbf{u}}[\Gamma_p]$, which allows us to conclude that M models $\mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$, as required.

Furthermore, by Lemma 6.4.17, Lemma 6.4.18 and the fact that the projected presuppositions of $CMTN_{\neq u}$ are logically valid, we have that every projected proof obligation of $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ is logically valid, as required. □

The third central proposition needed for the proof of the second completeness theorem for the proof checking algorithm states that in a certain sense the $CMTN_{\neq u}$ comprehension axiom can be proof-checked in *PTL* with a *sufficiently strong prover*. Before we can state the third central proposition needed in the proof of the second completeness theorem, we need to redefine the notion of a *sufficiently strong prover*. For this we first need a preliminary definition, similar to the definition of $\Gamma \Rightarrow \Phi$ above:

Definition 6.4.20. Given *PL* formula Φ, a list Γ of *PL* formulae and a list V of *PL* variable lists such that Γ and V are of the same length, we define $\Gamma \Rightarrow_V \Phi$ recursively as follows:

- $\langle\rangle \Rightarrow_{\langle\rangle} \Phi := \Phi$.

- $(\langle\Psi\rangle \oplus \Gamma) \Rightarrow_{\langle\langle v_1,\ldots,v_k\rangle\rangle\oplus V} \Phi := \forall_{\langle v_1,\ldots,v_k\rangle}(\Psi \to (\Gamma \Rightarrow_V \Phi))$.

One can easily see that if the variables in an element of V do not appear in an earlier element of Γ, then $\Gamma \Rightarrow_V \Phi$ is logically equivalent to $\forall_{\bigoplus V}(\bigwedge \Gamma \to \Phi)$.

The below redefinition of *sufficiently strong prover* is quite involved. The important point is that whether a certain proof obligation satisfies one of the properties listed there is a decidable condition, in contrast with the general condition of being a proof obligation whose conjecture follows from the premises, which is only a semidecidable condition. (See Enderton (1972) for an introduction to decidability and semidecidability.)

Definition 6.4.21. A prover P is called *sufficiently strong* if it satisfies the following properties:

1. If Γ is a premise list and a Φ is a *PL* formula such that $(\neg\Phi \to \bot) \in \Gamma$, then $P(\Gamma \vdash^? \Phi) = 1$.
2. If $\Phi \in \Gamma$ and $\neg\Phi \in \Gamma$, then $P(\Gamma \vdash^? \bot) = 1$.
3. If $\Phi \to \exists_{\langle v_1,\ldots,v_n\rangle}\ (\Psi_1 \wedge \ldots \wedge \Psi_n) \in \Gamma$ and Ψ_n does not contain free occurrences of $v_1, \ldots, v_n$, then $P(\Gamma \vdash^? (\Phi \to \Psi_n)) = 1$.
4. If $(\Phi \to \Psi) \in \Gamma$ and $\Phi \in \Gamma$, then $P(\Gamma \vdash^? \Psi) = 1$.
5. If $\neg\Phi \to \Psi \in \Gamma$, then $P(\Gamma \vdash^? \Phi \vee \Psi) = 1$.
6. If $\Phi \vee \Psi \in \Gamma$, then $P(\Gamma \vdash^? \neg\Phi \to \Psi) = 1$.
7. If T is a *PL* term and $\Psi_1\frac{T}{x}, \Psi_2\frac{T}{x} \in \Gamma$, then $P(\Gamma \vdash^? \exists_{\langle x\rangle}\ (\Psi_1 \wedge \Psi_2)) = 1$.
8. If $\exists_{\langle x\rangle}\ \Phi \in \Gamma$, then $P(\Gamma \vdash^? \exists_{\langle v\rangle}\ \Phi\frac{v}{x}) = 1$.
9. If T is a *PL* term, then $P(\Gamma \vdash^? T = T) = 1$.
10. If $T_1 = T_2 \in \Gamma$ and $\Phi\frac{T_1}{x} \in \Gamma$, then $P(\Gamma \vdash^? \Phi\frac{T_2}{x}) = 1$.
11. Suppose that $\forall_{\langle v_1,\ldots,v_n\rangle}(\Phi \to \Psi_1 \wedge \ldots \wedge \Psi_n) \in \Gamma$ and that $1 \leq k \leq n$. Suppose furthermore that $T_1, \ldots, T_n$ are $PL_{\neq u}$ terms such that $T_1 \neq u$, $\ldots$, $T_n \neq u$ and $\Phi\frac{T_1}{v_1}\ldots\frac{T_n}{v_n}$ are in Γ. Then $P(\Gamma \vdash^? (\Psi_k\frac{T_1}{v_1}\ldots\frac{T_n}{v_n})) = 1$.
12. If $T \neq u \in \Gamma$, then $P(\Gamma \vdash^? \exists_{\langle x\rangle}\ x = T) = 1$.
13. Suppose that $k \geq 1$ and that Γ contains the *CMTN* Map Extensionality Axiom for $n = k$ and all instances of the Undefinedness Axiom for which $n + m = k$. Suppose furthermore that $P(\bar{z})$ is some *PL* formula with parameters and that Γ contains premises of the form $\forall_{\langle z_1,\ldots,z_n\rangle}(\neg P(\bar{z}) \to \neg app_k(T_1, \bar{z}) \neq u)$, $\forall_{\langle z_1,\ldots,z_n\rangle}(P(\bar{z}) \to \Phi \wedge app_k(T_2, \bar{z}) = app_k(T_1, \bar{z}))$ and $\forall_{\langle z_1,\ldots,z_n\rangle}(P(\bar{z}) \leftrightarrow app_k(T_2, \bar{z}) \neq u)$, $M(T_1, n)$ and $M(T_2, n)$. Then $P(\Gamma \vdash^? T_2 = T_1) = 1$.
14. If Γ contains a premise of the form $\langle\bigwedge_{i=1}^{k_1}\Phi_1^i, \ldots, \bigwedge_{i=1}^{k_m}\Phi_m^i\rangle \Rightarrow_V (\Psi_1 \wedge \ldots \wedge \Psi_n \wedge X)$ as well as $\Phi_1^1, \ldots, \Phi_1^{k_1}, \ldots, \Phi_m^1, \ldots, \Phi_m^{k_m}$, then $P(\Gamma \vdash^? X) = 1$.
15. If Γ contains premises $\Phi_1, \ldots, \Phi_n$ and x is a variable, then $P(\Gamma \vdash^? \exists_x\ \bigwedge_{i=1}^{n}\Phi_i) = 1$.

16. Suppose that Δ is a finite set of non-comprehension $CMTN_{\neq u}$ axioms. Suppose that Γ contains $CMTN$ axioms for the arities covered in Δ (see section 6.1.6 for a clarification of what whis means). Suppose furthermore that Γ contains a formula of the form $\bigwedge(\Gamma' \cup \Delta) \rightarrow \Psi$, where Γ' is a finite subset of Γ. $P(\Gamma \vdash^? \Psi) = 1$.

Now we can state the third central proposition needed for the proof of the second completeness theorem for the proof checking algorithm:

Proposition 6.4.22. *For every $CMTN_{\neq u}$ comprehension axiom Θ, there is a PTL formula θ such that for every sufficiently strong prover P, $check_P(\theta) = \top$, and such that the premise list that is active after checking θ contains Θ.*

Proof. There are four $CMTN_{\neq u}$ comprehension axiom schemata. The proposition has to be proved for each of them separately.

We call a $PL_{\neq u}$ formula Ψ *legitimate* iff every atomic formula Φ in Ψ that is not of the form $T = u$ is part of a conjunction of the form $(T_1 \neq u \wedge \ldots \wedge T_n \neq u \wedge \Phi)$, where $T_1, \ldots, T_n$ are the complex terms appearing in Φ, ordered in such a way that if T_i is a proper subterm of T_j, then $i < j$.

Fix a sufficiently strong prover P.

Class Comprehension

The $CMTN_{\neq u}$ Class Comprehension Axiom Schema with parameters made explicit is as follows:

> Given a legitimate $PL_{\neq u}$ formula $F(\bar{p}, y)$ that does not have x among its free variables, the following is an axiom:
> $\forall_{\neq u}\bar{p}\ (\forall y \neq u\ (F(\bar{p}, y) \rightarrow L(y)) \rightarrow \exists x \neq u\ (C(x) \wedge \forall y \neq u\ (y \in x \leftrightarrow F(\bar{p}, y))))$

Let Θ be an instance of this axiom schema. We now define a PTL text θ with the required properties as follows:

$$\theta := \exists p_1\ \ldots \exists p_k\ \top \rightarrow ((\exists y\ F(\bar{p}, y) \rightarrow L(y)) \rightarrow \exists x\ (C(x) \wedge (\exists y\ \top \rightarrow (y \in x \leftrightarrow F(\bar{p}, y)))))$$

One can easily verify that the premise list that is active after checking θ contains Θ. We now need to show that the prover P successfully checks all proof obligations checked by $check(\theta)$.

Since $F(\bar{p}y)$ is legitimate, all presuppositional proof obligation triggered within θ have the conjecture among the premises and are hence certainly successfully checked. When processing the subformula $\exists x \neq u\ (C(x) \wedge \forall y \neq u\ (y \in x \leftrightarrow F(\bar{p}, y)))$ of θ, the proof checking algorithm calls *exist_check* with arguments of the following form:

$$exist\_check(1, \Gamma, \mathbb{T}, \exists_x\ (C(x) \wedge \forall_y(y \in x \leftrightarrow F(\bar{p}, y))), \mu)$$

Here Γ contains a premise of the form $\forall_y(F(\bar{p}, y) \rightarrow L(y))$. Since the existentially quantified formula that *exist_check* has to check is of the right form, the first clause in the definition of *exist_check* may be applied. The algorithm now has to check that $P(\Gamma \oplus \langle F(\bar{p}, y)\rangle \vdash^? L(y)) = 1$. Since Γ contains $\forall_y(F(\bar{p}, y) \rightarrow L(y))$, this follows directly from case 11 of the definition of *sufficiently strong prover*.

Set Comprehension

This case is similar to the above case, only that the second instead of the first clause in the definition of *exist_check* has to be applied.

Map Comprehension

This case is similar to the Functionality case detailed out below (and even somewhat simpler). It requires the properties 11 and 15 of the definition of *sufficiently strong prover*.

Functionality

The Functionality Axiom Schema says that under certain syntactic and semantic restrictions, all maps that can result from the Map Comprehension Axioms Schema are limited. We prove this in *PTL* by implicitly introducing a new map g that takes exactly the same values as the given map f. The precautions taken for Functionality in the $\varphi \to \theta$ case of the proof checking algorithm ensure that g is limited, and Map Extensionality ensures that $g = f$, which implies that f is limited.

The $CMTN_{\neq u}$ Functionality Axiom Schema with parameters made explicit is as follows:

> Given legitimate $PL_{\neq u}$ formulae $P(\bar{p}, \bar{z})$ and $R(\bar{p}, \bar{z}, x)$ that do not contain the symbol L, the following is an axiom:
> $\forall_{\neq u}\bar{p}\ (L(p_1) \wedge \ldots \wedge L(p_k) \to \forall_{\neq u}\bar{z}\ \forall_{\neq u}x\ (R(\bar{p}, \bar{z}, x) \to L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x)) \to \forall_{\neq u}f\ (M(f, n) \wedge \forall_{\neq u}\bar{z}\ (P(\bar{p}, \bar{z}) \to f(\bar{z}) \neq u \wedge R(\bar{p}, \bar{z}, f(\bar{z}))) \wedge \forall_{\neq u}\bar{z}\ (\neg P(\bar{p}, \bar{z}) \to \neg f(\bar{z}) \neq u) \to L(f)))$

Let Θ be an instance of this axiom schema. We will now define a *PTL* text $\theta := \theta_1\ \&\ \theta_2$ with the required properties. The subtext θ_2 of θ has the form

$$\exists p_1\ \ldots \exists p_k\ (L(p_1) \wedge \ldots \wedge L(p_k)) \to (\psi_1 \to (\exists f\ \psi_2(f) \to L(f))),$$

where

$$\begin{aligned} \psi_1 &:= \exists z_1\ \ldots \exists z_k\ \exists x\ R(\bar{p}, \bar{z}, x) \to L(z_1) \wedge \cdots \wedge L(z_n) \wedge L(x), \text{ and} \\ \psi_2(f) &:= M(f, n) \wedge (\exists z_1\ \ldots \exists z_k\ P(\bar{p}, \bar{z}) \to \mathrm{def}(f(\bar{z})) \wedge R(\bar{p}, \bar{z}, f(\bar{z}))) \wedge \\ &\quad (\exists z_1\ \ldots \exists z_k\ \neg P(\bar{p}, \bar{z}) \to \neg \mathrm{def}(f(\bar{z}))). \end{aligned}$$

One can easily see that θ_2 and hence θ gives rise to the premise Θ. The task is now to define θ_1 in such a way that $check_P(\theta_1\ \&\ \theta_2) = \top$ for every sufficiently strong prover P.

We define the *PTL* text ξ to be

$$(\exists z_1\ \ldots \exists z_n\ P(\bar{p}, \bar{z}) \to \mathrm{def}(f(\bar{z}))\ \&\ \exists g(\bar{z})\ g(\bar{z}) = f(\bar{z}))\ \&\ g = f.$$

Now θ_1 is defined to be

$$\exists p_1\ \ldots \exists p_k\ (L(p_1) \wedge \ldots \wedge L(p_k)) \to (\psi_1 \to (\exists f\ \psi_2(f) \to \xi \wedge L(f))).$$

We now need to show that the prover P successfully checks all proof obligations checked by $check(\theta_1\ \&\ \theta_2)$. Since $P(\bar{p}, \bar{z})$ and $R(\bar{p}, \bar{z}, x)$ are legitimate,

all presuppositional proof obligation checked triggered within ψ_1 and ψ_2 have the conjecture among the premises and are hence certainly successfully checked. Apart from these, there are five further proof obligations checked by *check*$(\theta_1 \,\&\, \theta_2)$:

- The def$(f(\bar{z}))$ in ξ triggers a proof obligation of the form $\Gamma \vdash^? f(\bar{z}) \neq u$. Here Γ contains $P(\bar{p}, \bar{z})$ and $\forall_{\langle z_1,\ldots,z_n\rangle}(P(\bar{p}, \bar{z}) \to f(\bar{z}) \neq u \wedge R(\bar{p}, \bar{z}, f(\bar{z})))$ as well as $z_1 \neq u$, $\ldots$, $z_n \neq u$. Hence property 11 of the definition of *sufficiently strong prover* ensures that $P(\Gamma \vdash^? f(\bar{z}) \neq u) = 1$.

- The $\exists g(\bar{z})\ g(\bar{z}) = f(\bar{z})$ in ξ triggers a proof obligation of the form $\Gamma \vdash^? \exists_{\langle x\rangle} x = f(\bar{z})$. Here $f(\bar{z}) \neq u \in \Gamma$, so property 12 of the definition of *sufficiently strong prover* ensures that $P(\Gamma \vdash^? \exists_{\langle x\rangle} x = f(\bar{z})) = 1$.

- The $g = f$ in ξ triggers a proof obligation of the form $\Gamma \vdash^? g = f$. Γ contains the premises $\forall_{\langle z_1,\ldots,z_n\rangle}(\neg P(\bar{p}, \bar{z}) \to \neg f(\bar{z}) \neq u)$, $\forall_{\langle z_1,\ldots,z_n\rangle}(P(\bar{p}, \bar{z}) \to f(\bar{z}) \neq u \wedge g(\bar{z}) = f(\bar{z}))$ and $\forall_{\langle z_1,\ldots,z_n\rangle}(P(\bar{p}, \bar{z}) \leftrightarrow g(\bar{z}) \neq u)$ as well as $M(f, n)$ and $M(g, n)$ (the third and fifth resulting from the processing of the implication in ξ that implicitly introduced g). By the explanations in section 6.1.6, the premise list gets extended by the *CMTN* axioms needed for an application of property 13 of the definition of *sufficiently strong prover*, which allows us to conclude that $P(\Gamma \vdash^? g = f) = 1$.

- The $L(f)$ at the end of θ_1 triggers a proof obligation of the form $\Gamma \vdash^? L(f)$. One can easily see that in the processing of the implication in ξ that implicitly introduced g, the conditions for setting $\alpha = 1$ are fulfilled, so that $L(g)$ is added to the active premise list at the end of processing this implication. Since $g = f$ is in the premise list, property 10 of the definition of *sufficiently strong prover* ensures that $P(\Gamma \vdash^? L(f)) = 1$.

- The $L(f)$ at the end of θ_2 triggers a proof obligation of the form $\Gamma \vdash^? L(f)$. Here Γ is of the form $\Gamma^- \oplus \langle L(p_1), \ldots, L(p_k), \Psi_1, \Psi_2(f)\rangle$, where the premise resulting from θ_1 in Γ has the form

  $$\langle L(p_1) \wedge \ldots \wedge L(p_k), \Psi_1, \Psi_2(f)\rangle \Rightarrow_{\langle\langle p_1,\ldots,p_k\rangle,\langle\rangle,\langle f\rangle\rangle} (X \wedge L(f)).$$

  Hence property 14 of the definition of *sufficiently strong prover* ensures that $P(\Gamma \vdash^? L(f)) = 1$. □

We now restate the second completeness theorem for the proof checking algorithm before proving it:

Theorem 6.4.8. *Let φ be a valid nice PTL formula and let P be a sufficiently strong prover. Then there is a PTL text θ such that $check_P(\theta \,\&\, \varphi) = \top$.*

Proof for the case that φ does no contain any ι. Let $\mathbb{P} = \{p_1, \ldots, p_k\}$ be the set of proof obligations checked by *check*(φ). For every $p \in \mathbb{P}$, we write Γ_p for the premises of p without the added *CMTN* axioms (i.e. just the active premise list at that point in the proof checking), and Ψ_p for the conjecture of p.

Now we want to apply Proposition 6.4.13 to φ to conclude that for every $p \in \mathbb{P}$ there is a finite subset Δ_p of $CMTN_{\neq u}$ such that $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \to \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ is a valid $PL_{\neq u}$ formula with valid projected presuppositions. But for the statement of Proposition 6.4.13 we assumed a weakened proof checking

algorithm in which the second clause in the definition of *check_limitedness* and the first two clauses in the definition of *exist_check* have been deleted. Note that these deleted clauses are used by the original proof checking algorithm only when the proof obligations triggered therein are successfully checked. Let $\mathbb{P}'$ be the set of proof obligations checked by *check*(φ) which are and not proof obligations triggered by one of these clauses and successfully checked by the prover. Then we have the above conclusion only for $p \in \mathbb{P}'$ and not for all $p \in \mathbb{P}$. But for $p \in \mathbb{P} \setminus \mathbb{P}'$, the proof checking is at any rate successful, so that these proof obligation do not cause any problems.

Since φ does not contain ι, $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ does not contain any skolem function symbols and can hence be easily seen to be a $PL_{\neq u}$ formula over the language of *CMTN*.

Now by Proposition 6.4.11, we have $\vdash_p \bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ for every $p \in \mathbb{P}'$. In other words, there is a derivation D_p in presuppositional natural deduction for each of these $PL_{\neq u}$ formulae of the form $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$. Note that any $PL_{\neq u}$ formula appearing in any of these D_p's is a $PL_{\neq u}$ formula over the language of *CMTN*. We will now transform each D_p into a *PTL* text δ_p in a similar way as we transformed D into δ in the proof of the first completeness theorem (Theorem 6.4.7).

Again we consider D_p to be written in Fitch's indentation notation. First all occurrences of the quantifier $\exists_{\neq u}$ are replaced by the *PTL* quantifier $\exists$. Next for every variable declaration "Declare: v" appearing in D_p, we replace "Declare: v" and the proof lines $v \neq u$ and Φ that follow it by a single proof line $\exists v\, \Phi$ of the same indentation. Afterwards, every indented subproof of the form

$$\begin{array}{|l} \Phi \\ \hline \Psi_1 \\ \vdots \\ \Psi_n \end{array}$$

is transformed into a *PTL* text of the form $\Phi \rightarrow \Psi_1 \,\&\, \ldots \,\&\, \Psi_n$. This transformation is recursively applied to subproof including subproofs, and so on. On the highest level the subproofs and proof steps are conjuncted with & in the original order. Call the result of this transformation of D_p δ_p.

Let $\Theta_1, \ldots, \Theta_n$ be the $CMTN_{\neq u}$ comprehension axioms appearing in $\bigcup_{p \in P} \Delta_p$. Let $\theta_1, \ldots, \theta_n$ be the *PTL* texts that prove these axioms in the way specified in Proposition 6.4.22.

Now the required *PTL* text θ is defined to be $\delta_{p_1} \,\&\, \ldots \,\&\, \delta_{p_k} \,\&\, \theta_1 \,\&\, \ldots \,\&\, \theta_n$. We now have to show that P successfully checks all proof obligations checked in *check*($\theta \,\&\, \varphi$).

First we consider the presuppositional proof obligations checked in the subtext $\delta_{p_1} \,\&\, \ldots \,\&\, \delta_{p_k}$ of θ. These are always of the form $\Gamma \vdash^? T \neq u$, where T is a complex term occurring in some line Φ of some D_p. By the syntactical limitation in the definition of the presuppositional natural deduction calculus, the projected presuppositions of Φ must appear as lines in D_p preceding Φ and of the same indentation. One can easily see that their *PL* counterparts hence appear in Γ. Let $\Gamma' \Rightarrow_V T \neq u$ be the projected presupposition triggered by the occurrence of T in Φ corresponding to the occurrence of T in $\delta_{p_1} \,\&\, \ldots \,\&\, \delta_{p_k}$ that

we are currently considering. Before the proof checking algorithm encounters this occurrence of T, it must syntactically analyse Φ until it encounters this occurrence of T. By comparing the definition of the proof checking algorithm with the definition of projected presuppositions, one can easily see that this syntactical analysis involves adding the formulae in Γ' to the active premise list. Hence $\Gamma' \subseteq \Gamma$. But now the required result $P(\Gamma \vdash^? T \neq u) = 1$ follows from property 14 of the definition of *sufficiently strong prover*.

Next we consider the non-presuppositional proof obligations checked in $\delta_{p_1} \,\&\, \ldots \,\&\, \delta_{p_k}$. Each such proof obligation results from a proof line in one of the D_p's. Based on the ten rules of inference of presuppositional natural deduction, we can distinguish ten cases. The proof in each of these cases is similar to that in the proof of Theorem 6.4.7.

Now we consider the proof obligations checked in $\theta_1 \,\&\, \ldots \,\&\, \theta_n$. For every $1 \leq i \leq n$, we know by Proposition 6.4.22 that $check_P(\theta_i) = \top$. In other words, every proof obligation checked in θ_i is successfully checked by the prover P. Now when θ_i gets checked as part of $\theta_1 \,\&\, \ldots \,\&\, \theta_n$, the proof obligations to be checked by the prover are the same as when θ_i is checked by itself, only that the premise lists may be larger. But enlarging the premise list cannot change a proof obligation that a sufficiently strong prover necessarily checks successfully into one which in can no longer check successfully. Hence the proof obligations checked in θ_i within the checking of $\theta_1 \,\&\, \ldots \,\&\, \theta_n$ are successfully checked by P.

Finally we are only left with the proof obligations checked in φ. These are the proof obligations in $\mathbb{P}$ with an augmented premise list: For $p \in \mathbb{P}$, the proof obligation corresponding to p checked in φ as part of checking $\theta \& \varphi$ is of the form $\Gamma_\theta \oplus \Gamma_p \vdash^? \Psi_p$, where Γ_θ is the premise list that is active after checking θ. The proof obligations in $\mathbb{P} \setminus \mathbb{P}'$ are at any rate successfully checked by the definition of $\mathbb{P}'$. Now fix $p \in \mathbb{P}'$. D_p was constructed to be a derivation of $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$. One can easily see that the premise list that is active after checking δ_p, and hence also Γ_θ, contains the premise $\bigwedge(\Gamma_p \cup \Delta_p) \rightarrow \Psi_p$. The $CMTN_{\neq u}$ comprehension axioms contained in Δ_p are contained in Γ_θ since they have been proved in $\theta_1 \,\&\, \ldots \,\&\, \theta_n$. The $CMTN$ correspondents of the $CMTN_{\neq u}$ non-comprehension axioms contained in Δ_p are added to the premise list of the proof obligation $\Gamma_\theta \oplus \Gamma_p \vdash^? \Psi_p$ according to the explanations in section 6.1.6. Hence $P(\Gamma_\theta \oplus \Gamma_p \vdash^? \Psi_p) = 1$ follows from property 16 of the definition of *sufficiently strong prover*. □

If φ may contain ι, the proof obligations in $\mathbb{P}$ may contain skolem function symbols. In this case, the derivations D_p contain skolem function symbols, and hence the above definition of the δ_p's does not yield PTL texts, as the δ_p's still contain skolem function symbols. In order to solve this problem, one needs to modify Proposition 6.4.13 in such a way that the $PL_{\neq u}$ formula whose validity it asserts does not contain skolem function symbols. For this one has to define a notion of *deskolemization* of proof obligations.

If all skolem function symbols were 0-ary (i.e. if we had only skolem constants), then deskolemization would be trivial: We could just replace skolem constants by free variables. But for skolem function symbols of arity greater than 0, deskolemization is not a trivial task.

First one needs to note by inspection of the proof checking algorithm that for any proof obligation of the form $\Gamma \vdash^?_S \Phi$ produced by the algorithm, we have the following two properties:

- For every n-ary skolem function symbol sk_i^n appearing in Γ, there is a formula list Γ' such that Γ' does not contain sk_i^n and for every formula Ψ in Γ containing sk_i^n, there is a list V of variable lists such that $\bigoplus V$ has length n and Ψ is of the form $\Gamma' \Rightarrow_V X$, where all occurrences of sk_i^n in X are in terms of the form $sk_i^n(\bigoplus V)$.
- Every skolem functions symbol appearing in Φ is 0-ary.

Now if our proof obligation contains just one skolem function symbol sk_i^n, we can explain deskolemization by the following case distinction:

- Case 1: $n = 0$

  In this case we just replace sk_i^n by some variable that appears nowhere else in the proof obligation.

- Case 2: $n \geq 1$.

  In this case, we let $\Psi_1, \ldots, \Psi_k$ be the formulae in Γ containing sk_i^n and we let Γ', $X_1, \ldots, X_k$ and $V_1, \ldots, V_k$ be as in the first property above. Then we replace $\Psi_1, \ldots, \Psi_k$ in Γ by $\Gamma' \Rightarrow_{V_i} \exists x \neq u \ (\bigwedge X_i \frac{sk_i(\oplus V_i)}{x})$ for a fresh variable x.

If our proof obligation contains more than one skolem function symbol, we apply the above transformation recursively, starting with the skolem function symbol whose first occurrence occurs the latest.

Now in the assertion of Proposition 6.4.13, we need to replace the $PL_{\neq u}$ formula $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi_p)$ by $\bigwedge(\mathbf{t}_{\neq \mathbf{u}}[\Gamma'_p] \cup \Delta_p) \rightarrow \mathbf{t}_{\neq \mathbf{u}}(\Psi'_p)$, where $\Gamma' \vdash^? \Psi'$ is the deskolemization of $\Gamma \vdash^? \Psi$. The proof of this modified version of Proposition 6.4.13 is similar to the proof of the previous version of Proposition 6.4.13.

In order for the proof of Theorem 6.4.8 to work with this modified version of Proposition 6.4.13, we need to modify property 16 of the definition of *sufficiently strong prover* in such a way that it takes care of this deskolemization.

6.5 A proof checking algorithm using all three prover output values

In this section we sketch how the *PTL* proof checking algorithm has to be adapted if one wants to take into account the difference between an ATP's time-out and its finding a counterexample for a proof obligation. By taking into account both this threefold distinction of possible ATP outputs and the threefold distinction between possible semantic values of a *PTL* text according to the validity function v, we can distinguish seven possible outputs for the proof checking algorithm:

- The algorithm has determined that $v(\Theta) = u$. In that case it does not make sense for the algorithm to try to determine whether $v(\Theta) = \top$ or $v(\Theta) = \bot$. We call this result of the proof checking algorithm (-1).
- The algorithm cannot determine whether $v(\Theta) = u$, but based on the assumption that $v(\Theta) \neq u$, it can determine that $v(\Theta) = \top$. This result is called $(0, 1)$.

- The algorithm cannot determine whether $v(\Theta) = u$, but based on the assumption that $v(\Theta) \neq u$, it can determine that $v(\Theta) = \bot$. This result is called $(0, -1)$.
- The algorithm cannot determine whether $v(\Theta) = u$, and using the assumption that $v(\Theta) \neq u$, it cannot determine whether $v(\Theta) = \top$ or $v(\Theta) = \bot$. This result is called $(0, 0)$.
- The algorithm can determine that $v(\Theta) = \top$. This result is called $(1, 1)$.
- The algorithm can determine that $v(\Theta) = \bot$. This result is called $(1, -1)$.
- The algorithm can determine that $v(\Theta) \neq u$, but it cannot determine whether $v(\Theta) = \top$ or $v(\Theta) = \bot$. This result is called $(1, 0)$.

The first coordinate always expresses whether $v(\Theta)$ is defined according to the algorithm, and unless it has been determined not to be so, there is a second coordinate that expresses whether $v(\Theta) = \top$ according to the algorithm. We call the set of these seven possible results of the proof checking algorithm ρ.

We now use ρ instead of $\{u, \bot, \top\}$ as the set of proof status values in the proof checking algorithm. The initial proof status value in the algorithm is now $(1, 1)$ instead of $\top$. We now still need to adapt the update function in order to make it work on these seven proof status values.

Recall that the proof checking algorithm calls the prover for two different purposes: For checking the definedness of v it checks *presuppositional proof obligations*, whereas for checking whether $v(\Theta) = \top$ or $v(\Theta) = \bot$ it checks *non-presuppositional proof obligations*. If the prover returns 0 for some presuppositional proof obligations, the final result of the proof checking algorithm is (-1). If the prover returns 1 for all presuppositional proof obligations, then we distinguish three cases:

- The prover returns 1 for all non-presuppositional proof obligations. In that case the final result is $(1, 1)$.
- The prover returns -1 for some non-presuppositional proof obligations. In that case the final result is $(1, -1)$.
- The prover does not return -1 for any non-presuppositional proof obligations, but does return 0 for some non-presuppositional proof obligations. In that case the final result is $(1, 0)$.

If the prover does not return -1 for any presuppositional proof obligations, but does return 0 for some of them, we distinguish the three then possible final results $(0, 1)$, $(0, -1)$ and $(0, 0)$ in a similar way based on the prover output for non-presuppositional proof obligations.

Here is the adapted definition of the update function:

Definition 6.5.1. We define an *update* function *update* from $\rho \times \{0, 1\} \times \{-1, 0, 1\}$ to ρ by

$$update(\mu, i, j) := \begin{cases} \mu & \text{if } j = 1 \\ (-1) & \text{if } \mu = (-1) \text{ or if } i = 0 \text{ and } j = -1 \\ (0, l) & \text{if } \mu = (k, l),\ i = 0 \text{ and } j = 0 \\ (k, \min(j, l)) & \text{if } \mu = (k, l) \text{ and } i = 1 \end{cases}$$

Remark. It is easily checked that the case distinction in the previous definition covers all possible input argument combinations, and that for the input argument combinations that are covered by more than one case, the defined output value of *update* is the same no matter which case is chosen.

Now the adapted proof checking algorithm only differs in the initial proof status value and in the definition of the update function. One can now proof a stronger soundness theorem for this adapted proof checking algorithm:

Theorem 6.5.2. *Suppose that θ is a nice PTL text and that check$(\theta) = \nu$.*

1. *If $\nu = (-1)$, then $v(\theta) = u$.*
2. *If $\nu = (1, -1)$, then $v(\theta) = \bot$.*
3. *If $\nu = (1, 1)$, then $v(\theta) = \top$.*

Assertion 3 of this theorem easily follows from the previous soundness theorem (Theorem 6.3.1). To prove the first two assertions, one needs to adapt assertion 5 of the Detailed Soundness Lemma to $\mu + update(\mu, 1, v(\theta, M, g)) \geq \nu$, where the $(i_1, j_1) \geq (i_2, j_2)$ means that $i_1 \geq i_2$ and $j_1 \geq j_2$ (the proof status value (-1) should be read as $(-1, -1)$ for the sake of this definition of $\geq$).

Chapter 7

The controlled natural language of Naproche

In this chapter we describe the Naproche CNL, i.e. the controlled natural language of the Naproche system. The description of its syntax is divided between sections 7.2, 7.3 and 7.4, which respectively explain the structure of Naproche CNL texts, the syntax of the textual parts and the syntax of the symbolic parts of the Naproche CNL. Section 7.5 describes the semantics of the Naproche CNL by defining a translation from the Naproche CNL to Proof Text Logic.

The sections about the Naproche CNL syntax contain – besides clarifications and design motivations – a semi-formal characterization of the Naproche CNL syntax. A formal grammar of the Naproche CNL can be found in appendix A.

The Naproche CNL grammar is described here as it is in the implemented system Naproche 0.52. The scope of grammatical constructs from the language of mathematics included in the Naproche CNL has been influenced by a number of factors: The current Naproche CNL started in late 2008 as an adaptation of Attempto Controlled English (ACE) to the language of mathematics. This first version was in many respects simpler than ACE, since it lacked many ACE constructs not needed in the language of mathematics; but it already contained some grammatical constructs characteristic for the language of mathematics, *inter alia* a relatively rich grammar for symbolic mathematics and text structure above the sentence level.[1] The further development of the Naproche CNL was guided by two main driving forces:

- The application of the Naproche system to the beginnings of Landau's *Grundlagen der Analysis* and Euclid's *Elements*, which motivated extensions of the Naproche CNL which made increasingly faithful CNL reformulations of these texts possible.

- A general analysis of the language of mathematics for linguistically and logically interesting features, which were included in the Naproche CNL.

[1] In ACE, a text is always considered a simple string of sentences, without any structure imposed on it. The Naproche CNL, on the other hand, provided for structure above the sentence level, e.g. theorem-proof blocks and the nested introduction and retraction of assumptions. In section 7.2 we describe this *macro-grammar* of the current Naproche CNL.

Extensions to the language were always a possible source of ambiguities, so that robust disambiguation principles had to be enforced, of course under the constraint that they coincide as well as possible with the way mathematicians naturally disambiguate.

There are still many ways in which the current Naproche CNL could be extended in order to make it more expressive and flexible. We discuss some of the possible extensions in section 7.7. Additionally, some extensions that could easily be implemented are already mentioned alongside the description of the existing grammar.

One of the issues that has to be surmounted in order to treat mathematical symbolism directly in a computer program is its two-dimensionality. Mathematicians extensively use superscripts and subscripts and put terms above other terms as in the fraction notation. Naproche has already for some time adopted LaTeX for its input, so that in this thesis we restrict ourselves to parsing and disambiguating the LaTeX code that is used for generating mathematical formulae.[2] The reversion of a pictorial symbolic input into a LaTeX input or another linearization of it is certainly an interesting undertaking, but outside the scope of this thesis.

In order to cope efficiently with the diversity of possible LaTeX codes for a given symbolic output – e.g. `a^b` and `a^{b}` both producing a^b – we normalize the LaTeX input before the actual parsing process, in this case to `a^{b}`. For the rest of this chapter, we use this normalized LaTeX code whenever it is necessary for the explanation; when the LaTeX code is not necessary for the explanation, we use the typographic notation that depicts the mathematical symbols as they are commonly drawn and printed.

Note that all example sentences that are not explicitly stated to be in the natural language of mathematics rather than in the Naproche CNL are Naproche CNL sentences adhering to the grammar described in this chapter.

7.1 Quantterms and anaphoric accessibility

The grammar of the Naproche CNL allows for a special kind of symbolic terms, which correspond to the quantifiable terms of *PTL* (see section 5.2.1), and which we call *quantterms*. The simplest kind of quantterms are variables that are used in a natural language quantifier; but just as *PTL* allows quantification over more complex terms than variables, so does the Naproche CNL. We will describe quantterms in more detail in section 7.4.6. But we already need to say some words now about a special role that quantterms play in the Naproche CNL, namely the role of *anaphoric antecedents*.

An *anaphora* is a linguistic expression whose interpretation depends on the interpretation of a previously occurring expression, called the *anaphoric antecedent*. For the kind of anaphora included in the Naproche CNL, one can actually say more precisely that the anaphora *corefers* with the anaphoric antecedent, i.e. refers to the same object in the discourse domain. A standard example of anaphora in natural language are pronouns which corefer with some

[2] We restrict ourselves to standard LaTeX, i.e. without any user-defined macros. Additionally, we in some respects require the author to use *neat* LaTeX, e.g. to write the sine function using `\sin` rather than `sin` in order to distinguish it from the concatenation of the three variables s, i and n.

previous noun phrase in the discourse. The Naproche CNL does not include any pronouns, but does include a kind of anaphora very characteristic of the language of mathematics, namely variables. Consider for example the following sentence:

(1) There is a natural number n such that $n + n = n^2$.

The first occurrence of n appears in a natural language quantification and serves as an anaphoric antecedent of the three occurrences of n in "$n + n = n^2$". The usage of a variable as an anaphoric antecedent is the simplest case of a quantterm. Instead of n we could have – for example – used n_k in all four positions of n in the above example, in order to show that n depends on some other previously introduced (and anaphorically accessible) variable k. The first occurrence of n_k in this modified example is an example of a more complex quantterm; the latter three occurrences of n_k are just considered subterms of the term $n_k + n_k = n_k^2$, and are not considered quantterms.

Given a certain position in a text, not all previously mentioned expressions that have the potential of being used as anaphoric antecedents may actually be used as an anaphoric antecedent for an anaphora in that position. Consider for example (2) and (3):

(2) There is a natural number n such that $n = 4k$. Clearly n is even.

(3) *There is no natural number n such that $n = 4k$. Clearly n is even.

In both examples, the quantterm n in the first sentence has the potential of being used as anaphoric antecedent: Actually it is used as an anaphoric antecedent for the n in $n = 4k$ in both examples. But while in (2), it is also an anaphoric antecedent for the occurrence of n in the second sentence, in (3) the anaphoric link between the quantterm n in the first sentence and the occurrence of n in the second sentence does not work: The quantterm used in a negative quantifier of the form "there is no" is not *anaphorically accessible* outside the scope of the quantifier, whereas in the case of an affirmative existential quantifier like "there is a" it is.

The work by Hans Kamp and others on Discourse Representation Theory (see Kamp & Reyle, 1993) contains a well-established theory for *anaphoric accessibility*, i.e. for determining which expressions may serve as anaphoric antecedents for anaphora at a given position in a discourse. In Dynamic Predicate Logic, we have the notion of *active quantifiers at a position in a formula* (see section 3.1.1), which models anaphoric accessibility in a way equivalent to that of Discourse Representation Theory: When a natural language text fragment is translated into *DPL* in a canonical way, the expressions anaphorically accessible at a position in the natural language text correspond to the active quantifiers at the corresponding position in the translation. In Proof Text Logic, we have also defined a notion of active quantifiers at a position in a *PTL* text (see section 5.2.3), which models the anaphoric accessibility relation that is in place in the Naproche CNL.

The details of this anaphoric accessibility relation will become clear when we define the translation of Naproche CNL texts into *PTL* texts in section 7.5. But already in the sections about the syntax of the Naproche CNL, we sometimes need to speak about the quantterms accessible at a given position in text.

7.2 Structure of Naproche CNL texts

As pointed out in section 1.1, mathematical texts are highly structured in an explicit way. Likewise Naproche CNL texts can be structured explicitly. We call the rules by which texts are structured above the sentence level the *macro-grammar* of the Naproche CNL. The macro-grammar incorporates the most common ways of structuring mathematical texts in a standardized way.

Additionally to the usage of LaTeX for typesetting mathematical formulae in the Naproche CNL, some LaTeX commands for structuring a text can also be used. However, all text structuring possible in the Naproche CNL is also possible without the usage of such LaTeX commands.

While we describe the macro-grammar, we sometimes need to refer to special kinds of Naproche CNL sentences:

- Simple declarative sentence
- Assertion
- Assumption
- Definition
- Variable type specification
- Alternative notation specification

These sentence kinds will be defined in section 7.3.5.

There are various kinds of structural blocks into which a Naproche text can be structured:

- Axiom blocks
- Assumption-consequences blocks
- Theorem-proof blocks
- Definition blocks
- Case distinction blocks
- Statement list blocks
- Note blocks
- Labelled text blocks

To some extent these can be nested into one another. Below, we explain each of these kinds of structural blocks, specifying its internal structure and how it can be nested in other structural blocks.

The global structure of a Naproche text is that of a sequence of concatenated assertions and structural blocks. When specifying the internal structure of the various structural blocks below, we sometimes say that at some point in the structural block there can be *text*. Here again this means a sequence of concatenated assertions and structural blocks, but now with the restrictions about the nesting of structural blocks in place.

Potential ambiguities on the level of the macro-grammar are avoided by strict rules that determine where a structural block starts and ends. The beginning of some structural blocks is marked by a sentence consisting of a special word (e.g. "Axiom" for axiom blocks) potentially followed by a number or word that names the block. Given a special word x to be used in such a sentence, we call a sentence of this form an x *heading* (so "Axiom." and "Axiom 5." are axiom headings).

For some kinds of structural blocks, the beginning of a new paragraph can mark the end of the block.[3] But in all places where the beginning of a new paragraph cannot mark the end of a block, new paragraphs may be started without any influence to the structuring of the text.

Axiom blocks

The beginning and end of an axiom can be either marked by an axiom heading and the beginning of a new paragraph or by the LaTeX environment commands `\begin{axiom}` and `\end{axiom}`. The content of the axiom must be stated as zero or more assumptions followed by one or more assertions. Axiom blocks may not be nested into other structural blocks apart from assumption-consequences blocks and labelled text blocks.

Assumption-consequences block

An assumption-consequences block always starts with an assumption. Its end may be marked by a sentence that starts with the word "Thus" followed by a simple declarative sentence; in that case, the sentence starting with "thus" itself no longer belongs to the assumption-consequences block. If its end is not marked, an assumption-consequences block ends when the block inside which it is immediately nested ends.

Assumption-consequences blocks may appear inside any structural block that allows for general text in it (i.e. inside other assumption-consequences blocks, inside the proofs of theorem-proof blocks, inside case distinction blocks and inside labelled text blocks).

Theorem-proof blocks

A theorem-proof block consists of two sub-blocks, a *theorem block* and a *proof block*, which must directly follow each other in this order.

A theorem block can have theorem type "theorem" or "lemma" (the theorem types "proposition" and "corollary" are so far not implemented). It starts with a theorem heading or lemma heading depending on its type. The content of a theorem follows in the form of zero or more assumptions followed by one or more assertions. The end of a theorem block is marked by the beginning of the corresponding proof block.

A proof block consists of a sentence consisting just of the word "Proof" followed by text followed by a sentence consisting just of the word "Qed".

[3] A line break marked in LaTeX by `\\` is also considered the beginning of a new paragraph for this purpose.

Theorem blocks and proof blocks may alternatively be delimited using the LaTeX environment command pairs `\begin{theorem}`/`\end{theorem}` (or `\begin{lemma}`/`\end{lemma}`) and `\begin{proof}`/`\end{proof}` respectively.

Theorem-proof blocks of type "theorem" may not be nested into other structural blocks apart from assumption-consequences blocks and labelled text blocks. Theorem-proof blocks of type "lemma" may be nested inside any structural block that allows for general text in it apart from other theorem-proof blocks of type "lemma".

Definition blocks

The core of a definition block is the definition sentence. It may optionally be preceded by a definition heading. Additionally, it is possible to enclose the definition sentence in the LaTeX environment commands `\begin{definition}` and `\end{definition}`. Definition blocks may appear in any structural block that allows for general text in it.

Case distinction blocks

The main part of a case distinction is a list of consecutive cases. Each case consists of three parts in the following order:

- A sentence consisting of the word "case" followed by a number or word that names the case
- A simple declarative sentence that characterizes that case
- A text containing the proof of the desired result for the case in question

The entire case list may be preceded by a sentence marking the beginning of a case introduction. Such a sentence generally has the form "There are n possible cases", where n should be replaced by a number word; additionally, there may be an assertion trigger (see section 7.3.5) like "Now" or "Hence" at the beginning of the sentence.

Furthermore, the end of a case distinction may be marked by a an assertion which starts with the words "in all cases" or "in both cases" (or has these words after its assertion trigger).

Case distinction blocks may be nested inside any structural block that allows for general text in it. When a case distinction is nested inside another case distinction, the marking of the beginning and the end of the inner case distinction is obligatory (otherwise it would not be clear which cases belong to the inner case distinction and which to the outer case distinction).

Statement list blocks

Simple declarative sentences, assertions and assumptions may contain a *cataphoric metalinguistic noun phrase* (see section 7.3.3) like "the following property" or "the following axioms", which announces a list of statements that follows the sentence in which it appears and about which some metalinguistic statement is made (e.g. "The following property does not hold:", "Assume that the following axioms hold:" or "At most one of the following cases holds:"). The

cataphoric metalinguistic noun phrase announces a type of statement ("property", "axiom" or "case"). Depending on the announced type, each list element starts with a property, axiom or case heading, followed by zero or more assumptions, followed by one or more assertions. The list elements in such a list of statements are separated by beginning new paragraphs.

The number of list elements must be coherent with the grammatical number used in the cataphoric metalinguistic noun phrase: If the noun phrase is singular, the list may only consist of one element; if it is plural, the list must consist of at least two elements.

Statement list blocks may appear after any simple declarative sentence, assertion or assumption containing a cataphoric metalinguistic noun phrase. This means that they can even appear at places where we said that there must be a list of assumptions followed by a list of assertions, i.e. inside axiom blocks, inside theorem blocks and inside list elements of other statement list blocks. When a plural statement list block appears inside another plural statement list block of the same statement type, there is no way to mark the end of the inner statement list block without marking the end of the outer statement list block. Hence the inner statement list block cannot appear inside a list element that is not the last list element of the outer statement list block.

Note blocks

The core of a note block is always a variable type specification or an alternative notation specification. It may optionally be preceded by a note heading.

A note block may appear in any structural block that allows for general text in it. Additionally, it may appear at the end of a theorem block.

Labelled text blocks

Labelled text blocks may be used to section a text (whether the global text or a nested text like the proof of a theorem-proof block) into various sections. Apart from helping the human reader to understand the logical structure of he text better, such sectioning can also help the Naproche system to understand where assumptions should be retracted: At the end of a labelled text block all assumption-consequences blocks that started inside the labelled text block end, i.e. all assumptions made since the beginning of the labelled text block get retracted.

A labelled text block consist of one or more sections, each of which consists of a label followed by the section's content in the form of general text. A label is a string of alphanumerical characters followed by ")". We distinguish different types of labels depending on the alphanumerical characters used: Capital Latin letters ("A)", "B)", "C)" etc.), small Latin letters ("a)", "b)", "c)" etc.), Arabic numerals ("1)", "2)", "3)" etc.) and Roman numerals ("i)", "ii)", "iii)" etc.). The only way to mark the end of a labelled text block is by ending a structured block inside which the labelled text block was nested. The end of a section is not marked explicitly, but deduced from the fact that a new section of the same label type begins.

Labelled text blocks may appear in any structural block that allows for general text in it. But when a labelled text block is nested inside another

labelled text block, it must use a different label type than any outer labelled text block.

7.3 Naproche CNL textual syntax

We first describe the general rules for forming sentences adhering to the grammar of the Naproche CNL. These general rules can cause some ambiguities, i.e. there are sentences that can be formed in more than one way from these rules. In section 7.3.6 below we will discuss additional disambiguation principles that free the Naproche CNL of ambiguities.

Most Naproche sentences are constructed by combining one or more *simple sentential phrases* using *sentential connectives*. We first discuss the simple sentential phrases, of which there are four kinds:

- *Formulae*: These will be explained in section 7.4 (where they are considered terms of type o). Formulae may optionally be preceded by "we have" or "we get".
- *NP-VP-sentences*: These are sentences that consist of a noun phrase (NP) followed by a verb phrase (VP). The possible forms of noun phrases and verb phrases will be discussed in sections 7.3.1 and 7.3.2 below.
- *Metasentences*: These are sentences that contain metalinguistic statements. They actually also have the form of a noun phrase followed by a verb phrase; but the noun phrases and verb phrases used in metasentences must be of certain limited forms that make them metalinguistic in character. We will call these metalinguistic noun phrases and verb phrases meta-NP and meta-VP, and reserve the names noun phrase (NP) and verb phrase (VP) for the usual non-metalinguistic noun phrases and verb phrases.
- *Quantified sentences*: These are sentences headed by a natural language quantification. We will discuss the possible forms of quantified sentences in section 7.3.4 below.

In NP-VP-sentences and metasentences the grammatical number of the noun phrase and verb phrase must coincide. Below we will say some more words on the grammatical number of complex noun phrases.

7.3.1 Noun phrases

Noun phrases are constructed by coordinating one or more *simple noun phrases* using the connectives "and" and "or". The two connectives may not be mixed within a single noun phrase. In a conjunction of more than two simple noun phrases, all but the last "and" can also be replaced by commas. For the purpose of agreement with a verb phrase in an NP-VP-sentence, the grammatical number of a complex noun phrase is determined as follows: If it is a conjunction of simple noun phrases (i.e. uses "and" for the coordination), it is always plural. If it is a disjunction of simple noun phrases of the same grammatical number, its grammatical number coincides with that of the simple noun phrases. If the disjuncts are of various grammatical numbers, the grammatical number of

the complex noun phrase is considered *mixed*, in which case it can no longer agree with a verb phrase of an NP-VP-sentence. Such a noun phrase of mixed grammatical number may still be used as an object or in a prepositional phrase, but as a subject of an NP-VP-sentence it may only be used if the verb is in the infinitive mode and hence does not have a determined grammatical number.

A *simple noun phrase* is either a symbolic term (see section 7.4) or a *determiner noun phrase*. Before presenting our semiformal description of determiner noun phrases, we first illustrate the possible forms of determiner noun phrases by some examples:

1. an integer
2. an even integer k
3. no finite sets
4. the set of odd prime numbers
5. some even numbers a_x, b_x satisfying the following properties
6. every circle C such that p lies on C
7. distinct points p_1, p_2 and p_3 on L such that $d(p_1, p_2) \leq d(p_2, p_3)$ or $d(p_1, p_2) > 1$
8. points not on L
9. no k such that $k^2 > n$

A determiner noun phrase consists of the following four parts in the following order: A determiner, optional adjectives, the core of the determiner noun phrase and optional postmodifiers. The determiner can be “a”, “an”, “some”, “the”, “every” or “no” in the singular and “some”, “all”, “no” or the zero determiner in the plural. In the list of optional adjectives there may be zero or more adjectives, with only one restriction: Certain adjectives like “parallel”, “coprime”, “distinct” and “disjoint”, which we term *transitive adjectives*, are used to express binary rather than unary relations. These may only be used in this list of optional adjectives if the noun phrase is in plural. Which adjectives are considered transitive adjectives is fixed in the lexicon of the Naproche CNL.

The core of a determiner noun phrase consists either of a noun or of a list of quantterms or of a noun followed by a list of quantterms. If the quantterm list contains two or more quantterms, the grammatical number of the noun phrase (and hence of the noun) must be plural. The quantterms in the quantterm list must be placed in separate LaTeX mathematics environment and may additionally be separated by commas or the word “and”.

There are four kinds of post-modifiers:

- *Collection complements* (e.g. “of odd prime numbers”): These consist of either the word “of” followed by a plural determiner noun phrase with zero determiner, or of the words “of objects called” followed by a plural noun. They may only be used if the noun in the core of the noun phrase denotes a collection of mathematical objects (like “set”, “class” or “collection”)[4] and must precede other post-modifiers.

[4]The lexicon of the Naproche CNL fixes which nouns are counted as collection nouns for this purpose.

- *Prepositional phrases* (e.g. "on C_1 and C_2" or "not on L"): These consist of a preposition followed by a noun phrase. The preposition may optionally be preceded by "not". Prepositional phrases must precede such-that clauses and *satisfying*-phrases.[5]

- *Such-that clauses* (e.g. "such that p lies on C"): These consist of the words "such that" followed by a simple declarative sentence (see section 7.3.5).

- *satisfying-phrases* (e.g. "satisfying the following properties"): These consist of the word "satisfying" followed by a *cataphoric meta-NP*, a special kind of meta-NP discussed in section 7.3.3. *satisfying*-phrases may not occur in determiner noun phrases that contain a such-that clause.

Such-that clauses and *satisfying*-phrases may only modify noun phrases that have been *named* using some symbolic expression. For example, in "every circle C such that p lies on C", the noun phrase is named using the quantterm C, which makes it possible to have a such-that clause. A special way in which a noun phrase can be considered named is by appearing in a predicative position in a verb phrase whose subject is a named noun phrase. For example, in "L is a line such that p lies on L", the noun phrase "a line such that p lies on L" is itself not named, but it is the predicative part of the verb phrase "is a line such that p lies on L" whose subject is the named noun phrase L, and hence is considered named as well.

When a such-that clause or *satisfying*-phrase modifies the last determiner noun phrase in a complex noun phrase, the syntactic interpretation of the complex noun phrase chosen by the Naproche CNL does not coincide with the natural reading, but turns out to be semantically equivalent. Consider for example sentence (4):

(4) There is a point q and a positive number x such that $d(p, q) < x$.

Since in the Naproche CNL a such-that clause can only modify a determiner noun phrase, the such-that clause modifies "a positive number x". In the natural reading, on the other hand, it modifies "a point q and a positive number x". But these two reading are at any rate logically equivalent in the *PTL*-based semantics that we give to the Naproche CNL in section 7.5, so that this is not a serious problem.

Note that there is no rule in the Naproche CNL that nouns have to consist of a single word. Indeed, in the Naproche CNL we consider "natural number" to be a two-word noun, as this corresponds to the way this expression is usually interpreted by mathematicians: When something is called a "natural number", this is not interpreted as two unary predicated being asserted of the object in question (as would be the case in usual adjective-noun expressions like "even integer"), but as a single unary predicate.

[5]Note that "of" is not listed as a proposition in the lexicon; hence there is no conflict between prepositional phrases and collection complements. In the language of mathematics, "of" is only used in combination with *transitive nouns* that require an *of*-complement. See section 7.7 for a clarification of *transitive nouns*, a linguistic construct so far not supported in the Naproche CNL. Collection nouns can be considered a special case of transitive nouns that is already supported in the Naproche CNL.

7.3.2 Verb phrases

Below we describe the various forms that affirmative verb phrases can have in the Naproche CNL. Additionally, for every affirmative verb phrase, there is a corresponding negated verb phrase formed by the standard rules for negating verb phrases in English: If the verb phrase is in the infinitive mode, it is negated by putting "not" before its head verb. If the verb phrase is not in the infinitive mode and already contains an auxiliary verb,[6] it is negated by putting "not" after the auxiliary verb. Otherwise it is negated by making the auxiliary verb "do" the head of the verb phrase and putting "not" between the inflected form[7] of "do" and the full verb of the original verb phrase, which now appears in the infinitive mode.

There are three kinds of affirmative verb phrases:

- An intransitive verb
- A transitive verb followed by its object, which may be any noun phrase
- An inflected form of the copula "to be" followed by a predicative expression, which may take one of the following forms:
  - A simple noun phrase which is either a term or a determiner noun phrase with "a", "an", "some" or "the" as determiner
  - An intransitive adjective
  - A transitive adjective (only possible if the verb phrase is in plural)
  - A transitive adjective followed by a prepositional phrase, where the preposition to be used in the prepositional phrase is fixed by the lexical entry of the transitive adjective (e.g. "to" for "parallel" and "coprime" and "from" for "distinct" and "disjoint")
  - A such-that clause (for this case we have the same kind of restriction as already mentioned at the end of section 7.3.1 above: The subject of the verb phrase must be named using some symbolic expression.)
  - A prepositional phrase (certain prepositions allow for verbs other than the copula to be used before them without a change in meaning; for example, one may write "to lie on L" instead of "to be on L".)

The Naproche CNL grammar takes care that verbs are inflected in the right way according to their grammatical number and mode (finite or infinitive). Moreover, it takes care of the difference between infinitives that do and infinitives that do not require a preposed "to".

Just as nouns can consist of more than one word, so can verbs. For example, "to belong to" is considered a transitive verb in the Naproche CNL.

[6] In the Naproche CNL the copula "to be" is the only auxiliary verb used in affirmative verb phrases.

[7] Whenever we speak of an *inflected form* of some word, the Naproche CNL grammar actually takes care that only inflected forms grammatically acceptable in English will be accepted.

7.3.3 Metalinguistic NPs and VPs

There are two kinds of metalinguistic noun phrases:

- *Anaphoric meta-NPs*: Simple anaphoric meta-NPs always consist of the word "case", "property" or "axiom" followed by the name of a previously introduced case. An anaphoric meta-NP coordinates one or more simple anaphoric meta-NPs in one of the following three ways:
  - As a conjunction, in which the simple anaphoric meta-NPs may be separated by "and" or commas.
  - As a disjunction, in which the simple anaphoric meta-NPs are separated by "or".
  - By an expression starting with "precisely one of" or "at most one of" followed by a list of simple anaphoric meta-NPs separated by "and" or commas.
- *Cataphoric meta-NPs*: These consist of the words "precisely one of the following", "at most one of the following" or "the following", followed by an inflected form of one of the words "case", "property" or "axiom" (in the first two cases they must be plural, in the third case they may be singular or plural).

A metalinguistic verb phrase is an inflected form of "to hold", "to be true", "to be correct", "to be incorrect", "to be false", "not to be true", "not to be correct", "not to hold" or "to be inconsistent". The last form may not be used in combination with a meta-NP which is a disjunction of meta-NPs or which contains the words "precisely one of" or "at most one of".

7.3.4 Quantified sentences

There are two kinds of quantified sentences, *universally quantified sentences* and *existentially quantified sentences*.

Universally quantified sentences consist of a *universally quantifying expression* followed by a potentially complex sentential phrase. These two parts may optionally be separated by a comma. A universally quantifying expression always consists of the word "for" followed by a determiner noun phrase with a universal determiner ("every" or "all"). The following is an example of a universally quantified sentence: "For all natural numbers m, n, $m + n = n + m$."

Existentially quantified sentences come in two different flavours: The first consists of an inflected form of "there to be at most one" or "there to be precisely one" followed by a singular determiner noun phrase truncated of its determiner. The second consists of an inflected form of "there to be" or "there to exist" followed by a conjunction of determiner noun phrases which are separated by "and" and whose determiner is indefinite or negative ("a", "an", "some", the zero determiner or "no"). If the conjunction consists of a single singular noun phrase, "there to be" or "there to exist" has to be inflected in the singular. If the conjunction contains at least one plural noun phrase, "there to be" or "there to exist" has to be inflected in the plural. If the conjunction consists of multiple singular noun phrases, "there to be" or "there to exist" may be inflected either in the singular or in the plural.

Here two examples of existentially quantified sentences:

- There is precisely one even prime number.
- There are no distinct parallel lines L_1 and L_2 such that p lies on L_1 and L_2.

7.3.5 Sentential connectives

We use the term *sentential connectives* in a rather broad sense to encompass all expressions that combine with one or more sentential phrases to form a new sentential phrase. A special kind of sentential connectives are *references*. A reference may be prefixed or postfixed to a sentential phrase. It consists of the word "by" followed by a list of one or more *reference cores*, separated by commas or "and". A reference core is of the form "axiom X", "theorem X", "lemma X", "definition X" or "induction", where "X" is to be replaced by the name of a previously stated axiom, theorem, lemma or definition.

Another special kind of sentential triggers are the *assertion triggers* and *assumption triggers*, which are used to distinguish between the sentence kinds *simple declarative sentences*, *assertions* and *assumptions*: An assertion is formed by preceding a simple declarative sentence by an assertion trigger, and an assumption is formed by preceding a simple declarative sentence by an assumption trigger. The assertion triggers are "also", "and", "but", "clearly", "finally", "furthermore", "hence", "i.e.", "in particular", "now", "observe that", "obviously", "recall that", "so", "therefore", "this (in turn) implies (that)",[8] and "trivially", as well as some combinations of these that are grammatically acceptable in English (e.g. "now recall that"). Additionally there is an empty assertion trigger, i.e. a simple declarative sentence may itself be considered an assertion. The assumption triggers are "(now) assume (that)", "(now) assume for a contradiction that", "(now) suppose that", "(now) let" and "(now) consider".

Assertions and assumptions may additionally be formed in the following ways: An assertion may just consist of a reference, of the word "trivial" or of the word "contradiction" optionally followed by a reference. An assumption may consist of a "(now) consider (arbitrary)" or "(now) fix (arbitrary)" followed by a quantterm list, or of "let" followed by a quantterm list, followed by "be given". The quantterm lists in these assumptions may optionally be followed by postmodifiers; the postmodifiers allowed here are the same as the ones allowed in noun phrases (see section 7.3.1), with collection complements excluded.

All other sentential connectives, which we call the *proper sentential connectives*, can be used in a nested way to form simple declarative sentences out of simple sentential phrases.[9]

1. (a) "implies"
   (b) "implies that"
2. "and"

[8] The brackets indicate optional parts of the assertion trigger.

[9] Note that the word "simple" has different meanings in "simple declarative sentences" and "simple sentential phrases" In the first case it means that the sentence lacks assertion and assumption triggers, in the second case it means that the sentential phrase is not formed out of simpler sentential phrases using sentential connectives.

3. "or"
4. ", and"
5. ", or"
6. (a) ", i.e."
   (b) ", so"
7. (a) "if ... then ..."
   (b) "... if ..."
   (c) "iff"
   (d) "if and only if"
8. Inflected variants of
   (a) "it to be false that"
   (b) "it not to be the case that"
   (c) "it to be the case that"

The sentential connectives listed under 1-7 are all binary. Their two arguments either precede and follow the connective, or are placed in the positions of the "..." indicated above. In the connectives under 7, the first argument may optionally be followed by a comma. The connectives under 8 are unary and precede their argument. If their argument is a complex sentential phrase formed out of simple sentential phrases with the connectives under 1-6, each of these simpler sentential phrases has to be preceded by "that". This rule makes it possible to use the semantically redundant "it to be the case that" for forcing a certain bracketing of complex sentential phrases. For example, to express $\neg(A \wedge B)$ in a more natural way in the Naproche CNL, one has to write (5). If one just writes (6), the interpretation will be $(\neg A) \wedge B$, since the lack of a "that" in front of B makes it impossible for B to be inside the scope of "it is not the case that".

(5) It is not the case that A and that B.

(6) It is not the case that A and B.

Further principles for disambiguating complex sentential phrases are discussed in section 7.3.6 below.

References only make sense and are naturally only accepted by human readers in positions where they modify a sentential phrase that becomes the conjecture of a proof obligation in the process of proof-checking the text. These positions can actually be recognized by purely grammatical means. For example, in assumptions and in *if*-clauses references never make sense. We have therefore included this restriction into the Naproche CNL grammar. In these positions, in which references may not be used, the connectives "i.e." and "so" can also not be used with the semantics of a dynamic conjunction that we give them, and are hence also excluded by restrictions included in the grammar.

In order to correctly predict the mode (finite or infinitive) of verbs, we also attach a mode to sentential phrases: A sentential phrase following one of the assumption triggers "let" or "consider" gets the mode "infinitive" or "*to*-infinitive"

respectively. In the case of a simple sentential phrase, this means that its head verb must be in the infinitive mode (and possibly be preceded by "to"). In the case of a sentential phrase formed by one of the connectives listed under 1-6, this mode gets inherited to its subordinated sentential phrases. In the case of a sentential phrase formed by one of the connectives under 8, that connective itself has to be inflected in the corresponding mode. Sentential phrases formed by one of the connectives under 7 can never have the mode "infinitive" or "to-infinitive". Hence an assumption can for example not consist of "let" or "consider" directly followed by an *if-then* construct; this of course is completely in line with restrictions that are in place in natural English.

7.3.6 Disambiguation principles

The grammatical rules described so far can cause some ambiguities, i.e. there are sentences that can be formed in more than one way from these rules. In this section we will present disambiguation principles which have been incorporated into the Naproche CNL in order to make it free from ambiguities. The disambiguation principles have been chosen in such a way that the reading they give preference to very often coincide with the reading a mathematician would actually give preference to. But readers of mathematical texts, just like people confronted with general natural language, use a large number of disambiguation principles weighted in a not well understood way. So they may in some circumstances prefer a reading which the disambiguation principles of the Naproche CNL discard. For this reason it is important for an author of Naproche CNL texts to understand the disambiguation principles used by the Naproche CNL.

First we present three special disambiguation principles which only apply in special circumstances. Finally we present a general disambiguation principle which removes all potential ambiguities not removed by the special disambiguation principles.

For the proper sentential connectives presented in the previous section, the numbers in the numbered list in which we presented them fix their *operator precedence*. So "and" binds stronger than "or", which binds stronger than ", and" etc.

References are always considered to modify the largest sentential clause that they could possibly modify given their position. For example, in (7), "by Lemma 12" modifies "$R(x)$ and $R(y)$", whereas in (8) it modifies only "$R(y)$".

(7) $R(x)$ and $R(y)$ by Lemma 12.

(8) $R(x)$ and by Lemma 12 $R(y)$.

As mentioned in section 7.3.1, the core of a noun phrase may be just a quantterm list. Given that the plural indefinite determiner may be empty, this means for example that "x" could be interpreted both as a term or as a determiner noun phrase consisting of a plural indefinite determiner followed by the quantterm list "x". If there is an anaphorically accessible occurrence of "x", the semantics of these two interpretations would be different: In the first case "x" would corefer with this anaphorically accessible occurrence of "x", whereas the second case would be interpreted as an existential quantification over x. In ordinary mathematical texts, the first reading would always be preferred. In order to force that reading in the Naproche CNL, we have added limitations as

to when the zero determiner may be used: It may only be used in determiner noun phrases that either contain a noun or are used as the core of an existential quantification. (9) and (10) contain examples of these two cases:

(9) A contains **points $\boldsymbol{x}$, $\boldsymbol{y}$** such that $d(x, y) = 1$.

(10) There are $\boldsymbol{x}$, $\boldsymbol{y}$ such that $d(x, y) = 1$.

Even though the three special disambiguation principles presented so far already remove many potential ambiguities, there are some potential ambiguities left for which these special disambiguation principles present no solution. Consider sentence (11) occurring in a context in which the variables k and l are accessible anaphoric antecedents:

(11) k is a prime number such that $k|l$ iff k is odd.

There are two potential readings for this sentence: In the first, it is a simple NP-VP-sentence, whose verb phrase contains the such-that clause "such that $k|l$ iff k is odd". In the second it is a bi-implication between "k is a prime number such that $k|l$" and "k is odd". Now the general disambiguation principle states that whenever there are such ambiguities not removed by any of the special disambiguation principles, the Naproche CNL grammar chooses the reading which closes all scopes as late as possible. In this example, the scope of the such-that clause is closed later by the first than by the second reading, so that the first reading is the reading chosen by the Naproche CNL grammar.

This general disambiguation principle also has an effect when we coordinate more than two simple sentential phrases with sentential connectives of the same operator precedence. One of the few cases where it actually makes a semantic difference are complex sentential phrases of the form "If A then B iff C". The general disambiguation principle gives preference to the reading in which "B iff C" is the second argument of the "If ... then ..."-construct.

In most cases, this general disambiguation principle chooses the reading that a mathematical reader would naturally prefer. However, there are of course exceptions. (12) presents an example of a sentence in which a mathematical reader is likely to choose a reading that differs from the reading chosen by Naproche CNL grammar:

(12) There is no square number k such that $k|n$, and n is a prime number.

Here the Naproche CNL grammar would consider "n is a prime number" to be part of the such-that clause. There are two reasons for a mathematical reader to prefer the reading in which the such-that clause ends at the comma: One is the orthographic hint provided by the comma; the other is the semantic-pragmatic reason that "n is a prime number" does not refer to the variable k postmodified by the such-that clause. These two reasons taken together make it very unlikely for a mathematical reader to naturally choose the reading chosen by the Naproche CNL grammar. But for the Naproche CNL grammar, the comma before the "and" could only make a difference in combination with other proper sentential connectives, and semantic-pragmatic considerations as the one relevant in this case are generally ignored.[10]

[10] The semantic-pragmatic natural disambiguation principle mentioned in this example could

7.3.7 Definitions

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

Here are some examples of bi-implicational definitions:

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

As can be seen from the examples, a bi-implicational definition can define an adjective (transitive or not), a noun, a verb (transitive or not) or a relational symbolic expression. In all cases it consists of the word "define" followed by a definiendum, followed by "iff" or "if and only if", followed by a definiens, which is a simple declarative sentence. The definiendum introduces some dummy variables; in the case of definitions of words, these may optionally be preceded by a specification consisting of an indefinite determiner, zero or more adjectives and a noun. These dummy variables are considered accessible in the definiens.

Bi-implicational definitions of words always have a verb in the definiendum: In the case that they define a verb, this is clear; in the case of an adjective, the copula is placed before the defined adjective; in the case of a noun, we also use the copula, but additionally place an indefinite article between the copula and the defined noun, as can be seen in example 2. The verb always has to be in the infinitive mode and preceded by "to". Definitions of verbs have an optionally specified variable in subject position, and in the case of transitive verbs, additionally also in object position. Definitions of nouns and intransitive adjectives only have one optionally specified variable, namely in subject position.

Transitive adjectives represent binary relations, so their definitions always have two optionally specified variables. But there are two different options for

actually be detected on purely syntactic grounds. For this reason, future versions of Naproche might include a disambiguation principle of this kind, or at least detect when the result of such a disambiguation principle conflicts with its usual disambiguation principles in order to warn the user that the reading chosen might not be the reading a mathematical reader would naturally choose.

arranging them in the definiendum: Either, as in example 4, both optionally specified variables appear in the subject separated by "and"; or, as in example 3, one appears in the subject and the other in a propositional phrase that postmodifies the transitive adjective and uses the preposition that is determined by the lexical entry of the transitive adjective (see section 7.3.2).

Bi-implicational definitions of relational symbolic expression have a definiendum that consists of a special kind of quantterm called *definition quantterm* that is described in section 7.4.6.

Here are some example of copula definitions:

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

In most cases, a copula definition consists just of the word "define" followed by a definiendum, followed by "to be", followed by a definiens. The definiendum of a copula definition is a definition quantterm (see section 7.4.6); the definiens is any term. The definiendum may introduce dummy variables, which serve as anaphoric antecedents for occurrences of the same variables in the definiens. If there are no such dummy variables, as in the first example, the definition is what one would normally call a definition of a constant symbol. If there are such dummy variables, as in the other three examples, the definition defines a function.

A copula definition may also be used to define the value of a function at a fixed argument, as in example 4. The fixed argument may be represented not only by a variable, but also by a complex term, which appears in the same form in the definiendum and in the definiens. The definition quantterm parser may have difficulties determining which symbols are part of the function being defined and which are part of the fixed complex argument; in order to ensure that the definition quantterm parser correctly parses the definition quantterm, one may optionally precede the copula definition by an expression of the form "For defining f at a", where f should be replaced by the name of the function being defined and a by the fixed argument at which we define the value of the function.

7.3.8 Notational specifications

There are two special kinds of sentences that can be used for specifying certain notational conventions.

Variable type specifications are sentences used to specify that certain variables will from now on be used only to refer to a certain kind of objects, e.g. (13):

(13) Small Latin letters will stand throughout for integers.

More precisely, variable type specifications always consist of a subject referring to some collection of letters, followed by the words "(will) always denote", "will be used throughout to denote" or "will stand throughout for", followed by a

plural noun. The subject is a conjunction of one or more expressions of the following form, separated by commas or "and": An optional capitalization adjective ("small" or "capital") followed by an alphabet adjective ("Latin, "Greek, "Fraktur" or "German", with the latter two being synonymous), followed by the word "letters".

Alternative notation specifications can currently only be used to specify that a given binary function expressed by an infix function symbol may also be expressed by concatenation, as in (14):

(14) Instead of $x \cdot y$ we also write xy.

Alternative notation specifications always have the form of (14), only that the infix function symbol $\cdot$ may be replaced by another infix function symbol, and that instead of x and y other variables may be used.

7.4 Symbolic mathematics in the Naproche CNL[11]

In section 1.1.2, we showed some of the problems involved in giving a syntactic description of symbolic mathematics, and in particular of parsing and disambiguating these expressions. We remind the reader about the example of the expression "$a(x+y)$", which can be understood in two completely different ways, depending on what kind of meaning is given to a: If a is a function symbol and $x + y$ denotes a legitimate argument for it, then $a(x + y)$ would be understood to be the result of applying the function a to $x + y$. If on the other hand a, x and y are – for example – all real numbers, then $a(x + y)$ would be understood as the product of a and $x + y$.

7.4.1 Possible approaches to disambiguation

If $a(x + y)$ is to be read as the value of a function a at $x + y$, then a has to be a function. This requirement can be understood in two different ways, which are nevertheless related and combinable: Either it is considered to be a presupposition of the symbolic expression $a(x + y)$; in this case, the linguistic theory of presuppositions becomes applicable (see section 3.2). Or it is considered to be a *type judgement* about a; in this case, it should be possible to formulate a type system for symbolic mathematics and reuse existing ideas from type theory to describe and work with this type system.

Since we had to include a treatment of presuppositions in Naproche at any rate (see section 6.1.3), one possible approach that we took into consideration for disambiguating symbolic expressions was to check their presuppositions already during the parsing process, so that readings which lead to wrong presuppositions would already be blocked during the parsing process. This approach, however, has turned out to be far too inefficient: It would involve constantly calling automatic theorem provers during the parsing process and waiting for their output before continuing the parsing.

Another approach is to rely on a type system rather than on presupposition fulfilment for disambiguating symbolic mathematics. In that case, one needs a very rich and flexible type system for symbolic mathematics. Such a type

[11] This section is partly taken over from Cramer et al. (2011).

system has been developed ingeniously by Ganesalingam (see Ganesalingam, 2009). However, to attain the richness of the type system required for handling all kinds of ambiguities that can arise, he was obliged to require the author of a text that is to be parsed by his system to write sentences whose sole function is to create types that are needed for certain disambiguations. Given that the goal of Naproche is to stay as close as possible to the language that mathematicians naturally use, this aspect of Ganesalingam's approach made it less attractive for us.

So we decided to take up a combined approach, in which there is a relatively simple type system capable of blocking most unwanted readings during the parsing process, with the remaining readings being filtered by checking their presuppositions.

7.4.2 A type system for symbolic mathematics

In the type system that Naproche uses for handling symbolic mathematics, there are two basic types: i for individuals and o for formulae expressing propositions. Apart from these, there are function types of the form $[t_1, ...t_n] \to t$, where $t_1, \ldots, t_n$ are the types of the arguments the function takes and t is the type of the term that we get when we apply this function to legitimate arguments. So unlike in the Simple Theory of Types (STT) (see Church, 1940), we have an inherent way of handling multi-argument functions. In STT, multi-argument functions must be simulated by their curried counterparts (see section 5.0.1). We, however, want to use types to describe how mathematical formulae are structured in actual mathematical texts, and for this purpose it is better to have multi-argument functions inherently in the type system.

Note that formulae are also considered terms (namely terms of type o), and that the logical connectors are considered functions of type $[o, o] \to o$ or $[o] \to o$. Even quantifiers are considered to be functions, namely two-place functions whose first argument has to be a variable and whose second argument is a term of type o that may depend on the variable. We formalize this by writing the type of quantifiers as $[var(\_, X), X - o] \to o$, where $var(\_, X)$ means that the first argument is a variable X of type $\_$ (i.e. of any type), and $X - o$ means that the second argument is a term of type o possibly depending on X.[12]

Notational types

As already discussed in section 1.1.2, functions can behave in syntactically different ways. For example, + is generally used as an infix function symbol ("$a+b$"), whereas the notation $f(x)$ uses a function symbol f in prefix position with its argument in brackets. In Naproche, we distinguish six basic ways in which function symbols behave syntactically, and call these the *basic notational types*[13] of the corresponding function symbols:

1. `infix`: Two-argument function symbol placed between its arguments (e.g. + in $n + m$).

[12] We use Prolog-like notation, i.e. capital letters for variables and $\_$ for an anonymous variable, when describing the type system.

[13] In Cramer et al. (2011), we used the term *syntactic type* rather than *notational type*. We now consider *notational type* to be less prone to misunderstanding than *syntactic type*.

2. `suffix`: One-argument function symbol placed after its argument (e.g. ! in $n!$).

3. `prefix`: One-argument function symbol placed before its argument (e.g. sin in $\sin x$).

4. `classical`: Function symbol with one or more arguments preceding its arguments, which are bracketed and separated by commas (e.g. f in $f(x)$ or $f(x, y)$).

5. `quantifier`: Two-place function symbol placed before its two arguments, where the arguments have to have types of the form $var(t_1, X)$ and $X - t_2$, and where the first argument position may be filled with a variable list rather than a single variable (e.g. $\forall x, y\ R(x, y)$).

6. `circumfix`: Expression for a function with one or more arguments, which are embedded into a predefined string of symbols, with at least one symbol at the beginning, at the end and between any two successive arguments (e.g. the degree of a field extension, $[K : k]$, considered as a two-place function depending on K and k). The *name* of a circumfix function is this predefined string with `[arg]` denoting the positions of its arguments. For example, the name of the field extension function is `[[arg]:[arg]]`.

A *notational type* is a finite list of basic notational types. For motivating this definition, we first consider an example from real analysis: The differentiation function is a function from differentiable real functions to real functions, sending any f to its derivative f'. When written in this $'$-notation, this function clearly has notational type `suffix`. But when we write $f'(x)$, we use the complex function name f' as a function with notational type `classical`. Now this does not seem to depend on the notational type of f: Suppose we have defined an extension of the factorial function ! to the reals (e.g. by $x! := \Gamma(x+1)$ using the Gamma function (see Heuser, 1991, p. 195)). If we then apply its derivative $!'$ to some real x, we would write $!'(x)$ and not $x!'$.[14] So it seems to be inherent in the way the differentiation function symbol $'$ is used that the complex function name it produces is of notational type `classical`. We formalize this by saying that $'$ is of notational type `[suffix,classical]`. This means that its basic notational type is `suffix`, and the notational type of any function name whose head is $'$ is `[classical]`.

This machinery makes it possible to correctly handle many complicated notations: For example, exponentiation is treated as a function of notational type `[circumfix,suffix]` and of type $[i] \to ([i] \to i)$ (so in this case the notation used makes us treat this multiple-argument function in a curried way rather than using an inherent multiple-argument function type), where the name of the circumfix function is `^{[arg]}`. In the case of `x^{y}`, this function is first applied to `y`, yielding `^{y}`, which is considered a suffix function, so that applying it to x yields `x^{y}`.

When concatenation is used to express a binary function (as is usually done for multiplication, as in nm for $n \cdot m$), we consider the function to be expressed by an empty infix function symbol. The possibility of expressing an

[14] Since this is a made-up example, we should add that our intuition as to what notation would be appropriate here has been confirmed by a number of mathematicians from the University of Bonn.

infix function by concatenation is announced by an *alternative notation specification* as discussed in section 7.3.8. In some cases, the way a seemingly infix function is introduced makes the system consider it a function of notational type `[prefix,suffix]` or `[suffix,prefix]`.[15] Such a function may also be expressed by concatenation, i.e. the function symbol of such a function may also be empty.

As already mentioned in section 7.3, Naproche distinguishes two different kinds of symbolic expressions:

- *Terms* serve either as definite noun phrases (e.g. $2x - 1$) or, if they have type o, as formulae and thus as sentential phrases (e.g. $x = y^2$).
- *Quantterms* correspond to the quantifiable terms of *PTL* and can, possibly together with a noun, make up the core of a determiner noun phrase.

7.4.3 Term Grammar

Below we describe the term grammar semi-formally by first listing (in a formal definite-clause-grammar notation (see Pereira & Warren, 1980) with Prolog-like syntax) a list of simplified grammar rules that any term must obey and then describing informally additional constraints that any term must satisfy in order to be actually parsed by the grammar. The constituent "term" used in the DCG rules below has an argument specifying the notational type of the term (i.e. a list of basic notational types). We use the variable name NT for a variable ranging over notational types.

Simplified term grammar

term(NT) → term([classical|NT]), ['('], term_list, [')'].
term(NT) → term(_), term([suffix|NT]).
term(NT) → term([prefix|NT]), term(_).
term(NT) → term([quantifier|NT]), variable_list, term(_).
term(NT) → term(_), term([infix|NT]), term(_).
term(NT) → circumfix_term(NT).
term(NT) → ['('], term(NT), [')'].
term(NT) → variable(NT).

term_list → term(_), [','], term_list.
term_list → term(_).

variable_list → variable(_), [','], variable_list.
variable_list → variable(_).
variable(_) → [_].

[15] An example for this are the addition and multiplication signs in the proofs of theorems 4 and 28 respectively of the Naproche CNL reformulation of chapter 1 of Landau's *Grundlagen der Analysis*, which is included in appendix B and discussed up to theorem 4 in chapter 8.

For every predefined variable or accessible quantterm V of notational type NT, add a rule of the following form to the grammar:

variable(NT) → V.

For every accessible circumfix function of notational type NT and name $S_1^1 \dots S_1^{n_1}[arg]S_2^1 \dots S_2^{n_2}[arg] \dots [arg]S_m^1 \dots S_m^{n_m}$*, add a rule of the following form to the grammar:*

circumfix_term(NT) →

$[S_1^1], \dots, [S_1^{n_1}], term(\_), [S_2^1], \dots, [S_2^{n_2}], term(\_), \dots, term(\_), [S_m^1], \dots, [S_m^{n_m}]$.

Infix relation symbols (i.e. infix function symbols with type of the form $[\_, \_] => o$) may be used for chained formulae, e.g. $t_1 = t_2 = t_3 = t_4$. In this case, the parse tree we produce for the formula is the same as if the formula had been $t_1 = t_2 \wedge t_2 = t_3 \wedge t_3 = t_4$.

Operator precedence

Syntactic disambiguation principles like the precedence of multiplication and division operators over addition and subtraction operators are encoded into the grammar using predefined operator priorities. We use the following operator priorities (in the order of decreasing precedence):

- $+$, $-$, $\rightarrow$ and $\leftrightarrow$
- Prefix functions
- Suffix functions
- Other infix functions

Additionally, there is a principle which overrides the above operator priorities, namely that the operators used to form atomic formulae always have a higher precedence than the operators used to combine atomic formulae into complex formulae.

As an example for the functioning of these syntactic disambiguation principles,

(15) $x + yz = \sin an! \wedge x = y \rightarrow z - y + z = 0$

is disambiguated as

(16) $(((x + (yz)) = \sin(a(n!))) \wedge (x = y)) \rightarrow (((z - y) + z) = 0)$.

In all cases that we are aware of, these syntactic disambiguation principles lead to an intuitive reading of the symbolic expression.

Defaultness of the notational type `classical`

As already alluded in section 1.1.2, the notational type `classical` is the default notational type for newly introduced functions. This principle is implemented into the grammar by an additional constraint that in the second to fifth DCG rule specified above, as well as in the rule "variable(_) → [_].", the notational

type of a term may not be instantiated to `infix`, `prefix`, `quantifier`, `suffix` or `circumfix`. For example, the requirement of the final term to have "suffix" as notational type in the second rule means that this notational type must already be associated with the term when parsing it and may not be attached to the term afterwards. There is a limited list of predefined infix function symbols ($\cdot$, $+$, $-$, $*$, $.$, $\circ$, $/$, $\in$, $<$, $>$, $\leq$, $\geq$) for which this constraint does not apply.

In practice, this constraint means that when you are quantifying over a function, this function may be used with classical notational type or, if a preferred infix function symbol is used, with infix notational type, but not with prefix, suffix or quantifier notational type. So (17) and (18) are allowed, but (19), (20) and (21) (with z read as an infix, f as a prefix and g as a suffix function symbol) are not allowed.

(17) $\exists f\ f(a) = 0$

(18) $\exists *\ x * x = x$

(19) $\exists z\ xzx = x$

(20) $\exists f\ fa = 0$

(21) $\exists g\ ag = 0$

The defaultness of the notational type `classical` is one of the two reasons why we do not formalize functions used in this notational way as circumfix functions. This would theoretically be possible: A one-argument classical function f could also be considered a circumfix function with name `f([arg])`. However, this way we would not be able to account for the fact that a function that was introduced without fixing its notational type can be used with notational type `classical`. The second reason for avoiding this solution is that non-circumfix function symbols can be used to refer to the function itself: If f is a classical function symbol, we can use f to refer to the function; for circumfix functions no way of referring to the function has been implemented.[16]

Simple and complex variables

In the above *simplified term grammar*, variables are always single symbols. In the actually implemented term grammar, there is more flexibility: Additionally to the single-symbol *simple variables*, there are multi-symbol *complex variables*. Complex variables always consist of a single symbol followed by a sequence of subscript digits, e.g. x_1 or y_{12}.

Predefined variables

It should be noted that we do not make the distinction between variables and constants that is usually made in the syntax of first-order logic and many other logical systems. In the semi-formal language of mathematics, there is a continuum between variable-like and constant-like expressions; this continuum is

[16] An accepted way of referring to circumfix functions in the language of mathematics is by the use of $-$ or $\bullet$ in the argument positions of a circumfix function. For example, the circumfix function with name `[[arg]:[arg]]` can be referred to as $[- : -]$ or $[\bullet : \bullet]$.

captured in Naproche through the use of dynamic quantification inherent in *DPL*, so that the bivalent distinction used in first-order logic is not needed.

However, logical constants are still treated in a special way, namely as "predefined variables". These are also given a predefined type and notational type as follows:

Predefined variable	Type	Notational type
$\rightarrow$, $\leftrightarrow$, $\wedge$ and $\vee$	$[o, o] \rightarrow o$	`infix`
$\neg$	$[o] \rightarrow o$	`prefix`
$\forall$ and $\exists$	$[var(\_, X), X - o] \rightarrow o$	`quantifier`
$=$	$[T, T] \rightarrow o$[17]	`infix`
$\neq$	$[\_, \_] \rightarrow o$[18]	`infix`
$\in$	$[i, i] \rightarrow o$	`infix`

Kinds of variables

In the parsing process we distinguish different kinds of variables:

- Predefined variables (logical constants)
- Bound variables
- Variables that were implicitly introduced earlier on in the symbolic expression and are now reused
- Accessible variables whose antecedent is in the same sentence
- Accessible variables whose antecedent is in a preceding sentence
- Implicitly introduced variables

When trying to parse a variable, we always first try to parse it according to a variable kind higher up in the above list before trying the kinds lower down in the list. Once a variable has been parsed in one way, it may no longer be parsed in such a way as to be of a kind that is mentioned later in the above list than the kind that it has already been assigned. This means, for example, that if x is accessible and we parse $\exists x\ x + x = x$, then all instances of x in this formula are bound by the existential quantifier; none of the instances of x refers to the accessible variable.

Coverage of the term grammar

The term grammar can cope with almost all terms that serve as definite noun phrases and formulae found in mathematical texts. Here is a list of formulae

[17] i.e. the two arguments must be of the same type
[18] i.e. the two arguments may be of distinct types

that can be correctly parsed and disambiguated by it:

$$x(y+z)=0$$
$$x=y<z$$
$$x *_G x = x$$
$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$
$$x_0 \lim_{x \to x_0} f(x^2) = 2f(\frac{x_0^{x_0}}{2}) \neq f'(N!)$$
$$T = m_0 \frac{l^2}{2}((cos\varphi_0\varphi_0')^2 + (-sin\varphi_0\varphi_0')^2)$$

Of course, these formulae can only be parsed if the types and notational types of the function symbols appearing in them are known in advance. This information is created by the *quantterm* grammar described in section 7.4.6 when the functions are introduced.

There are some limitations of the current implementation of the term grammar that we are aware of: Firstly our term grammar can only handle variable binding if the occurrence of the variable that binds the other occurrences precedes the bound occurrences. Hence the term grammar cannot handle the integral notation of the form $\int f(x)dx$, where the first occurrence of x is bound by the final occurrence of x. Furthermore, the term grammar can currently not cope with formula fragments like "$= 0$" nor with formulas containing triple dots like "$n \in \{1, \ldots, N\}$". However, we believe that the approach presented in this section constitutes a framework for tackling even these harder cases, i.e. that the current limitations are not due to principle limitations of our approach, but rather due to the prototypical character of the implementation.

7.4.4 Disambiguation after Parsing

As mentioned in section 7.4.1, the type system is not capable of blocking all unwanted readings. This is due to the fact that our type system is not fine-grained enough. All objects that are not functions are of the same type, namely i. So, for example, both natural numbers and sets would be of the type i. If one has defined that for sets A, B, the expression A^B denotes the set of functions from A to B, and one has furthermore defined that for natural numbers m, n, the expression m^n denotes the n-th power of m, then one has defined two functions of notational type `[circumfix,suffix]` and type $[i] \to ([i] \to i)$, both named `^{[arg]}`. Since their name, type and notational type are identical, they are indistinguishable during the parsing process. Thus, the ambiguity arising from this notational clash has to be resolved after the parsing process.

In such cases we keep track of all possible readings until the proof checking process.[19] As described in section 6.1.3, the proof checking involves checking the presuppositions of the *PTL* text. The two just mentioned functions of equal name, type and notational type would trigger different presuppositions: The first would trigger the presupposition that both of its arguments are sets,

[19] More precisely, the *PTL* text that we use to translate the Naproche CNL text is in such a case actually an *underspecified PTL text*, in which some subformula or subterm is not determined, but may be filled in with one of a number of possible subformulae or subterms.

whereas the second would trigger the presupposition that both of its arguments are numbers.[20] Since it is not possible for both of these presuppositions to be fulfilled for a given pair of arguments, the ambiguity can certainly be removed in the process of checking the presuppositions.

It is of course also possible that more than one reading fulfils the presupposition. Consider for example the following text, appearing in a context in which n is an accessible integer variable and in which $>$, $\geq$ and $\leq$ are accessible binary infix function symbols of type $[i, i] \to o$ and triggering the presupposition that both its arguments must be integers:

(22) There is an integer k such that $k > n$. Hence there is an integer k such that $k \geq n$. Then $n \leq k$.

Because of the dynamic nature of natural language existential quantification, both the k from the first sentence and the k from the second sentence is accessible in the third sentence. Hence $n \leq k$ has two readings, depending on whether k refers to the k introduced in the first sentence or the k introduced in the second sentence. Both of them fulfil the presuppositions of $\leq$. For disambiguating such formulae, the Naproche system uses the principle that a later anaphoric antecedent is preferred over an earlier one. Hence the Naproche system would choose the second reading of $k \geq n$, as a mathematical reader would naturally do as well.[21]

7.4.5 Type dependency graphs

It is also possible that the type information needed for disambiguating a symbolic expression is only available after the completion of the parsing process for that expression. Suppose, for example, that sentence (23) appears in a context where two binary relations named $>$ are accessible, one defined on natural numbers and one defined on functions of natural numbers, and where the symbol 1 is accessible both as a name for the natural number 1 and as a name for the identity function, but where the variable x is not accessible.

(23) If $x > 1$ and $x^2 + 1$ is prime, we have $R(x)$.

If the exponential notation x^2 is only defined for numbers and not for functions, then this sentence can be disambiguated using type information: x has to be of type i in "$x^2 + 1$" and therefore also in "$x > 1$", and so the "$>$" in "$x > 1$" refers to the relation on numbers and not the one on functions. But this type-based disambiguation of "$x > 1$" was not possible during the process of parsing "$x > 1$", because at that point "$x^2 + 1$" had not yet been parsed. In order to handle such type-based disambiguations that occur after that parsing of an

[20] The information that these functions trigger these presuppositions gets extracted from their definitions: In section 7.5.4 below, we define how function definitions are translated by implications in such a way that the functions they define are dynamically implicitly introduced in the *PTL* translation of the definition. Hence the explanations in section 6.1.4 about extracting information about the domain of a dynamically implicitly introduced function applies to function definitions.

[21] Note that already when parsing $k \geq n$, both the k from the first sentence and the k introduced at the beginning of the second sentence are accessible. But in this case, the ambiguity is already resolved by the principle mentioned under the heading "Kinds of variable" above, since one potential anaphoric antecedent is in the same sentence and the other is not; so the disambiguation principle mentioned here is not needed for $k \geq n$.

expression, we use *type-dependency graphs*, which specify which reading of an expression depends on which type judgements.

The parsing module of the Naproche system constructs a type-dependency graph for every sentence in a Naproche text. The type-dependency graph is modified whenever a symbolic expression is encountered. There are two kinds of vertices and two kinds of directed edges in a type-dependency graph:

- For every symbolic expression appearing in the sentence and every variable implicitly introduced in some term, there is an *E-vertex* representing this symbolic expression or variable.
- For every possible reading of a term and every possible type assignment for a quantterm or implicitly introduced variable, there is an *R-vertex* representing this reading or type assignment.
- Whenever a reading represented by vertex r_1 depends on a type assignment represented by vertex r_2, there is a *D-edge* from r_1 to r_2.
- Whenever an R-vertex r represents a reading of type assignment of a symbolic expression or variable represented by an E-vertex e, there is an *R-edge* from r to e.

Type-assignments are graphically represented by a variable or quantterm with its type as a subscript index. Since in the examples we consider, the different readings of a term always differ by the type assignments of their variables, we graphically represent readings by writing the types of the variables occurring in them as subscript indices.

After parsing $x > 1$, the type-dependency graph of (23) contains two R-vertices for the two possible readings of $x > 1$ and two further R-vertices for the two corresponding type assignments of x. The term $x^2 + 1$ is then parsed with only a single reading, in which x has to be of type i. Hence the R-vertex of the type assignment $x_{[i]\to i}$ gets deleted from the graph, and consequently the reading $x_{[i]\to i} >_{[[i]\to i,[i]\to i]\to o} 1_{[i]\to i}$ of $x > 1$ that depends on this type assignment also gets deleted from the graph, thus indicating that $x > 1$ has been disambiguated based on type information.

We now describe the algorithm that takes care of these modifications of the type-dependency graph. In order to increase the readability of this description, we write "r_1 depends on r_2" instead of "there is a D-edge from r_1 to r_2 in the current type-dependency graph" and "r is a reading of e" instead of "there is an R-edge from r to e in the current type-dependency graph". When we say "add a vertex" or "add an edge", we mean that it should be added to the current type-dependency graph. When we say "delete v" for some vertex v of the current type-dependency graph, we mean that the vertex v and all edges that begin or end in v should be deleted from the current type-dependency graph.

After parsing a symbolic expression E,

1. add an E-vertex e representing E,
2. for every reading R of E,
   2.1. add a vertex r representing R,

2.2. for every accessible variable v in E whose antecedent is in the current sentence, add a D-edge from r to the R-vertex representing the type assignment that v has according to the reading R,

2.3. for every variable v implicitly introduced in E,

2.3.1. add an E-vertex e_v representing the variable v,

2.3.2. add an R-vertex r_v representing the type assignment that v has according to the reading R,

2.3.3. add an R-edge from r_v to e_v,

2.3.4. add a D-edge from r to r_v,

2.4. for every R-vertex t (representing a type assignment) such that r depends on t,

2.4.1. for every R-vertex t' such that t and t' are distinct readings of the same E-vertex and such that no reading $r' \neq r$ of e depends on t',

2.4.1.1. for every R-vertex r' depending on t',

2.4.1.1.1. let $e_{r'}$ be the E-vertex that r' is a reading of,

2.4.1.1.2. for every R-vertex $t'' \neq t'$ such that r' depends on t'' and such that no other reading of $e_{r'}$ depends on t'', do 2.4.1.1 and 2.4.1.2 with t' replaced by t'',

2.4.1.1.3. delete r',

2.4.1.2. delete t'.

7.4.6 Quantterm grammar

As already mentioned in section 7.1, quantterms correspond to the quantifiable terms of *PTL* (see section 5.2.1). However, the syntax of quantterms is much more flexible than that of the quantifiable terms of *PTL*. In *PTL* all functions are written in a way that corresponds to the `classical` notational type of the Naproche CNL. This also holds for the quantifiable terms of *PTL*. Quantterms on the other hand allow for all notational types apart from `quantifier`.

This additional syntactic flexibility is also a potential source of ambiguities. Below we will discuss the disambiguation principles used by the quantterm grammar. But in order to make the potential ambiguities tractable, we needed to impose some very natural syntactic limitation to quantterms that the quantifiable terms of *PTL* are not subject to. This limitation is natural in the sense that it is usually followed in the language of mathematics for purely pragmatic reasons: The usage of quantterms that do not follow this limitation would not increase the expressibility of the language, but would make quantterms very hard to parse and disambiguate.[22]

This additional syntactic limitation can be phrased as follows: While the head function of a complex quantterm may recursively be any quantterm, its arguments may only be accessible quantterms. Let us illustrate this limitation

[22] In the case of the quantifiable terms of *PTL*, we left out this additional syntactic limitation, since the limited syntax of *PTL* makes parsing *PTL* texts completely unproblematic at any rate, so that this additional syntactic limitation would have been an unnecessary complication at that point of the theoretic development.

by considering some examples. Examples (24), (25) and (26) should be considered to appear at a position in a text at which a unary classical function symbol f and a ternary classical relation symbol R are accessible, but at which no variable named z is accessible.

(24) For all x, y there is some $g_x(y)$ such that $R(x, y, g_x(y))$.

(25) *For all x, y there is some $g_x(f(y))$ such that $R(x, y, g_x(f(y)))$.

(26) *For all x, y there is some $g_z(y)$ such that $R(x, y, g_z(y))$.

(24) has the same truth conditions as (27), but additionally implicitly introduces a function symbol of notational type `[circumfix,classical]` named `g_{[arg]}`.

(27) For all x, y there is some z such that $R(x, y, z)$.

The quantterm $g_x(y)$ in (24) fulfils the additional syntactic limitation, since its argument y is accessible and its head function g_x is again a quantterm satisfying this syntactic limitation, as its argument is the accessible variable x.

The quantterm $g_x(f(y))$ in (25) does not satisfy the additional syntactic limitation, since its argument $f(y)$ is not an accessible quantterm. So (25) is not accepted in the Naproche CNL. And indeed it does not seem to be acceptable in the natural language of mathematics either. The reason for this seems to be that is it not clear what it should mean to make $g_x(f(y))$ dependent on $f(y)$ by mentioning $f(y)$ as an argument. If one were forced to interpret (25) in some sensible way, one could maybe with an increased amount of imagination interpret it as (28); but if this meaning was really intended, one would usually write it as in (28) and not in the hardly interpretable manner of (25).

(28) For all x, z such that for some y $z = f(y)$, there is some $g_x(z)$ such that $R(x, y, g_x(z))$.

The quantterm $g_z(y)$ in (26) also does not fulfil the additional syntactic limitation: Even though its only argument y is an accessible quantterm, its head function g_z does not fulfil the additional syntactic limitation, as its argument z is not accessible. So (26) is not accepted in the Naproche CNL. And indeed it does not seem to be acceptable in the natural language of mathematics either: It makes no sense to mention an argument like z in a quantterm as if the value of the quantterm was dependent on the value of z, when z is not accessible and hence only a meaningless symbol.

Now the quantterm grammar without the additional disambiguation principles to be presented below can be concisely described as follows: A quantterm can either be simple, in which case it is any symbol or circumfix function name, or it can be a complex quantterm, in which case it is a function of a given notational type, which itself is a quantterm according to this grammar, applied to arguments which are accessible quantterms, where of course the application of the function to its arguments follows the rules of the given notational type.

Disambiguating quantterms

Now one problem is that the quantterm grammar finds a number of possible readings for any input. For example, $f(x, y)$ can be interpreted in four ways:

1. as two-place classical function f (depending on x and y)
2. as two-place circumfix function `f([arg],[arg])` (depending on x and y)
3. as one-place circumfix function `f([arg],y)` (depending on x)
4. as one-place circumfix function `f(x,[arg])` (depending on y).

Here we want to choose the first reading as the preferred reading to be used by the program. This is done by a special algorithm for selecting the preferred reading, which works as follows:

- Non-`circumfix` readings are always preferred over `circumfix` readings.
- Between two `circumfix` readings, one is preferred over the other if its `circumfix` name has an `[arg]` at a place, where the other has a symbol.
- A reading that has `classical` in the second position of the notational type list is preferred over one that does not. (This principle is needed, for example, to ensure that in $f'(x)$, $'$ is interpreted as a suffix function making f' classical rather than as a classical function making $'(x)$ a suffix function.)
- When none of the above rules decides which reading is better, we recursively check which head function is preferred by those rules.

Anaphoric accessibility

As mentioned in section 7.1, the anaphoric accessibility of quantterms is modelled by the notion of *active quantifiers* (**aq**) in *PTL* (see definition 5.2.4 in section PTL scope). The only essential difference between the notions of active quantifiers in *DPL* and *PTL* is that in *PTL* a quantifier occurring inside the scope of an implication can still be active outside the scope of the implication, because the implication may have implicitly introduced a function symbol through that quantifier. We repeat the definition of **aq** for implications:

$$\begin{aligned}\mathbf{aq}(\varphi \to \theta) := \{(\exists t, t_0) \mid\ & \mathbf{aq}(\varphi) = \{(\exists t'_1, t_1), \ldots, (\exists t'_n, t_n)\} \text{ for } n \geq 1,\\ & (\exists t, t') \in \mathbf{aq}(\theta) \text{ and there is an } n\text{-place argument filler } \sigma\\ & \text{such that } t' = t_0^{\sigma}(t_1, \ldots, t_n)\}\end{aligned}$$

When $t' = t_0^{\sigma}(t_1, \ldots, t_n)$ holds for some n-place argument filler σ, we say that t' is a *quantterm for the function* t_0 *dependent on* $t_1, \ldots, t_n$. In order to keep track of which quantterms are anaphorically accessible, the Naproche system needs to check for each implication whether any quantterm introduced in the consequence of the implication is a quantterm for some function dependent on the quantterms introduced in the antecedent of the implication. As we will see below, the procedure for checking this will also be needed for correctly parsing some expressions that introduce functions in a more explicit way than the implicit dynamic function introduction.

Dependent quantterms

In the above example (24), the quantterm $g_x(y)$ depended on the two variables x and y, which had to be quantificationally introduced beforehand. Another way of introducing variables on which a quantterm may depend is with a $\mapsto$-construct:

(29) There is some function $x, y \mapsto g_x(y)$ such that for all x, y, $R(x, y, g_x(y))$.

Such $\mapsto$-constructs are called *dependent quantterms*. A dependent quantterm always consists of a list of variables, separated by commas, followed by $\mapsto$, followed by a quantterm. The quantterm following $\mapsto$ must be a quantterm for some function dependent on the variables preceding $\mapsto$. This criterion has preference over the disambiguation criteria mentioned above: If there are any readings fulfilling this criterion, the chosen reading is that reading fulfilling this criterion that ranks highest according to the disambiguation criteria above.

Definition quantterms

As mentioned in section 7.3.7 above, there is a special kind of quantterms for definitions. Just like dependent quantterms, *definition quantterms* can be considered a mechanism for introducing functions in an explicit rather than implicit way.

Let us first consider a simple example of a definition quantterm. Suppose (30) occurs in a context where x is not accessible:

(30) Define $x\&$ to be $x + x$.

In this definition, x is used as a dummy variable for defining the unary suffix function &. One problem faced by any system aimed at parsing such definitions is that the system has to recognize which parts of the definition quantterm (in this case $x\&$) make up dummy variables. In order to make this problem tractable, we impose the very sensible restriction that the dummy variable may not be accessible.[23] Now in order to determine which parts of the quantterm make up dummy variables, we actually parse the definiens (here $x+x$) first, and determine which variables are *implicitly introduced variables* according to the definition in section 7.4.3. In this example, this would be only x, which would hence be considered a dummy variable also for parsing and disambiguating $x\&$.

In such simple cases, the disambiguation of definition quantterms works in the same way as the disambiguation of the quantterm following $\mapsto$ in a dependent quantterm, where the variables implicitly introduced in the definiens take over the role of the variables preceding $\mapsto$ in a dependent quantterm.

The parsing and disambiguation of definition quantterms is a bit more complex in the case of definitions that define the value of a function at a fixed argument. Consider for this example (31) appearing in a context where x is an accessible variable and $'$ is an accessible unary suffix function:

(31) For defining ! at x', define $x'!$ to be $x' \cdot x!$.

[23] Using an accessible variable as a dummy variable for a definition would at any rate be considered very bad style in the natural language of mathematics.

The x' in "For defining ! at x'" is parsed according to the term grammar, whereas the ! is parsed according to the quantterm grammar. We now know that the quantterm $x'!$ has to define ! at the value x'. With this restriction, it is clear that $x'!$ can only be parsed as a unary suffix function applied to the argument x', which is the intended interpretation. This example is especially simple, since the quantterm $x'!$ does not contain dummy variables. We now consider a somewhat more complex example with a dummy variable, which comes from the Naproche adaptation of Landau's *Grundlagen der Analysis* discussed in chapter 8. Suppose that (32) appears in a context where x is an accessible variable, $'$ is an accessible unary suffix function and $+$ is an accessible function of notational type `[suffix,prefix]`, which has so far only been defined at the value x (in other words, $+$ can be treated as if it were a binary infix function so far only defined when its first argument is x).

(32) For defining $+$ at x', define $x' + y$ to be $(x + y)'$.

(32) now defines $+$ at x'. Applying the `[suffix,prefix]` function $+$ to x' gives a prefix function $x'+$, which in $x' + y$ is further applied to the dummy variable y. The definition quantterm parser has to automatically recognize this interpretation of the definition quantterm $x'+y$. Since y is implicitly introduced in $(x + y)'$, the parser has the information that y is a dummy variable. Now for the parsing process, the additional syntactic restriction of quantterms has to be loosened somewhat: The arguments of a complex definition quantterm may not only be variables, but also the term that we expect as a fixed argument (in this case x'). With this loosened restriction, the parser produces in total 16 readings:[24]

1. $+_{\texttt{[infix]}}(x', y)$
2. $+_{\texttt{[suffix,infix]}}(')_{\texttt{[infix]}}(x, y)$
3. $'_{\texttt{[prefix,infix]}}(+)_{\texttt{[infix]}}(x, y)$
4. $'_{\texttt{[suffix,prefix,prefix]}}(x)_{\texttt{[prefix,prefix]}}(+)_{\texttt{[prefix]}}(y)$
5. $'_{\texttt{[prefix,suffix,prefix]}}(+)_{\texttt{[suffix,prefix]}}(x)_{\texttt{[prefix]}}(y)$
6. $'_{\texttt{[prefix,prefix,suffix]}}(+)_{\texttt{[prefix,suffix]}}(y)_{\texttt{[suffix]}}(x)$
7. $+_{\texttt{[prefix,suffix]}}(y)_{\texttt{[prefix]}}(x')$
8. $+_{\texttt{[suffix,prefix]}}(x')_{\texttt{[prefix]}}(y)$
9. $+_{\texttt{[prefix,suffix,suffix]}}(y)_{\texttt{[suffix,suffix]}}(')_{\texttt{[suffix]}}(x)$
10. $+_{\texttt{[suffix,prefix,suffix]}}(')_{\texttt{[prefix,suffix]}}(y)_{\texttt{[suffix]}}(x)$
11. $+_{\texttt{[suffix,suffix,prefix]}}(')_{\texttt{[suffix,prefix]}}(x)_{\texttt{[prefix]}}(y)$
12. $y_{\texttt{[suffix,suffix]}}(+)_{\texttt{[suffix]}}(x')$

[24] In order to write down a reading, we write its functional applications in classical notation, but note as a subscript index to the corresponding function the actual notational type that the function has according to that reading. When x' appears as a fixed argument, we just write x' without indicating its parse tree.

13. $y_{[\texttt{suffix},\texttt{suffix},\texttt{suffix}]}(+)_{[\texttt{suffix},\texttt{suffix}]}(')_{[\texttt{suffix}]}(x)$
14. $x_{[\texttt{prefix},\texttt{prefix},\texttt{prefix}]}(')_{[\texttt{prefix},\texttt{prefix}]}(+)_{[\texttt{prefix}]}(y)$
15. $\texttt{x}[\texttt{arg}]+_{[\texttt{circumfix},\texttt{prefix}]}(')_{[\texttt{prefix}]}(y)$
16. $'[\texttt{arg}]\texttt{y}_{[\texttt{circumfix},\texttt{suffix}]}(+)_{[\texttt{prefix}]}(x)$

Now the restrictions for choosing the preferred reading are as follows:

1. The fixed argument (in this case x') has to appear as a single argument to the function mentioned at the beginning of the definition (in this case x).
2. The dummy variables (in this case only y) have to make up all arguments of a sequence of consecutive function applications. (In a case of a single dummy variable, this restriction simplifies to the restriction that the dummy variable must be the single argument of some function application.)

In our example, the only reading that fulfils both restrictions is reading 8. If more than one reading fulfils these requirements, the reading that ranks highest according to the disambiguation criteria above is chosen.

7.4.7 Comparison to Ganesalingam's solution

The only work outside Naproche we are aware of that recognizes the problem of parsing and disambiguating symbolic mathematics as intertwined with the natural language component of mathematical texts and as of a completely different kind than parsing formal languages is Ganesalingam (2009). There are, however, two main differences between Ganesalingam's approach and ours:

Firstly, Ganesalingam has the methodological principle that no mathematical content is encoded directly into his theory, and he considers such syntactic disambiguation principles as the precedence of multiplication over addition as part of mathematical content.[25] Thus he does not encode such principles into his theory, but requires the author to write sentences of the following form in order to get the desired disambiguation of arithmetic expressions:

(33) If m, n and k are natural numbers, then "$m + nk$" means "$m + (nk)$".

We on the other hand do not want to require the author to write things that mathematicians do not normally write, and so decided to encode some basic syntactic disambiguation principles directly into our theory.

Secondly, as already alluded in section 7.4.1, Ganesalingam relies much more heavily on a type system than we do for disambiguating symbolic mathematics. This is due to the fact that he does not include presuppositions into the disambiguation machinery. By making use of presuppositions for disambiguation, we were able to attain similar goals as Ganesalingam with a much more coarse type system. One of the benefits of the coarseness of the type system is that we do not require the author to make statements whose only goal is to influence the typing of symbolic material.

[25] See page 105 in Ganesalingam (2009).

7.5 Naproche CNL semantics

We endow the Naproche CNL with a semantics by specifying a translation from Naproche CNL texts to *PTL* texts. In this way, the Naproche CNL does not only inherit the model-theoretic semantics of *PTL* defined in section 5.2.2, but also a procedural semantics based on the proof checking algorithm defined on *PTL*. For example, the Naproche CNL text fragments (34) and (35) are translated into *PTL* texts of the form (36) and (37) respectively, which are equivalent according to the model-theoretic semantics of *PTL* defined in section 5.2.2:

(34) There is an x such that $P(x)$. Then $R(x)$.

(35) There is an x such that $P(x)$ and $R(x)$.

(36) $\exists x\ P(x)\ \&\ R(x)$

(37) $\exists x\ (P(x) \wedge R(x))$

But the proof checking algorithm treats (36) and (37) differently: For proof-checking (36), it lets the automated theorem prover first check the conjecture $\exists x\ P(x)$ based on the active premise list, and next check the conjecture $R(x)$ based on the premise list now extended by the new premise $P(x)$. For proof-checking (37) on the other hand, it only sends one proof obligation to the automated theorem prover, namely to check $\exists x\ (P(x) \wedge R(x))$ based on the active premise list. This captures the difference that mathematicians feel between (34) and (35).

One can easily convince oneself of the fact that *PTL* is expressive enough to be used as a translation for Naproche CNL texts. Indeed, someone who has some experience of reformulating mathematical statements in first-order logic and who has studied the particularities of *PTL* presented in chapter 5 would not find it difficult to translate Naproche CNL texts to *PTL* texts, maybe with some exceptions: It is not intuitively clear how to treat the quantifiers in bi-implications and reversed implications and how translate definitions and some macro-grammatical structures like case distinctions. These special difficulties will be treated in sections 7.5.9, 7.5.4 and 7.5.5 below. But even those parts of the translation that a logically trained human might find intuitive are difficult to define formally. This is of course due to the very different syntactical nature of the Naproche CNL and *PTL*. Much of this chapter will be about the formal details of this Naproche-CNL-to-*PTL* translation.

Additionally to the syntactic disambiguation principles treated in section 7.3.6, we need to mention one important semantic disambiguation principle which the Naproche-CNL-to-*PTL* translation presupposes: A quantifier that is introduced earlier in a sentence is always given wider scope than a quantifier introduced later in the sentence. Here "quantifier" refers both to determiners that get rendered by quantifiers in *PTL* and to the natural language quantifiers in quantified sentences as discussed in section 7.3.4.

This disambiguation principle coincides with the natural reading of sentences with more than one quantifier in the language of mathematics,[26] with one exception, namely quantifiers appearing in the complement of a transitive noun:

[26] Of course, the preference for this reading also exists in common language use, but is much more strictly followed in the language of mathematics.

(38) A contains some divisor of every number in B.

In (38), "every" is naturally given a wider scope than "some". But since transitive nouns are so far not supported in the Naproche CNL (see section 7.7 below), this exception does not need to be taken into account in the Naproche CNL.

In sections 7.5.1 to 7.5.5, we define the translation from Naproche CNL texts to *PTL* for a restricted class of Naproche CNL texts, in order to simplify the exposition. The restriction in place here is that the Naproche CNL text to be translated does not contain any plurals, complex noun phrases coordinated with "and" or "or", variable type specifications, dependent quantterms, metalinguistic constructs, bi-implications (with "iff" or "if and only if") outside definitions or reversed implications (with the antecedent following the consequence of the implication, i.e. the sentential connective 7b from section 7.3.5). Variable type specifications will be treated in section 7.5.6; dependent quantterms will be treated in section 7.5.7; metalinguistic constituents will be treated in section 7.5.8; bi-implications and reversed implications will be treated in section 7.5.9; plurals and complex noun phrases will be treated in section 7.6.

7.5.1 *PTL* variables and IDs

In the definition of *PTL* syntax, we assumed a countably infinite supply of variables without specifying what form these variables take. For the translation of a Naproche CNL text T into *PTL*, we use the following variables:

- For every variable x used in T as a non-predefined variable and every natural number $n \geq 1$, we use x^n as a *PTL* variable.
- For every circumfix function name N of a circumfix function used in T and every natural number $n \geq 1$, we use N^n as a *PTL* variable.
- For every noun, verb, adjective or preposition w and every natural number $n \geq 1$, we use w^n as a *PTL* variable.
- We use v', v'', v''' etc. as *PTL* variables.

We use the *PTL* variable x^1 to translate the occurrence of the variable x in T where x is first introduced as well as all occurrences of x that have this first occurrence of x as their anaphoric antecedent. If x is introduced a second time, now not anaphorically linked to the first introduction of x, we translate it as x^2. x^3, x^4 etc. are used in a similar way. The same can be said about *PTL* variables of the form N^n, where N is a circumfix function name.

The predefined variables of the symbolic part of the Naproche CNL naturally correspond to logical relation symbols, connectives and quantifiers of *PTL*,[27] and are hence not translated by *PTL* variables.

In the case of nouns, verbs, adjectives and prepositions, we consider every definition to be a dynamic existential introduction of the defined word. Additionally, for nouns, a collection complement of the form "of objects called w", where w is a plural noun, is also considered to existentially introduce the noun w. The notion of anaphoric accessibility is also applied to occurrences

[27] $\leftrightarrow$ and $\forall$ do not have correspondents in basic *PTL*; but taking into account that the abbreviations $\forall t\ \varphi$ and $(\varphi \leftrightarrow \psi)$ have been defined for the *PTL* formulae $(\exists t\ \top \rightarrow \varphi)$ and $(\Diamond(\varphi \rightarrow \psi) \wedge \Diamond(\psi \rightarrow \varphi))$, the intended translations for $\leftrightarrow$ and $\forall$ in *PTL* become obvious.

of nouns, verbs, adjectives and prepositions: These existential introductions of such words can serve as antecedents for later occurrences of the same word, provided that the constraints of anaphoric accessibility defined through the notion of active quantifiers are fulfilled (compare section 7.1). If a word is existentially introduced more than once and more than one of its existential introductions is anaphorically accessible at a given point in the text, we consider the most recent existential introduction of the word to be its anaphoric antecedent.

Occurrences of nouns, verbs, adjectives and prepositions in positions where such an existential introduction is not accessible are called *global uses* of these words. For translating a global use of a word w, we use w^1. The occurrence of a word in a definition or collection complement that introduces that word for the first time in a text is translated by w^2, and so are all occurrences of the same word anaphorically linked to this introduction. If a word is introduced through a definition or collection complement more than once, its later introductions are translated by w^3, w^4 etc.

The variables v', v'', v''' etc. are used in the translation whenever we need a variable which does not have a direct correspondent in the Naproche CNL text that we are translating. The first time we need such a variable we use v', the second time v'' etc. In the definition of the semantics of expressions that trigger the use of such variables, we use the symbol v to denote that such a variable should be used; in the actual translation algorithm, such a v would have to be replaced by the first variable in the sequence $v', v'', v''', \ldots$ that has so far not been used.

Similarly to the case of *PTL* variables, we assumed in the definition of *PTL* syntax a countably infinite supply of IDs without specifying what form these IDs take. In a Naproche CNL text, what corresponds to these IDs of *PTL* syntax are the names of axioms, theorems, lemmas and definitions, always used after one of the words "axiom", "theorem", "lemma" and "definition" in a heading or reference. For the translation of a Naproche CNL text T into *PTL*, we use IDs of the form t_N, where t is to be replaced by the ID type ("axiom", "theorem", "lemma" and "definition"), N is to be replaced by the name used in T corresponding to the ID and n is to be replaced by a natural number in order to distinguish IDs in case that the same name is introduced in combination with the same ID type more than once in T, just as we did in the case of natural numbers.

7.5.2 Simplified Naproche-CNL-to-*PTL* translation

In this section, we define the Naproche-CNL-to-*PTL* translation for simple declarative sentences. Additionally to the restrictions already mentioned above, we here assume one additional restriction, namely that the sentence to be translated does not contain terms with implicitly introduced variables. Implicitly introduced variables will be treated in section 7.5.3 below.

For symbolic expressions that have been parsed and disambiguated according to the rules specified in section 7.4, the translation into a *PTL* text can be read of directly from the disambiguated parse tree: We just need to ignore the notational types writing all function applications in the `classical` syntax, replace variables and circumfix functions by the corresponding *PTL* variables as specified in section 7.5.1 above, and ensure that chained formulae are translated by conjunctions as specified in section 7.4.3. Furthermore, since the existential

quantifier $\exists$ appearing in mathematical formula usually does not have the dynamic interpretation of the $\exists$ of *PTL* but the static interpretation of the $\exists$ in *PL*, we prefix the static operator $\diamond$ to any existentially quantified subformula of the *PTL* translation of a symbolic expression.

For defining the translation of NP-VP-sentences and quantified sentences, we need to define semantics for such constituents as nouns, noun phrases, verb and verb phrases. It does not make much sense to define their semantics to be certain fixed *PTL* terms or formulae. Instead, we will make use of a common technique in formal semantics, namely to define the semantics of such constituents using a variety of the *lambda calculus*. For a detailed exposition of the application of the lambda calculus to formal linguistics, see for example Blackburn and Bos (2005).

In our case, we use the lambda calculus to describe the construction of certain strings of symbols, which turn out to be *PTL* texts. For example, $\lambda x.\exists v\ (x \wedge P(v))$ denotes the function that maps any string x to the string resulting from concatenating the strings "$\exists v\ ($", x and "$\wedge P(v))$". When this lambda term is applied to $R(v)$, which we write as $\lambda x.\exists v\ (x \wedge P(v))@R(v)$, then the result is the string $\exists v\ (R(v) \wedge P(v))$, which is a well-formed *PTL* text.

In some cases, the semantics that we need to give to a certain constituent has two parts. In such cases, we write the semantics as a pair (a_1, a_2), where a_1 and a_2 are lambda terms. In order to conveniently refer to the elements a_1 and a_2 of such a pair $a = (a_1, a_2)$, we use the standard notation $\pi_1(a)$ and $\pi_2(a)$ for a_1 and a_2 respectively.[28] When we say "lambda term", we mean any term constructed from strings of *PTL* symbols using lambda abstraction, application with @, pairing and the functions π_1 and π_2.

In order to define a semantics for every possible constituent of the Naproche CNL, we map every terminal constituent of the textual grammar to a lambda term, which constitutes the semantics of this constituent, and additionally map every grammatical rule of the textual grammar to a lambda term, which we call the semantics of the rule. If a grammatical rule is used to form a constituent c out of the constituents $c_1, \ldots c_n$, then the semantics of c is defined to be the semantics of the rule applied sequentially to the lambda terms representing the semantics of $c_1, \ldots c_n$. Let us illustrate this through a simple example. Consider the following *PTL* text:

(39) Some integer k is even.

By what we have already said, the semantics of the quantterm k is the *PTL* term k^1. By the explanations in section 7.5.1, "integer" and "even" will be translated with the help of the variables $integer^1$ and $even^1$. More precisely, the semantics that we give to the noun "integer" and the adjective "even" are $\lambda x.integer^1(x)$ and $\lambda x.even^1(x)$. The determiner "some" gets the semantics $\lambda x.\lambda y.\exists \pi_2(x)\ (\pi_1(x)@\pi_2(x) \wedge y@\pi_2(x))$.

The grammatical rule that forms the core of a noun phrase out of a noun and a quantterm[29] is given the semantics $\lambda x.\lambda y.(x, y)$. Hence the semantics

[28] Note that this pair notation should not be confused with the tuple notation τ_n defined inside *PTL*. *PTL* terms and texts are now just viewed as strings of symbols, and among the allowed symbols in these strings are these τ_n. Additionally we now have a notation for talking about pairs of strings.

[29] Since we are currently ignoring plurals, the quantterm list in a noun phrase whose core does not lack quantterms altogether must consist of precisely one quantterm.

of the noun phrase core "integer k" is $\lambda x.\lambda y.(x, y)$ applied sequentially to $\lambda x.integer^1(x)$ and k^1, i.e. $(\lambda x.integer^1(x), k^1)$. The grammatical rule for forming a noun phrase out of a determiner and a noun phrase core is given the semantics $\lambda x.\lambda y.x@y$. Hence the semantics of "some integer k" is $\lambda x.\lambda y.x@y$ applied sequentially to $\lambda x.\lambda y.\exists\pi_2(x)\ (\pi_1(x)@\pi_2(x) \wedge y@\pi_2(x))$ and $(\lambda x.integer^1(x), k^1)$, i.e. $\lambda y.\exists k^1\ (integer^1(k^1) \wedge y@k^1)$.

The copula gets the semantics $\lambda x.x$, and the grammatical rule for forming a verb phrase out of the copula and an intransitive adjective is given the semantics $\lambda x.\lambda y.x@y$. Hence the semantics of "is even" is $\lambda x.\lambda y.x@y$ applied sequentially to $\lambda x.x$ and $\lambda x.even^1(x)$, i.e. $\lambda x.even^1(x)$. The rule for forming an NP-VP-sentence out of a noun phrase and a verb phrase is also given the semantics $\lambda x.\lambda y.x@y$. So the semantics of (39) is $\lambda x.\lambda y.x@y$ applied sequentially to $\lambda y.\exists k^1\ (integer^1(k^1) \wedge y@k^1)$ and $\lambda x.even^1(x)$, i.e. the following *PTL* text:

(40) $\exists k^1\ (integer^1(k^1) \wedge even^1(k^1))$

It is easily seen that (40) reflects the natural meaning of (39). Furthermore, the usage of k^1 as variable for the dynamic existential quantification ensures that if further sentences are added to the Naproche CNL text (39) and these sentences anaphorically refer to the k in (39), then the *PTL* translation will have k^1 as a translation of k, which will be bound by the existential quantifier $\exists k^1$ of (40), as semantically required.

We get precisely the same result as above if the rule for forming a verb phrase out of the copula and an intransitive adjective is given the semantics $\lambda x.\lambda y.y$ instead of $\lambda x.\lambda y.x@y$. What we do in this case is that we just ignore the semantics of the copula. When we specify the semantics of all grammatical rules below, we will actually always ignore the semantics of the copula in grammatical rules that involve the copula. Hence the copula does not need to be given any semantics. Other constituents that similarly do not need to be given any semantics are "such that", "satisfying", the comma or "and" used to separate reference cores in a reference, the "then" of an *if-then*-construct and the "of (objects called)" in a collection complement.

We now define the semantics of all terminal constituents and grammatical rules needed in simple declarative sentences that adhere to the restrictions mentioned at the beginning of this section. There is not much we can say in the way of explaining these formal definitions, besides appealing to the interested reader to try out the functioning of these definitions in some simple example sentences, in order to convince himself of the fact that these definitions do coincide with the intuitive way that a logically trained person with knowledge of *PTL* would translate from the Naproche CNL to *PTL*.

We start with the semantics of the terminal constituents:

- A noun N which is represented by the *PTL* variable N^n according to the explanation in section 7.5.1: $(\lambda x.N^n(x), N^n)$
- An intransitive verb V which is represented by the *PTL* variable V^n according to the explanation in section 7.5.1: $\lambda x.V^n(x)$
- A transitive verb V which is represented by the *PTL* variable V^n according to the explanation in section 7.5.1: $\lambda x.\lambda y.V^n(y, x)$
- An intransitive adjective A which is represented by the *PTL* variable A^n according to the explanation in section 7.5.1: $\lambda x.A^n(x)$

- A transitive adjective A which is represented by the *PTL* variable A^n according to the explanation in section 7.5.1: $\lambda x.\lambda y.A^n(y,x)$
- A preposition P which is represented by the *PTL* variable P^n according to the explanation in section 7.5.1: $\lambda x.\lambda y.P^n(y,x)$
- A reference core R which is represented by the *PTL* ID i_R according to the explanation in section 7.5.1: i_R
- The indefinite determiners "a", "an" and "some": $\lambda x.\lambda y.\exists\pi_2(x)\ (\pi_1(x)@\pi_2(x) \wedge y@\pi_2(x))$
- The negative determiner "no": $\lambda x.\lambda y.\neg\exists\pi_2(x)\ (\pi_1(x)@\pi_2(x) \wedge y@\pi_2(x))$
- The universal determiner "every": $\lambda x.\lambda y.\exists\pi_2(x)\ \pi_1(x) \rightarrow y@\pi_2(x)$
- The definite determiner "the": $\lambda x.\lambda y.y@\iota\pi_2(x)\ \pi_1(x)$
- Inflected forms of "there to be at most one": $\lambda x.\exists\pi_2(x)\ \exists v\ (\pi_1(x)@\pi_2(x) \wedge \pi_1(x)@v) \rightarrow \pi_2(x) = v$
- Inflected forms of "there to be precisely one": $\lambda x.\exists\pi_2(x)\ (\pi_1(x)@\pi_2(x) \wedge (\exists v\ \pi_1(x)@v \rightarrow \pi_2(x) = v))$
- Inflected forms of "there to be" and "there to exist": $\lambda x.x@\top$
- "by" used in references: $\lambda x.\lambda y.\mathit{ref}(x,y)$
- The sentential connectives "and", ", and", ", i.e." and ", so": $\lambda x.\lambda y.x \wedge y$
- The sentential connectives "or" and ", or": $\lambda x.\lambda y.x \vee y$
- "if" and "implies": $\lambda x.\lambda y.x \rightarrow y$
- Inflected forms of "it to be false that" and "it not to be the case that": $\lambda x.\neg x$
- Inflected forms of "it to be the case that": $\lambda x.x$

Some terminal constituents have a special logical or *CMTN*-theoretical meaning when they are used globally (see section 7.5.1 above for the definition of *global use*). The semantics presented for them below are exceptions to some of the classes presented above. In the case of the nouns in this list, the second element of the pair that defines their semantics is irrelevant (since it only plays a role in forming collection complements starting with "of objects called", and the usage of a noun in such a collection complement is never a global use; see section 7.5.1). Hence we just place a dummy v in the position of this irrelevant element in the below definition.

- The transitive adjective "distinct": $\lambda x.\lambda y.\neg y = x$
- Inflected forms of "class": $(\lambda x.C(x), v)$
- Inflected forms of "set": $(\lambda x.L(x) \wedge C(x), v)$
- Inflected forms of "to belong to": $(\lambda x.\lambda y.y \in x, v)$

- Inflected forms of "to contain": $(\lambda x.\lambda y.x \in y, v)$
- Inflected forms of "map": $(\lambda x.\Diamond\exists v\ M(x,v), v)$
- Inflected forms of "function": $(\lambda x.L(x) \wedge \Diamond\exists v\ M(x,v), v)$
- Inflected forms of "relation": $(\lambda x.\Diamond\exists v\ M(x,v), v)$
- The intransitive adjective "unary": $\lambda x.M(x, s(0))$
- The intransitive adjective "binary": $\lambda x.M(x, s(s(0)))$
- The intransitive adjective "ternary": $\lambda x.M(x, s(s(s(0))))$
- The preposition "in": $\lambda x.\lambda y.y \in x$
- Inflected forms of "tuple": $\lambda x.\exists v\ T(x,v)$
- Inflected forms of "natural number": $\lambda x.N(x)$
- Inflected forms of "object": $\lambda x.\top$

We will now present the semantics of the grammatical rules needed for constructing simple declarative sentences that adhere to the above mentioned restrictions. The expressions we use to refer to the rules are not intended as complete descriptions of the rules; some optional commas are ignored and some additional syntactic limitations may hold for them, as specified in section 7.3. But these expressions do allude to the intended way of dividing the constructed constituent into smaller constituents, since the way this division is carried out is important for using the defined lambda term for constructing a semantic representation. In these expressions, we use $\overline{\text{N}}$ to denote the part of a determiner noun phrase that follows the determiner. In the terminology of section 7.3.1, this is a noun phrase core possibly preceded by adjectives and possibly followed by postmodifiers.

We start with the grammatical rules needed for forming noun phrases:

- The rule for forming an NP out of a term: $\lambda x.\lambda y.y@x$
- The rule for forming an NP out of a determiner and an $\overline{\text{N}}$: $\lambda x.\lambda y.x@y$
- The rule for forming an $\overline{\text{N}}$ out of a noun and a quantterm: $\lambda x.\lambda y.(\pi_1(x), y)$
- The rule for forming an $\overline{\text{N}}$ out of a noun: $\lambda x.(\pi_1(x), v)$
- The rule for forming an $\overline{\text{N}}$ out of a quantterm: $\lambda x.(\top, y)$
- The rule for forming an $\overline{\text{N}}$ out of an adjective and an $\overline{\text{N}}$: $\lambda x.\lambda y.(\lambda z.(x@z \wedge \pi_1(y)@z), \pi_2(y))$
- The rule for forming an $\overline{\text{N}}$ out of an $\overline{\text{N}}$ and a collection complement or a propositional phrase: $\lambda x.\lambda y.(\lambda z.(\pi_1(x)@z \wedge y@z), \pi_2(x))$
- The rule for forming an $\overline{\text{N}}$ out of an $\overline{\text{N}}$ and a such-that clause: $\lambda x.\lambda y.(\lambda z.(\pi_1(x)@z \wedge y), \pi_2(x))$
- The rule for forming a such-that clause out of "such that" and a sentential phrase: $\lambda x.\lambda y.y$

- The rule for forming a prepositional phrase out of a preposition and a noun phrase: $\lambda x.\lambda y.\lambda z.y@(x@z)$
- The rule for forming a collection complement out of "of objects called" and a noun:
  $\lambda x.\lambda y.\lambda z.\exists \pi_2(y)\ (M(\pi_2(y), s(0)) \wedge \forall v\ B(\pi_1(y)@v) \wedge \forall v\ (v \in z \leftrightarrow \pi_1(y)@v))$
- The rule for forming a collection complement used in a definite noun phrase out of "of" and an $\overline{\text{N}}$: $\lambda x.\lambda y.\lambda z.\exists \pi_2(y)\ \top \rightarrow (\pi_2(y) \in z \leftrightarrow \pi_1(y)@\pi_2(y))$
- The rule for forming a collection complement used in a non-definite noun phrase out of "of" and an $\overline{\text{N}}$:[30]
  $\lambda x.\lambda y.\lambda z.\exists \pi_2(y)\ \top \rightarrow (\pi_2(y) \in z \rightarrow \pi_1(y)@\pi_2(y))$

Now we define the semantics of the rules needed for forming verb phrases:

- The rule for forming a VP out of an intransitive verb: $\lambda x.x$
- The rule for forming a VP out of a transitive verb and an NP:
  $\lambda x.\lambda y.\lambda z.y@(x@z)$
- The rule for forming a VP out of the copula and an NP:
  $\lambda x.\lambda y.\lambda z.(y@z)@\lambda w.w = z$
- The rule for forming a VP out of the copula and an intransitive adjective:
  $\lambda x.\lambda y.y$
- The rule for forming a VP out of the copula followed by a transitive adjective, followed by its fixed preposition, followed by a noun phrase:
  $\lambda x.\lambda y.\lambda z.\lambda w.\lambda u.w@(y@u)$
- The rule for forming a VP out of the copula and a such-that clause:
  $\lambda x.\lambda y.\lambda z.y$
- The rule for forming a VP out of the copula and a prepositional phrase:
  $\lambda x.\lambda y.y$
- The rule for transforming an affirmative into a negative VP:[31] $\lambda x.\lambda y.\neg x@y$

Finally, we consider the rules needed for forming sentential phrases:

- The rule for forming an NP-VP-sentence out of an NP and a VP:
  $\lambda x.\lambda y.x@y$
- The rule for forming a universally quantified sentence out of "for", a determiner noun phrase and a sentential phrase: $\lambda x.\lambda y.\lambda z.y@(\lambda w.z)$
- The rule for forming an existentially quantified sentence out of "there to be" or "there to exists" and a noun phrase: $\lambda x.\lambda y.x@y$

[30] The semantics of a collection complement with "of" depends on whether it is used as a postmodifier in a noun phrase whose specifier is "the" or not. For example, "the set of integers" contains all integers (and only them), whereas "a set of integers" may not contain all integers (so that the only requirement is that it contains only integers). This difference is reflected by the usage of $\leftrightarrow$ and $\rightarrow$ respectively in the formal definitions of the semantics of collection complements.

[31] We consider the affirmative VP to be the only constituent in the negative VP formed by this rule.

- The rule for forming an existentially quantified sentence out of "there to be at most one" or "there to be precisely one" and an $\overline{\text{N}}$: $\lambda x.\lambda y.x@y$
- The rule for forming a reference core list out of a reference: $\lambda x.x$
- The rule for forming a reference core list out of a reference core list followed by a comma or "and", followed by a reference core: $\lambda x.\lambda y.\lambda z.xz$
- The rule for forming a reference out of "by" and a reference core: $\lambda x.\lambda y.x@y$
- The rule for forming a sentential phrase out of a sentential phrase followed by a reference: $\lambda x.\lambda y.y@x$
- The rule for forming a sentential phrase out of a reference followed by a sentential phrase: $\lambda x.\lambda y.x@y$
- The rule for forming a sentential phrase out of a sentential phrase followed by an infix sentential connective, followed by a further sentential phrase: $\lambda x.\lambda y.\lambda z.(y@x)@z$
- The rule for forming a sentential phrase out of "if" followed by a sentential phrase, followed by "then", followed by a further sentential phrase: $\lambda x.\lambda y.\lambda z.\lambda w.(x@y)@w$
- The rule for forming a sentential phrase out of a unary prefix sentential connective followed by a sentential phrase: $\lambda x.\lambda y.x@y$

In two special cases, the formal definition of the semantics gives rise to a *PTL* representation that seems a bit more complicated than necessary. The first relates to the usage of equality for translating copula VPs whose predicative expression is a noun phrase:

(41) x is an integer.

(42) $\exists v\ (\mathit{integer}^1(v) \wedge v = x^1)$

(43) $\mathit{integer}^1(x^1)$

(41) gets translated as (42), even though the simpler equivalent (43) might seem a more intuitive translation. The translation with equality as in (42) has been chosen in order to conserve compositionality in the definition of the semantics of such VPs and in order to avoid complicated and unnecessary case distinctions. The kind of simplification that is needed to get to (43) from (42) is performed by the Naproche system at the point where *PTL* is translated to *PL* in the proof checking module.

The second such complication relates to the unnecessary appearances of $\top$ as in the translation (45) of (44):

(44) There is an integer.

(45) $\exists v\ (\mathit{integer}^1(v) \wedge \top)$

(46) $\exists v\ \mathit{integer}^1(v)$

The simpler equivalent (46) might seem more intuitive; the source of this complication is again compositionality and the avoidance of case distinctions. In the actual Naproche system, this problem does not appear because of the usage of Proof Representation Structures instead of *PTL* (see appendix C).

When we present the *PTL* translation of extended fragments of Naproche CNL text in chapter 8, we – for the sake of readability – avoid these complications and use the simplified variants.

7.5.3 Implicitly introduced variables

Consider the following sentence, appearing in a context where a unary classical relation symbol R is accessible, but x is not accessible:

(47) If $R(x)$, then x is even.

In this sentence, x is implicitly introduced in the formula $R(x)$. The occurrence of x in $R(x)$ does not have an anaphoric antecedent. But in the *PTL* translation of (47), x has to be bound by some quantifier, since otherwise the resulting *PTL* text would not be ground and hence not have the niceness properties of *PTL* texts that we defined in chapter 5 and assumed to hold for all *PTL* texts that result from Naproche texts.

For every variable implicitly introduced in a formula, we add an existential quantifier quantifying over the translation of that variable in front of the translation of the formula. Thus the translation of (47) is (48), which is also its natural reading:

(48) $\exists x^1\ R^1(x^1) \rightarrow even^1(x^1)$

Variables can also get introduced implicitly in terms that are not formulae:

(49) If $f(x)$ is even, then x is even.

In this case we can't add the existential quantifiers directly in front of the translation of the term. Instead, we add them directly in front of the smallest *PTL* formula containing the translation of the term:

(50) $\exists x^1\ even^1(f^1(x^1)) \rightarrow even^1(x^1)$

If one wants to account for the appearance of these existential quantifiers using the lambda-calculus formalism used above, we need to define the semantics of a term[32], in which the variables $v_1, \ldots, v_n$ get introduced implicitly, to be a pair consisting of the actual translation of the term into *PTL* and the sequence $\exists v_1\ \ldots \exists v_n$. This quantifier sequence is in a similar way added as additional information to superordinated constituents, until we reach a constituent that is translated by a *PTL* formula, e.g. an NP-VP-sentence. There it is prefixed to the translation defined above.

7.5.4 Definitions

Until now we have only defined the semantics of simple declarative sentences. Before we can go on to define the semantics of complete texts, we need to define the semantics of definitions.

[32]Remember that a formula is just considered a special case of a term.

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

In definitions without dummy variables, we make direct use of this analogy by rendering the definition by an existentially quantified *PTL* formula:[33]

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

(52) $\exists c^1\ c^1 = +^1(a^1, g^1(a^1))$

The translation (52) of (51) existentially introduces the *PTL* variable c^1 that corresponds to the defined symbol c. In the proof checking algorithm, this existential *PTL* formula makes the premise $c^1 = +^1(a^1, g^1(a^1))$ available for proving later assertions, just as would be expected for definition (51).

There is one point in the proof checking algorithm where one might think that the analogy between definitions and existential *PTL* formulae breaks down: The existential *PTL* formula triggers a proof obligation with an existential conjecture; in our example, this means that $\exists x_1\ (x_1 \neq u \wedge x_1 = +^1(a^1, g^1(a^1)))$ has to be proven to follow from the active premise list. For a definition, on the other hand, one would not expect any proof obligation, since it does not make an assertion but just expands the language. But note that the premise list that is active when $\exists x_1\ (x_1 \neq u \wedge x_1 = +^1(a^1, g^1(a^1)))$ has to be proven already contains the presuppositional premise $+^1(a^1, g^1(a^1)) \neq u$, since the *read_term* function in the proof checking algorithm has already checked the presuppositions of $+^1(a^1, g^1(a^1))$ (i.e. has checked that $+^1(a^1, g^1(a^1))$ is defined), and has added this presuppositional premise to the premise list. But using this premise, it becomes completely trivial to prove the conjecture $\exists x_1\ (x_1 \neq u \wedge x_1 = +^1(a^1, g^1(a^1)))$. So this difference between existential assertions and definitions is not a real issue. Hence we can say that our choice to translate definitions with existential *PTL* formulae is justified.

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

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

(54) $\exists x^1\ \top \rightarrow \exists f^1(x^1)\ f^1(x^1) = +^1(x^1, x^1)$

The principle of implicit dynamic function introduction in *PTL* now ensures that the translation (54) dynamically introduces the function symbol f^1 as a possible antecedent for subsequent parts of a *PTL* text. The conjecture of the proof obligation which (54) triggers is still an existential claim that trivially follows from the active premise list, namely $\exists x_1\ (x_1 \neq u \wedge x_1 = +^1(x^1, x^1))$ (the active premise list contains the presuppositional premise $+^1(x^1, x^1) \neq u$).

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

As mentioned in section 7.3.7, copula definitions may be preceded by an expression of the form "For defining ... at ...". These expressions may influence the way that the definition quantterm is disambiguated, but apart from that they do not influence the semantics of the definition. Given that we now already assume all symbolic expressions to be disambiguated, we can thus completely ignore these expressions when defining the translation from Naproche definitions to *PTL* formulae.

Now let us consider an example of a bi-implicational definition:

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

(56) $\exists x^1\ \mathit{integer}^1(x^1) \rightarrow \exists \mathit{even}^2(x^1)\ (\mathit{even}^2(x^1) \leftrightarrow x^1 = g^1(x^1))$

The translation (56) now triggers a proof obligation with conjecture

$$\exists x_1\ (x_1 \neq u \wedge (x_1 = \top \leftrightarrow x^1 = g^1(x^1))).$$

But this conjecture follows directly from three *CMTN* axioms which according to the explanation in section 6.1.6 get added to the active premise list, namely the Boolean axioms $\top \neq \bot$ and $\forall x\ (B(x) \leftrightarrow x = \top \vee x = \bot)$ and the sort disjointness axiom, which together with the second Boolean axiom ensures that $\top \neq u$ and $\bot \neq u$.

Having illustrated the translation of definitions through these examples, we can now proceed to giving the formal definition of this translation. We assume that the semantics of a term is as explained at the end of the previous section, i.e. a pair consisting of the actual translation of the term into *PTL* and a quantifier sequence for existentially quantifying over the implicitly introduced variables of the term.

Just as before, there are some constituents that do not need their semantics to be defined, as it would at any rate be ignored by the composition rules. But now the list of constituents that do not need a semantics does not only include terminal constituents, but also the optional premodifier to a copula definition of the form "For defining ... at ...". So the grammatical rule for forming this premodifier does not need to be given a semantics. The terminal constituents that do not need to be given a semantics are "define", the "iff" in bi-implicational definitions and the "and" in a definiendum containing a transitive adjective. We will actually not need to present the semantics of any terminal constituents, since all other terminal constituents appearing in definitions have already had their semantics defined above.

Here are the semantics of the grammatical rules needed for constructing bi-implicational definitions:

- The rule for forming a bi-implicational definition out of "define", a definiendum, "iff" or "if and only if" and a simple declarative sentence:
  $\lambda x.\lambda y.\lambda z.\lambda w.\pi_1(y) \rightarrow \exists \pi_2(y)\ (\pi_2(y) \leftrightarrow w)$
- The rule for forming a definiendum out of an $\overline{\mathrm{N}}$, "to be" and an intransitive adjective: $\lambda x.\lambda y.\lambda z.(\exists \pi_2(x)\ \pi_1(x)@\pi_2(x), z@\pi_2(x))$
- The rule for forming a definiendum out of an $\overline{\mathrm{N}}$ followed by "to be", followed by a transitive adjective, followed by its fixed preposition, followed by an $\overline{\mathrm{N}}$:
  $\lambda x.\lambda y.\lambda z.\lambda w.\lambda u.(\exists \pi_2(x)\ \exists \pi_2(u)\ (\pi_1(x)@\pi_2(x) \wedge \pi_1(u)@\pi_2(u)), (z@\pi_2(u))@\pi_2(x))$

- The rule for forming a definiendum out of an $\overline{\text{N}}$ followed by "and", followed by an $\overline{\text{N}}$, followed by "to be", followed by a transitive adjective: $\lambda x.\lambda y.\lambda z.\lambda w.\lambda u.(\exists\pi_2(x)\ \exists\pi_2(z)\ (\pi_1(x)@\pi_2(x)\wedge\pi_1(z)@\pi_2(z)), (u@\pi_2(z))@\pi_2(x))$

- The rule for forming a definiendum out of an $\overline{\text{N}}$, "to be", an indefinite determiner and an intransitive adjective: $\lambda x.\lambda y.\lambda z.\lambda w.(\exists\pi_2(x)\ \pi_1(x)@\pi_2(x), w@\pi_2(x))$

- The rule for forming a definiendum out of an $\overline{\text{N}}$ and an intransitive verb: $\lambda x.\lambda y.(\exists\pi_2(x)\ \pi_1(x)@\pi_2(x), y@\pi_2(x))$

- The rule for forming a definiendum out of an $\overline{\text{N}}$, a transitive verb and an $\overline{\text{N}}$: $\lambda x.\lambda y.\lambda z.(\exists\pi_2(x)\ \exists\pi_2(z)\ (\pi_1(x)@\pi_2(x)\wedge\pi_1(z)@\pi_2(z)), (y@\pi_2(z))@\pi_2(x))$

- The rule for forming a definiendum out of a term: $\lambda x.(\pi_2(x)\ \top, \pi_1(x))$

Here are the semantics of the grammatical rules needed for constructing copula definitions:

- The rule for forming a copula definition out of "define", a quantterm, "to be" and a term: $\lambda x.\lambda y.\lambda z.\lambda w.\pi_2(w)\ \top \rightarrow \exists y\ y = \pi_1(w)$

- The rule for forming a copula definition out of a premodifier of the form "For defining ... at ..." and a copula definition: $\lambda x.\lambda y.y$

7.5.5 Macro-grammatical semantics

So far we have defined the Naproche-CNL-to-*PTL* translation for simple declarative sentences and definitions. In this section we will extend the translation to complete Naproche texts.

Below we define a provisional translation for Naproche CNL texts. In the case a Naproche CNL text does not contain globally used words (see section 7.5.1), the provisional translation is the final translation for the Naproche CNL text.[34] But if a Naproche CNL text contains globally used words, the intended interpretation of the text is that what it asserts about the globally used words should hold for all relations of the corresponding arity (unary or binary).[35] This intended interpretation is formally achieved as follows: Let $w_1, \ldots, w_n$ be the words globally used in the Naproche CNL text that express unary relations (nouns and intransitive verbs and adjectives), and let $W_1, \ldots, W_m$ be the words globally used in the Naproche CNL text that express binary relations (prepositions and transitive verbs and adjectives). Then the Naproche CNL text is

[34] For the purpose of this section, we do not consider the global uses of words that were given a special logical or *CMTN*-theoretical meaning to be globally used words.

[35] Usually, a Naproche CNL text containing globally used words would start with some axioms or assumptions containing properties assumed of the relations expressed by these words. In that case, what the text asserts about the globally used words is that if they satisfy these assumed properties, then they also satisfy whatever follows these axioms or assumption. So what follows the axioms or assumptions only has to be satisfied by all relations satisfying the assumed properties, and not by all relations whatsoever.

translated by a *PTL* text of the form

$$\begin{aligned}
&\exists w_1^0\ (M(w_1^0, s(0)) \wedge \forall v\ B(w_1^0(v))) \\
&\wedge \ldots \wedge \\
&\exists w_n^0\ (M(w_n^0, s(0)) \wedge \forall v\ B(w_n^0(v))) \\
&\wedge \\
&\exists W_1^0\ (M(W_1^0, s(s(0))) \wedge \forall v\ \forall v'\ B(W_1^0(v, v'))) \\
&\wedge \ldots \wedge \\
&\exists W_m^0\ (M(W_m^0, s(s(0))) \wedge \forall v\ \forall v'\ B(W_m^0(v, v'))) \\
&\rightarrow \theta,
\end{aligned}$$

where θ is the provisional translation of the Naproche CNL text.

Assertions and assumptions that are composed of a trigger and a simple declarative sentence are translated in the same way as the simple declarative sentence in them. The semantic difference between assertions and assumptions is thus not captured in their translation, but in the way this translation gets used to form the translation of a Naproche text containing them.

We now define the translation of assertions and assumptions formed in other ways:

- The rule for forming an assertion out of the word "trivial": $\lambda x.\top$
- The rule for forming an assertion out of a reference: $\lambda x.x@\top$
- The rule for forming an assertion out of "contradiction" and a reference: $\lambda x.\lambda y.y@\bot$
- The rule for forming an assumption our of "(now) consider (arbitrary)" or "(now) fix (arbitrary)" and a quantterm: $\lambda x.\lambda y.\exists y\ \top$
- The rule for forming an assumption out of "let", a quantterm and "be given": $\lambda x.\lambda y.\lambda z.\exists y\ \top$

For defining the semantics of Naproche texts, it is useful to revise one aspect of the division of texts into structural blocks: Instead of considering axiom blocks structural blocks, we consider there to be a kind of structural blocks called *axiom-consequences block*. An axiom-consequences block always consist of what is called an axiom block in section 7.2 and text. But a text may no longer contain axiom blocks directly, but only axiom-consequences blocks. This revised division of a text into structural blocks corresponds better to the semantics we will define for texts.

Below we will define how to translate the various structural blocks into *PTL*. Only note blocks will not be given any semantics. The provisional translation of a Naproche text, and likewise the translation of *text* embedded in a structural block, is constructed by connecting with & the translations of the assertions and structural blocks – apart from note blocks – that the text is made of. Whether the & is used in a left-associative or right-associative way for forming this large conjunctions does not matter semantically.

An axiom-consequences block is always translated by a *PTL* text of the form $\varphi \rightarrow \theta$, where φ is the translation of the axiom block and θ the translation of the

text following it. If the axiom block contains no assumptions, its translation is just the conjunction of the translations of the assertions in it. If it does contain assumptions, its translation has the form $\varphi_1 \to \varphi_2$, where φ_1 is the conjunction of the translations of the assumptions and φ_2 is the conjunction of the translations of the assertions.

An assumption-consequences block is always translated by a *PTL* text of the form $\varphi \to \theta$, where φ is the translation of the assumption and θ the translation of the text following it.

If the theorem block in a theorem-proof block contains no assumption, the translation of the theorem-proof block has the form $thm(\vartheta, \varphi, \theta)$, where ϑ is the theorem type of the theorem block, φ is the conjunction of the translations of the assertions in the theorem block, and θ is the translation of the text in the proof block. If the theorem block does contain assumptions, the translation of the theorem-proof block has the form $\chi \to thm(\vartheta, \varphi, \theta)$, where χ is the conjunction of the translations of the assumptions in the theorem block, and ϑ, φ and θ are as in the previous case. (Thus the assumptions in a theorem block are not only available for the assertions in the theorem block, but also in the proof block.)

Now we define how to translate case distinction blocks. Suppose that the case distinction that we want to translate is as follows, where $N_1, \ldots, N_k$ are case names, $S_1, \ldots, S_k$ are simple declarative sentences and $T_1, \ldots, T_k$ are texts:

Case N_1: S_1.
T_1.

$\vdots$

Case N_k: S_k.
T_k.

Let $t(S_1), \ldots, t(S_k)$ be the translations of $S_1, \ldots, S_k$, and let $t(T_1), \ldots, t(T_k)$ be the translations of $T_1, \ldots, T_k$. Then the translation of this case distinction block is the following *PTL* text:

$$(S_1 \to T_1)\ \&\ \ldots\ \&\ (S_k \to T_k)\ \&\ (S_1 \vee \ldots \vee S_k)$$

For both the conjunction and the disjunction it does not matter semantically whether the connective is interpreted in a left-associative or right-associative way.

The translation of a definition block is just the translation of the definition it contains. The translation of a labelled text block is just the translation of the text in it. Statement list blocks can only occur after sentences with a cataphoric meta-NP. Thus they cannot appear in the restricted kind of Naproche texts that we are now considering; we will discuss them together with other metalinguistic features in section 7.5.8 below.

7.5.6 Variable type specifications

By now we have defined the Naproche-CNL-to-*PTL* translation for all Naproche CNL text adhering to the restrictions mentioned just before section 7.5.1. Now we will consider each of these restrictions and explain how the translation can be expanded to texts that do not have this restriction. The first restriction

that we will drop in this way is the restriction that the text should not contain variable type specifications.

Variable type specifications link a predicate to a class of variables. For example, the variable type specification (57) links the predicate $\lambda x.integer^1(x)$ to variables that are small Latin letters.

(57) Small Latin letters will stand throughout for integers.

Note that it may happen that two variable type specifications link different predicates to the same class of variables.

When a variable x gets introduced in a Naproche CNL text as a quantterm or implicitly in a term, its first appearance in the translation of the text is in an existentially quantified *PTL* formula $\exists x^n\ \varphi$. If x belongs to a class of variables to which some variable type specification has linked a predicate, we replace $\exists x^n\ \varphi$ by $\exists x^n\ (P@x^n \wedge \varphi)$, where P is the predicate which has most recently been linked to a variable class containing x by a variable type specification.

There is one special case in which this way of treating variables affected by a variable type specification does not give the desired result, namely the case of variables introduced in collection complements. Consider the following example sentence, appearing in a text in which sentence 57 has been stated:

(58) A is the set of k such that 2 divides k.

According to the above defined treatment of variables affected by a variable type specification, the translation of (58) would be (59), whereas the natural reading of (58) is (60):

(59) $A^0 = \iota v\ (C(v) \wedge L(v) \wedge \exists k^0\ (integer^0(k^0) \wedge \top) \rightarrow (k^0 \in v \leftrightarrow divide^0(2^0, k^0)))$

(60) $A^0 = \iota v\ (C(v) \wedge L(v) \wedge \exists k^0\ \top \rightarrow (k^0 \in v \leftrightarrow integer^0(k^0) \wedge divide^0(2^0, k^0)))$

The difference is that in (59), the specified set is allowed to contain non-integers; only its integer members are specified to be divisible by 2. In the natural reading (60), only integers can be members of the specified set. (The unwanted reading (59) faces the additional problem that its uniqueness presupoposition would not be fulfilled, since there is more than one set which is a superset of the even integers.)

So we need a special treatment of variables introduced in collection complements and affected by a variable type specification. Let x be a variable linked to a predicate P by an active variable type specification. Before considering the effect of variable type specifications, a collection complement whose NP has the variable x as its quantterm list is translated by a *PTL* formula of the form $\exists x\ \top \rightarrow (x \in y\ \square\ \varphi)$, where $\square$ is either $\leftrightarrow$ or $\rightarrow$. For taking the variable type specification into account, we replace this translation by $\exists x\ \top \rightarrow (x \in y\ \square\ P(x) \wedge \varphi)$.

7.5.7 Dependent quantterms

Now we drop the restriction that the text should not contain dependent quantterms. See section 7.4.6 for the definition of dependent quantterms.

We will first explain the desired translation of an example sentence involving a dependent quantterm before explaining how to translate dependent quantterms in general. Suppose that the following sentence appears in a context where the ternary relation symbol R is accessible:

(61) There is some map $x, y \mapsto F(x)(y)$ such that for every x there is a y such that $R(x, y, F(x)(y))$.

The desired *PTL* translation of (61) is (62):

(62) $(\exists x^0\ \exists y^0\ \top \rightarrow \exists F^0(x^0)(y^0)\ \top) \wedge \exists v\ M(F^0, v) \wedge$
$(\exists x^1\ \top \rightarrow \exists y^1\ R^0(x^1, y^1, F^0(x^1)(y^1)))$

Compare this to the *PTL* translation (64) of the similar sentence (63) not involving a dependent quantterm:

(63) There is some map F such that for every x there is a y such that $R(x, y, F(x)(y))$.

(64) $\exists F^0\ (\exists v\ M(F^0, v) \wedge (\exists x^1\ \top \rightarrow \exists y^1\ R^0(x^1, y^1, F^0(x^1)(y^1))))$

The only difference between (61) and (64) is in the way the *PTL* variable F^0 is dynamically introduced: In (64) it is introduced through the explicit existential quantification $\exists F^0$, whereas in (61) it is implicitly introduced through the implication $\exists x^0\ \exists y^0\ \top \rightarrow \exists F^0(x^0)(y^0)\ \top$.

Note that dependent quantterms are especially useful in combination with a variable type specification affecting the variables it depends on. Suppose for example that sentence (61) appears in a text in which the variable type specification (57) from section 7.5.6 has been stated. Then its *PTL* translation is (65):

(65) $(\exists x^0\ (\mathit{integer}^0(x^0) \wedge \exists y^0\ (\mathit{integer}^0(y^0) \wedge \top)) \rightarrow \exists F^0(x^0)(y^0)\ \top) \& \exists v\ M(F^0, v)$
$\wedge (\exists x^1\ (\mathit{integer}^0(x^0) \wedge \top) \rightarrow \exists y^1\ (\mathit{integer}^0(y^0) \wedge R^0(x^1, y^1, F^0(x^1)(y^1))))$

Here the way F^0 gets dynamically introduced allows us to conclude that it is a unary function defined on integers, whose value at every integer is again a unary function defined on integers.

Now we explain how to translate dependent quantterms in general. After parsing and disambiguating a dependent quantterm in the way explained in section 7.4.6, we have a parse tree for the functional core of the dependent quantterm (i.e. for F in the above example), a parse tree for the part following the $\mapsto$ (i.e. for $F(x)(y)$ in the above example) and a list $x_1, \ldots, x_n$ of variables listed in front of the $\mapsto$. Let t and t' be the *PTL* terms corresponding respectively to these two parse trees in the way explained in section 7.5.1 and at the beginning of section 7.5.2.

First we produce a provisional translation, in which we pretend that instead of a dependent quantterm we have a quantterm whose parse tree is the parse tree of the functional core of the dependent quantterm. In the above example, (64) is the provisional translation of (61).

Next we consider the formula of the form $\exists t\ \varphi$, where $\exists t$ is the quantification that corresponds to the quantterm that we imagined in place of the dependent quantterm. We replace $\exists t\ \varphi$ by $(\exists x_1\ \ldots \exists x_n\ \rightarrow \exists t'\ \top)\ \&\ \varphi$ in the provisional translation, thus producing the final translation. Note that this replacement should be in effect before taking care of variable type specifications in the way described in section 7.5.6.

Additionally to this general definition of the translation, we need to take care of a special case, namely when the dependent quantterm appears in an expression starting with "precisely one" or "at most one". Consider the following

example sentence, appearing in a text in which the variable type specification (57) from section 7.5.6 has been stated:

(66) There is precisely one map $x, y \mapsto F(x)(y)$ such that for every x there is a y such that $R(x, y, F(x)(y))$.

According to the above explanation, its translation would be (67):

(67) $(\exists x^0\ (\mathit{integer}^0(x^0) \wedge \exists y^0\ (\mathit{integer}^0(y^0) \wedge \top)) \rightarrow \exists F^0(x^0)(y^0)\ \top) \& \exists v\ M(F^0, v)$
$\wedge(\exists x^1\ (\mathit{integer}^0(x^0) \wedge \top) \rightarrow \exists y^1\ (\mathit{integer}^0(y^0) \wedge R^0(x^1, y^1, F^0(x^1)(y^1))))$
$\wedge(\exists v'\ \exists v\ M(v', v) \wedge (\exists x^1\ (\mathit{integer}^0(x^0) \wedge \top) \rightarrow$
$\exists y^1\ (\mathit{integer}^0(y^0) \wedge R^0(x^1, y^1, v(x^1)(y^1)))) \rightarrow F^0 = v)$

The difference between the quantification with "some" in (61) and the quantification with "precisely one" in (66) is expressed using an implication whose antecedent introduces a new variable v' and asserts of it the same properties as we have asserted of F^0, and whose consequence is the equation $F^0 = v$. The idea is of course that any object having the properties stated about F^0 is identical to F^0, i.e. that there is only one object with the stated properties. But v' is introduced explicitly using the quantification $\exists v'$, whereas F^0 is introduced implicitly using the implication

$$\exists x^0\ (\mathit{integer}^0(x^0) \wedge \exists y^0\ (\mathit{integer}^0(y^0) \wedge \top)) \rightarrow \exists F^0(x^0)(y^0)\ \top.$$

The additional information contained in this implication is not asserted of v'.

This problem is due to the fact that in the provisional translation we only changed the quantification $\exists F^0$, leaving the quantification $\exists v'$ unchanged. So what we need to do is to also replace $\exists v'\ \varphi$ by

$$(\exists x^0\ (\mathit{integer}^0(x^0) \wedge \exists y^0\ (\mathit{integer}^0(y^0) \wedge \top)) \rightarrow \exists v'(x^0)(y^0)\ \top)\ \&\ \varphi.$$

In general, the provisional translation of a sentence involving a quantification over a dependent quantterm with "precisely one" or "at most one" contains a formula of the form $\exists v\ \varphi$, where φ asserts of v the properties previously asserted of the function t introduced by the quantterm. This occurrence of $\exists v\ \varphi$ has to be replaced by $(\exists x_1\ \ldots \exists x_n\ \rightarrow \exists t'(v)\ \top)\ \&\ \varphi$, where $x_1, \ldots, x_n$ are as above and $t'(v)$ is a modification of the above t', in which the functional core t of t' has been replaced by v.

7.5.8 Metalinguistic constituents

Now we drop the restriction that the text should not contain metalinguistic constituents.

The basic idea of how to translate metalinguistic constituents is very simple. Suppose for example that in one place a Naproche CNL text contains a fragment of the following form, where S_1 and S_2 are simple declarative sentences:

Case 1: S_1.

Case 2: S_2.

Case 3: S_3.

If we let $t(S_1)$, $t(S_2)$ and $t(S_3)$ denote the translations of S_1, S_2 and (S_3) respectively, then the basic idea is to translate metasentences involving anaphoric meta-NPs referring back to these cases as shown in the following examples:

- Case 1 holds: $t(S_1)$
- Case 1 does not hold: $\neg t(S_1)$
- Case 1, case 2 and case 3 do not hold: $\neg t(S_1) \wedge \neg t(S_2) \wedge \neg t(S_3)$
- Case 1, case 2 and case 3 are inconsistent: $\neg(t(S_1) \wedge t(S_2) \wedge t(S_3))$
- At most one of case 1, case 2 and case 3 holds: $\neg(t(S_1) \wedge t(S_2)) \wedge \neg(t(S_2) \wedge t(S_3)) \wedge \neg(t(S_1) \wedge t(S_3))$
- Precisely one of case 1, case 2 and case 3 holds: $\neg(t(S_1) \wedge t(S_2)) \wedge \neg(t(S_2) \wedge t(S_3)) \wedge \neg(t(S_1) \wedge t(S_3)) \wedge (t(S_1) \vee t(S_2) \vee t(S_3))$

However, there is an issue with this basic idea. Consider, for example, the following fragment from the Naproche CNL adaptation of Landau's *Grundlagen der Analysis*:[36]

> Theorem 9: Fix x, y. Then precisely one of the following cases holds:
> Case 1: $x = y$.
> Case 2: There is a u such that $x = y + u$.
> Case 3: There is a v such that $y = x + v$.
> Proof:
> A) [...]
> B) Fix x. Let $\mathfrak{M}$ be the set of y such that precisely one of case 1, case 2 and case 3 holds.

At the position in the text where the metasentence appears, the variable x has two possible anaphoric antecedents: The occurrence of x fixed at the beginning of the theorem, and the occurrence fixed at the beginning of part B of the proof. Similarly, the variable y has two possible anaphoric antecedents, the second one being the quantterm y appearing in the expression "the set of y such that". Assuming that no instances of x and y appear before this text fragment, the translation of the sentence originally called case 1 is $x^0 = y^0$. But the intention of "case 1" in the metasentence is $x^1 = y^1$ and not $x^0 = y^0$.

One possible solution to this problem is to keep track of the surface form of a named sentence instead of its translation, and reparse this surface form when parsing the metasentence. In the above example, we would keep track of the fact that case 1 is the sentence "$x = y$", and not of the fact that the translation of case 1 is $x^0 = y^0$. When reparsing "$x = y$" in the course of parsing the metasentence, we have the later introduced x and y as possible anaphoric antecedents, which would get preferred according to the disambiguation principle mentioned at the end of section 7.4.4. Thus we would get the reading $x^1 = y^1$, as required.

But there is a serious problem with this proposed solution: If the sentence involves ambiguities which can get resolved by presupposition checking as described in section 7.4.4, the sentence might get disambiguated in a different way

[36] See appendix B for the complete text from which this is a fragment. The issue discussed here appears in the same form in Landau's original text, i.e. is not a product of the adaptation of the text to the Naproche CNL.

when reparsed. The difference between different readings might amount not just to a different anaphoric antecedent for a variable, as in the above example, but even to a different syntactical structure of a parsed formula. Such a difference in interpretation can however never be intended in the case of anaphoric meta-NPs. Hence we need a different solution.

The solution is that we do keep track of the translation of the named sentence. But when we reuse this translation in the course of translating a metasentence, we reconsider the possible anaphoric antecedents for the variables appearing freely in the terms of the named sentence. Since the set of possible anaphoric antecedents might be different than at the position where the named sentence was originally parsed, it can happen that a variable is given another anaphoric antecedent than in the original translation. But since we just modify anaphoric antecedents in the translation and do not reparse the sentence, we cannot get the problem that the previously proposed solution had.[37]

There are two more problems with the translation of metasentences involving anaphoric meta-NPs. The first is a purely syntactical technicality: If the translation of the named sentence existentially introduces a *PTL* variable, inserting a copy of this translation at a later point will cause the niceness properties of *PTL* texts defined in chapter 5 to be violated. In order to adhere to the niceness properties, we need to rename every variable x^n existentially introduced in the translation of the named sentence to x^m, where m is the smallest integer such that x^m does not yet appear in our translation.

The second problem is of a more semantic nature: If the translation of the named sentences existentially introduces a *PTL* variable x^n, the solution described so far would cause the metasentence to make a variable x^m anaphorically accessible. This does not agree with actual use in the language of mathematics. Suppose for example that a text contains the following fragment:

(68) Case 1: There is an integer x such that $R(x)$.

If at a later point, where no x is accessible, we write "Hence case 1 holds", this does not allow us to speak of x in the subsequent sentence in the way that the alternative assertion "Hence there is an integer x such that $R(x)$" does. In order to correctly model the anaphoric accessibility relation of the language of mathematics, we prefix the already modified translation of the named sentence by the operator $\diamond$, which blocks the accessibility of variables introduced in the translation.

So far we have only discussed anaphoric meta-NPs. For cataphoric meta-NPs, the basic idea is practically the same, but the problems discussed above do not arise in the same way. In the case of cataphoric meta-NPs, the sentences whose translations have to be inserted into the metasentence are only parsed after the metasentence. They have not been parsed previously, so the first problem discussed above does not arise at all. The syntactical technicality relating to the niceness property of *PTL* texts can arise, but in a different manner: In the case of metasentences involving "at most one" or "precisely one", the translation of the named sentence may have to be inserted more than once into the translation of the metasentence. In this case, we also have to rename existentially introduced *PTL* variables in all copies of the translation besides the first.

[37] Readers with a computer science background will note that this solution is comparable to the *dynamic scoping* of variables that is possible in some programming languages.

As for the last problem discussed for anaphoric meta-NPs above, cataphoric meta-NPs behave differently on this issue. Consider for example the following fragment:

> Now the following properties hold:
>
> Property 1: There is an integer k such that $R(k)$.
>
> Property 2: Some odd prime number p divides k.
>
> Observe that 3 does not divide k, so $p \neq 3$.

As this example illustrates, variables introduced by a statement in the statement list block announced by the cataphoric meta-NP may become accessible both for later statements in the statement list block and for the text following the statements list block. This, however, depends on the form of the metasentence. If instead of "the following properties hold" we have "precisely one of the following properties hold", the accessibility is blocked both between statements in the statement list block and between the statement list block and subsequent text. And if instead of "the following properties hold" we have "the following properties are inconsistent", the the accessibility between statements in the statement list block is conserved, whereas the accessibility between the statement list block and subsequent text is blocked.

The Naproche-CNL-to-*PTL* translation takes care of all these issues in the case of metasentences.

7.5.9 Bi-implications and reversed implications

Now we drop the restriction that the text should not contain bi-implications and reversed implications.

In bi-implications and reversed implications, a phenomenon similar to that of the donkey sentences discussed in section 3.1 can be observed:

(69) 2 divides an integer x iff x is even.

(70) $\forall x\ (\mathit{integer}(x) \to (\mathit{divide}(2, x) \leftrightarrow \mathit{even}(x)))$

The natural interpretation of (69) in *PL* is (70), i.e. the variable x introduced in an indefinite noun phrase in the left part ("antecedent") of the bi-implication is interpreted as globally universally quantified. This corresponds to the interpretation of mathematical *donkey sentences* like (71), where a variable introduced in an indefinite noun in the antecedent of a usual implication is interpreted as globally universally quantified:

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

The phenomenon appears in the same way in reversed implications:

(72) 2 divides an integer x if x is even.

(73) $\forall x\ (\mathit{integer}(x) \to (\mathit{even}(x) \to \mathit{divide}(2, x)))$

Reversed implications – unlike bi-implications – are common not only in the language of mathematics but also in general language use. Nevertheless, there has to our knowledge not been any systematic study of this donkey-sentence-like phenomenon for reversed implications. It might be the case that in common language it can be reduced to the phenomenon of *generic readings* of indefinite noun phrases (see A. Cohen (2002) for an overview over generic interpretation of noun phrases). A generic reading differs semantically from a universal reading in that it is restricted to *typical* members of a category. But in the language of mathematics this typicality restriction of generic readings is generally dropped and generic noun phrases are interpreted in the same way as universal quantifiers. Hence, for the purpose of our interpretation of reversed implications and bi-implications in the Naproche CNL, we can ignore this possible linguistic difference between usual donkey sentence and the phenomenon discussed here.

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

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

The first problem is that the conjunct $\mathit{integer}(x) \wedge \mathit{divide}(2, x)$ in the semantics $\exists x\ (\mathit{integer}(x) \wedge \mathit{divide}(2, x))$ that we have so far given to "2 divides an integer x" has to be split up in order to attain the intended interpretation (70): $\mathit{integer}(x)$ has to restrict the universal quantification, while $\mathit{divide}(2, x)$ has to become one argument of the logical bi-implication. So we cannot make use of the semantics $\exists x\ (\mathit{integer}(x) \wedge \mathit{divide}(2, x))$ in a compositional way.

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

The second problem is that not every existential quantification in the left part of a bi-implication is to be interpreted as a universal quantifier outside the scope of the bi-implication. Consider sentence (74), whose natural interpretation in PL is (75) and not (76):

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

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

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

If we interpreted all existential quantifications in the left part of a bi-implication in a universal way, we would get the interpretation (76), which however is not equivalent to the natural interpretation (75).

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

The basic idea for solving this second problem is simple: We give a universal interpretation only to those existential quantifiers introduced in the left part of the bi-implication which serve as anaphoric antecedents for an expression in the right part ("succedent") of the bi-implication. However, there is a problem with this basic solution. Consider for example sentence (77):

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

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

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

According to the basic solution just proposed, only l would be interpreted in a universal way, as in (78). But since l is introduced after k in the left part of the bi-implication, it is felt to be somehow dependent on k, which makes it very unnatural to give it wider scope than k. Hence the interpretation (79) which gives wide scope and a universal interpretation to both k and l is naturally preferred.

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

Before we proceed to explaining how this desired interpretation is actually produced as a *PTL* translation, we need to say a word about anaphoric accessibility in bi-implications and reversed implications: Because of this special semantic handling, the general principle mentioned in section 7.1 that the *PTL* notion of active quantifiers at a given position defines which quantterms may serve as anaphoric antecedents at the corresponding position in the text cannot be applied. Instead, we need to say that all quantterms anaphorically accessible at the end of the left part of the bi-implication or reversed implication are accessible in the right part.

Now we define the *PTL* translation that leads to the above interpretation. Suppose that we have a bi-implication or reversed implication whose left and right parts are respectively translated by the *PTL* formulae φ and ψ. For every term t such that $(\exists t, t) \in \mathbf{aq}(\varphi)$, we check whether t appears in ψ. If no such t appears in ψ, the translation of the bi-implication or reversed implication is $\varphi \leftrightarrow \psi$ or $\psi \rightarrow \varphi$ respectively. If there is such a t appearing in ψ, let t_0 be the one that is introduced latest in φ. Now let $\exists t_n, \ldots, \exists t_0$ be the occurrences of existential quantifiers $\exists t$ with $(\exists t, t) \in \mathbf{aq}(\varphi)$ that precede or are identical with $\exists t_0$. Now for every term t among $t_0, \ldots, t_n$ whose existential introduction

translates an indefinite noun phrase, the existential subformula of φ introducing t has the form $\exists t\ (\chi \wedge \bar{\chi})$. Let $\chi_1, \ldots, \chi_k$ be the collection of all subformulae of φ appearing in the position of χ in such an existential subformula of φ. Let φ' be the formula resulting from φ by removing $\exists t_n, \ldots, \exists t_0$ and $\chi_1, \ldots, \chi_k$ from φ.[38] Then the translation of the bi-implication or reversed implication is $\exists t_n \ldots \exists t_0\ (\chi_1 \wedge \ldots \wedge \chi_k) \rightarrow (\varphi' \leftrightarrow \psi)$ or $\exists t_n \ldots \exists t_0\ (\chi_1 \wedge \ldots \wedge \chi_k) \rightarrow (\psi \rightarrow \varphi')$ respectively.

7.5.10 Accommodation of presuppositions

In this section, we do not discuss a semantic phenomenon already implemented in the Naproche system, but a phenomenon which could be implemented in future versions of Naproche.

As discussed in section 3.2.4, there can be no global accommodation of presuppositions in mathematical texts, but only local accommodation. We will now discuss how local accommodation could be implemented within the framework that we have developed so far.

In the current Naproche system, a failure to prove a presupposition will always lead to the proof text not being accepted. However, as already mentioned in section 3.2.4, local accommodation of presuppositions is sometimes required for interpreting real mathematical texts, as in the following example already discussed in section 3.2.4:

> Suppose that f has n derivatives at x_0 and n is the smallest positive integer such that $f^{(n)}(x_0) \neq 0$.
>
> (Trench, 2003, p. 102)

Given the machinery developed so far, we can characterize the positions at which local accommodation is possible as follows: They are those positions of a text whose *PTL* translation gets processed within a *read_text* process of the proof checking algorithm. Remember that in *PTL* text fragments processed by a *check_text* process not embedded in a *read_text* process, every assertion has to be checked, whereas *PTL* text fragments processed within a *read_text* process do not have to be checked but just get translated to *PL*; nevertheless, the presuppositions of *PTL* text fragment processed within a *read_text* process do have to be checked.

Let us illustrate this using a simple Naproche text fragment as example. Suppose that (80) appears in a context where a binary relation $>$ has been defined on the reals and a function $x \mapsto x^{-1}$ has been defined for all reals $x \neq 0$:

(80) For every real x such that $x^{-1} > 0$, $x > 0$.

When a mathematician reads such a sentence, he does not stop at $x^{-1} > 0$ to protest that x^{-1} might not be defined since x might be zero, but instead locally accommodates that x^{-1} is defined, i.e. that $x \neq 0$. So he adds the assumptions that x is real, that $x \neq 0$ and that $x^{-1} > 0$ to the local context under which he

[38] Of course, this removal cannot be performed naively on the string of symbols that constitutes the *PTL* formula φ. For example, if φ is $\exists t_0\ (\chi_0 \wedge \bar{\chi})$, we of course intend φ' to be $\bar{\chi}$ and not the ungrammatical string $(\wedge\bar{\chi})$.

then considers the formula $x > 0$. Let us now look at the *PTL* translation (81) of (80):[39]

(81) $\exists x\ (\mathit{real}(x) \wedge x^{-1} > 0) \rightarrow x > 0$

Suppose that the (81) gets processed by *check_text*. The premise list that is active before checking (81) is assumed to contain information about the presuppositions of the function $x \mapsto x^{-1}$ in the form of the following formula:

$$\forall v\ (\mathit{real}(v) \wedge v \neq 0 \leftrightarrow v^{-1} \neq u).$$

The definition of *check_text* for formulae of the form $\varphi \rightarrow \psi$ specifies that first $\exists x\ (\mathit{real}(x) \wedge x^{-1} > 0)$ gets processed by *read_text*; so by the above characterization, local accommodation is possible while processing $\exists x\ (\mathit{real}(x) \wedge x^{-1} > 0)$. When the proof checking algorithm processes $x^{-1} > 0$, it produces a presuppositional proof obligation whose conjecture is $x^{-1} \neq u$, and whose premise list contains $\mathit{real}(x)$ as the only information about x. Since nothing in the premise list informs us that $x \neq 0$, this presuppositional proof obligation will not be proved by the automated theorem prover.

With local accommodation in place, this must no longer mean that the proof checking fails at this point. Instead, we modify the *PTL* translation in such a way that the presuppositional proof obligation no longer fails. More precisely, we modify the original *PTL* translation (81) to (82):

(82) $\exists x\ (\mathit{real}(x) \wedge \mathrm{def}(x^{-1}) \wedge x^{-1} > 0) \rightarrow x > 0$

Since in the context where this *PTL* text fragment appears, $\mathrm{def}(x^{-1})$ is equivalent to $\mathit{real}(x) \wedge x \neq 0$, this models our informal explanation of local accommodation in this example.

Having illustrated how local accommodation can work in Naproche, we want to illustrate with a similar example why accommodation only makes sense within a *read_text* process:

(83) For every real x such that $x \geq 0$, $x^{-1} \geq 0$.

(84) $\exists x\ (\mathit{real}(x) \wedge x \geq 0) \rightarrow x^{-1} \geq 0$

In this example, the processing of $x^{-1} \geq 0$ does not take place within a *read_text* process, so that we do not only have to prove the presuppositions of $x^{-1} \geq 0$ using the premise list that is active when encountering $x^{-1} \geq 0$, but also have to prove $x^{-1} \geq 0$ itself. $x^{-1} \geq 0$ produces a presuppositional proof obligation with conjecture $x^{-1} \neq u$, which can only be proven if $x \neq 0$. But nothing in the premise list of this proof obligation tells us that $x \neq 0$. If we now modified (84) to (85), we could prove the presuppositional proof obligation produced by $x^{-1} \geq 0$.

(85) $\exists x\ (\mathit{real}(x) \wedge x \geq 0) \rightarrow \mathrm{def}(x^{-1}) \wedge x^{-1} \geq 0$

[39] For readability, we have left out superscripts from the *PTL* variables in this *PTL* translation of (80), and have written the suffix function $^{-1}$ in suffix notation and the infix relation $>$ in infix notation. Below we use analogous notation for modified versions of this *PTL* text fragment and for *PL* formulae produced from this *PTL* text fragment.

But now we do not only have to prove the non-presuppositional proof obligation whose conjecture is $x^{-1} \geq 0$, but also a further non-presuppositional proof obligation whose conjecture is $x^{-1} \neq u$. Of course this new proof obligation can only be proved if $x \neq 0$, so the proof checking fails at any rate.

Of course, what happened here is that the accommodation of the presupposition of $x^{-1} \geq 0$ is a global accommodation, since inserting $\mathrm{def}(x^{-1})$ into the *PTL* formula results in a modification of the global premise list, i.e. of the global context. We already explained in section 3.2.4 why this is not possible in mathematical texts; the above example and its clarifications only show how this explanation can be recast with the formal machinery developed in the course of this thesis.

Now the proposed solution for generally allowing local accommodation in Naproche CNL texts is as follows: Whenever a presuppositional proof obligation called within a *read_text* process fails and t is the term whose definedness was checked by this presuppositional proof obligation, we replace the atomic *PTL* formula φ containing t by $\mathrm{def}(t) \wedge \varphi$.

The choice in this proposed solution to insert $\mathrm{def}(t)$ in front of the atomic formula φ containing t (rather than in front of some more complex formula containing t) amounts to local accommodation always getting performed on the most local level possible. Compare the discussion at the end of section 3.2.4, where we gave an example of a text where local accommodation is possible at more than one level. We do not think that mathematicians have a clear intuition as to which level to prefer for local accommodation in such cases, so we could have proposed a different choice for such cases. But the solution we have proposed has the advantage that it is relatively easy to explain to a mathematician what this amounts to: Our solution amounts to interpreting any atomic statement involving an undefined term as false.

There is one serious problem with the proposed solution as presented above: According to our definition of *PTL*, $\mathrm{def}(t)$ is only a legitimate *PTL* formula if t is an ι-free *PTL* term. However sometimes, for example in the citation from Trench (2003) at the beginning of this section, we need to locally accommodate presuppositions of definite descriptions, i.e. of *PTL* terms of the form $\iota x\ \varphi$. In order to solve this problem, we define an extension of *PTL* which allows for *PTL* formulae of the form $\mathrm{def}(t)$ for arbitrary *PTL* terms t. In this case, the definition of the semantics of $\mathrm{def}(t)$ needs to be modified a bit. More precisely, the only modification needed is in the definition of when $[\![\mathrm{def}(t)]\!]_g^M$ is defined (recall that previously $[\![\mathrm{def}(t)]\!]_g^M$ was always defined):

> *def*$([\![\mathrm{def}(t)]\!]_g^M)$ iff for every subterm of t of the form $\iota x\ \varphi$ that is not a subterm of another subterm of t of this form, *def*$([\![\varphi]\!]_g^M)$.

The reason for this modification is to ensure that inserting $\mathrm{def}(t)$ corresponds to accommodating on the most local level possible: For example, if $[\![\mathrm{def}(\iota x\ x = \iota y\ \varphi(x,y))]\!]_g^M$ were always defined, inserting $\mathrm{def}(\iota x\ \iota y\ \varphi(x,y))$ in front of an atomic formula containing the term $\iota x\ \iota y\ \varphi(x,y)$ would accommodate not only the presuppositions triggered by ιx, but also those triggered by ιy. But the presuppositions triggered by ιy should be accommodated within the scope of the atomic formula $x = \iota y\ \varphi(x,y)$ in order to be accommodated at the most local level possible.

The proof checking algorithm now also has to be modified in order to handle this extension of *PTL* correctly: The presuppositional proof obligations

originating from within the scope of an ι have to be marked in a special way. For this we introduce a special marker $\iota\mathbb{P}$, which is treated just like the usual marker $\mathbb{P}$ for presuppositional proof obligations in all cases apart the one spelled out below. Additionally, we need a fourth proof status value u_ι for signalling that some presuppositions within the scope of an ι have failed. Now the only parts of the definition of the proof checking algorithm that need to be modified in a significant way are the definitions of *check_text*$(\mathrm{def}(t), \Gamma, \mathbb{T}, \mu)$ and *read_term*$(\iota x\ \varphi, \Gamma, \mathbb{T}, \mu)$:

check_text$(\mathrm{def}(t), \Gamma, \mathbb{T}, \mu) = (\Gamma_1, \mathbb{T}, \nu)$:-
  read_term$(t, \Gamma, \mathbb{T}, \mu) = (\Gamma', \_, \mu')$,
  $\Gamma_1 = \langle \Phi^{\iota\mathbb{P}} \mid \Phi^{\iota\mathbb{P}} \in \Gamma' - \Gamma \rangle$,
  $\Gamma_2 = \langle \Phi^{0} \mid \Phi^{\mathbb{P}} \in \Gamma' - \Gamma \rangle$,
  $sk_{i_1}, \ldots, sk_{i_n}$ are the skolem function symbols appearing in Γ_2,
  $\Gamma' = \Gamma \oplus \langle \exists_{\langle x_1,\ldots,x_n\rangle} \bigwedge \Gamma_2 \frac{x_1}{sk_{i_1}} \cdots \frac{x_n}{sk_{i_n}} \rangle$,
  if $\mu = u_\iota$ or $\mu' = u_\iota$:
    $\nu = u_\iota$,
  else:
    if $\mu = u$:
      $\nu = u$,
    else:
      if $\mu' = u$:
        $\nu = \bot$,
      else:
        $\nu = \mu$.

read_term$(\iota x\ \varphi, \Gamma, \mathbb{T}, \mu) = (\Gamma_1, sk^{new}, \nu)$:-
  read_text$(\varphi, \langle x \rangle, \Gamma, \mathbb{T}, \mu) = (\Gamma_0, \mathbb{T}_0, \Phi, \mu_1)$,
  exist_check$(0, \Gamma_0, \exists x\ \exists_{\mathbb{T}_0}\ \Phi, \mu_1) = (\mu_2)$,
  if $\mu_1 = u_\iota$:
    $\nu = u_\iota$,
  else:
    $\nu = update(\mu_2, 0, P(\Gamma_0 \oplus \langle \exists_{\mathbb{T}_0}\ \Phi \frac{sk^{new}}{x} \rangle \vdash^{?} \forall x\ (\exists_{\mathbb{T}_0}\ \Phi \to x = sk^{new})))$,
  Γ_0' is Γ_0 with all occurrences of the marker $\mathbb{P}$ replaced by $\iota\mathbb{P}$,
  $\Gamma_1 = \Gamma_0' \oplus \langle (\exists_{\mathbb{T}_0}\ \Phi \frac{sk^{new}}{x})^{\mathbb{P}}, (\forall x\ (\exists_{\mathbb{T}_0}\ \Phi \to x = sk^{new}))^{\mathbb{P}} \rangle$.

The soundness proof for the proof checking algorithm can be adapted to show that this modified proof checking algorithm is still sound. The details of this adaptation go beyond the scope of this thesis.

7.6 Complex noun phrases and plurals[40]

In this section we describe how the semantics of sentences involving complex and plural noun phrases is defined in the Naproche CNL. For this we first need to discuss some ambiguities that plurals, both in common language and in the language of mathematics, give rise to.

The following sentence in common language is ambiguous:[41]

[40] Most parts of this section are adapted parts of Cramer and Schröder (2012).
[41] A comprehensive overview over plural readings is given by Link (1991).

(86) Three men lifted a piano.

It can mean either that three men lifted a piano together (in a single lifting act), or that there were three lifting acts, each of which involved a different man lifting a piano. The first is called the *collective* reading, the second the *distributive* reading.[42] The ambiguity arises because the agent of a lifting event can either be a collection of individuals or a single individual.

In the language of mathematics, both the collective and the distributive reading exist:

(87) 12 and 25 are coprime.

(88) 2 and 3 are prime numbers.

Instead of (87), one could also say "12 is coprime to 25." So the adjective "coprime" can be used in two grammatically distinct ways, but in both cases refers to the same mathematical binary relation: either it is (predicatively or attributively) attached to a plural NP that gets a collective reading, or it has as a complement a prepositional phrase with "to". When used in the first way, we call this usage of "coprime" a *collective usage of a transitive adjective*, when used in the second way, a *transitive usage of a transitive adjective*. We say that the two logical arguments of "coprime" can be *grouped* into one collective linguistic argument, a plural NP with a collective reading. In general, mathematical adjectives expressing a symmetric binary relation have these two uses (cf. "parallel", "equivalent", "distinct", "disjoint"; in the case of "distinct" and "disjoint", the preposition used for the transitive case is "from" rather than "to"). Other cases of grouped arguments are "x and y commute" (cf. "x commutes with y") and "x connects y and z" (cf. "x connects y to z"). "x is between y and z" is an example of an expression with a grouped argument for which there is no corresponding expression without grouped arguments.

Since "prime number" expresses a unary relation, it is not possible to group two of its logical arguments into a single linguistic argument; this explains why (88) cannot have a collective reading of the sort that (87) has. Which expressions can have grouped arguments is coded into the lexicon of the Naproche CNL.

An ambiguity like that of (86) can only arise when an expression (here the verb "to lift") has a linguistic argument that can be either a collectively interpreted plural NP or a singular NP (and can hence also be a distributively interpreted plural NP). Such expressions are extremely rare in the language of mathematics. One example that we are aware of is the adjective "inconsistent":

(89) φ and ψ are inconsistent.

(89) can be mean either that the set of formulae $\{\varphi, \psi\}$ is an inconsistent set of formulae, or that φ is inconsistent and ψ is inconsistent. This ambiguity is avoided in Naproche by not marking "inconsistent" as an expression with grouped arguments in our lexicon, so that (89) only has the distributive reading; the collective reading can only be expressed with explicit set notation in Naproche.

[42] We ignore cumulative readings here, because they play a negligible role in the mathematical contexts we have in mind.

7.6.1 Scope ambiguity

Another kind of ambiguity of special interest for our treatment of plurals and noun phrase conjunctions is a scope ambiguity that arises in certain sentences containing a noun phrase conjunction and a quantifier:

(90) A and B contain some prime number.

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

Sometimes certain considerations of reference or variable range force one of the two readings, as in (91) and (92).

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

(92) x and y are prime numbers p such that some odd prime number q divides $p + 1$.[43]

(91) only has a narrow-conjunction-scope reading, because the existentially introduced entity is linked via a predicate ("divides") to a term ("$x + y$") that refers to the coordinated noun phrases individually. (92) on the other hand only has a wide-conjunction-scope reading, because the variable p must range over the values of both x and y, and q depends on p.

Recall the semantic disambiguation principle mentioned at the beginning of section 7.5, namely that a quantifier that is introduced earlier in a sentence is always given wider scope than a quantifier introduced later in the sentence. With the addition of complex noun phrases, we extended this principle to their scopes, with the exception of the cases like (91) where another reading is forced by certain syntactical considerations. Section 7.6.4 contains an account of how cases like (91) are identified.

7.6.2 Pairwise interpretations of collective plurals

In mathematical texts, one often sees sentences like (93) and (94), which are interpreted in a pairwise way as in (95) and (96):[44]

(93) 7, 12 and 25 are coprime.

(94) All lines in A are parallel.

(95) $\mathit{coprime}(7, 12) \wedge \mathit{coprime}(12, 25) \wedge \mathit{coprime}(7, 25)$

[43] Given that this example is made up, one might ask whether it really occurs in mathematical texts that a plural noun followed by a variable is predicatively linked to a conjunction of terms as in this example. One real example that we found comes from page 4 of G. L. Cohen (2003): "Notice that 13, 37, 61, ..., are primes p such that $p^3 + 2$ and $p^3 + 1$ are squarefree."

[44] In this section, we use ordinary *PL* formulae to spell out interpretations of example sentences.

(96) $\forall x, y \in A\ (x \neq y \rightarrow \mathit{parallel}(x, y))$[45]

Sometimes, especially in connection with the negative collective adjectives "distinct" and "disjoint", this interpretation is reinforced through the use of the word "pairwise", in order to ensure that one applies the predicate to all pairs of objects collectively referred to by the plural NP. But given that this pairwise interpretation is at any rate the standard interpretation of such sentences even in the absence of the adverb "pairwise", we decided not to require the use of the word "pairwise" in the Naproche CNL.

7.6.3 Non-plural complex noun phrases

Consider the five following example sentences:

(97) 3 and 4 are coprime.

(98) 3 and the smallest even positive square number are coprime.

(99) ?3 and some even integer are coprime.

(100) *Every odd integer and some even integer are coprime.

(101) *3 or 4 is coprime.

While (97) and (98) are normal expressions, (99) is a somewhat unusual wording, but still intelligible, whereas (100) and (101) do not make any sense. The reason is that the subject of "coprime" must refer to a collection of two or more objects to which we can apply the pairwise reading discussed in section 7.6.2 above. A conjunction of terms can always be interpreted as referring to the collection of objects referred to by the conjuncts. But noun phrases with a universal or negative determiner do not refer to to a fixed object, so that conjunctions involving them cannot be interpreted as referring to a fixed collection of objects. (In the case of a noun phrase with an indefinite determiner as in (99), it is possible to consider it to both dynamically introduce an object and to refer to that introduced object. In this way it is possible to make sense of (99).) Disjunctions can never be interpreted as referring to a collection of objects.

This motivates the following distinction: A complex noun phrase is called a *plural complex noun phrase* iff it is a conjunction of noun phrases which are terms or which are determiner noun phrases with a definite or indefinite determiner. The semantics of plural complex noun phrases will be defined through the plural interpretation algorithm described in section 7.6.4 below. In this section we describe the semantics of non-plural complex noun phrases.

A non-plural noun phrase conjunction is always translated by a conjunction of formulae, where each conjunct in the translation is the translation of the sentence resulting from replacing the noun phrase conjunction by one of its conjuncts. Consider for example sentence (102), whose translation is (103):

(102) Every odd integer and some even integer are prime.

(103) $(\exists v'\ \mathit{odd}^0(v') \wedge \mathit{integer}^0(v') \rightarrow \mathit{prime}(v')) \wedge \exists v''\ (\mathit{even}^0(v'') \wedge \mathit{integer}^0(v'') \wedge \mathit{prime}(v''))$

[45] The distinctness condition here can be ignored in the case of reflexive relations like "parallel", but is certainly needed for non-reflexive relations like "coprime" or "disjoint".

The first conjunct of (103) can be considered to be a translation of "Every odd integer are prime", ignoring the ungrammaticality of this sentence that is due to the lack of agreement between the subject and the verb with respect to grammatical number (note that the definition of our semantics does not depend on this agreement being fulfilled). Similarly the second conjunct of (103) can be considered to be a translation of "Some even integer are prime".

The same principle can be adapted to noun phrase disjunctions, as in the following example:

(104) n divides every odd integer or some even integer.

(105) $(\exists v'\ \mathit{odd}^0(v') \wedge \mathit{integer}^0(v') \rightarrow \mathit{divide}(n, v')) \vee \exists v''\ \mathit{even}^0(v'') \wedge \mathit{integer}^0(v'') \wedge \mathit{divide}(n, v'')$

If more than one non-plural complex noun phrase appears in a sentence, we begin the replacement of complex noun phrases by one of their conjuncts or disjuncts at the first non-plural complex noun phrase in the sentence. The sentence resulting from this first replacement has one non-plural complex noun phrase less. So by recursion this defines a procedure to translate sentences involving any number of non-plural complex noun phrases. The choice to start at the first non-plural complex noun phrase was made in order to ensure the adherence to the semantic disambiguation principle mentioned above at the end of section 7.6.1 that a complex noun phrase that is introduced earlier in a sentence is always given wider scope than a complex noun phrase introduced later in the sentence.

7.6.4 The plural interpretation algorithm

We have implemented a plural interpretation algorithm in Naproche which can cope with plurals, plural ambiguity resolution and pairwise interpretations as explained in the introduction to section 7.6 and in sections 7.6.1 and 7.6.2. We illustrate how the algorithm treats plurals by considering the following example sentence (appearing in a context where x and y are accessible):

(106) x and y are distinct primes p such that $p+1$ is a square number and some odd prime divides $x + y$.[46]

This example has only one natural reading, and illustrates all the natural disambiguation methods mentioned in the previous sections: The plural construction "x and y" is modified by one predicate ("distinct") that needs to be interpreted collectively and by one predicate ("prime") that needs to be interpreted distributively. One of the existential NPs in the such-that clause ("a square number") has to be given a narrow scope, while the other ("some odd prime") has to be given a wide scope. The algorithm specifies a formal procedure to attain this natural reading.

[46] For the sake of simplicity, we consider "square number" a single two-word noun, and translate it to *PTL* as *square*.

The algorithm works by first producing a preliminary translation into an extension of *PTL*. The preliminary translation of (106) is as follows:[47]

$$\begin{aligned} \mathit{plural}(u_{x,y}, &\exists p\ (\mathit{distinct}(p) \wedge \mathit{prime}(p) \wedge \exists v\ (\mathit{square}(v) \wedge v = +(p,1)) \wedge \\ &\exists w\ (\mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y))) \wedge u_{x,y} = p)) \end{aligned}$$

The extension of *PTL* used here has two features that do not exist in *PTL*:

- It allows for *plural variables*, which are written like normal *PTL* variables but which have a list of *PTL* terms as subscript. Plural variables are used when translating plural complex noun phrases. The terms in the subscript of a plural variable correspond to the single conjuncts of such a conjunction.
- It allows for formulae of the form $\mathit{plural}(x, \varphi)$, where x is a plural variable and φ is a formula. The φ is the translation of everything in the logical scope of the plural complex noun phrase translated by x (e.g. the complete NP-VP-sentence of which this complex noun phrase is a subject).

The above explanations already suggest how the preliminary translation can be obtained from a Naproche CNL sentence. Since the construction of such preliminary translations works along the same lines as the Naproche-CNL-to-*PTL* translation defined above, we will not say more on this, but instead concentrate on the algorithm needed for transforming this preliminary translation into the final *PTL* translation of the sentence.

Note that in this preliminary translation, the relation symbols that translate transitive adjectives can be used in a unary way, i.e. with a single argument. They are translated in this way when used without being followed by a preposition phrase indicating the second argument of the transitive adjective. As explained in section 7.3, such uses of transitive nouns are only allowed in plural NPs and VPs. The plural interpretation algorithm ensures that the unary uses of these relation symbols get replaced by binary uses in the final *PTL* translation. Furthermore, note that the transitive adjective "distinct", which has a special semantics ($\lambda x.\lambda y.\neg y = x$), is translated as *distinct*. The special semantics of "distinct" is taken care of as soon as the unary usage of *distinct* is replaced by a binary usage.

The goal of the algorithm is to eliminate the plural variables in favour of the terms they subordinate. This has to be done separately for the distributively and collectively interpreted parts. Certain variables introduced in the scope of the plural may depend on collective uses of the plural variable (compare example (92) in section 7.6.1 above, in which q depended on p).

The algorithm consists of one preliminary normalization step followed by the following five steps: For each plural variable:

1. Mark the collective uses of the plural variable.
2. Mark the distributive uses of the plural variable and dependent variables.
3. Separate the scope of distributive uses of the plural variable from the rest.

[47] For the sake of readability, we leave out superscripts in the *PTL* variables and use u, v, w instead of v', v'', v'''.

4. Replace collective variable occurrences.

5. Replace distributive variable occurrences.

Now we describe each of the steps in more detail:

0. Normalization:

First we normalize the $plural(x_{y_1,\ldots,y_n}, \varphi)$-construct. This normalization can be divided into two sub-steps: In the first, the formula φ in this construct is replaced by a logically equivalent *PTL* formula φ', in which all existential quantifiers that are active quantifiers in φ appear at the onset of the formula. The following diagram shows how this affects the preliminary translation in our example:

$$plural(u_{x,y}, \exists p\ (distinct(p) \wedge prime(p) \wedge \exists v\ (square(v) \wedge v = +(p,1)) \wedge \exists w\ (odd(w) \wedge prime(w) \wedge divide(w, +(x,y))) \wedge u_{x,y} = p))$$

$$\wr$$

$$plural(u_{x,y}, \exists p\ \exists v\ \exists w\ (distinct(p) \wedge prime(p) \wedge square(v) \wedge v = +(p,1) \wedge odd(w) \wedge prime(w) \wedge divide(w, +(x,y)) \wedge u_{x,y} = p))$$

In what follows, we will often need to refer to the conjuncts of the conjunction that follows the quantifiers in φ'; for this we will just use the term "conjunct" without any further specification.

If the existential quantifiers at the onset of the thus produced φ' introduce a variable z which is equated by one of the conjuncts with the plural variable $x_{y_1,\ldots,y_n}$ of the $plural(x_{y_1,\ldots,y_n}, \varphi)$-construct, we delete $\exists z$ and the equating conjunct from φ' and replace both $x_{y_1,\ldots,y_n}$ and z by $z_{y_1,\ldots,y_n}$ throughout. In our example, $\exists p$ and $u_{x,y} = p$ are deleted and $u_{x,y}$ and p are replaced by $p_{x,y}$ throughout, as shown below:

$$plural(u_{x,y}, \exists p\ \exists v\ \exists w\ (distinct(p) \wedge prime(p) \wedge square(v) \wedge v = +(p,1) \wedge odd(w) \wedge prime(w) \wedge divide(w, +(x,y)) \wedge u_{x,y} = p))$$

$$\wr$$

$$plural(p_{x,y}, \exists v\ \exists w\ (distinct(p_{x,y}) \wedge prime(p_{x,y}) \wedge square(v) \wedge v = +(p_{x,y},1) \wedge odd(w) \wedge prime(w) \wedge divide(w, +(x,y))))$$

1. Marking the collective uses of the plural variable:

We mark every conjunct which consists of a predicate that has the plural variable as grouped argument ("distinct(p)" in the example formula, marked by boldface below). That the plural variable is a grouped argument is derived from the fact that the number of arguments, with which the predicate appears in the conjunct, is one less than its logical number of arguments fixed in the lexicon, and from

the fact that the lexicon specifies the possibility of grouping two of its arguments into one.

$$\begin{aligned} &\mathit{plural}(p_{x,y},\exists v\ \exists w\ (\mathit{distinct}(p_{x,y}) \wedge \mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y},1) \wedge \\ &\qquad \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)))) \end{aligned}$$

$$\wr$$

$$\begin{aligned} &\mathit{plural}(p_{x,y},\exists v\ \exists w\ (\mathbf{distinct}(\boldsymbol{p_{x,y}}) \wedge \mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y},1) \wedge \\ &\qquad \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)))) \end{aligned}$$

2. Marking the distributive uses of the plural variable and dependent variables:

We recursively mark (in the example by underlining) all conjuncts that were not marked in step 1 and contain the plural variable or a marked variable, and all variables contained in a conjunct marked in this way, until no more conjuncts and variables can be marked by this process:

$$\begin{aligned} &\mathit{plural}(p_{x,y},\exists v\ \exists w\ (\mathbf{distinct}(\boldsymbol{p_{x,y}}) \wedge \mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y},1) \wedge \\ &\qquad \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)))) \end{aligned}$$

$$\wr$$

$$\begin{aligned} &\mathit{plural}(p_{x,y},\exists \underline{v}\ \exists w\ (\mathbf{distinct}(\boldsymbol{p_{x,y}}) \wedge \underline{\mathit{prime}(p_{x,y})} \wedge \underline{\mathit{square}(v)} \wedge \underline{v = +(p_{x,y},1)} \wedge \\ &\qquad \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)))) \end{aligned}$$

3. Separating the scope of distributive uses of the plural variable from the rest:

All variables (together with their quantifiers) and conjuncts not marked in step 2 get pulled out of the $\mathit{plural}(x_{y_1,\ldots,y_n},\varphi)$-construct and inserted to the left of this construct:[48]

$$\begin{aligned} &\mathit{plural}(p_{x,y},\exists \underline{v}\ \exists w\ (\mathbf{distinct}(\boldsymbol{p_{x,y}}) \wedge \underline{\mathit{prime}(p_{x,y})} \wedge \underline{\mathit{square}(v)} \wedge \underline{v = +(p_{x,y},1)} \wedge \\ &\qquad \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)))) \end{aligned}$$

$$\wr$$

$$\begin{aligned} &\exists w\ \mathit{distinct}(p_{x,y}) \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w,+(x,y)) \wedge \\ &\mathit{plural}(p_{x,y},\exists v\ (\mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y},1))) \end{aligned}$$

[48] Since this step moves quantifiers and conjuncts around, one might wonder whether it can cause formerly bound variables to become free. This, however, is impeded by the recursive procedure in step 2: If a certain variable stays in the $\mathit{plural}(x_{y_1,\ldots,y_n},\varphi)$-construct, no condition containing this variable can be pulled out of this construct.

4: Replacing collective variable occurrences:

For every formula $R(x_{y_1,\ldots,y_n})$ with grouped argument $x_{y_1,\ldots,y_n}$, and every pair $(y_i, y_j) \in \{y_1, \ldots, y_n\}$ with $i \neq j$, we create a formula of the form $R(y_i, y_j)$ and remove the original formula $R(x_{y_1,\ldots,y_n})$ by a conjunction of all formulae thus created. This is also the place where binary usages of *distinct* are replaced by the special semantics $\lambda x.\lambda y.\neg y = x$ of "distinct" (in our example this amounts to replacing $\mathit{distinct}(p_{x,y})$ by $\neg x = y$):

$$\begin{aligned}&\exists w\ \mathit{distinct}(p_{x,y}) \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\mathit{plural}(p_{x,y}, \exists v\ (\mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y}, 1)))\end{aligned}$$

$$\wr$$

$$\begin{aligned}&\exists w \neg x = y \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\mathit{plural}(p_{x,y}, \exists v\ (\mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y}, 1)))\end{aligned}$$

5. Replacing distributive variable occurrences:

Consider the remaining $\mathit{plural}(x_{y_1,\ldots,y_n}, \varphi)$-construct. For every term y_i subordinated to the plural variable $x_{y_1,\ldots,y_n}$, we make a copy of $\Diamond\varphi$ in which every instance of $x_{y_1,\ldots,y_n}$ is replaced by y_i. The conjunction of these copies of $\Diamond\varphi$ replaces the $\mathit{plural}(x_{y_1,\ldots,y_n}, \varphi)$-construct:

$$\begin{aligned}&\exists w \neg x = y \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\mathit{plural}(p_{x,y}, \exists v\ (\mathit{prime}(p_{x,y}) \wedge \mathit{square}(v) \wedge v = +(p_{x,y}, 1)))\end{aligned}$$

$$\wr$$

$$\begin{aligned}&\exists w \neg x = y \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\Diamond \exists v\ (\mathit{prime}(x) \wedge \mathit{square}(v) \wedge v = +(x, 1)) \wedge \\ &\Diamond \exists v\ (\mathit{prime}(y) \wedge \mathit{square}(v) \wedge v = +(y, 1))\end{aligned}$$

There is a technical problem related to this step: In our example, the variable v gets existentially introduced twice, once in each copy of $\Diamond\varphi$ produced at this step. This violates the niceness properties of *PTL* texts defined in chapter 5. In order to adhere to the niceness properties, we additionally rename every variable z existentially introduced in the y_i-copy of $\Diamond\varphi$ by z_{y_i}:

$$\begin{aligned}&\exists w \neg x = y \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\Diamond \exists v\ (\mathit{prime}(x) \wedge \mathit{square}(v) \wedge v = +(x, 1)) \wedge \\ &\Diamond \exists v\ (\mathit{prime}(y) \wedge \mathit{square}(v) \wedge v = +(y, 1))\end{aligned}$$

$$\wr$$

$$\begin{aligned}&\exists w \neg x = y \wedge \mathit{odd}(w) \wedge \mathit{prime}(w) \wedge \mathit{divide}(w, +(x,y)) \wedge \\ &\Diamond \exists v_x\ (\mathit{prime}(x) \wedge \mathit{square}(v_x) \wedge v_x = +(x, 1)) \wedge \\ &\Diamond \exists v_y\ (\mathit{prime}(y) \wedge \mathit{square}(v_y) \wedge v_y = +(y, 1))\end{aligned}$$

This final *PTL* formula corresponds to the natural reading of sentence (106) that we described at the beginning of this section.

7.7 Coverage of the Naproche CNL

In this section we will give some clarifications about the coverage and the limitations of the Naproche CNL and discuss some possible extensions to this CNL. Since the coverage of the symbolic part of the Naproche CNL was already discussed in section 21, we will focus here mainly on the textual parts of the language, and to a lesser extent on the macro-grammatical coverage.

As explained in section 1.3.2 of the introduction, mathematical proofs can in general be modelled by proofs in some formal system, such as first-order logic together with the set-theoretic axioms of ZFC. Since the Naproche CNL has a much higher expressivity than such formal systems, mathematical proofs can similarly in general be modelled in the Naproche CNL. But as described in section 1.3.2, the modelling of mathematical proofs in ZFC is not very faithful to the original: The translation of mathematical statements into the basic language of ZFC with $\in$ as the only non-logical symbol causes a massive blow-up in the length of the text. Furthermore, there is also a massive blow-up in the number of proof steps needed in order to model a normal mathematical proof in one of the standard fine-grained proof calculi for first-order logic.

The goal of the Naproche CNL is, of course, to be more faithful to the original and to avoid this blow-up as much as possible. To which degree the number of proof-steps needs to be increased depends on the strength of the automated theorem provers used in the proof-checking module of Naproche; so this does not directly depend on the expressive strength of the CNL that we want to focus on in this section. Hence we will focus on the question how faithful a Naproche CNL adaptation of a mathematical text can be if we ignore the problem of it possibly containing proof steps that the proof-checking module cannot follow.

As should already be evident from the description of the Naproche CNL syntax in the previous sections and from the example sentences presented in order to illustrate this description, the Naproche CNL contains a large number of natural language constructs commonly used in mathematical texts. Given a mathematical text that needs to be adapted to the Naproche CNL, these constructs make it possible to leave many parts of such a text unchanged, and make it possible to adapt the parts that have to be changed in a way that still sounds relatively natural and does not use much more lengthy expressions than the original.

So far, the coverage of the Naproche CNL has not been evaluated quantitatively: Such an evaluation would require choosing diverse mathematical texts and trying to translate them into the Naproche CNL, staying as close as possible to the original, and, when this is not possible, choosing the most concise Naproche CNL expressions logically equivalent to the original. This is, of course, a highly non-trivial task. Given that the research conducted for this thesis did not just aim at a rich mathematical CNL, but also at successful proof checking of mathematical texts adapted to the CNL, we did not make the effort to adapt mathematical texts which would at any rate not have been proof-checkable. The texts that we did adapt to the Naproche CNL are the first chapter of Edmund Landau's *Grundlagen der Analysis*, which can be found in appendix B, and which is discussed in detail in chapter 8, as well as the first five theorems of Euclid's *Elements* together with an axiomatization of geometry based on the system E by Avigad et al. (2009), consisting of more than eighty axioms. The successful adaptation of these texts to the Naproche CNL must of course be

considered in the light of the fact that the language was extended in order to make certain constructs appearing in these texts be more faithfully represented in the Naproche CNL.

Given that the Landau's *Grundlagen der Analysis* uses the textual parts of the language of mathematics very schematically, the same holds for its adaptation to the Naproche CNL, so that this text does not give a good picture of what is possible in the textual part of the Naproche CNL. In order to illustrate better the expressiveness of the textual part of the Naproche CNL, we have written the following geometric text in the Naproche CNL. All sentences in this text are interpreted by the Naproche system in the way that a mathematical reader would naturally interpret them.

> Axiom 1: For all points p, q, there is precisely one line L such that p and q are on L.
>
> Axiom 2: If L_1 and L_2 are distinct lines, then there is at most one point p such that p is on L_1 and L_2.
>
> Suppose that there is a set $\mathbb{R}$ of objects called numbers and a binary function $+$ and a binary relation $\leq$ and a number 0 in $\mathbb{R}$ satisfying the following axioms:
>
> Axiom 3: For all points p, q there is a number $d(p, q)$.
>
> Axiom 4: For all points p, q and o, $d(p, o) \leq d(p, q) + d(q, o)$.
>
> Axiom 5: Let p be a point and r be a number. Then there is a point q such that $d(p, q) = r$.
>
> Axiom 6: For every number r, there is a number $-r$ such that $r + (-r) = 0$.
>
> Definition 7: Define line L and line M to be parallel iff $L = M$ or no point is on L and M.
>
> Lemma 9: If L and M are parallel lines, then there is a number $D(L, M)$ such that the following properties hold:
> Property A: There is a point p on L and a point q on M such that $D(L, M) = d(p, q)$.
> Property B: For all points p, q such that p is on L and q is on M, $D(L, M) \leq d(p, q)$.
> Proof: Trivial. Qed.
>
> Theorem 10: Suppose that L is a line. Then there are functions f_L, g_L satisfying the following properties:
> Property A: For every line M such that M is parallel to L, $f_L(M)$ is a number such that $f_L(M) = D(L, M)$ or $f_L(M) = -D(L, M)$.
> Property B: For every number r, $g_L(r)$ is a line and $g_L(r)$ is parallel to L.
> Property C: For every line M, $g_L(f_L(M)) = M$.
> Property D: For every number r, $f_L(g_L(r)) = r$.
> Proof:

> Clearly there is a point p not on L and a line N such that p is on N and L and N are not parallel. Then there is precisely one point q such that q is on L and N.
>
> Let M be a line such that M is parallel to L. Then there is precisely one point p_M such that p_M is on M and N. Now if $L \neq M$, then precisely one of the following cases holds:
> Case 1: $d(q, p_M) + d(p_M, p) = d(q, p)$.
> Case 2: $d(p_M, q) + d(q, p) = d(p_M, p)$.
>
> So there is precisely one number $f_L(M)$ such that case 1 holds and $f_L(M) = D(L, M)$ or case 2 holds and $f_L(M) = -D(L, M)$. Thus property A holds. Furthermore, for every number r there is precisely one line $g_L(r)$ such that $f_L(g_L(r)) = r$ and $g_L(r)$ is parallel to L. Now this in turn implies that property B, property C and property D hold. Qed.

There are, of course, many common constructs in the language of mathematics that have not yet been included in the Naproche CNL. Some of them would require new techniques for parsing, disambiguating and semantically interpreting them to be included, whereas others could easily be added to the system as it is, and have only not been added so far for the lack of need to do so. In order to give the reader a feeling of what kind of limitations the Naproche CNL has, we now present a list of such missing linguistic constructs that we are aware of. We start with constructs that could easily be added to the current system:

- Transitive nouns, i.e. nouns that express binary relations and similarly to transitive adjectives use a postposed prepositional phrase with a fixed preposition, in most cases "of" (e.g. "divisor of n" or "subgroup of G")
- More flexibility with expressions similar to "at most one" and "precisely one":
  - The expression "at least one" besides "at most one" and "precisely one" for forming existentially quantified sentences
  - These three expressions as determiners for determiner noun phrases and not only for existentially quantified sentences with "there to be" or "there to exist"
  - "at least", "at most" and "precisely" followed by other number words than "one"
  - The adjective "unique" with the same semantics as the determiner "precisely one"
- The $\{\ldots \mid \ldots\}$ notation for sets (e.g. $\{x \mid x \subseteq A\}$) additionally to its already supported textual counterpart "the set of ... such that ..."
- Transitive adjectives followed by a propositional phrase (e.g. "parallel to L") as postmodifiers and not only as predicatives in copula verb phrases
- Existential statements with "for some ..." analogously to the existing "for all"

- Assumptions of the form "Fix ...", "Consider ..." and "Let ... be given" with indefinite determiner noun phrases (e.g. "an integer k") instead of just quantterm lists in the position of the dots
- More flexibility in the wording of definitions:
  - Expressions like "we say" and "we define" as alternatives to the currently supported "define" at the beginning of definitions; definitions preceded by a definition heading may even lack such an expression (so "n is even iff there is some k such that $n = 2k$" could be a definition if it follows a definition heading).
  - Definitions with ":=" (e.g. $f(x) := x^2$) additionally to their already supported textual counterpart "Define ... to be ..."
  - Bi-implicational definitions with "if" instead of "iff" (compare footnote 4 in section 1.2)
  - Definitions of transitive adjective with plural determiner noun phrases containing a two-variable variable list (e.g. "Define lines L and M to be parallel iff ...") instead of a conjunction of two determiner noun phrases (e.g. "Define line L and line M to be parallel iff ...")
  - Definitions that fix the meaning of a preposition (e.g. "Define a to be above b iff $\pi_2(a) > \pi_2(b)$."
- Case distinctions inside symbolic material, e.g. $|x| = \begin{cases} x & \text{if } x \geq 0 \\ -x & \text{otherwise.} \end{cases}$
- Inflected forms of "to be defined" (e.g. "$\frac{1}{k}$ is defined")
- Symbolic terms prefixed to nouns, adjectives, verbs or suffixes (e.g. $\mathbb{F}$-module, $(\mathbb{Z} \times \mathbb{Z})$-module, p-adic, n-ary)
- Verb phrases coordinated with "and" and "or"
- Ending proofs by contradictions with expressions like "which contradicts Theorem 7" postfixed to a sentence, instead of with a separate sentence of the form "contradiction by Theorem 7"
- Variable type specifications that link a predicate not to all variable symbols of a predefined class (like small Greek letters), but to a list of variable symbols explicitly stated (e.g. "α, β and γ will stand throughout for ordinals.")
- A larger lexicon of nouns, verbs and adjectives usable in the Naproche CNL than that currently supported[49]

Now we present missing linguistic constructs that would require additional techniques for their interpretation or disambiguation. How involved the elaboration of these required additional techniques would be varies greatly between the different constructs: For some of them one could take over existing interpretation and disambiguation techniques from the linguistic literature or from

[49] See the lexicon at the end of the formal textual grammar in section A.2 of appendix A for a list of the currently supported nouns, verbs and adjectives.

existing applications of computational linguistics, whereas others would require novel techniques adapted to the language of mathematics to be developed. For each linguistic construct presented in this list, we briefly touch on the kind of techniques required and the expected difficulty of elaborating these techniques.

- Inflected forms of "to have" as well as "with" or "without" followed by a noun phrase whose head noun is a transitive noun which has "of" as its fixed preposition (e.g. "n has an odd prime divisor", "$\emptyset$ has no proper subset", "a number with an odd prime divisor", "a number without odd prime divisor")

  (It is easy to develop an interpretation algorithm for a construct of a fixed form, e.g. for "to have" followed by an indefinite noun phrase with such a transitive noun. But it is not a priori clear in which way the different possible forms (noun phrases with various determiners preceded by either "to have", "with" or "without") can be given a unified semantic treatment.)

- The expressions "at least", "at most" and "precisely" followed by a natural number variable instead of a number word

  (If the system is based on a foundational theory that knows of natural numbers and of the relation $<$ on them, then this is easy to implement. *CMTN* knows of natural numbers, but $<$ is not axiomatized in it, though it could easily be expanded to include an axiomatization of $<$. But, as described in appendix C, *CMTN* is not implemented in the current system, nor does the foundation theory implemented know anything about natural numbers.)

- A noun phrase postmodifier of the form "defined on ..." (e.g. "a function defined on $\mathbb{R}$" or "a binary relation defined on positive integers")

  (The only difficulty here is the flexibility required: "defined on" may be followed by singular noun phrase explicitly referring to a set or by a plural noun phrase that can be considered to refer to a collection; the interpretation of "defined on ..." depends on the arity of the function or relation referred to by the noun phrase preceding "define on"; the ideal case would be that the interpretation of "defined on ..." is even successful if the arity of this function or relation is only specified later, though this is likely to be hard to implement.)

- Certain restricted uses of the ellipsis ("...") used in symbolic parts of the language of mathematics, for example "$1 + \cdots + n$" or "for all $v_1, \ldots, v_n$"

  (A CNL cannot include all possible uses of the ellipsis found in mathematical texts, since its usage is too diverse and its interpretation too much based on heuristics and mathematical intuition to define a deterministic interpretation algorithm for all possible uses of the ellipsis. But single uses of the ellipsis, like those cited above, can be given a determinate formal semantics.[50] One should study the possibility of giving a unified semantic account of as many such uses of the ellipsis as is possible under

[50] The formal semantics of "for all $v_1, \ldots, v_n$" must universally quantify over a function defined on $\{1, \ldots, n\}$; in other words, it should be given the same semantics as the expression "for every function $i \mapsto v_i$ defined on $\{1, \ldots, n\}$".

the restriction that the interpretation algorithm for these uses should be deterministic and not dependent on heuristics.)

- The word "respectively" in expressions like "$x + z < y + z$, $x + z = y + z$ and $x + z > y + z$ imply $x < y$, $x = y$ and $x > y$ respectively"

  (The interpretation of plurals discussed in section 7.6 needs to allow for special cases when the word "respectively" is used in such ways. The single special cases are certainly easy to elaborate, but it is likely to be more difficult to ensure that all ways in which "respectively" is used are included in the CNL.)

- It should be possible to postfix quantterms with a binary infix relation symbol followed by a second argument for the relation (e.g. "there is some $x \in A$")

  (The problem here is that the quantterm as well as the second argument to the relation may be complex, which makes it hard to recognize where the relation symbol is. Currently the quantterm and term grammar are completely separated and even use different parsing techniques, so unifying them is technically difficult. Additionally, new disambiguation principles would be needed for this kind of symbolic expressions: Note that quantterms may even reintroduce a symbol previously introduced, now with a new meaning; one needs to ensure that the infix relation symbol is not the head of the quantterm and is not being reintroduced with a new meaning.)

- Terms consisting of an accessible function applied to quantterms, used in a quantterm position (e.g. "There is a fraction $\frac{x}{y}$")

  (Again, the problematic case are complex quantterms and the interaction between the quantterm grammar and a construct currently implemented only in the term grammar. In a previous version of Naproche, when only simple variables could be used in positions where we now have quantterms, this kind of expression was already possible, but it had to be removed in order to make the flexible quantterm grammar possible. If we want to reintroduce it, we need to develop disambiguation techniques to distinguish such terms from usual quantterms.)

- Allowing universal and existential quantifications with "for all" or "for some" to be postposed instead of preposed to their scope (e.g. "$x + y' = (x + y)'$ for all x, y")

  (Here a special disambiguation principle for determining where the scope of such a quantification starts in a complex sentence might be required.)

- Relative clauses with "which" and "that" (e.g. "a number that divides 12" or "a number which 2 and 3 divide")

  (If the relative pronoun is to be allowed to have other syntactic functions in the relative clause than that of its subject, then one needs to take care of the gap left in the relative clause. For example, in "a number which 2 and 3 divide", no object follows the transitive verb "divide", since the relative pronoun "which" already has the syntactic function of an object.)

- The determiner "any"

  (For this, one needs to include a syntactic distinction that specifies in which position "any" is interpreted like an indefinite determiner and in which positions like a universal quantifier.)

- Natural language anaphora:

  - Anaphoric pronouns ("it" and "they")

    (These would require principles for anaphora resolution, i.e. for determining the antecedent of such anaphoric pronouns, to be included in the Naproche CNL. These principles should be simple enough so that authors of Naproche CNL texts can ensure that the anaphoric pronouns they use get resolved in the intended way. These principles could be taken over from existing general purpose CNLs like Attempto Controlled English. However, it might be desirable to consider specifics of the language of mathematics with respect to anaphora resolution.)

  - The determiner "such a" for referring to a recently mentioned property (e.g. "such a function" for a function satisfying certain properties of functions that were recently mentioned)

    (Just as for "it" and "they", anaphora resolution principles would be required. Unlike for "it" and "they", such principles probably have not yet been developed for "such a" in any existing CNL.)

  - The expression "otherwise" for introducing an assumption that negates a previously made statement

    (Just as for "it", "they" and "such a", the CNL would have to include anaphora resolution principles, in this case for determining which previously made assertion is being negated. In the case of "otherwise" these anaphora resolution principles certainly need to take into account specifics of the language of mathematics.)

  - Anaphoric definite noun phrases, i.e. anaphoric usage of "the" (e.g. "the group" to refer to a group previously mentioned; compare section 3.2.1)

    (New disambiguation principle would be required to distinguish anaphoric definite noun phrases from the already supported definite descriptions with "the".)

- Constructs like "We still need to show that ..."

  (Such constructs only make sense if the proof-checking module can follow a goal-oriented approach, similar in nature to the backward reasoning briefly described in the subsection about HOL in section 1.3.3 of the introduction: The assertion of a theorem or lemma can be considered a goal of the proof that follows it, and certain proof steps can be considered to simplify this goal. Once such a goal-oriented approach is implemented in the proof-checking module, it is both natural and unproblematic to include linguistic constructs like "We still need to show that ...".)

- It should be possible for definitions of nouns, verbs and adjectives to dynamically extend the lexicon, so that one can introduce such words in definitions even if they do not appear in the lexicon of the Naproche CNL.

(Given that in the current Naproche system relation and function symbols are not listed in a lexicon but have to be introduced in some way in the text, e.g. by definitions, the machinery for such dynamic extensions of the lexicon already exist. But for nouns and verbs we would need additional rules for determining their various inflected forms once some base form has been introduced in a definition. The preposition to be used with a transitive adjective dynamically added to the lexicon would either have to be stated in the definition or be recognized at its first usage.)

Note that some of the proposed extensions could, in combination with some existing disambiguation principles, lead to a different interpretation of sentences already included in the current Naproche CNL. For example, the addition of anaphoric definite noun phrases would cause some noun phrases starting with "the" to be interpreted as anaphoric definite noun phrases, which would currently be interpreted as definite descriptions.

A different kind of limitation of the Naproche CNL is caused by some strict syntactic postulations that were included in order to make the CNL unambiguous. For example, every assumption that is to be retracted without closing the structural block in which it was introduced needs to be retracted by a sentence starting with "thus", and this is the only allowed way of using "thus". This, of course, does not correspond to the usual usage of "thus" in the language of mathematics. The reason for this unnatural aspect in the Naproche CNL is that the language of mathematics does not have clear signs for marking the retraction of an assumption. Readers of mathematical texts use their understanding of the proof progress together with various typographical heuristics to determine where assumptions are retracted. Nevertheless, it might be worthwhile to study the ways in which assumptions are retracted in actual mathematical texts with the goal of possibly making this aspect of the Naproche CNL more natural.

There is also a serious problem with giving a controlled natural language a wide coverage and sophisticated interpretation algorithms: One of the central ideas of a controlled natural language is that an author of a CNL text can have control over the interpretation of the text by the system that processes it. But if the CNL has a very wide coverage and a number of sophisticated interpretation algorithms, it becomes hard for an author to understand the CNL and its interpretation well enough to ensure that the text he writes has the intended meaning. The current Naproche CNL already faces this problem to some extent. For example, a Naproche CNL author has to know that indefinite noun phrases do not have a generic interpretation in the Naproche CNL, but only a dynamic existential interpretation. But in the language of mathematics, indefinite noun phrases also appear with a generic reading (which is, as mentioned in section 7.5.9, in a mathematical context semantically indistinguishable from a universal reading), e.g. "A homotopy $f_t : X \to X$" in (107):

(107) A homotopy $f_t : X \to X$ that gives a deformation retraction of X onto a subspace A has the property that $f_t|A = \mathbb{1}$ for all t. (Hatcher, 2002, p. 3)

But in implications the dynamic existential reading does lead to a reading which is equivalent to a universal quantification outside the scope of the implication (this is the donkey-sentence phenomenon discussed in section 7). Furthermore, in bi-implications and reversed implications, there is a sophisticated algorithm for determining whether an indefinite noun phrase gets a universal reading or

not (see section 7.5.9). A Naproche CNL author has to be aware of these facts in order to have control over the interpretation that the system will give to his text.

If the Naproche CNL gets extended further, it will be even more problematic than now to assume an author to be capable of being aware of all these issues. In order to minimize this problem, the Naproche system would have to give semantic feedback to the author. One possible way of giving semantic feedback to a CNL author is by providing him with *paraphrases* of the sentences that he writes. This method has been explored and partly implemented for Attempto Controlled English, as discussed in Kaljurand (2009). The basic idea is that a paraphrase is a sentence semantically equivalent to but syntactically distinct from the sentence written by the author. The paraphrases the system produces should preferably be held within a particularly simple subset of the CNL, in order to ensure that they are understood correctly by the author.

Chapter 8

A case study: Landau's *Grundlagen der Analysis*

In this chapter we illustrate the theory described in this thesis and the functioning of the Naproche system on the beginnings of the Naproche CNL adaptation of the first chapter of Landau's *Grundlagen der Analysis*.

Landau's *Grundlagen der Analysis* is a text on the foundations of number systems needed for analysis: Starting from the Peano axiomatization of the natural numbers, Landau in turn extends each number system using set theoretic methods, building up first the positive rationals, next the positive reals, next the number system of all reals, and finally the complex numbers.

Since Landau's text explicitly introduces the natural numbers through Peano's axiomatization, his text – unlike most mathematical texts – does not take the natural numbers for granted. For this reason, we will assume the background theory for this text to be *CMT* instead of *CMTN*. (Recall from section 4.3.1 of chapter 4 that *CMT* is a theory of classes, maps and tuples, so unlike *CMTN* it does not have the natural numbers among its primitives.)

In his text, Landau freely talks about sets, functions and tuples and uses properties of these fundamental mathematical objects without formally defining a theory for these. *CMT* can be used to model Landau's background assumptions about these fundamental objects.[1] Below we will show how *CMT* models the set-theoretic and function-theoretic machinery used by Landau in the fragment considered in this chapter (which does not contain talk about tuples).

Landau stated the Induction Axiom in set-theoretic terms, just as Peano had done in his original axiomatization of the natural numbers. Nowadays, Peano's axioms are most well-known in their first-order reformulation, in which there is not a single Induction Axiom, but an Induction Axiom Schema. In this first-order reformulation of Peano's axiomatization, addition and multiplication have to be assumed to be given and their basic properties have to be axiomatized. But with the set-theoretic formulation of the Induction Axiom, one can show the existence and uniqueness of the addition and multiplication functions with the desired properties. The only function assumed in the axioms is the successor

[1] Since all objects that Landau talks about can by typed, one could just as well model these background assumptions using a type theory with set types, function types and tuple types among its type constructs.

function.

The beginning of Landau's text that we consider in this chapter contains the Peano axiomatization of the natural numbers, three basic theorems about the successor function, and the theorem that proves the existence and uniqueness of the addition function with the desired properties.

In our discussion of the Naproche CNL adaptation of this text fragment, its *PTL* translation and the functioning of the proof checking on it, we will start with a very detailed analysis. When discussing later parts of the text fragment, our analysis becomes less detailed and focuses on those aspects that are not similar to what we have already discussed.

8.1 Peano's axioms

In the first section of his first chapter, Landau introduces Peano's axiomatization of the natural numbers. Here are the paragraphs of this section that we actually translated into the Naproche CNL:[2]

> Wir nehmen als gegeben an:
>
> Eine Menge, d.h. Gesamtheit, von Dingen, natürliche Zahlen genannt, mit den nachher aufzuzählenden Eigenschaften, Axiome genannt.
>
> [...]
>
> Kleine lateinische Buchstaben bedeuten in diesem Buch, wenn nichts anderes gesagt wird, durchweg natürliche Zahlen.
>
> [...]
>
> **Axiom 1:** 1 *ist eine natürliche Zahl.*
>
> [...]

[2] In Landau's text, all formulae are displayed on separate lines. When we cite from Landau's original and its translation by Steinhardt (trans., 1951), we put short formulae in-line, so that the citations do not take up too much space.

Here is the English translation of the cited paragraphs by Steinhardt (trans., 1951):

> We assume the following to be given:
>
> A set (i.e. totality) of objects called natural numbers, possessing the properties—called axioms—to be listed below.
>
> [...]
>
> Unless otherwise specified, small italic letters [see footnote 4 on page 261] will stand for natural numbers throughout this book.
>
> [...]
>
> **Axiom 1:** 1 *is a natural number.*
>
> [...]
>
> **Axiom 2:** *For each x there exists exactly one natural number, called the successor of x, which will be denoted by x'.*
>
> [...]
>
> **Axiom 3:** *We always have $x' \neq 1$.*
>
> [...]
>
> **Axiom 4:** *If $x' = y'$ then $x = y$.*
>
> [...]
>
> **Axiom 5** (Axiom of Induction): *Let there be given a set $\mathfrak{M}$ of natural numbers, with the following properties:*
>
> I) 1 *belongs $\mathfrak{M}$.*
>
> II) *If x belongs to $\mathfrak{M}$ then so does x'.*
>
> *Then $\mathfrak{M}$ contains all natural numbers.*

Axiom 2: *Zu jedem x gibt es genau eine natürliche Zahl, die der Nachfolger von x heißt und mit x' bezeichnet werden möge.*
[...]
Axiom 3: *Stets ist $x' \neq 1$.*
[...]
Axiom 4: *Aus $x' = y'$ folgt $x = y$.*
[...]
Axiom 5 (Induktionsaxiom): *Es sei $\mathfrak{M}$ eine Menge natürlicher Zahlen mit den Eigenschaften:*
I) 1 *gehört zu $\mathfrak{M}$.*
II) *Wenn x zu $\mathfrak{M}$ gehört so gehört x' zu $\mathfrak{M}$.*
Dann umfaßt $\mathfrak{M}$ alle natürlichen Zahlen.

In the omitted paragraphs, indicated by "[...]", Landau introduces the syntax and semantics of $=$ and $\neq$, mentions some bracketing conventions and presents some clarifications of the axioms.

8.1.1 Naproche CNL adaptation and *PTL* translation

Below we present the Naproche CNL adaptation of these paragraphs together with the *PTL* translation of this Naproche CNL text. The Naproche CNL adaptation has a completely natural appearance, and could even go through as a translation – though not perfectly literal – of the original text into natural English. Nevertheless, there are some noteworthy differences, which we will discuss below.

The *PTL* translation is formatted in such a way that the *PTL* subformula corresponding to a certain sentence appears directly to the right of that sentence. For the sake of readability, we present the *PTL* translation with some syntactical simplifications and adaptations:[3]

- We use the simplification mentioned at the end of section 7.5.2: For example, "x is a natural number" gets translated as $N(x)$ instead of $\exists v\ (N(v) \wedge v = x)$, and "There is a natural number" gets translated as $\exists x\ N(x)$ instead of $\exists x\ (N(x) \wedge \top)$.

- Unnecessary brackets are dropped (as already practised in the previous chapter); usual conventions of operator priorities hold as mentioned in chapter 2.

- We generally write v for v', v'', v''', etc. In one place, we use the semantically motivated $\mathbb{N}$ instead.

- Superscripts of *PTL* variables are dropped (except in the case of *PTL* variables of the form u^i for $i \in \mathbb{N}$, since dropping the superscript in this case would make the variable indistinguishable from the undefinedness constant u used in the *PL* translation of the *PTL* text).

- The successor function $'$ is written in suffix instead of classical notation, as in the original text.

[3]The clarifications of simplifications listed lower in the list already presuppose the simplifications listed above them. This list of simplifications also contains simplifications that will only be needed in sections 8.2 and 8.3 below.

- We write N instead of *natural_number*.
- We write $a_\bullet$ and $b_\bullet$ for the *PTL* variables `a_{[arg]}` and `b_{[arg]}` respectively.
- *PTL* IDs are abbreviated: For example, instead of *axiom_1*, we write *ax*1
- We write $\forall x\ \varphi$ for $\exists x\ \top \to \varphi$.
- We write $\varphi \leftrightarrow \psi$ for $(\Diamond\varphi \to \Diamond\psi) \wedge (\Diamond\psi \to \Diamond\varphi)$.

In the *PTL* translation of this axiom text, we use the variable θ to indicate the position where the translation of subsequent text will be inserted.

Below on the left is the Naproche CNL adaptation of the paragraphs under discussion, and on the right is its *PTL* translation:

Assume that there is a set of objects called natural numbers.	$\exists \mathbb{N}\ (C(\mathbb{N}) \wedge L(\mathbb{N}) \wedge \exists N\ (M_1(N) \wedge$ $\forall v\ B(N(v)) \wedge \forall v\ (v \in \mathbb{N} \leftrightarrow N(v)))) \to$
Small Latin letters will stand throughout for natural numbers.	
Axiom 1: 1 is a natural number.	$(\mathit{label}(\mathit{ax}1, \exists 1\ N(1)) \to$
Axiom 2: For every x, there is a natural number x'.	$(\mathit{label}(\mathit{ax}2, \exists x\ N(x) \to \exists x'\ N(x')) \to$
Axiom 3: For every x, $x' \neq 1$.	$(\mathit{label}(\mathit{ax}3, \exists x\ N(x) \to \neg x' = 1) \to$
Axiom 4: If $x' = y'$, then $x = y$.	$(\mathit{label}(\mathit{ax}4, \exists x\ (N(x) \wedge \exists y\ (N(y) \wedge$ $x' = y')) \to x = y) \to$
Axiom 5: Suppose $\mathfrak{M}$ is a set of natural numbers satisfying the following properties:	$(\mathit{label}(\mathit{ax}5, \exists \mathfrak{M}\ (C(\mathfrak{M}) \wedge L(\mathfrak{M}) \wedge$ $(\exists v\ v \in \mathfrak{M} \to N(v)) \wedge$
Property 1: 1 belongs to $\mathfrak{M}$.	$1 \in \mathfrak{M} \wedge$
Property 2: If x belongs to $\mathfrak{M}$, then x' belongs to $\mathfrak{M}$.	$(\exists x\ (N(x) \wedge x \in \mathfrak{M}) \to x' \in \mathfrak{M})) \to$
Then $\mathfrak{M}$ contains all natural numbers.	$(\exists v\ N(v) \to v \in \mathfrak{M})) \to \theta)))))$

We will first discuss some differences between the Naproche CNL adaptation and the original text, before we go on to explain the *PTL* translation.

First consider the sentence introducing the set of natural numbers. The first difference is that the original text contains an additional parenthetic comment ("d.h. Gesamtheit") for the sake of clarifying the word "Menge" ("set"), supposedly for readers not familiar with basic set theoretic terminology. We generally left out such clarifying parenthetic comments in the Naproche CNL adaptations. The other difference is the appearance of a cataphoric meta-NP in the original text ("den nachher aufzuzählenden Eigenschaften, Axiome genannt") which is absent in our adaptation. The reason for this absence is that in the Naproche CNL, the cataphoric antecedent of a cataphoric meta-NP must always directly follow the sentence containing the cataphoric meta-NP. Note that ax-

ioms in a statement list block following an assumption that never gets retracted are semantically equivalent to axioms in axiom blocks, so that dropping this cataphoric meta-NP makes no semantic difference.

The variable type declaration and Axiom 1 in the Naproche CNL adaptation are perfect translations of the original.[4]

In Axiom 2 there are again two differences: One difference is that the Naproche CNL does not introduce a textual expression for the successor function, but only the suffix function symbol $'$. In order to introduce the expression "successor of" using an expression of the form "called the successor of x", we would have to introduce two new linguistic constructs into the Naproche CNL: Transitive nouns (as already discussed in section 7.7) and construction of the form "called the ...". But Landau does not use the expression "successor" outside clarifying comments in the subsequent text, so that there is not much need for introducing this expression here.

The second difference between the Naproche CNL adaptation and the original in Axiom 2 is that we have used a normal existential quantification with "there is a", where the original has a quantification with "there is precisely one" ("gibt es genau eine"). The problem is that after removing the expression "called the successor of x", quantifying with "there is precisely one" would be semantically wrong: "For every x, there is precisely one natural number x'" would have the same truth conditions as "For every x, there is precisely one natural number y", i.e. it would imply that there is at most one natural number. Quantifying with "there is a" makes Axiom 2 a normal implicit dynamic function introduction for the function symbol $'$: The choice principle included in our treatment of implicit dynamic function introductions ensures that the uniqueness restriction is not needed. We can only change this quantification to the more faithful "there is precisely one" if we also use the expression "called the successor of x" here; in this case the semantics of such expressions with "called the" would have to ensure that the quantification with "precisely one" no longer implies that there is at most one natural number.

In Axiom 3 we only had to replace the somewhat imprecise allusion to a universal quantification with "Stets ist" ("We always have") by the more explicit universal quantification with "For every x". Axiom 4 is again a perfect translation of the original. In the quite verbose Axiom 5, the only adaptation we had to make was in the names of the listed properties ("Property 1" and "Property 2" instead of "I)" and "II)").

Let us now consider the *PTL* translation of the axioms. The *PTL* translation of the first sentence existentially introduces a set $\mathbb{N}$ and a unary relation N (corresponding to the noun "natural number"), and asserts that the elements of $\mathbb{N}$ are precisely the objects satisfying N. The fact that N is a unary relation is expressed using the *CMT* function symbol M_1 for unary functions (in *CMTN* we would have $M(N, s(0))$ instead of $M_1(N)$) and the formula $\forall v\ B(N(v))$, which asserts that all values of this function are Booleans.

The variable type declaration does not have a correspondent on the *PTL* side, but does influence the *PTL* translation of the subsequent text: Whenever a small Latin letter x is used for a newly introduced variable, the additional

[4] Steinhardt (trans., 1951) wrongly translates "lateinische" in this variable type declaration as "italic" rather than "Latin". Given that the Naproche CNL adaptation was made from the German original rather than from the English translation, we have conserved the meaning of the original German text in this respect.

subformula $N(x)$ is added to the *PTL* translation.

In Axiom 1, the variable "1" is implicitly introduced in the term "1". Given that implicitly introduced variables are rendered in *PTL* as existentially quantified, the *PTL* translation of the content of Axiom 1 starts with $\exists 1$. Because of the dynamic nature of *PTL*'s existential quantifier, the translation of Axiom 1 has precisely the intended effect, which in the usual metalanguage of mathematicians would be phrased as follows: "Axiom 1 introduces a constant symbol 1, which may be used in the subsequent text, and which is asserted to have the property 'natural number' (i.e. to fulfil the predicate N)."

In the fragment currently considered, there are four more implicitly introduced variables: The x and y in Axiom 4 get implicitly introduced in the formula $x' = y'$; the $\mathfrak{M}$ in Axiom 5 gets implicitly introduced in the term $\mathfrak{M}$ after "Suppose"; and the x in Property 2 of Axiom 5 gets implicitly introduced in the term x after "If". The existential quantifiers corresponding to these four implicitly introduced variables all appear in the φ of a formula of the form $\varphi \to \theta$. The effect of this is that the introduced variable may only be used in the text corresponding to the implication $\varphi \to \theta$, and that after closing the scope of this implication, the variable may be read as universally quantified over the scope of the implication. The 1 implicitly introduced in Axiom 1 also appears in the φ of a formula of the form $\varphi \to \theta$, but in this case the implication $\varphi \to \theta$ spans the rest of the *PTL* translation, so that 1 may be used for the rest of the Naproche CNL text and is never felt as universally quantified.

Given these clarifications, the rest of the *PTL* translation of the fragment considered becomes self-explanatory.

8.1.2 Proof checking

We will now consider what the proof checking algorithm does with the *PTL* translation. For conveniently writing down the *PL* formulae produced by the proof checking algorithm, we will make use of the following definition:

Definition 8.1.1. Given a *PL* formula φ and a variable x, we define $\exists_x \varphi$ and $\forall_x \varphi$ to be $\exists x\ (x \neq u \wedge \varphi)$ and $\forall x\ (x \neq u \to \varphi)$ respectively.

This definition is of course a simplified variant of definition 6.1.8 (which defined the notation $\exists_{\mathbb{T}}\ \varphi$ and $\forall_{\mathbb{T}}\ \varphi$ for a term list $\mathbb{T}$), which we made abundant use of when defining the *PL* translation of *PTL* formulae produced by the proof checking algorithm. Definition 6.1.8 additionally required a substitution of terms by variables, which we will now make explicit. For this we will have to choose variables for substituting terms (compare footnote 10 to definition 6.1.8); we will make this choice in such a way that it supports comprehensibility.

When the *PTL* translation of the Naproche CNL text starting with the currently considered axiom text is handed to the proof checking algorithm, the algorithm will first process the *PTL* translation of the axioms. Given that each translation of an axiom appears to the left of an implication sign, it will be processed within a *read_text* process, and hence the only proof obligations that need to be checked are presuppositional proof obligations. The only presuppositional proof obligations appearing in the axioms are those that check that terms involving the successor function are well-defined (for which the argument of the successor function has to be an natural number). For discussing these presuppositional proof obligations, we first need to discuss the proof checking of the

PTL translation of Axiom 2, which is the axiom that introduces the successor function.

Before encountering the *PTL* translation of Axiom 2, the proof checking algorithm translates the *PTL* formulae preceding it and translates them to *PL* formulae. The *PL* formulae thus created by the algorithm get annotated by *PTL* IDs to make up the premise list Γ, which is the premise list that is active when the algorithm encounters the *PTL* translation of Axiom 2:

$$\begin{aligned}\Gamma = \langle & C(\mathbb{N}),\\ & L(\mathbb{N}),\\ & M_1(N),\\ & \forall_v B(N(v)),\\ & \forall_v(v \in \mathbb{N} \leftrightarrow N(v)),\\ & ax1 : N(1)\rangle\end{aligned}$$

The *PTL* translation of the content of Axiom 2 is $\exists x\ N(x) \to \exists x'\ N(x')$. For understanding how the proof checking algorithm handles this *PTL* formula, you need to keep in mind the definition of *check_text* on formulae of the form $\varphi \to \theta$ from page 108. The proof checking algorithm needs to calculate

$$\mathit{check\_text}(\exists x\ N(x) \to \exists x'\ N(x'), \Gamma, \langle \mathbb{N}, N, 1\rangle, \top).$$

For this it first determines that

$$\mathit{read\_text}(\exists x\ N(x), \Gamma, \langle \mathbb{N}, N, 1\rangle, \top) = (\Gamma, \langle x\rangle, N(x), \top)$$

and that

$$\mathit{check\_text}(\exists x'\ N(x'), \Gamma \oplus \langle N(x)\rangle, \langle \mathbb{N}, N, 1, x\rangle, \top) = (\Gamma \oplus \langle N(x), N(x')\rangle, \langle \mathbb{N}, N, 1, x, x'\rangle, \top),$$

as the interested reader can easily verify.

Next the proof checking algorithm checks whether it may apply Functionality to the map that gets introduced implicitly in this implication. Since L does not appear in $\langle N(x), N(x')\rangle$, this amounts to ensuring that

$$\mathit{check\_limitedness}(\Gamma \oplus \langle N(x), N(x')\rangle, \langle N(x), N(x')\rangle, \langle \mathbb{N}, N, 1\rangle, N)$$

holds. Since $\Gamma \oplus \langle N(x), N(x')\rangle$ together with the non-comprehension axioms of *CMT* does not imply $L(N)$, this can certainly not be established using the first clause in the definition of *check_limitedness*. But the special case formalized in the second clause of this definition does work: Γ contains the premise $\forall_v(v \in \mathbb{N} \leftrightarrow N(v))$, so that it is enough to check that

$$\mathit{check\_limitedness}(\Gamma \oplus \langle N(x), N(x')\rangle, \langle N(x), N(x')\rangle, \langle \mathbb{N}, N, 1\rangle, \mathbb{N})$$

holds. This can now be established using the first clause in the definition of *check_limitedness*: For this the proof obligation

$$\Gamma \oplus \langle N(x), N(x')\rangle \vdash^{?}_{\langle\rangle} L(\mathbb{N})$$

is given to the ATP. Since Γ already contains $L(\mathbb{N})$, this proof obligation is trivially solved by any ATP.

Now the proof checking algorithm proceeds to calculating

$$\mathit{make\_functions}(\langle x\rangle, \langle x'\rangle, \Gamma, \Gamma \oplus \langle N(x), N(x')\rangle, N(x), 1, \top),$$

which is where it will determine that the successor function $'$ is introduced implicitly in the implication $\exists x\ N(x) \to \exists x'\ N(x')$, at the same time determining the domain information of $'$ (i.e. the conditions under which applying $'$ to an argument results in a defined term).

The central part in calculating this *make_functions* term consists of calculating

$$\mathit{make\_function}(\langle x\rangle, x', \Gamma \oplus \langle N(x), N(x')\rangle, N(x), 1, \top).$$

For this the algorithm first needs to find a term T and 1-place argument filler σ such that x' (i.e. $'(x)$) is $T^\sigma(x)$. The choice $T ='$ and $\sigma = (id_{\{1\}}, (0,1))$ satisfies this property. Next the algorithm determines that $'(x)$ is the only function-head subterm of $'(x)$ that contains $'$ as a proper subterm. It then constructs the formula

$$\Psi_1 = \forall_x(N(x) \leftrightarrow x' \neq u)$$

which encodes the domain information of $'$.

Now the algorithm sends the proof obligation

$$\Gamma \oplus \langle N(x), N(x')\rangle \vdash^?_{\langle\rangle} L(x') \tag{8.1}$$

to the ATP. Since $\forall_v(v \in \mathbb{N} \leftrightarrow N(v))$ is in Γ and the *CMT* Element Axiom $\forall x, y\ (L(y) \wedge x \in y \to L(y))$ is among the axioms added to the premise when handing (8.1) to the ATP (see section 6.1.6), (8.1) is solvable by any state-of-the-art ATP.[5]

Since the conditions for applying Functionality were fulfilled (i.e. the α in the definition of *make_function* is 1), the algorithms constructs the formulae $L(')$ and $\forall_x(N(x) \to L(x'))$ and concludes that

$$\begin{aligned}&\mathit{make\_function}(\langle x\rangle, x', \Gamma \oplus \langle N(x), N(x')\rangle, N(x), 1, \top)\\ &= (\langle'\rangle, \langle\rangle, \langle\forall_x(N(x) \leftrightarrow x' \neq u), L('), \forall_x(N(x) \to L(x'))\rangle, \top).\end{aligned}$$

For concluding the calculation of the *make_functions* term, the algorithm now still needs to do two things: Firstly, it lets the ATP check the proof obligation

$$\Gamma \oplus \langle N(x)\rangle \vdash^?_{\langle\rangle} L(x),$$

which can be solved in the same way as (8.1). Secondly, it constructs the formula $\forall_x(N(x) \to L(x))$. Finally, it concludes that

$$\begin{aligned}&\mathit{make\_functions}(\langle x\rangle, x', \Gamma \oplus \langle N(x), N(x')\rangle, N(x), 1, \top)\\ &= (\langle'\rangle, \langle\rangle, \langle\forall_x(N(x) \leftrightarrow x' \neq u), L('), \forall_x(N(x) \to L(x'))\rangle, \langle\forall_x(N(x) \to L(x))^{\mathbb{P}}\rangle, \top).\end{aligned}$$

For concluding the calculation of the processing of the implication under *check_text*, the algorithm now still determines that

$$\mathit{pull\_out\_pres}(\langle\rangle, \langle x, x'\rangle, \Gamma, \Gamma \oplus \langle N(x), N(x')\rangle) = (\Gamma, \langle N(x), N(x')\rangle, \langle\rangle)$$

[5] In this chapter, we will several times claim a proof obligation to be solvable using a state-of-the-art ATP. All proof obligations about which we make such a claim correspond to proof obligations actually produced by Naproche 0.52 on the text under consideration and solved in fractions of a second by the ATP integrated in Naproche 0.52 (E 1.2; for general information about the prover E, (see Schulz, 2004)).

and constructs the formula $\forall_x \forall_y (N(x) \wedge N(y) \rightarrow L(y))$. Finally it can conclude that

$$
\begin{aligned}
&\mathit{check\_text}(\exists x\ N(x) \rightarrow \exists x'\ N(x'), \Gamma, \langle \mathbb{N}, N, 1\rangle, \top)\\
&= (\Gamma \oplus \langle \forall_x (N(x) \rightarrow L(x))^{\mathbb{P}},\\
&\qquad \forall_x \forall_y (N(x) \wedge N(y) \rightarrow L(y))^{\mathbb{P}},\\
&\qquad \forall_x (N(x) \rightarrow N(x')),\\
&\qquad \forall_x (N(x) \leftrightarrow x' \neq u),\\
&\qquad L('),\\
&\qquad \forall_x (N(x) \rightarrow L(x'))\rangle,\\
&\quad \langle \mathbb{N}, N, 1, '\rangle, \top).
\end{aligned}
$$

The fact that the successor function has been introduced in Axiom 2 can now be seen from the fact that the list of accessible terms has been extended from $\langle \mathbb{N}, N, 1\rangle$ to $\langle \mathbb{N}, N, 1, '\rangle$. The formula $\forall_x (N(x) \leftrightarrow x' \neq u)$ encodes the domain information of this introduced function, while the formula $\forall_x (N(x) \rightarrow N(x'))$ can be said to express what Axiom 2 explicitly states about the implicitly introduced function. The other formulae added to the premise express *CMT*-theoretic details about limitedness.

Now we want to consider what presuppositional proof obligations need to be checked in the subsequent axioms. Axiom 3 contains the formula $x' \neq 1$. Before encountering this formula, the active premise list already gets extended by the formula $N(x)$. The term x' in $x' \neq 1$ gets processed by the third clause of *read_term*, for which a presuppositional proof obligation with the conjecture $x' \neq u$ is sent to the ATP. Since the formulae $\forall_x (N(x) \leftrightarrow x' \neq u)$ and $N(x)$ are in the premise list of this proof obligation, the conjecture can be concluded by any state-of-the-art ATP. Similarly, the formula $x' = y'$ in Axiom 4 triggers two presuppositional proof obligations and the term x' in Property 2 of Axiom 5 triggers one presupposition, all of which can be solved in an analogous way to the one just discussed.

The premise list Γ_0 that is active when the algorithm finishes the processing of the *PTL* translation of the five axiom is as follows:

$$
\begin{aligned}
\Gamma_0 = \langle & C(\mathbb{N}),\\
& L(\mathbb{N}),\\
& M_1(N),\\
& \forall_v B(N(v)),\\
& \forall_v (v \in \mathbb{N} \leftrightarrow N(v)),\\
& ax1 : N(1),\\
& ax2 : \forall_x (N(x) \rightarrow L(x))^{\mathbb{P}},\\
& ax2 : \forall_x \forall_y (N(x) \wedge N(y) \rightarrow L(y))^{\mathbb{P}},\\
& ax2 : \forall_x (N(x) \rightarrow N(x')),\\
& ax2 : \forall_x (N(x) \leftrightarrow x' \neq u),\\
& ax2 : L('),\\
& ax2 : \forall_x (N(x) \rightarrow L(x')),\\
& ax3 : \forall_x (N(x) \rightarrow x' \neq u)^{\mathbb{P}},
\end{aligned}
$$

$$
\begin{aligned}
&ax3 : \forall_x(N(x) \to x' \neq 1),\\
&ax4 : \forall_x\forall_y(N(x) \wedge N(y) \wedge x' = y' \to x' \neq u)^{\mathbb{P}},\\
&ax4 : \forall_x\forall_y(N(x) \wedge N(y) \wedge x' = y' \to y' \neq u)^{\mathbb{P}},\\
&ax4 : \forall_x\forall_y(N(x) \wedge N(y) \wedge x' = y' \to x = y),\\
&ax5 : \forall_{\mathfrak{M}}(C(\mathfrak{M}) \wedge L(\mathfrak{M}) \wedge \forall_v(v \in \mathfrak{M} \to N(v)) \wedge 1 \in \mathfrak{M} \to\\
&\qquad \forall_x(N(x) \wedge x \in \mathfrak{M} \to x' \neq u))^{\mathbb{P}},\\
&ax5 : \forall_{\mathfrak{M}}(C(\mathfrak{M}) \wedge L(\mathfrak{M}) \wedge \forall_v(v \in \mathfrak{M} \to N(v)) \wedge 1 \in \mathfrak{M} \wedge\\
&\qquad \forall_x(N(x) \wedge x \in \mathfrak{M} \to x' \in \mathfrak{M}) \to \forall_v(N(v) \to v \in \mathfrak{M}))\rangle
\end{aligned}
$$

Note that for every presuppositional proof obligation that was checked while processing the *PTL* translation of the axioms, we have one presuppositional premise (marked by a superscript $\mathbb{P}$) in Γ_0. The interested reader can check for himself how the *pull_out_pres* function constructs these presuppositional premises from the premise lists and conjectures of the presuppositional proof obligations. For each of Axiom 3, 4 and 5 we have one non-presuppositional premise expressing the actual content of the axiom.

8.2 Theorems 1-3: Properties of the successor function

The second section of Landau's first chapter is named *Addition*, since Landau proves the existence of the addition function and its basic properties in this section. But the first three theorems in this section are preliminary work that establishes some useful properties of the successor function:[6]

Satz 1: *Aus* $x \neq y$ *folgt* $x' \neq y'$.
Beweis: Sonst wäre $x' = y'$, also nach Axiom 4 $x = y$.
Theorem 2: $x' \neq x$.
Beweis: $\mathfrak{M}$ sei die Menge der x, für die dies gilt.

[6] Here is the English translation of this fragment by Steinhardt (trans., 1951):

Theorem 1: *If* $x \neq y$ *then* $x' \neq y'$.
Proof: Otherwise we would have $x' = y'$ and hence, by Axiom 4, $x = y$.
Theorem 2: $x' \neq x$.
Proof: Let $\mathfrak{M}$ be the set of all x for which this holds true.
I) By Axiom 1 and Axiom 3, $1' \neq 1$; therefore 1 belongs to $\mathfrak{M}$.
II) If x belongs to $\mathfrak{M}$, then $x' \neq x$, and hence by Theorem 1, $(x')' \neq x'$, so that x' belongs to $\mathfrak{M}$.
By Axiom 5, $\mathfrak{M}$ therefore contains all the natural numbers, i.e. we have for each x that $x' \neq x$.
Theorem 3: *If* $x \neq 1$, *then there exists one* (hence, by Axiom 4, exactly one) u *such that* $x = u'$.
Proof: Let $\mathfrak{M}$ be the set consisting of the number 1 and of all those x for which there exists such a u. (For any such x we have of necessity that $x \neq 1$ by Axiom 3.)
I) 1 belongs to $\mathfrak{M}$.
II) If x belongs to $\mathfrak{M}$, then, with u denoting the number x, we have $x' = u'$, so that x' belongs to $\mathfrak{M}$.
By Axiom 5, $\mathfrak{M}$ therefore contains all the natural numbers; thus for each $x \neq 1$ there exists a u such that $x = u'$.

I) Nach Axiom 1 und Axiom 3 ist $1' \neq 1$; also gehört 1 zu $\mathfrak{M}$.

II) Ist x zu $\mathfrak{M}$ gehörig, so ist $x' \neq x$, also nach Satz 1 $(x')' \neq x'$, also x' zu $\mathfrak{M}$ gehörig.

Nach Axiom 5 umfaßt also $\mathfrak{M}$ alle natürlichen Zahlen, d.h. für jedes x ist $x' \neq x$.

Theorem 3: *Ist $x \neq 1$, so gibt es ein* (also nach Axiom 4 genau ein) *u mit $x = u'$.*

Beweis: $\mathfrak{M}$ sei die Menge, die aus der Zahl 1 und denjenigen x besteht, zu denen es ein solches u gibt. (Von selbst ist jedes derartige $x \neq 1$ nach Axiom 3.)

I) 1 gehört zu $\mathfrak{M}$.

II) Ist x zu $\mathfrak{M}$ gehörig, so ist, wenn unter u die Zahl x verstanden wird, $x' = u'$, also x' zu $\mathfrak{M}$ gehörig.

Nach Axiom 5 umfaßt also $\mathfrak{M}$ alle natürlichen Zahlen; zu jedem $x \neq 1$ gibt es also ein u mit $x = u'$.

8.2.1 Naproche CNL adaptation and *PTL* translation

Here is our Naproche CNL adaptation of these theorems together with its *PTL* translation:

Theorem 1: If $x \neq y$ then $x' \neq y'$.	$thm(thm, label(thm1, \exists x\ (N(x) \wedge \exists y\ (N(y) \wedge \neg x = y)) \rightarrow \neg x' = y'),$
Proof:	
Assume that $x \neq y$ and $x' = y'$. Then by Axiom 4, $x = y$. Qed.	$\exists x\ (N(x) \wedge \exists y\ (N(y) \wedge \neg x = y)) \wedge x' = y' \rightarrow ref(\langle ax4 \rangle, x = y))$ &
Theorem 2: For all x $x' \neq x$.	$thm(thm, label(thm2, \exists x\ N(x) \rightarrow \neg x' = x),$
Proof:	
Let $\mathfrak{M}$ be the set of x such that $x' \neq x$.	$\exists \mathfrak{M}\ \mathfrak{M} = \iota v\ (C(v) \wedge L(v) \wedge \forall x\ (x \in v \leftrightarrow N(x) \wedge \neg x' = x)) \rightarrow$
By Axiom 1 and Axiom 3, $1' \neq 1$, i.e. 1 belongs to $\mathfrak{M}$.	$ref(\langle ax1, ax3 \rangle, \neg 1' = 1 \wedge 1 \in \mathfrak{M})$ &
If x belongs to $\mathfrak{M}$, then $x' \neq x$, i.e. by Theorem 1 $(x')' \neq x'$, i.e. x' belongs to $\mathfrak{M}$.	$(\exists x\ (N(x) \wedge x \in \mathfrak{M}) \rightarrow (\neg x' = x \wedge ref(\langle thm1 \rangle, (\neg x'' = x' \wedge x' \in \mathfrak{M}))))$ &
By Axiom 5 $\mathfrak{M}$ contains all natural numbers, i.e. for every x $x' \neq x$. Qed.	$ref(\langle ax5 \rangle, (\exists v\ N(v) \rightarrow v \in \mathfrak{M}) \wedge (\exists x\ N(x) \rightarrow \neg x' = x)))$ &
Theorem 3: If $x \neq 1$ then there is a u such that $x = u'$.	$thm(thm, label(thm3, (\exists x\ (N(x) \wedge x \neq 1) \rightarrow \exists u^0\ x = u^{0'})),$
Proof:	
Let $\mathfrak{M}$ be the set of x such that $x = 1$ or there is a u such that $x = u'$.	$(\exists \mathfrak{M}\ \mathfrak{M} = \iota v\ (C(v) \wedge L(v) \wedge \forall x\ (x \in v \leftrightarrow (N(x) \wedge (x = 1 \vee \exists u^0\ x = u^{0'})))) \rightarrow$
1 belongs to $\mathfrak{M}$.	$1 \in \mathfrak{M}$ &
Suppose x belongs to $\mathfrak{M}$. Now if $u = x$ then $x' = u'$. So x' belongs to $\mathfrak{M}$.	$(\exists x\ (N(x) \wedge x \in \mathfrak{M}) \rightarrow ((\exists u^0\ (N(u^0) \wedge u^0 = x) \rightarrow x' = u^{0'})\ \&\ x' \in \mathfrak{M}))$ &

Thus by Axiom 5, $\mathfrak{M}$ contains all natural numbers. Hence for every x such that $x \neq 1$, there is a u such that $x = u'$. Qed.	*ref*$\langle ax5 \rangle$, $(\exists v\ N(v) \rightarrow v \in \mathfrak{M})$) & $(\exists x\ (N(x) \wedge \neg x = 1) \rightarrow \exists u^0\ \neg x = u^{0\prime})))$

We will now discuss the differences between this Naproche CNL adaptation and the original. Note that in general we refrain from mentioning differences completely analogous to differences already previously discussed.

The assertion of the first theorem is a normal implication. The original text implicitly assumes at the beginning of the proof that the antecedent of this implication holds. In other words, one can say that the proof is inside the scope of the implication. In the Naproche CNL, assumptions made in a theorem assertion only have their scope extended in this way into the proof if they are made in a separate sentence. The scope of an assumption introduced with "if" on the other hand always ends at latest at the end of the sentence. Hence we had to make the assumption $x \neq y$ explicit at the beginning of the proof. Furthermore, the proof in the original starts with the anaphoric "sonst" ("otherwise"), which introduces as a further assumption the negation of the proof goal $x' \neq y'$ (compare the proposal to extend the Naproche CNL by "otherwise" made in section 7.7 of chapter 7). By double negation elimination, Landau immediately concludes $x' = y'$ from this. We instead added $x' = y'$ directly to the assumption that we had to add at the beginning of the proof.

In the original text, x is implicitly introduced in the assertion of Theorem 2, but is understood to be universally quantified. Since implicitly introduced variables are always interpreted in a dynamically existentially quantified way in the Naproche CNL, we had to add "For all x" at the beginning of the assertion of Theorem 2.

For characterizing the elements of the set $\mathfrak{M}$ introduced at the beginning of the proof of Theorem 2, Landau uses the anaphoric expression "für die dies gilt" ("for which this holds true"), where "dies" ("this") is anaphorically linked to the formula $x' \neq x$ in the theorem assertion. Since this kind of anaphora is not supported in the Naproche CNL, we had to repeat the formula $x' \neq x$ in this place.

Landau uses the labels "I)" and "II)" to mark parts of the proof of Theorem 2 that correspond to the properties mentioned with the names "I)" and "II)" in Axiom 5. One might wonder why we dropped these labels completely rather than writing a labelled text block with the labels "I)" and "II)" in order to make Naproche CNL adaptation more faithful to the original. In the original, the scope of each of these labels is precisely the paragraph which it started, so the last paragraph of the proof is outside the scope of these labels. But in the Naproche CNL, a new paragraph does not mark the end of a labelled text block; a labelled text block may only end if a superordinated structural block ends. In this case, this could be done by retracting the assumption introduced at the beginning of the proof (thus ending the assumption-consequences block) or by ending the proof. But the last sentence of the proof of Theorem 2 needs to be inside the scope of the assumption. So in this case, we cannot have labelled text blocks in the proof. (Note that in the proof of Theorem 4 we have a labelled proof block of the kind that the Naproche CNL supports, with labels "A)" and "B)".)

We had to make two adaptations to the expression that characterizes the elements of the set $\mathfrak{M}$ introduced at the beginning of the proof of Theorem 3: Firstly, since we cannot characterize a set by an expression of the form "the set containing 1 and those x for which ...", we had to introduce the variable x a bit earlier, writing "the set of x such that $x = 1$ or ...". Secondly, the anaphoric expression "ein solches u" ("such a u") had to be resolved as "a u such that $x = u'$". (Compare the proposal to extend the Naproche CNL by "such a" made in section 7.7 of chapter 7.)

Note the word "Thus" appearing at the beginning of the last paragraph of Theorem 3. It ensures that the assumption "Suppose x belongs to $\mathfrak{M}$" gets retracted. In the original there is of course no such clear marker for the retraction of this assumption. For a reader, this usage of "Thus" in no way reduces the naturality of the text. But an author of a Naproche CNL text has to be careful to put the word "Thus" in the right positions in the text.

At the end of the proof of Theorem 3, we had to render the quantification "zu jedem $x \neq 1$" ("for each $x \neq 1$") as "for every x such that $x \neq 1$". (Compare the proposal made in section 7.7 to extend the Naproche CNL by the possibility to postfix quantterms with a binary infix relation symbol followed by a second argument for the relation.)

Most parts of the *PTL* translation of the three theorems now under consideration are self-explanatory. One detail is worth mentioning: The three references appearing in the proof of Theorem 2 ("By Axiom 1 and Axiom 3", "by Theorem 1" and "By Axiom 5") all appear at the beginning of a simple sentential phrase which is linked to one or more subsequent simple sentential phrases using "i.e.". In the *PTL* translation the scope of these references includes the translations of these subsequent simple sentential phrases, but in the natural and intended reading their scope is limited to the simple sentential phrase that they introduce. The reason for their extended scope in the *PTL* translation is the disambiguation principle mentioned in section 7.3.6 that references are always considered to modify the largest sentential clause that they could possibly modify given their position. This disambiguation principle does not give natural results in the case of complex sentences involving "i.e.". One might consider the possibility of introducing an exception to this disambiguation principle for such complex sentences; but the advantages of introducing such an exception would have to be weighed against the disadvantage of thus making the interpretation rules of the CNL more complex and hence less learnable for potential authors (compare the discussion at the end of section 7.7).

8.2.2 Proof checking

We will now consider what the proof checking algorithm does with the *PTL* translation of the three theorems under consideration. Since the presuppositional proof obligations and presuppositional premises triggered by the successor function were already discussed in section 8.1.2 above, we will ignore these in the discussion that follows.

When checking a theorem-proof block $thm(\vartheta, \varphi, \theta)$, the proof checking algorithm always first checks the proof *PTL* text θ. In the case of Theorem 1, θ has the form of an implication whose right hand side is a conjunction of the two *PTL* formulae $\exists x\ (N(x) \wedge \exists y\ (N(y) \wedge \neg x = y))$ and $x' = y'$, and whose left hand side is $ref(\langle ax4 \rangle, x = y)$. For checking the implication, the proof checking algo-

rithm first extends Γ by the premises $x \neq u^{\mathbb{P}}$, $N(x)$, $y \neq u^{\mathbb{P}}$, $N(y)$, $\neg x = y$ and $x' = y'$ corresponding to the left hand side of the implication. It then proceeds to checking the *PTL* formula $ref(\langle ax4 \rangle, x = y)$ using this extended premise list. The only proof obligation sent to the ATP for checking this *PTL* formula is the following:

$$\Gamma \oplus \langle x \neq u^{\mathbb{P}}, N(x), y \neq u^{\mathbb{P}}, N(y), \neg x = y, x' = y' \vdash^{?}_{\langle ax4 \rangle} x = y$$

Since Γ contains $ax4 : \forall_x \forall_y (N(x) \wedge N(y) \wedge x' = y' \rightarrow x = y)$, this proof obligation is of course quickly solved by any state-of-the-art theorem prover. Having checked this proof obligation, the proof checking algorithm determines that the *PL* translation of the whole proof *PTL* text θ is

$$\Theta = \forall_x \forall_y (N(x) \wedge N(y) \wedge \neg x = y \wedge x' = y' \rightarrow x = y).$$

The proof obligation for the assertion of Theorem 1 now is

$$\Gamma_0 \oplus \langle \Theta, N(x), N(y), \neg x = y \rangle \vdash^{?}_{\langle\rangle} \neg x' = y',$$

which is quickly solved by any state-of-the-art theorem prover.

The premise list that is active after processing Theorem 1 is

$$\Gamma_1 = \Gamma_0 \oplus \langle thm1 : \forall_x \forall_y (N(x) \wedge N(y) \wedge \neg x = y \rightarrow \neg x' = y') - thm \rangle.$$

Note that the formula that has been added to Γ_0 corresponds only to the assertion of Theorem 1. The premise Θ corresponding to the proof of Theorem 1 has been dropped and is no longer accessible for the subsequent proof checking.

The proof of Theorem 2 contains the first use of a definite description, so we will briefly discuss how this is handled in the proof checking. The *PTL* translation of the first sentence of this proof contains the *PTL* term

$$\iota v \ (C(v) \wedge L(v) \wedge \forall x \ (x \in v \leftrightarrow N(x) \wedge \neg x' = x)),$$

which gets processed using *read_term*. For this, the value of the following *exist_check* term has to be calculated:

$$exist\_check(0, \Gamma_1, \exists v \ (C(v) \wedge L(v) \wedge \forall x \ (x \in v \leftrightarrow N(x) \wedge \neg x' = x)), \top)$$

Calculating the value of this term corresponds to checking the existential presupposition of the definite description "the set of x such that $x' \neq x$". Note that the second clause of the definition of *exist_check*, which formalizes the application of the *CMTN* (or *CMT*) Set Comprehension Axiom Schema, may be applied to this *exist_check* term. After checking the proof obligation

$$\Gamma_1 \oplus \langle N(x) \wedge \neg x' = x \rangle \vdash^{?}_{\langle\rangle} L(x)$$

and checking the limitedness of N and $'$ using *check_limitedness*, the algorithm concludes that

$$exist\_check(0, \Gamma_1, \exists v \ (C(v) \wedge L(v) \wedge \forall x \ (x \in v \leftrightarrow N(x) \wedge \neg x' = x)), \top) = (\top).$$

Next the algorithm sends the proof obligation

$$\Gamma_1 \oplus \langle C(sk_0) \wedge L(sk_0) \wedge \forall_x (x \in sk_0 \leftrightarrow N(x) \wedge \neg x' = x) \rangle$$
$$\vdash^{?}_{\langle\rangle} \forall_w (C(w) \wedge L(w) \wedge \forall_x (x \in v \leftrightarrow N(x) \wedge \neg x' = x) \rightarrow w = sk_0)$$

to the ATP. This proof obligation corresponds to checking the uniqueness presupposition of the definite description "the set of x such that $x' \neq x$" and can be solved because of the fact that the *CMT* Set Extensionality Axiom is added to the premise list when handing it to the ATP.

After processing the translation of the first sentence of the proof, the active premise list is

$$\begin{aligned}\Gamma_1^+ = \Gamma_1 \oplus \langle & C(sk_0) \wedge L(sk_0) \wedge \forall_x(x \in sk_0 \leftrightarrow N(x) \wedge \neg x' = x)^{\mathbb{P}}, \\ & \forall_w(C(w) \wedge L(w) \wedge \forall_x(x \in v \leftrightarrow N(x) \wedge \neg x' = x) \rightarrow w = sk_0)^{\mathbb{P}}, \\ & \mathfrak{M} = sk_0\rangle.\end{aligned}$$

At the end of the proof, Landau establishes that for every x, $x' \neq x$. This is what we wanted to establish, so when reading this, we have the feeling that the proof is finished. But in the Naproche CNL adaptation there is a small problem: The assumption with which the proof started ("Let $\mathfrak{M}$ be the set of x such that $x' \neq x$") has still not been retracted. It gets retracted at the "Qed". After this, what we have proved is just that "for every x, $x' \neq x$" follows from this assumption. Of course the reason why we intuitively feel that the proof is already finished when Landau writes "for every x, $x' \neq x$" is that we do not really feel this assumption as an assumption, but just as an introduction of the temporary constant $\mathfrak{M}$ with a defined meaning. We can account for this intuition in the framework of the theory developed in this thesis as follows:

The non-presuppositional content of the assumption is trivial. It is represented by the premise $\mathfrak{M} = sk_0$, where $\mathfrak{M}$ is a newly introduced variable. After retracting the assumption, we thus have a premise of the form

$$\forall_{\mathfrak{M}}(\mathfrak{M} = sk_0 \rightarrow \forall_x x' \neq x)$$

in our active premise list. This premise is trivially equivalent to the conjecture $\forall_x x' \neq x$ of the proof obligation produced for the theorem assertion.[7] Since the non-presuppositional content of the assumption is trivial and leads to this premise trivially equivalent to the desired result, we do not feel the assumption to have any content at all, and hence do not feel it to be an assumption in the first place.

The remainder of the proof checking of Theorems 2 and 3 does not contain any interesting features not discussed so far. Γ_3 is the premise list that is active after checking the *PTL* translation of Theorem 3:[8]

$$\begin{aligned}\Gamma_3 = \Gamma_1 \oplus \langle & thm2 : \forall_x(N(x) \rightarrow x' \neq x) - thm \\ & thm3 : \forall_x(N(x) \wedge x \neq 1 \rightarrow \exists_{u^0} x = u^{0'})\end{aligned}$$

[7] In the Naproche system, the formula simplification mentioned at the end of section 7.5.2 actually ensures that the premise will be simplified in such a way that it becomes identical with the conjecture.

[8] Note that we are ignoring the presuppositional premises resulting from applications of the successor function.

8.3 Theorem 4: The addition function

In Theorem 4, Landau introduces the addition function, proving its existence and uniqueness with the desired properties:[9]

> **Satz 4,** zugleich **Definition 1:** *Auf genau eine Art läßt sich jedem Zahlenpaar x, y eine natürliche Zahl, $x+y$ genannt* ($+$ sprich: plus), *so zuordnen, daß*
>
> 1) $x + 1 = x'$ *für jedes* x
> 2) $x + y' = (x + y)'$ *für jedes* x *und jedes* y.
>
> *$x + y$ heißt die Summe von x und y oder die durch Addition von y zu x entstehende Zahl.*
>
> **Beweis:** A) Zunächst zeigen wir, daß es bei jedem festen x höchstens eine Möglichkeit gibt, $x + y$ für alle y so zu definieren, daß $x + 1 = x'$ und $x + y' = (x + y)'$ für jedes y.

[9] Here is the English translation of this theorem by Steinhardt (trans., 1951):

> **Theorem 4,** and at the same time **Definition 1:** *To every pair of numbers x, y, we may assign in exactly one way a natural number, called $x + y$* ($+$ to be read "plus"), *such that*
>
> 1) $x + 1 = x'$ *for every* x
> 2) $x + y' = (x + y)'$ *for every* x *and every* y.
>
> *$x + y$ is called the sum of x and y, or the number obtained by addition of y to x.*
>
> **Proof:** A) First we will show that for each fixed x there is at most one possibility of defining $x + y$ for all y in such a way that $x + 1 = x'$ and $x + y' = (x + y)'$ for every y.
>
> Let a_y and b_y be defined for all y and be such that
>
> $$a_1 = x', \qquad b_1 = x',$$
> $$a_{y'} = (a_y)', \quad b_{y'} = (b_y)' \quad \textit{for every } y.$$
>
> Let $\mathfrak{M}$ be the set of all y for which $a_y = b_y$.
>
> I) $a_1 = x' = b_1$; hence 1 belongs to $\mathfrak{M}$.
>
> II) If y belongs to $\mathfrak{M}$, then $a_y = b_y$, hence by Axiom 2, $(a_y)' = (b_y)'$, therefore $a_{y'} = (a_y)' = (b_y)' = b_{y'}$, so that y' belongs to $\mathfrak{M}$.
>
> Hence $\mathfrak{M}$ is the set of all natural numbers; i.e. for every y we have $a_y = b_y$.
>
> B) Now we will show that for every x it is actually possible to define $x + y$ for all y in such away that $x + 1 = x'$ and $x + y' = (x + y)'$ for every y.
>
> Let $\mathfrak{M}$ be the set of all x for which this is possible (in exactly one way, by A)).
>
> I) For $x = 1$, the number $x + y = y'$ is as required, since
>
> $$x + 1 = 1' = x',$$
> $$x + y' = (y')' = (x + y)'.$$
>
> Hence 1 belongs to $\mathfrak{M}$.
>
> II) Let x belong to $\mathfrak{M}$, so that there exists an $x + y$ for all y. Then the number $x' + y = (x + y)'$ is the required number for x', since
>
> $$x' + 1 = (x + 1)' = (x')'$$
>
> and
>
> $$x' + y' = (x + y')' = ((x + y)')' = (x' + y)'.$$
>
> Hence x' belongs to $\mathfrak{M}$.
>
> Therefore $\mathfrak{M}$ contains all x.

Es seien a_y und b_y für alle y definiert und so beschaffen, daß

$$a_1 = x', \qquad b_1 = x',$$
$$a_{y'} = (a_y)', \quad b_{y'} = (b_y)' \quad \textit{für jedes } y.$$

$\mathfrak{M}$ sei die Menge der y mit $a_y = b_y$.

I) $a_1 = x' = b_1$; 1 gehört also zu $\mathfrak{M}$.

II) Ist y zu $\mathfrak{M}$ gehörig, so ist $a_y = b_y$, also nach Axiom 2 $(a_y)' = (b_y)'$, also $a_{y'} = (a_y)' = (b_y)' = b_{y'}$, also y' zu $\mathfrak{M}$ gehörig.

Daher ist $\mathfrak{M}$ die Menge aller natürlichen Zahlen; d.h. für jedes y ist $a_y = b_y$.

B) Wir zeigen jetzt, daß es zu jedem x eine Möglichkeit gibt, $x + y$ für alle y so zu definieren, daß $x + 1 = x'$ und $x + y' = (x + y)'$ für jedes y.

$\mathfrak{M}$ sei die Menge der x, zu denen es eine (also nach A) genau eine) solche Möglichkeit gibt.

I) Für $x = 1$ leistet $x + y = y'$ das Gewünschte. Denn

$$x + 1 = 1' = x',$$
$$x + y' = (y')' = (x + y)'.$$

Also gehört 1 zu $\mathfrak{M}$.

II) Es sei x zu $\mathfrak{M}$ gehörig, also ein $x + y$ für alle y vorhanden. Dann leistet $x' + y = (x + y)'$ das Gewünschte bei x'. Denn

$$x' + 1 = (x + 1)' = (x')'$$

und

$$x' + y' = (x + y')' = \big((x + y)'\big)' = (x' + y)'.$$

Also gehört x' zu $\mathfrak{M}$.

Daher umfaßt $\mathfrak{M}$ alle x.

8.3.1 Naproche CNL adaptation and *PTL* translation

Here is our Naproche CNL adaptation of these theorems together with its *PTL* translation:

Theorem 4: There is precisely one function $x, y \mapsto x + y$ such that for all x, y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$.	$thm(thm, label(thm1, (\exists x\ (N(x) \wedge \exists y\ N(y))$ $\rightarrow \exists +\,(x, y)\ \top) \wedge M_2(+) \wedge L(+) \wedge$ $(\exists x\ (N(x) \rightarrow \exists y\ (N(y) \rightarrow N(+(x, y))$ $\wedge +(x, 1) = x' \wedge +(x, y') = +(x, y)'))) \wedge$ $((\exists x\ (N(x) \wedge \exists y\ N(y)) \rightarrow \exists v(x, y)\ \top) \wedge$ $M_2(v) \wedge L(v) \wedge (\exists x\ (N(v) \rightarrow \exists y\ (N(y) \rightarrow$ $N(v(x, y)) \wedge v(x, 1) = x' \wedge$ $v(x, y') = v(x, y)'))) \rightarrow + = v)),$
Proof: A) Fix x. Suppose that there are functions $y \mapsto a_y$ and $y \mapsto b_y$ such that $a_1 = x'$ and $b_1 = x'$ and for all y, $a_{y'} = (a_y)'$ and $b_{y'} = (b_y)'$.	$(\exists x\ N(x) \rightarrow ((\exists y\ N(y) \rightarrow \exists a_\bullet(y)\ \top) \wedge$ $(\exists y\ N(y) \rightarrow \exists b_\bullet(y)\ \top) \wedge M(a_\bullet) \wedge L(a_\bullet) \wedge$ $M(b_\bullet) \wedge L(b_\bullet) \wedge a_\bullet(1) = x' \wedge b_\bullet(1) = x' \wedge$ $(\exists y\ N(y) \rightarrow a_\bullet(y') = a_\bullet(y)' \wedge$ $b_\bullet(y') = b_\bullet(y)') \rightarrow$

Let $\mathfrak{M}$ be the set of y such that $a_y = b_y$.
$a_1 = x' = b_1$, so 1 belongs to $\mathfrak{M}$.
If y belongs to $\mathfrak{M}$, then $a_y = b_y$, i.e. by Axiom 2 $(a_y)' = (b_y)'$, i.e. $a_{y'} = (a_y)' = (b_y)' = b_{y'}$, i.e. y' belongs to $\mathfrak{M}$.
So $\mathfrak{M}$ contains all natural numbers. Thus for all y, $a_y = b_y$.
Thus there is at most one function $y \mapsto x + y$ such that $x + 1 = x'$ and for all y, $x + y' = (x + y)'$.

B) Now let $\mathfrak{M}$ be the set of x such that there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$.
Suppose $x = 1$. Define $x + y$ to be y'. Then $x + 1 = 1' = x'$, and for all y, $x + y' = (y')' = (x + y)'$.
Thus 1 belongs to $\mathfrak{M}$.
Let x belong to $\mathfrak{M}$. Then there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$. For defining $+$ at x', define $x' + y$ to be $(x + y)'$.
Then $x' + 1 = (x + 1)' = (x')'$ and for all y, $x' + y' = (x + y')' = ((x + y)')' = (x' + y)'$.

So x' belongs to $\mathfrak{M}$.
Thus $\mathfrak{M}$ contains all x. So for every x, there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$. Qed.

$(\exists \mathfrak{M}\, \mathfrak{M} = \iota v\ (C(v) \wedge L(v) \wedge \forall y\ (y \in v \leftrightarrow$
$N(y) \wedge a_\bullet(y) = b_\bullet(y))) \rightarrow$
$a_\bullet(1) = x' \wedge x' = b_\bullet(1) \wedge 1 \in \mathfrak{M}\ \&$
$(\exists y\ y \in \mathfrak{M} \rightarrow a_\bullet(y) = b_\bullet(y) \wedge$
$\mathit{ref}(\langle ax2 \rangle, a_\bullet(y)' = b_\bullet(y)' \wedge a_\bullet(y') = a_\bullet(y)' \wedge$
$a_\bullet(y)' = b_\bullet(y)' \wedge b_\bullet(y)' = b_\bullet(y') \wedge y' \in \mathfrak{M}))\ \&$

$\exists v\ N(v) \rightarrow v \in \mathfrak{M})\ \&$
$(\exists y\ N(y) \rightarrow a_\bullet(y) = b_\bullet(y)))\ \&$
$((\exists y\ N(y) \rightarrow \exists +(x)(y)\ \top) \wedge$
$(\exists y\ N(y) \rightarrow \exists v(x)(y)\ \top) \wedge M_2(+)$
$\wedge L(+) \wedge +(x, 1) = x' \wedge (\exists y\ N(y) \rightarrow$
$+(x, y') = +(x, y)') \wedge M_2(v) \wedge L(v) \wedge$
$v(x, 1) = x' \wedge (\exists y\ N(y) \rightarrow$
$v(x, y') = v(x, y)') \rightarrow + = v))\ \&$
$(\exists \mathfrak{M}\, \mathfrak{M} = \iota v\ (C(v) \wedge L(v) \wedge \forall x\ (x \in v \leftrightarrow$
$(\exists y\ N(y) \rightarrow \exists +(x)(y)\ \top) \wedge (\exists y\ N(y) \rightarrow$
$N(+(x)(y)) \wedge +(x)(1) = x' \wedge$
$+(x)(y') = +(x)(y'))))) \rightarrow$

$(\exists x\ (N(x) \wedge x = 1) \rightarrow (\exists y\ N(y) \rightarrow$
$\exists +(x)(y)\ +(x)(y) = y')\ \&$
$+(x)(1) = 1' \wedge 1' = x' \wedge (\exists y\ N(y) \rightarrow$
$+(x)(y') = y'' \wedge y'' = +(x)(y'))\ \& 1 \in \mathfrak{M}\ \&$
$(\exists x\ (N(x) \wedge x \in \mathfrak{M}) \rightarrow (\exists y\ N(y) \rightarrow$
$\exists +(x)(y)\ \top) \wedge (\exists y\ N(y) \rightarrow N(+(x)(y)) \wedge$
$+(x)(1) = x' \wedge +(x)(y') = +(x)(y)')\ \&$

$(\exists y\ N(y) \rightarrow$
$\exists +(x')(y)\ +(x')(y) = +(x)(y')\ \&$
$+(x')(1) = +(x)(1)' \wedge +(x)(1)' = x'' \wedge$
$(\exists y\ N(y) \rightarrow +(x')(y') = +(x)(y')' \wedge$
$+(x)(y')' = +(x)(y)'' \wedge$
$+(x)(y)'' = +(x')(y')\ \&$
$x' \in \mathfrak{M})\ \&$
$(\exists x\ N(x) \rightarrow x \in \mathfrak{M})\ \& (\exists x\ N(x) \rightarrow$
$(\exists y\ N(y) \rightarrow \exists +(x)(y)\ \top) \wedge (\exists y\ N(y) \rightarrow$
$N(+(x)(y)) \wedge +(x)(1) = x' \wedge$
$+(x)(y') = +(x)(y'))))$

We will now discuss the differences between this Naproche CNL adaptation and the original.

The first difference is only a minor one, namely that in the Naproche CNL adaptation we could not mention the alternative name "Definition 1" of Theorem 4. But we do want to call the reader's attention to the fact that Landau has called a theorem with an existential assertion a definition. This lends additional support for our semantic treatment of definitions, according to which definitions are translated into *PTL* in the same way as certain existential assertions (compare section 7.5.4).

Now we turn our attention to a more significant modification of the original, in the assertion of Theorem 4: Where the adaptation makes an explicitly existential statement about the existence of a certain function, the original speaks more informally of the possibility to assign a natural number to every pair of natural numbers. So far the Naproche CNL does not allow this alternative more informal way of phrasing an assertion about the existence of a function. It is possible to extend the Naproche CNL by such a means of expression, but before doing this, one should not only survey whether this means of expression is used by various mathematical authors, but also whether it is considered good style by modern mathematicians. At any rate, we believe that for formal mathematicians, even when given the means to express themselves in a natural input language, it is intuitive and reasonable to adapt such an informal and implicit means of expression by a more explicit one, as we have done.

The reader might wonder why we used the somewhat cumbersome and unusual expression "function $x, y \mapsto x + y$" instead of the simpler "function $+$" in this existential assertion. The reason is that if we had simply written "function $+$", we would not have expressed the fact that the function should be defined precisely on the natural numbers. The properties we require of the function in the rest of the sentence imply that it must be defined at least on all pairs of natural numbers, but they do not exclude the possibility of it being defined on a larger domain including the pairs of natural numbers. Because of the uniqueness assertion made in this sentence, the sentence would even turn out false: If we may consider functions defined on larger domains, there is more than one function satisfying the stated properties. In the original text, Landau speaks of assigning $x + y$ to a pair of numbers x, y, and the fact that he uses small Latin letters x, y implies that these must be natural numbers. In this way it is clear that the domain of the intended function must be precisely the set of pairs of natural numbers. In our adaptation, this is modelled well by the usage of a dependent quantterm $x, y \mapsto x + y$, in which it is also the usage of the small Latin letters x and y that gives us this information about the domain of the intended function.

In the original, the fact that $x + y$ is always a natural number is expressed as part of the assertion about ways of assigning $x + y$ to pairs x, y. In our adaptation we had to mention it after the explicit existential quantification as first property required of the function in question.

Where the original mentions the two further required properties in a list and with postposed universal quantifications ("für jedes x" and "für jedes x und jedes y"), we have stated these two properties in-line inside the scope of a single preposed universal quantification ("for all x, y"), whose scope also contains the assertion that $x + y$ is a natural number. As we have mentioned in section 7.7, postposed universal quantification is so far not possible in the Naproche CNL. The usage of a single universal quantification over x and y is motivated by the fact that the Naproche CNL has no means for closing the scope of a universal quantification with "for all" without closing the scope of some superordinated constituent, and justified by the fact that $\forall \bar{x}\ (\varphi_1(\bar{x}) \wedge \ldots \wedge \varphi_n(\bar{x}))$ is equivalent to $\forall \bar{x}\ \varphi_1(\bar{x}) \wedge \ldots \wedge \forall \bar{x}\ \varphi_n(\bar{x})$.

Landau's proof is divided into two sections, and at the beginning of each section, he announces what he will prove in this section. As mentioned in section 7.7, the Naproche CNL does not yet support such expressions announcing subgoals of a proof goal, as it does not support any goal-oriented proving. So

we have replaced these announcements of subgoals made at the beginning of each section by corresponding assertions at the end of each section.

In both sections Landau introduces a function + in a similar way as in the theorem assertion. But instead of talking about ways of assigning a number $x + y$ to a pair x, y of numbers, he now speaks of ways of defining a number $x+y$ for a number y, where x has been previously fixed. The difference between the verbs "assign" and "define" is here semantically irrelevant and does not influence the Naproche CNL adaptation. But the fact that we now fix x in advance and hence make $x + y$ dependent only on y means that instead of the dependent quantterm "$x, y \mapsto x + y$" we now use the dependent quantterm "$y \mapsto x + y$" in the Naproche CNL adaptation. In particular, this means that all occurrences of the function symbol + in the proof are not, like the function symbol + in the theorem assertion, binary function symbols of notational type `infix`, but have notational type `[suffix,prefix]`. In other words, they are interpreted in a curried way: They are applied first as a suffix function to the preposed argument x to form a function $x+$, which is then applied as a prefix function to y (or some other possible argument, e.g. y').

Apart from this, the first section of Landau's proof did not have to be modified in a note-worthy way. In the second section, Landau twice uses an expression of the form "leistet ... das Gewünschte" ("... is as required"). These are anaphoric expressions referring to part of the subgoal announced at the beginning of the second section. The sentences involving these expressions implicitly define the function + at certain values. We have made the definitional character of these sentences explicit and dropped the anaphoric expression.

In the long chained equation at the end of the proof, the variable y is implicitly introduced in the original, but interpreted in a universally quantified way. Since implicitly introduced variables are interpreted existentially in the Naproche CNL, we have made the universal quantification explicit by adding "for all y" in front of the equation.

We now turn our attention to the *PTL* translation of Theorem 4. The translation of the theorem assertion is very long and may scare off the reader. The main reason for this length is the doubling of the semantic contribution of the $\overline{\mathrm{N}}$ following "precisely one": The *PTL* translation of the theorem assertion is of the form $\varphi(+) \wedge (\varphi(v) \to + = v)$, where $\varphi(+)$ is the following *PTL* formula, which implicitly dynamically introduces the function + (compare the explanation of the translations of dependent quantterm in section 7.5.7) and states its intended properties:

$$\begin{aligned}\varphi(+) =&(\exists x\ (N(x) \wedge \exists y\ N(y)) \to \exists +(x, y)\ \top) \wedge M_2(+) \wedge L(+) \wedge \\ &(\exists x\ (N(x) \to \exists y\ (N(y) \to N(+(x, y)) \wedge +(x, 1) = x' \wedge \\ &+(x, y') = +(x, y)')))\end{aligned}$$

A similar doubling of a long *PTL* formula occurs in the last sentence of part A) of the proof due to the usage of "at most one".

Note that the different syntactic treatment of the function symbol + in the theorem assertion and proof, which we discussed above, is of course also reflected in the *PTL* translation: While + takes two arguments at once in the *PTL* translation of the theorem assertion, it takes its two arguments one after the other in the *PTL* translation of the proof.

Apart from this, the translation of the proof of Theorem 4 does not contain any surprises. In part B) of the proof the reader can see examples of how definitions get translated in the Naproche CNL, of course in line with the explanations in section 7.5.4.

8.3.2 Proof checking

The interesting part in the proof checking of Theorem 4 is how the existence and uniqueness of a function with the properties desired for the addition function is attained. We will focus on this point and leave out the details of the proof checking that are not relevant for this point, as they are at any rate similar to what we have already seen in the previous sections.

The *PTL* translation of part A) of the proof is of the form $\exists x\ N(x) \to \theta(x)\ \&\ \varphi(x)$, where $\varphi(x)$ encodes the information that there is at most one function $y \mapsto x + y$ satisfying the intended properties and $\theta(x)$ encodes an argument for concluding $\varphi(x)$. $\varphi(x)$ encodes this information by an implication of the form $\psi(+) \wedge \psi(v) \to + = v$, where $\psi(+)$ dynamically introduces $+$ and states its intended properties. $\theta(x)$ has a similar form:

$$\begin{aligned}\theta(x) =&(\exists y\ N(y) \to \exists a_\bullet(y)\ \top) \wedge (\exists y\ N(y) \to \exists b_\bullet(y)\ \top) \wedge \psi'(a_\bullet) \wedge \psi'(b_\bullet)\\ &\to \xi(a_\bullet, b_\bullet) \wedge (\exists y\ N(y) \to a_\bullet(y) = b_\bullet(y))\end{aligned}$$

Here $\psi'(a_\bullet)$ states the intended properties of the function $a_\bullet$ in the same way as $\psi(+)$, but without dynamically introducing the function. Instead, what precedes $\psi'(a_\bullet)$ in $\theta(x)$ dynamically introduces $a_\bullet$ and $b_\bullet$. The *PTL* text $\xi(a_\bullet, b_\bullet)$ contains an argument for concluding $(\exists y\ N(y) \to a_\bullet(y) = b_\bullet(y))$.

Note that $\theta(x)$ would have been more similar to $\varphi(x)$ if it ended in $a_\bullet = b_\bullet$ instead of $(\exists y\ N(y) \to a_\bullet(y) = b_\bullet(y))$. Of course, the first follows from the second by Map Extensionality. But since $a_\bullet = b_\bullet$ is not mentioned in $\theta(x)$, Map Extensionality is not needed when proof-checking $\theta(x)$, but when using the premise list produced by $\theta(x)$ for proof-checking $\varphi(x)$. So the proof-checking of $\varphi(x)$ can only succeed because we add an appropriate instance of the *CMT* Map Extensionality Axiom Schema to the premise list of the proof obligation for concluding $+ = v$, as explained in section 6.1.6.

Part B) of the proof establishes the existence of the addition function. It does not directly establish the existence of the intended binary addition function, but of a curried version thereof. This is done by establishing that for every x there is a function $y \mapsto x + y$ with the intended properties. The fact that such a function exists for every x means that by the principle of implicit dynamic function introduction applied in the same way as in Axiom 2 (see section 8.1.2 above), $+$ is a `[suffix,prefix]` function with the desired properties. We will discuss below how the existence of this curried function together with the uniqueness of such a curried function established in part A) implies the existence and uniqueness of the intended uncurried function.

Now we want to consider how part B) of the proof establishes that for every x there exists a function with the intended properties. Note that the Landau text contains no axioms about the existence of functions, and the *CMT* axioms that we add to the premise list during the proof checking also do not contain the axioms of the Map Comprehension Axiom Schema. The reason why we can nevertheless prove the existence of functions is because of the principle of

implicit dynamic function introduction included in *PTL* and the proof checking algorithm. We will now show how this works in practice by discussing the case $x = 1$ of the proof.

The definition "Define $x + y$ to be y'" is translated as $\exists y\ N(y) \rightarrow \exists + (x)(y)\ + (x)(y) = y'$. The crucial part is the existential quantification $\exists + (x)(y)\ + (x)(y) = y'$. When proof-checking this part of the *PTL* translation, the algorithm will send a proof obligation with conjecture $\exists z\ z = y'$ to the ATP. Here the complex term $+(x)(y)$ has been replaced by a new variable z in order to make the conjecture a *PL* formula. As can be easily seen, this conjecture is trivially valid and can hence be checked by any ATP. But in the premise (8.2) that is then added to the active premise list, the complex term $+(x)(y)$ is not replaced by z, but is left as a complex term, only rewritten using the application function app_1:

$$app_1(app_1(+, x), y) = y' \tag{8.2}$$

When closing the scope of the implication $\exists y\ N(y) \rightarrow \exists + (x)(y)\ + (x)(y) = y'$, (8.2) gives rise to the premise $\forall_y(N(y) \rightarrow app_1(app_1(+, x), y) = y')$.

In this way, the proof checking algorithm has already introduced the function $+$. The following sentence establishes that it has the desired properties. Here again the premise list gets extended by premises involving $+$ as first argument of app_1. When in the next sentence the proof checking algorithm has to check that $1 \in \mathfrak{M}$, it has among the active premises the premise characterizing the elements of $\mathfrak{M}$ as natural numbers x for which a function with certain properties (involving x) exists, and has premises involving $app_1(+, x)$ and stating of it that it is a function with these properties for the case $x = 1$. Hence it can successfully check that $1 \in \mathfrak{M}$.

This concludes our discussion of the function existence proof in the case $x = 1$ of the proof. In the case of x' it works completely analogously.

After checking the proof of Theorem 4, the proof checking algorithm needs to check the theorem assertion. The only problem here is, as already mentioned above, the fact that the theorem asserts the existence and uniqueness of a binary function, whereas the proof has established the existence and uniqueness of the corresponding curried function. At this point, we will need the currying-uncurrying axioms discussed in section 6.1.6, which are added to the premise list of proof obligations in the same way as *CMT* axioms. For proving the existence of the binary addition function, we need the Uncurrying Axiom for $unc_{1,1}$, since this axiom transforms a doubly unary curried function into a binary uncurried function. For proving the uniqueness of the binary addition function, we need the Currying Axiom for $cur_{1,1}$ and the *unc-cur* Axiom for $unc_{1,1}$ and $cur_{1,1}$, as the interested reader can easily check.

Chapter 9

Conclusion and outlook

In this work we have given a thorough analysis of the language of mathematics and have proposed a theoretical framework for proof-checking mathematical texts in controlled natural language. This theoretical framework consists of three main parts:

- A formal language *PTL* (*Proof Text Logic*), which captures dynamic and text-structural aspects of natural mathematical texts and whose semantics was built on the foundational theory *CMTN*, a theory with classes, maps tuples, natural numbers and Booleans as primitive objects and equiconsistent to *ZFC*.
- A proof checking algorithm for *PTL*, which makes use of an automated theorem prover for standard first-order predicate logic for checking the correctness of given *PTL* text.
- A controlled natural language for mathematical texts, whose semantics was specified by defining a translation from this controlled natural language to the formal language *PTL*.

We have motivated the formal language *PTL* by certain constructs in the language of mathematics. Especially noteworthy is the *implicit dynamic function introduction* found in the language of mathematics, which to our knowledge had not been previously described or formalized by other logicians or linguists.

The proof checking algorithm has been shown to be sound and correct both with respect to *PTL* semantics and with respect to the semantics of standard first-order predicate logic.

As an example for the practical functioning of the theoretic framework proposed in the thesis, we have shown in detail how it can be applied to the beginning of Landau's (1930) *Grundlagen der Analysis*.

The theory described in this thesis has largely been implemented in the Naproche system; for the differences between the actual implementation and the theory as described here, see appendix C.

9.1 Outlook

In the course of the thesis, we have already mentioned some ways in which the theoretical framework proposed here can be developed further or adapted.

We now discuss some of these possible further developments that we consider especially promising.

Less controlled input language

In the introduction we mentioned the goal of having an input language that is natural for the potential users, in our case for the mathematicians. But in order to make the problem tractable, we limited ourself to the usage of a *controlled natural language* with a limited syntax and limited semantic interpretation rules. The main reason for this is that in the application we had in mind – the verification of mathematical proofs – absolute reliability of the program is very important. However, absolute reliability of uncontrolled natural language processing is not attainable. Nevertheless, there is a way in which we can attain absolute reliability while allowing the input language to leave the stringent requirements of a controlled natural language: For the statement of the axioms, definitions and the assertions of theorems, it is important to have an absolutely reliable interpretation of the natural language input, and for this the controlled natural language approach should be kept. But inside the proofs to the stated theorems, we could allow for a more flexible natural input language, for whose interpretation one could use heuristics based on statistical methods for natural language processing. There is then a risk of misinterpretation by the system; but if a proof is found to be correct, we know for sure that the system has used the input by the author to find some valid mathematical proof of the stated theorem, and the statement of the theorem was certainly interpreted correctly, as it is still written in a controlled natural language.

This envisioned division between controlled natural language in axioms, definitions and theorem statements on the one hand and more flexibility in proofs on the other hand reflects an actually existing difference between these parts of a proof text in actual mathematical practice: Mathematicians usually use natural language much more careful in axioms, definitions and theorem statements, since they are aware of the fact that misunderstanding in these places can be very grave. Inside a proof, on the other hand, they often use natural language in a more sloppy way, as they know that a misunderstanding will usually be detected by a careful reader based on the fact that the misunderstood statement no longer functions as a valid proof step in the proof of the theorem in question.

Goal-oriented proving

In the proof-checking as described in this thesis, we only took account of forward reasoning, in which one builds up proven facts until one attains the fact that one wanted to show. But mathematicians usually state the desired result at the beginning of the proof, and as they are proving it, they might also use backward reasoning: The assertion of a theorem or lemma can be considered a goal of the proof that follows it, and certain proof steps can be considered to simplify this goal. For example, if the goal is to show an implication, then assuming the antecedent of this implication simplifies the goal to the conclusion of the implication. Mathematicians use expressions like "We still need to show that ..." in order to guide the reader through such a goal-oriented proof.

In text linguistics, there is a model called the *Quaestio* model for analysing how a text in its totality as well as parts of it are aimed at answering a certain

question, called *Quaestio* in this model; see Klein and Stutterheim (1987). We hope that this linguistic model in combination with existing methods for formalizing goal-oriented proving can give us new insights as to how goal-oriented proving works in practice, and how it should be implemented in a natural language proof system like the Naproche system.

Making use of type theory

In section 3.3, we briefly mentioned the possibility to use a type-theoretic approach to avoiding the paradoxes of unrestricted function comprehension. The reason we gave for developing an untyped theory for avoiding the paradoxes is that mathematicians sometimes make use of functions that do not fit into the corset of strict typing, e.g. a function defined on both real numbers and real functions. We have, however, made use of some type-theoretic machinery in the disambiguation of the symbolic parts of the Naproche CNL. Additionally, since most of the times mathematicians do actually work with objects that fit into a the corset of strict typing, one can argue that requiring such strict typing in a system like the Naproche system is comparable to requiring controlled natural language rather than allowing unrestricted natural language input.

In the actual implementation of the Naproche system (see appendix C), we do actually use the type-theoretic restriction introduced for disambiguation purposes for avoiding the paradoxes of unrestricted function comprehension. Ackermann-like function theory is not implemented in the system. So based on the actual state of the system, we could further develop the type-theoretic approach.

It is an interesting undertaking to explore how mathematicians actually make use of type information in their texts. Ganesalingam (2009) has made some research in this area, and has come up with an ingenious novel type system for typing the objects that a given mathematical text refers to. However, he uses his type system only for purposes of linguistic disambiguation. It would be interesting to explore further whether this function of his type system can be combined with the function of avoiding paradoxes. Additionally, a type system can make some presuppositional calls to the prover redundant and hence make the system more effective.

Appendix A

Formal grammar of the Naproche CNL

This appendix provides formal grammars of the partial grammars that make up the grammar of the Naproche CNL, namely of the macro-grammar, the textual grammar and the quantterm grammar. The fourth component of the grammar of the Naproche CNL, namely the term grammar, is not presented as a purely formal grammar, but is also given a more formal characterization than in chapter 7. The interaction between the component grammars is not defined formally, but described semi-formally.

All formal grammars in this appendix are written in the *definite clause grammar* (DCG) formalism (see Pereira & Warren, 1980) with Prolog syntax. Additionally to standard Prolog syntax, we also make use of the syntax for feature structures that is defined by GULP 4, a package for SWI-Prolog (see Covington, 1994b, 2007). Furthermore, we have added a predicate `change_feature/4` to GULP that can be used to change the value of a single feature: `change_feature(FeatureStructureIn, Feature,NewValue,FeatureStructureOut)` holds precisely if `FeatureStrutureOut` coincides with `FeatureStructureIn` on all features other than `Feature`, and takes the value `NewValue` at the feature `Feature`.

The formal grammars are commented to some extend, in order to make them more comprehensible. Comments always appear in lines starting with `%`. Additionally, we make use of the semi-formal type- and mode-description that is part of the structured comment style defined by SWI Prolog (see Wielemaker, Schrijvers, Triska, & Lager, 2012). A comment of the form

```
predicate(+Arg1,+Arg2,-Arg3)
```

means that we will now define a three-place predicate, whose first two arguments can be considered input and whose last argument can be considered output.

The macro and textual grammars are intended for a top-down-parser (see Covington, 1994a, p. 151), the quantterm grammar for a chart parser (see Covington, 1994a, p. 167).

We consider the input given to these grammars to be tokenized and preprocessed: A text is presented as a list of sentences. Paragraph boundaries

are represented as sentences whose only word is "##". A sentence is presented as a list of its words. A mathematical expression coming from a single LaTeX mathematics environment is treated like a single word inside a sentence, and is presented as a list of the symbols appearing in it, with the normalization described in section 7.4 already realized. LaTeX environments like `\begin{axiom} ... \end{axiom}` are represented in the same way as a sequence of sentences of the form `axiom. ... End_axiom.`.

A.1 Macro-grammar

The macro-grammar defines how different kinds of sentences can be put together to a Naproche text. The form of the different kinds of sentences is defined in the textual grammar. Here in the macro-grammar the different kinds of sentences defined in the textual grammar are the terminal symbols.

The beginning of a new paragraph may serve as an indicator that some text segment (e.g. an axiom) is finished. However, in contexts where they cannot serve as such indicators, new paragraphs may be freely used without any influence on the parsing. For formalizing this behaviour, the macro-grammar has two undefined predicates, `begin_limited_new_paragraphs` and `end_limited_new_paragraphs`, which can appear in this order among the goals of a clause. This should be thought of as an indication to the parser of the macro-grammar that during the parsing process that takes place between the encounter of the first and the second of these two corresponding predicate occurrences, a new paragraph may only be parsed where there is a `new_paragraph` terminal symbol in the grammar. In all other cases, new paragraphs may be parsed even where there is no `new_paragraph` terminal symbol in the grammar.

Here is the commented formal macro-grammar:

```
%%  text(?Features,?Emptyness)
%
%   A text can contain different kinds of construct, depending on its
%   features. It may always contain simple assertions.
%   The argument Features is a feature list with features "in_proof"
%   (taking values "no", "lemma" and "theorem") and "in_case" (taking
%   values "yes" and "no").
%   The argument Emptyness takes values "yes" or "no".
%
%   For parsing a Naproche text, we initialize Features with
%   in_proof~no..in_case~no..label~nothing.

text(F,no) -->
    text_block_sequence(F,_,no).

text(F,no) -->
    [assertion],
    text(F,_).
```

```
text(F,no) -->
    { F = in_proof∼no..in_case∼no },
    axiom,
    text(F,_).

text(F,no) -->
    theorem(TheoremType),
    {
    \+ F = in_proof∼lemma,
    ( TheoremType = theorem -> ( F = in_proof∼no..in_case∼no ) ; true )
    },
    text(F,_).

text(F,no) -->
    definition,
    text(F,_).

text(F,no) -->
    [assumption],
    text(F,no),
    optional_closing_and_text(F).

text(F,no) -->
    note(F),
    text(F,_).

text(F,no) -->
    { F = subtype∼beginning..length∼greater_than_1 },
    cases(F),
    optional_case_closing_and_text(F).

text(_,yes) -->
    [].

%%  text_block_sequence(?Features,?Type,?Emptyness)

text_block_sequence(F,Typ,no) -->
    text_block(F,Typ),
    text_block_sequence(F,Typ,_).

text_block_sequence(_,_,_,yes) --> [].

%%  text_block(?Features,?Typ)
%
%   A text block starts with a label of the form 'word)'. There is a feature
```

```
%   "label" which indicates the type of  enumeration (lower-case, upper-case,
%   roman or arabic numbers). If within a text segment a label of the same
%   type as at the beginning of the text segment is encountered, it fails.
%   Therefore a new text segment is started.

text_block(F,Typ) -->
    { F = label~NoTyp },
    [label(Typ,NoTyp)],
    { change_feature(F,label,Typ,NewF) },
    text(NewF,no).

%%  axiom
%
%   An axiom consists of an axiom heading, followed by a possibly empty
%   list of assumptions, followed by a non-empty list of assertions,
%   followed by some marking of the end of the axiom, which may just be
%   a new paragraph. Since a new paragraph marks the end of the axiom,
%   there may be no new paragraph within the axiom.

axiom -->
    [heading(axiom)],
    { begin_limited_new_paragraphs },
    assumptions,
    assertions,
    axiom_ending,
    { end_limited_new_paragraphs }.

axiom_ending -->
    [end(axiom)],
    optional_new_paragraphs.

axiom_ending -->
    new_paragraphs.

%   new_paragraphs
%
%   One or more new paragraph sentences.

new_paragraphs -->
    [new_paragraph],
    optional_new_paragraphs.

%   optional_new_paragraphs
%
%   Zero or more new paragraph sentences.
```

```
optional_new_paragraphs -->
    [new_paragraph],
    optional_new_paragraphs.

optional_new_paragraphs -->
    [].

%   optional_closing_and_text(?Features)
%
%   Parses either an assumption closing followed by possibly empty text,
%   or nothing.

optional_closing_and_text(F) -->
    [closing],
    text(F,_).

optional_closing_and_text(_) -->
    [].

%%  note(?Features)

note(F) -->
    [heading(note)],
    note_core(F).

note(F) -->
    note_core(F).

note_core(F) -->
    { F = in_proof~no..in_case~no },
    [var_type_fix].

note_core(_) -->
    [alternative_notation].

optional_note(F) -->
    note(F).

optional_note(_) -->
    [].

%  theorem(-TheoremType)
%
%   A theorem consists of a heading, a goal text, the marker "Proof", a body
```

```
%   text (which possibly includes lemmas) and the marker "Qed".

theorem(TheoremType) -->
    [theorem_heading(TheoremType)],
    assumptions,
    assertions,
    optional_note(in_proof∼no..in_case∼no),
    optional_end(TheoremType),
    [proof],
    text(in_proof∼TheoremType..in_case∼no..label∼nothing,no),
    [proof_end].


optional_end(TheoremType) -->
    [end(TheoremType)].

optional_end(_) -->
    [].

%   assumptions
%
%   0 or more assumptions.

assumptions -->
    [assumption],
    assumptions.

assumptions -->
    [].

%   assertions
%
%   1 or more assertions.

assertions -->
    [assertion],
    optional_assertions.

optional_assertions -->
    [assertion],
    optional_assertions.

optional_assertions -->
    [].
```

```
%   definition
%
%   definition parses a definition possibly preceeded with a sentence
%   declaring a name for the definition (e.g. "Definition 4:").

definition -->
    [heading(definition)],
    [definition],
    optional_end(definition).

definition -->
    [definition].


%%  cases(+Features)
%
%   The Features argument consists of four features:
%   - "in_case" can either have the values "yes" or "no".
%   - "in_proof" can either have the values "yes" or "no".
%   - "subtype" can either have the value "beginning" or "rest-list".
%   - "length" can have the values "greater_than_1", "greater_than_0" and "0".
%
%   The "length" feature is used to ensure that a case distinction has at
%   least two cases.

cases(in_case~X..in_proof~Y..subtype~Z..length~Length) -->
    case_introduction(in_case~X..subtype~Z),
    [case_id],
    [case],
    text(in_case~yes..in_proof~Y..label~nothing,_),
    { Length == greater_than_1 -> SubLength = greater_than_0 ; true },
    cases(in_case~yes..in_proof~Y..subtype~rest-list..length~SubLength).

cases(subtype~rest-list..length~0) -->
    [].

%%  case_introduction(?Features)
%
%   Depending on its features, case_introduction either parses nothing or
%   a sentence that announces the beginning of a case distinction: A case
%   distinction made within a case distinction (i.e. when the feature
%   "in_case" is not "yes") must be announced; else the announcement is
%   optional. If the feature "subtype" is "rest-list", then we are not at
%   the beginning of a case distinction, but at the beginnning of a new
```

```
%   case of an already open case distinction. Hence there may be no
%   announcement of the beginning of a case distinction in that case.

case_introduction(in_case~no) -->
    [].

case_introduction(subtype~rest-list) -->
    [].

case_introduction(subtype~beginning) -->
    [case_intro].

optional_case_closing_and_text(F) -->
    [case_closing],
    text(F,_).

optional_case_closing_and_text(in_case~no) -->
    [].

%%  list(+FollowingType,+Number)
%
%   A list has to be parsed after a sentence containing a following_np
%   (see textual grammar). The values of FollowingType and Number depend
%   on the following_np.

list(FollType,singular) -->
    {FollType = axiom ; FollType = case ; FollType = property},
    list_element(FollType).

list(FollType,plural) -->
    {FollType = axiom ; FollType = case ; FollType = property},
    list_element(FollType),
    list_element(FollType),
    optional_list(FollType).

optional_list(FollType) -->
    list_element(FollType),
    optional_list(FollType).

optional_list(_) -->
    [].

list_element(FollType) -->
    [heading(FollType)],
    { begin_limited_new_paragraphs },
```

```
    assumptions,
    assertions,
    list_element_ending(FollType),
    { end_limited_new_paragraphs }.

list_element_ending(axiom) -->
    [end(axiom)],
    optional_new_paragraphs.

list_element_ending(_) -->
    new_paragraphs.
```

A.2 Textual grammar

The textual grammar defines the grammar on the sentence level. Since it makes extensive use of feature structures, we first explain all features used in the grammar:

- `number`: This feature marks the grammatical number (`singular` or `plural`) of a noun phrase, specifier, noun, verb phrase, verb or similar component. Additionally, it can be used to count whether a list (e.g. a list of quant-terms) has one or more than one element. In that case it can also take the value `null`, meaning that the list is empty.

- `mode`: This feature indicates whether a verb or verb phrase is in the finite mode (`finite`) or in the infinitive mode (`infinitive` or `to-infinitive`, depending on whether it is an infinitive without or with "to"). Additionally, sentential phrases may take this mode feature, in which case it indicates the mode of the head verb of the sentential phrase (or of any of a number of coordinated sentential phrases).

- `transitive`: This feature is used to distingish between transitive verbs (for which it takes the value `plus`), intransitive verbs (`minus`) and the copula "to be" (`copula`).

- `adj_trans`: This feature indicates whether an adjective is transitive or not (see section 7.3.1), and – in the case of transitive adjectives – specifies which preposition is used for the complement. Its value is either `no` (for intransitive adjectives) or the name of the preposition used for the complement of a transitive adjective.

- `specifier_type`: The specifier (i.e. determiner) of a noun phrase may be of `specifier_type` `definite` ("the"), `indefinite` (e.g. "a", "some" and the empty plural specifier), `negative` ("no") or `universal` ("all" and "every"). The specifier type is also inherited by the noun phrases headed by the specifier in question. Noun phrases that are just symbolic terms and hence lack a specifier get specifier type `term`.

- `subordinated`: This feature of a sentential phrase takes values `yes` or `no` in order to indicate whether the sentential phrase is subordinated in such a way that it may not take references (see section 7.3.5).

- `typ`: This feature serves various purposes: It is used to distinguish various kinds of sentence type triggers (`assertion`, `ass` (assumption), `variable_declaration`, `ass_closing` (assumption closing), `case_closing`), it is used to distinguish between `existential` and `universal` natural language quantifiers, and to distinguish between different styles of itemization labels (see comment at the definition of `label/2` in the formal grammar below).

- `noun_type`: Nouns that denote collections (e.g. "set", "class" and "collection") can be used in special syntactic ways. In order to account for that, these are marked with a `noun_type` feature `collection`; for other nouns this feature takes the value `normal`.

- `alt_copulas`: For some prepositions, the copula in predicative usages of the preposition (e.g. "to be on ...") may be replaced by another verb without a shift in meaning (e.g. "to lie on ..."). Which verbs may in this way be used as alternatives to the copula is indicated in the lexical entry of a preposition using the `alt_copulas` feature: Its value is a list of l verbs other than the copula that may be used in its place.

- `comma`: Certain assertion triggers may optionally take a comma after them, while others may not. This is marked in their lexical entry using the `comma` feature, which can take the value `optional` or `no`.

- `named`: Such-that clauses may only modify noun phrases that have been named using some symbolic term. But in the case of a predicative noun phrase, it is enough if the subject noun phrase predicated by it is named. This feature is used to keep track of whether a such-that clause will be allowed according to these rules: It takes the value `yes` on named noun phrases, on noun phrases predicating named noun phrases and on verb phrases whose subject is a named noun phrase. Otherwise it taked the value `no`.

- `empty`: Special care has to be taken to ensure that the empty plural specifier is not postulated at spurious places (see section 7.3.6). This feature is used to take care of this. It can take the value `yes` or `no`.

Note that the grammatical rules in the formal textual grammar presented below have been extracted from rules containing semantic information. Hence there are sometimes separate rules that could – from a purely syntactic point of view – be easily unified, but which are separated because of different semantic behaviour.

The names of the predicates (i.e. non-terminal symbols) in the grammar are as in the actual code of the Naproche system. A significant number of the predicates go back to ACE's terminology. We consider some of the predicate names, especially those naming different kinds of sentential phrases, not to be well-chosen terminology. For example, note that the predicate `sentence` refers to what we have termed NP-VP-sentence in section 7.3. When the word

`sentence` appears in complex predicate names, it means any simple sentential phrase. In the comments to the formal grammar, I use the word "sentence" according to the terminology used in the thesis.

Here is the commented formal textual grammar:

```
assertion -->
    [trivial].

assertion -->
    necessary_references.

assertion -->
    trigger(typ~assertion),
    references,
    proposition_coord(mode~finite..subordinated~no),
    references.

assertion -->
    [contradiction],
    references.

definition -->
    [define],
    definiendum,
    iff,
    proposition_coord(mode~finite..subordinated~yes).

definition -->
    optional_definition_intro,
    [define],
    symbolic:definition_quantterm,
    copula(mode~to-infinitive),
    symbolic:term(_Type).

optional_definition_intro -->
    [for,defining],
    symbolic:quantterm,
    [at],
    symbolic:term(_Type),
    comma(comma~optional).

optional_definition_intro -->
    [].

definiendum -->
    symbolic:definition_quantterm.
```

```
definiendum -->
    optionally_specified_variable,
    copula(mode~to-infinitive..number~singular),
    indefinite_article(number~singular),
    noun(_,number~singular).

definiendum -->
    optionally_specified_variable,
    copula(mode~to-infinitive..number~singular),
    adjective(number~singular).

definiendum -->
    optionally_specified_variable,
    copula(mode~to-infinitive..number~singular),
    adjective_parser(adj_trans~T..number~singular),
    { \+ T = no },
    [T],
    optionally_specified_variable.

definiendum -->
    optionally_specified_variable,
    [and],
    optionally_specified_variable,
    copula(mode~to-infinitive),
    adjective_parser(adj_trans~T..number~plural),
    { \+ T = no }.

definiendum -->
    optionally_specified_variable,
    intransitive_verb(_,mode~to-infinitive..number~singular).

definiendum -->
    optionally_specified_variable,
    transitive_verb(mode~to-infinitive..number~singular),
    optionally_specified_variable.

%%  optionally_specified_variable
%
%   This predicate parses a variable optionally preceded by an indefinite
%   noun phrase.

optionally_specified_variable -->
    simple_indefinite_np,
    symbolic:variable.
```

```
optionally_specified_variable -->
    symbolic:variable.

simple_indefinite_np -->
    specifier(specifier_type~indefinite),
    simple_nbar.

simple_nbar -->
    noun(_,number~singular).

simple_nbar -->
    adjective(number~singular),
    simple_nbar.

assumption -->
    trigger(typ~variable_declaration),
    quantterm_list_bar(_).

assumption -->
    [let],
    quantterm_list_bar(_),
    [be],
    [given].

assumption -->
    trigger(typ~ass..mode~Mode),
    proposition_coord(mode~Mode..subordinated~yes).

%%  proposition_cord(?Features)
%
%   propoosition_coord parses the core of a sentence. This can be a complex
%   sentential phrase consisting of numerous simple sentential phrases. The
%   following is a detailed characterisation of proposition_coord in terms
%   of sentence_coord:
%   proposition_coord can be either a simple sentence_coord, or a number of
%   sentence_coords linked with "if...then" or "iff".

proposition_coord(mode~Mode..subordinated~S1) -->
    { var(S1); S1 = no; S2 = yes },
    sentence_coord(mode~Mode..subordinated~S2),
    proposition_coord_tail.

proposition_coord(mode~finite..subordinated~Sub) -->
    [if],
```

```
    sentence_coord(mode~finite..subordinated~yes),
    comma(comma~optional),
    [then],
    references,
    trigger(typ~conseq),
    references,
    proposition_coord(mode~finite..subordinated~Sub).

proposition_coord(mode~that..subordinated~Sub) -->
    [that],
    [if],
    sentence_coord(mode~finite..subordinated~yes),
    comma(comma~optional),
    [then],
    references,
    trigger(typ~conseq),
    references,
    proposition_coord(mode~finite..subordinated~Sub).

proposition_coord_tail -->
    comma(comma~optional),
    [if],
    sentence_coord(mode~finite..subordinated~yes).

proposition_coord_tail -->
    comma(comma~optional),
    iff,
    sentence_coord(mode~finite..subordinated~yes).

proposition_coord_tail -->
    [].

%%  sentence_coord(?Features)
%
%   sentence_coord links a number of topicalised_sentences with "and",
%   "or", ", and", ", or" and "i.e." in such a way that the bracketing
%   is unambiguous.

sentence_coord(mode~Mode..subordinated~S) -->
    sentence_coord_0(mode~Mode),
    sentence_coord_tail(mode~Mode..subordinated~S).

sentence_coord_tail(mode~Mode..subordinated~no) -->
    references,
    conseq_conjunct_marker,
```

```
    references,
    sentence_coord(mode~Mode).

sentence_coord_tail(mode~Mode..subordinated~no) -->
    references,
    comma(comma~optional),
    [and],
    necessary_references,
    sentence_coord(mode~Mode).

sentence_coord_tail(mode~Mode..subordinated~no) -->
    necessary_references,
    comma(comma~optional),
    [and],
    sentence_coord(mode~Mode).

sentence_coord_tail(_) -->
    [].

sentence_coord_0(mode~Mode) -->
    sentence_coord_1(mode~Mode),
    sentence_coord_0_tail(mode~Mode).

sentence_coord_0_tail(mode~Mode) -->
    [,],
    [or],
    sentence_coord_0(mode~Mode).

sentence_coord_0_tail(_) -->
    [].

sentence_coord_1(mode~Mode) -->
    sentence_coord_2(mode~Mode),
    sentence_coord_1_tail(mode~Mode).

sentence_coord_1_tail(mode~Mode) -->
    [,],
    trigger(typ~conjunction),
    sentence_coord_1(mode~Mode).

sentence_coord_1_tail(_) -->
    [].

sentence_coord_2(mode~Mode) -->
    sentence_coord_3(mode~Mode),
```

```
    sentence_coord_2_tail(mode~Mode).

sentence_coord_2_tail(mode~Mode) -->
    [or],
    sentence_coord_2(mode~Mode).

sentence_coord_2_tail(_) -->
    [].

sentence_coord_3(mode~that) -->
    [that],
    topicalised_sentence(mode~finite),
    sentence_coord_3_tail(mode~that).

sentence_coord_3(mode~Mode) -->
    topicalised_sentence(mode~Mode),
    sentence_coord_3_tail(mode~Mode).

sentence_coord_3_tail(mode~Mode) -->
    trigger(typ~conjunction_or_comma),
    sentence_coord_3(mode~Mode).

sentence_coord_3_tail(_) -->
    [].

%%  topicalised_sentence(?Features)
%
%   A topicalised_sentence can be a quantified sentential phrase, two
%   composite_sentences linked with "implies that", or just one
%   composite_sentence.

topicalised_sentence(mode~Mode) -->
    existential_topic(mode~Mode).

topicalised_sentence(mode~finite) -->
    universal_topic,
    comma(comma~optional),
    proposition_coord(mode~finite..subordinated~no).

topicalised_sentence(mode~finite) -->
    composite_sentence(mode~finite),
    comma(comma~optional),
    implies,
    composite_sentence(mode~finite).
```

```
topicalised_sentence(mode~Mode) -->
    composite_sentence(mode~Mode).

%%  existential_topic(?Features)
%
%   An existential_topic is a sentential phrase headed by a natural language
%   existential quantification.

existential_topic(Features) -->
    { Features = typ~existential },
    quantifier(Features),
    existential_np_coord(Features).

existential_topic(Features) -->
    { Features = typ~at_most_one },
    quantifier(Features),
    nbar(Features).

existential_topic(Features) -->
    { Features = typ~precisely_one },
    quantifier(Features),
    nbar(Features).

existential_np_coord(number~Number..typ~Typ) -->
    np(specifier_type~ST..number~Number1..typ~Typ),
    { ST = indefinite ; ST = negative },
    existential_np_coord_tail(number~NumberTail..typ~Typ),
    {
    NumberTail = null ->
        Number = Number1
        ;
        ( Number1 = singular , NumberTail = singular
          ;
          Number = plural
        )
    }.

existential_np_coord_tail(Features) -->
    [and],
    existential_np_coord(Features).

existential_np_coord_tail(number~null) -->
    [].

%%  universal_topic
```

```
%
%   A universal_topic is a sentential phrase headed by a natural language
%   universal quantification.

universal_topic -->
    [for],
    np(specifier_type~universal).

%%  composite_sentence(?Features)
%
%   A composite_sentence is either a simple sentential phrase  or a
%   proposition_coord prefixed with a sentence_init.

composite_sentence(mode~Mode) -->
    sentence_init(mode~Mode),
    proposition_coord(mode~that..subordinated~yes).

composite_sentence(mode~Mode) -->
    trigger(typ~formula..mode~Mode),
    symbolic:term(o).

composite_sentence(mode~Mode) -->
    sentence(mode~Mode).

composite_sentence(mode~Mode) -->
    metasentence(mode~Mode).

%%  sentence(?Features)
%
%   A sentence is what we termed NP-VP-sentence in the thesis: A noun
%   phrase followed by a verb phrase.

sentence(mode~Mode) -->
    np_coord(number~Number..named~Named,_),
    vp(mode~Mode..number~Number..named~Named).

%%  np_coord(?Features,-Connective)
%
%   np_coord parses a noun phrase of any complexity: It may coordinate
%   simple noun phrases (np) with "and" or "or". In a conjunction of
%   more than two simple noun phrases, all but the last "and" can also
%   be replaced by commas.

np_coord(number~plural..named~Named,and) -->
    simple_np_conjunction(named~Named).
```

```
np_coord(number~plural..named~Named,and) -->
    np(named~Named),
    comma_or_and(obligatory),
    np_coord(named~Named,and).

np_coord(number~plural..named~Named,and) -->
    np(named~Named),
    [and],
    np(named~Named).

np_coord(number~CoordNumber..named~Named,or) -->
    np(number~Number1..named~Named),
    [or],
    np_coord(number~Number2..named~Named,or),
%   In an NP disjunction, the grammatical number of the single disjuncts
%   affects the grammatical number of the disjunction.
%   If the disjuncts have the same number, the disjunction takes over this
%   number.
%   If the disjuncts have different numbers, the disjunction number is
%   "mixed".
%   Since verb agreement is not possible for NP disjunctions with number
%   "mixed", such NP disjunctions can only be used as objects or as subjects
%   of infinitive constructs.
    { Number1 = Number2 -> CoordNumber = Number1 ; CoordNumber = mixed }.

np_coord(number~CoordNumber..named~Named,or) -->
    np(number~Number1..named~Named),
    [or],
    np(number~Number2..named~Named),
    { Number1 = Number2 -> CoordNumber = Number1 ; CoordNumber = mixed }.

np_coord(Features,no) -->
    np(Features).

%%  simple_np_conjunction(?Features)
%
%   simple_np_conjunction parses a conjunction of noun phrases that are not
%   headed by a universal or negative specifier.

simple_np_conjunction(named~Named) -->
    np(specifier_type~SpecifierType..named~Named),
    {
    \+ SpecifierType = universal,
    \+ SpecifierType = negative
```

```
    },
    comma_or_and(obligatory),
    simple_np_conjunction(named~Named).

simple_np_conjunction(named~Named) -->
    np(specifier_type~SpecifierType1..named~Named),
    {
    \+ SpecifierType1 = universal,
    \+ SpecifierType1 = negative
    },
    [and],
    np(specifier_type~SpecifierType2..named~Named),
    {
    \+ SpecifierType2 = universal,
    \+ SpecifierType2 = negative
    }.

%%  np(?Features)
%
%   np parses a simple noun phrase, i.e. either a term or a specifier
%   followed by an nbar.

np(number~singular..specifier_type~term) -->
    symbolic:term(_Type).

np(Features) -->
    specifier(Features),
    {
    % Variables without "there are" or a noun in front of them
    % should never be read as plural nbars with empty indefinite
    % specifiers, but only as terms:
    ( Features = empty~no ; subsumes(typ~existential,Features) )
    ->
    true
    ;
    Features = noun~obligatory
    },
    nbar(Features).

%%  nbar(?Features)
%
%   An nbar is a noun, possibly preceeded by an adjective, and possibly
%   followed by a quantterm_list_bar. Alternitavely, it is just a
%   quantterm_list_bar possibly preceeded by an adjective.
```

```
nbar(Features) -->
    nbar1(Features).

nbar(Features) -->
    adjective(Features),
    nbar(Features).

nbar1(Features) -->
    {Features = noun_type~collection},
    noun(_,Features),
    quantterm_list_bar(Features),
    collection_complement.

nbar1(Features) -->
    {Features = noun_type~collection},
    noun(_,Features),
    {
    Features = named~Named,
    ( Named == yes -> true ; Named = no )
    },
    collection_complement,
    optional_ppst(Features).

nbar1(Features) -->
    noun(_,Features),
    quantterm_list_bar(Features).

nbar1(Features) -->
    noun(_,Features),
    {
    Features = named~Named,
    ( Named == yes -> true ; Named = no )
    },
    optional_ppst(Features).

nbar1(Features) -->
    { Features = noun~optional },
    quantterm_list_bar(Features).

collection_complement -->
    [of,objects,called],
    noun(_,number~plural).

collection_complement -->
    [of],
```

```
    nbar(number~plural).

%%  quantterm_list_bar(?Features)
%
%   A quantterm_list_bar is list of quantterms possibly followed by a
%   propositional phrase and/or a such_that_clause.

quantterm_list_bar(Features) -->
    {
    change_feature(Features,number,VariableListNumber,VariableListFeatures)
    },
    quantterm_list(VariableListFeatures),
    {
    \+ VariableListNumber = null,
    ( VariableListNumber = plural -> Features = number~plural; true )
    },
    optional_ppst(named~yes).

quantterm_list(number~Number) -->
    symbolic:dependent_quantterm,
    quantterm_rest_list(number~RestNumber),
    { RestNumber = null -> Number = singular; Number = plural }.

quantterm_list(number~Number) -->
    symbolic:quantterm,
    quantterm_rest_list(number~RestNumber),
    { RestNumber = null -> Number = singular; Number = plural }.

quantterm_rest_list(Features) -->
    comma_or_and(optional),
    quantterm_list(Features).

quantterm_rest_list(number~null) -->
    [].

%%  optional_ppst(?Features)
%
%   This predicate optionally parses a (possibly negated) prepositional
%   phrase, a such_that_clause or a prepositional phrase followed by a
%   such_that_clause.

optional_ppst(Features) -->
    optionally_negated_pp(Features),
    optional_such_that_clause(Features).
```

```
optional_ppst(Features) -->
    optional_such_that_clause(Features).

optionally_negated_pp(Features) -->
    [not],
    pp(Features).

optionally_negated_pp(Features) -->
    pp(Features).

%%  pp(?Fetures)
%
%   pp parses a prepositional phrase, i.e. a preposition followed by a
%   noun phrase.

pp(Features) -->
    preposition(Features),
    np_coord(_,_).

optional_such_that_clause(Features) -->
    { Features = named~yes },
    such_that_clause.

optional_such_that_clause(_) -->
    satisfying_clause.

optional_such_that_clause(_) -->
    [].

%%  such_that_clause
%
%   A such_that_clause is a subclause starting with "such that"
%   followed by a proposition_coord.

such_that_clause -->
    comma(comma~optional),
    [such],
    [that],
    proposition_coord(mode~finite..subordinated~yes).

%%  satisfying_clause
%
%   A satisfying_clause is a postposed adjectival phrase
%   consisting of "satisfying" followed by a following_np.
```

```
satisfying_clause -->
    satisfying,
    following_np(_).

satisfying -->
    [satisfying].

satisfying -->
    [not],
    [satisfying].

%%  vp(?Features)
%
%   vp parses a verb phrase.

vp(Features) -->
    negation(Features),
    {
    Features = number~Number,
    VBarFeatures = mode~infinitive..number~Number
    },
    vbar(VBarFeatures).

vp(Features) -->
    vbar(Features).

vp(Features) -->
    optionally_negated_copula(Features),
    {
    Features = named~Named,
    ( Named == no -> true ; Named = yes )
    },
    specifier(Features),
    nbar(Features),
    {
    Features = specifier_type~indefinite
    ;
    Features = specifier_type~definite
    ;
    Features = specifier_type~term
    }.

vp(Features) -->
    optionally_negated_copula(Features),
    adjective(Features).
```

```
vp(Features) -->
    optionally_negated_copula(Features),
    adjective_parser(Features),
    {
    Features = adj_trans~T,
    \+ T = no
    },
    [T],
    np_coord(_,_).

vp(Features) -->
    optionally_negated_verb(Verb,Features),
    pp(Features),
    {
    Features = alt_copulas~AltCopulas,
    member(Verb,[be|AltCopulas])
    }.

vp(Features) -->
    { Features = named~yes },
    optionally_negated_copula(Features),
    such_that_clause.

vbar(Features) -->
    transitive_verb(Features),
    np_coord(_,_).

vbar(Features) -->
    intransitive_verb(_,Features).

negation(Features) -->
    { Features = mode~finite },
    intransitive_verb(do,Features),
    [not].

negation(Features) -->
    { \+ Features = mode~finite },
    [not].

optionally_negated_copula(Features) -->
    negated_copula(Features).

optionally_negated_copula(Features) -->
    copula(Features).
```

```
negated_copula(Features) -->
    { Feature = mode~finite },
    copula(Features),
    [not].

negated_copula(Features) -->
    { \+ Feature = mode~finite },
    [not],
    copula(Features).

optionally_negated_verb(be,Features) -->
    optionally_negated_copula(Features).

optionally_negated_verb(Verb,Features) -->
    negation(Features),
    {
    Features = number~Number,
    VerbFeatures = mode~infinitive..number~Number
    },
    intransitive_verb(Verb,VerbFeatures).

optionally_negated_verb(Verb,Features) -->
    intransitive_verb(Verb,Features).

%%  label(-Type,?NoType)
%
%   label parses sentences of the form "i)", "ii)", "A)" etc. Given the
%   way these are pre-tokenized, we can say that label always parses
%   a string of single-character words, of which the last must be ")"
%   (and no previous single-character word may be "(").
%
%   The Type argument can take the values "capital" (for labels of the
%   form A), B), C) etc.), "latin" (for a), b), c) etc.), "number" (for
%   1), 2), 3) etc.), "roman" (for i), ii), iii) etc.) and "unknown"
%   (for any other string followed by ")"). The NoType argument may indicate
%   which value the Type argument may not take. If all values are allowed
%   for Type, then NoType is "nothing".

label(Type,NoType) -->
    alnum(Identifier),
    [')'],
    {
    dcg_lexicon([Identifier],enumeration,typ~Type),
    \+ NoType = Type
```

```
    }.

alnum(Out) -->
    [Char],
    { atom(Char),
      \+ Char = '(' },
    alnum(TmpOut),
    { atom_concat(Char, TmpOut, Out) }.

alnum(Char) -->
    [Char],
    { atom(Char),
      \+ Char = '(' }.

%%  metasentence(?Features)
%
%   metasentence parses a simple sentential phrase that either announces
%   a list of statements starting in the following sentence (using
%   following_np) or talks about previously introduced statements. (For
%   the second case, only talk about previously introduced cases of a
%   case distinction is implemented.)

metasentence(mode~Mode) -->
    following_np(number~Number),
    meta_vp(mode~Mode..number~Number).

metasentence(mode~Mode) -->
    meta_np(_,number~Number),
    meta_vp(mode~Mode..number~Number).

%%  following_np(?Features)
%
%   following_np parses a noun phrase that contains the word "following"
%   followed by "axiom", "property" or "case" (possibly in plural form),
%   and is used to announce a list of statements starting in the following
%   sentence.

following_np(number~singular) -->
    [precisely,one,of,the,following],
    noun(Noun,number~plural),
    { Noun = axiom ; Noun = property ; Noun = case }.

following_np(number~singular) -->
    [at,most,one,of,the,following],
    noun(Noun,number~plural),
```

```
    { Noun = axiom ; Noun = property ; Noun = case }.

following_np(Features) -->
    [the,following],
    noun(Noun,Features),
    { Noun = axiom ; Noun = property ; Noun = case }.

meta_np(and,number~plural) -->
    [case,_Identifier],
    comma_or_and(obligatory),
    meta_np(and,_).

meta_np(at_most_one,number~singular) -->
    [at,most,one,of],
    [case,_Identifier],
    comma_or_and(obligatory),
    meta_np(and,_).

meta_np(xor,number~singular) -->
    [precisely,one,of],
    [case,_Identifier],
    comma_or_and(obligatory),
    meta_np(and,_).

meta_np(or,number~singular) -->
    [case,_Identifier],
    [or],
    meta_np(or,number~singular).

meta_np(or,number~singular) -->
    [case,_Identifier1],
    [or],
    [case,_Identifier2].

meta_np(and,number~singular) -->
    [case,_Identifier].

meta_vp(Features) -->
    copula(Features),
    [correct].

meta_vp(Features) -->
    copula(Features),
    [true].
```

```
meta_vp(Features) -->
    intransitive_verb(hold,Features).

meta_vp(Features) -->
    copula(Features),
    [incorrect].

meta_vp(Features) -->
    copula(Features),
    [false].

meta_vp(Features) -->
    negated_copula(Features),
    [correct].

meta_vp(Features) -->
    negated_copula(Features),
    [true].

meta_vp(mode~Mode..number~Number) -->
    negation(mode~Mode..number~Number),
    intransitive_verb(hold,mode~infinitive..number~Number).

meta_vp(Features) -->
    copula(Features),
    [inconsistent].

% LEXICAL ITEMS

noun(Noun,Features) -->
    {
    dcg_lexicon(DeclinedNoun,noun,Features,Noun)
    },
    DeclinedNoun.

%%  adjective(?Features)
%
%   adjective parses adjectives in positions where no propositional
%   complement to transitive adjectives is possible. Hence transitive
%   adjectives may only be parsed by adjective if the number feature is
%   "plural".

adjective(Features) -->
    [Adjective],
    {
```

```
    Features = adj_trans~no,
    dcg_lexicon([Adjective],adjective,Features)
    }.

adjective(number~plural) -->
    [Adjective],
    {
    dcg_lexicon([Adjective],adjective,adj_trans~T),
    \+ T = no
    }.

%%  adjective_parser(?Features)
%
%   adjective_parser parses any adjective, i.e. also transitive adjectives
%   that have to be followed by a propositional complement.

adjective_parser(Features) -->
    [Adjective],
    {
    dcg_lexicon([Adjective],adjective,Features)
    }.

transitive_verb(Features) -->
    {
    Features = transitive~plus,
    dcg_lexicon(ConjugatedVerb,verb,Features,_)
    },
    ConjugatedVerb.

intransitive_verb(Verb,Features) -->
    {
    Features = transitive~minus,
    dcg_lexicon(ConjugatedVerb,verb,Features,Verb)
    },
    ConjugatedVerb.

copula(Features) -->
    {
    Features = transitive~copula,
    dcg_lexicon(ConjugatedVerb,verb,Features,_)
    },
    ConjugatedVerb.

specifier(Features) -->
    { dcg_lexicon(Specifier,specifier,Features) },
```

```
    Specifier.

number -->
    [Number],
    { dcg_lexicon([Number],number) }.

preposition(AltCopulas) -->
    { dcg_lexicon(Preposition,preposition,AltCopulas,_,_) },
    Preposition.

quantifier(Features) -->
    { dcg_lexicon(Quantifier,quantifier,Features) },
    Quantifier.

trigger(Features) -->
    { dcg_lexicon(Trigger,trigger,Features) },
    Trigger,
    comma(Features).

sentence_init(mode~Mode) -->
    { dcg_lexicon(SentenceInit,sentence_init,mode~Mode) },
    SentenceInit.

comma(comma~optional) -->
    [','].

comma(_) -->
    [].

iff -->
    [iff].

iff -->
    [if],
    [and],
    [only],
    [if].

implies -->
    [implies],
    [that].

implies -->
    [implies].
```

```
indefinite_article(number∼singular) -->
    [a].

indefinite_article(number∼singular) -->
    [an].

indefinite_article(number∼plural) -->
    [].

comma_or_and(_) -->
    [’,’].

comma_or_and(_) -->
    [and].

comma_or_and(optional) -->
    [].

comma_or_or -->
    [’,’].

comma_or_or -->
    [or].

conseq_conjunct_marker -->
    [’,’],
    trigger(typ∼ie).

conseq_conjunct_marker -->
    comma(comma∼optional),
    [and],
    trigger(typ∼conseq_conjunct).

references -->
    necessary_references.

references -->
    [].

necessary_references -->
    [by],
    reference_list,
    comma(comma∼optional).

reference_list -->
```

```
    reference,
    comma_or_and(obligatory),
    reference_list.

reference_list -->
    reference.

reference -->
    [axiom],
    [_Identifier].

reference -->
    [theorem],
    [_Identifier].

reference -->
    [lemma],
    [_Identifier].

reference -->
    [definition],
    [_Identifier].

reference -->
    [induction].

% SPECIAL SENTENCE TYPES

closing -->
    trigger(typ~ass_closing),
    references,
    proposition_coord(mode~finite..subordinated~no),
    references.

theorem_heading -->
    { Type = theorem ; Type = lemma },
    heading(Type).

heading(Type) -->
    [Type].

heading(Type) -->
    [Type],
    [_Identifier].
```

```
proof -->
    [proof].

proof_end -->
    [qed].

proof_end -->
    ['End_proof'].

end(Type) -->
    { atom_concat('End_',Type,End_Type) },
    [End_Type].

case_id -->
    [case],
    [_Identifier].

case_intro -->
    trigger(typ~assertion),
    [there,are],
    number,
    [cases].

case_closing -->
    trigger(typ~assertion),
    trigger(typ~case_closing),
    proposition_coord(mode~finite..subordinated~no).

case -->
    proposition_coord(mode~finite..subordinated~yes).

new_paragraph -->
    [##].

% VARIABLE TYPE FIXING

%%  var_type_fix
%
%   var_type_fix parses a sentence that links certain variable symbol
%   collections to certain predicates (named 'types' here).

var_type_fix -->
    symbol_collections,
    infix_var_type_fix,
    noun(_,number~plural).
```

```
symbol_collections -->
    symbol_collection,
    comma_or_and(obligatory),
    symbol_collections.

symbol_collections -->
    symbol_collection,
    [and],
    symbol_collection.

symbol_collections -->
    symbol_collection.

symbol_collection -->
    optional_capitalization,
    alphabet,
    [letters].

optional_capitalization -->
    [small].

optional_capitalization -->
    [capital].

optional_capitalization -->
    [].

alphabet -->
    [latin].

alphabet -->
    [greek].

alphabet -->
    [fraktur].

alphabet -->
    [german].

infix_var_type_fix -->
    [always],
    [denote].

infix_var_type_fix -->
```

```
    [will],
    [always],
    [denote].

infix_var_type_fix -->
    [will],
    [be],
    [used],
    [throughout],
    [to],
    [denote].

infix_var_type_fix -->
    [will],
    [stand],
    [throughout],
    [for].

% ALTERNATIVE NOTATION

%%  alternative_notation
%
%   alternative_notation parses a sentence that announces the possibility
%   of concatenative notation for a binary predicate that was introdced
%   with infix notation.

alternative_notation -->
    [instead],
    [of],
    [math([X,_Infix,Y])],
    [we],
    [also],
    [write],
    [math([X,Y])].


%=========
% LEXICON
%=========

%----------------------------
% Nouns, pronouns and numbers
%----------------------------

% Simple nouns
dcg_lexicon([set],noun,number~singular..noun_type~collection,set).
```

```
dcg_lexicon([sets],noun,number~plural..noun_type~collection,set).
dcg_lexicon([class],noun,number~singular..noun_type~collection,class).
dcg_lexicon([classes],noun,number~plural..noun_type~collection,class).
dcg_lexicon([collection],noun,number~singular..noun_type~collection,collection).
dcg_lexicon([collection],noun,number~plural..noun_type~collection,collection).
dcg_lexicon([element],noun,number~singular..noun_type~normal,element).
dcg_lexicon([elements],noun,number~plural..noun_type~normal,element).
dcg_lexicon([number],noun,number~singular..noun_type~normal,number).
dcg_lexicon([numbers],noun,number~plural..noun_type~normal,number).
dcg_lexicon([integer],noun,number~singular..noun_type~normal,integer).
dcg_lexicon([integers],noun,number~plural..noun_type~normal,integer).
dcg_lexicon([real],noun,number~singular..noun_type~normal,real).
dcg_lexicon([reals],noun,number~plural..noun_type~normal,real).
dcg_lexicon([ordinal],noun,number~singular..noun_type~normal,ordinal).
dcg_lexicon([ordinals],noun,number~plural..noun_type~normal,ordinal).
dcg_lexicon([point],noun,number~singular..noun_type~normal,point).
dcg_lexicon([points],noun,number~plural..noun_type~normal,point).
dcg_lexicon([line],noun,number~singular..noun_type~normal,line).
dcg_lexicon([lines],noun,number~plural..noun_type~normal,line).
dcg_lexicon([circle],noun,number~singular..noun_type~normal,circle).
dcg_lexicon([circles],noun,number~plural..noun_type~normal,circle).
dcg_lexicon([segment],noun,number~singular..noun_type~normal,segment).
dcg_lexicon([segments],noun,number~plural..noun_type~normal,segments).
dcg_lexicon([angle],noun,number~singular..noun_type~normal,angle).
dcg_lexicon([angles],noun,number~plural..noun_type~normal,angle).
dcg_lexicon([area],noun,number~singular..noun_type~normal,area).
dcg_lexicon([areas],noun,number~plural..noun_type~normal,area).
dcg_lexicon([triangle],noun,number~singular..noun_type~normal,triangle).
dcg_lexicon([triangles],noun,number~plural..noun_type~normal,triangle).
dcg_lexicon([axiom],noun,number~singular..noun_type~normal,axiom).
dcg_lexicon([axioms],noun,number~plural..noun_type~normal,axiom).
dcg_lexicon([property],noun,number~singular..noun_type~normal,property).
dcg_lexicon([properties],noun,number~plural..noun_type~normal,property).
dcg_lexicon([case],noun,number~singular..noun_type~normal,case).
dcg_lexicon([cases],noun,number~plural..noun_type~normal,case).

% Type nouns
dcg_lexicon([function],noun,number~singular..noun_type~normal,'').
dcg_lexicon([functions],noun,number~plural..noun_type~normal,'').
dcg_lexicon([relation],noun,number~singular..noun_type~normal,'').
dcg_lexicon([relations],noun,number~plural..noun_type~normal,'').
dcg_lexicon([object],noun,number~singular..noun_type~normal,'').
dcg_lexicon([objects],noun,number~plural..noun_type~normal,'').

% Complex nouns
```

```
dcg_lexicon([natural,number],noun,
            number~singular..noun_type~normal,natural_number).
dcg_lexicon([natural,numbers],noun,
            number~plural..noun_type~normal,natural_number).

% Pronouns
dcg_lexicon([it],pronoun,number~singular).

% Numbers
dcg_lexicon([two],number).
dcg_lexicon([three],number).
dcg_lexicon([four],number).
dcg_lexicon([five],number).
dcg_lexicon([six],number).
dcg_lexicon([seven],number).
dcg_lexicon([eight],number).
dcg_lexicon([nine],number).
dcg_lexicon([ten],number).


%-----------------------------------
% Adjectives, verbs and prepositions
%-----------------------------------

dcg_lexicon([empty],adjective,adj_trans~no).
dcg_lexicon([even],adjective,adj_trans~no).
dcg_lexicon([natural],adjective,adj_trans~no).
dcg_lexicon([odd],adjective,adj_trans~no).
dcg_lexicon([prime],adjective,adj_trans~no).
dcg_lexicon([compound],adjective,adj_trans~no).
dcg_lexicon([composite],adjective,adj_trans~no).
dcg_lexicon([positive],adjective,adj_trans~no).
dcg_lexicon([transitive],adjective,adj_trans~no).
dcg_lexicon([square],adjective,adj_trans~no).
dcg_lexicon([rational],adjective,adj_trans~no).
dcg_lexicon([irrational],adjective,adj_trans~no).
dcg_lexicon([finite],adjective,adj_trans~no).
dcg_lexicon([infinite],adjective,adj_trans~no).
dcg_lexicon([nonzero],adjective,adj_trans~no).
dcg_lexicon([trivial],adjective,adj_trans~no).
dcg_lexicon([nontrivial],adjective,adj_trans~no).

dcg_lexicon([distinct],adjective,adj_trans~from).
dcg_lexicon([disjoint],adjective,adj_trans~from).
dcg_lexicon([parallel],adjective,adj_trans~to).
```

```
dcg_lexicon([coprime],adjective,adj_trans~to).

% Type adjectives
dcg_lexicon([unary],adjective,adj_trans~no).
dcg_lexicon([binary],adjective,adj_trans~no).
dcg_lexicon([ternary],adjective,adj_trans~no).


% Verbs

dcg_lexicon([succeeds],verb,transitive~minus..mode~finite..number~singular,
            succeed).
dcg_lexicon([to,succeed],verb,transitive~minus..mode~to-infinitive,succeed).
dcg_lexicon([succeed],verb,Features,succeed) :-
        Features = transitive~minus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([holds],verb,transitive~minus..mode~finite..number~singular,hold).
dcg_lexicon([to,hold],verb,transitive~minus..mode~to-infinitive,hold).
dcg_lexicon([hold],verb,Features,hold) :-
        Features = transitive~minus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([does],verb,transitive~minus..mode~finite..number~singular,do).
dcg_lexicon([to,do],verb,transitive~minus..mode~to-infinitive,do).
dcg_lexicon([do],verb,Features,do) :-
        Features = transitive~minus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([goes],verb,transitive~minus..mode~finite..number~singular,go).
dcg_lexicon([to,go],verb,transitive~minus..mode~to-infinitive,go).
dcg_lexicon([go],verb,Features,go) :-
        Features = transitive~minus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([lies],verb,transitive~minus..mode~finite..number~singular,lie).
dcg_lexicon([to,lie],verb,transitive~minus..mode~to-infinitive,lie).
dcg_lexicon([lie],verb,Features,lie) :-
        Features = transitive~minus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).

dcg_lexicon([contains],verb,transitive~plus..mode~finite..number~singular,
            contain).
dcg_lexicon([to,contain],verb,transitive~plus..mode~to-infinitive,contain).
dcg_lexicon([contain],verb,Features,contain) :-
        Features = transitive~plus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([belongs,to],verb,transitive~plus..mode~finite..number~singular,
            '\\in').
```

```
dcg_lexicon([to,belong,to],verb,transitive~plus..mode~to-infinitive,'\\in').
dcg_lexicon([belong,to],verb,Features,'\\in') :-
        Features = transitive~plus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([divides],verb,transitive~plus..mode~finite..number~singular,
            divide).
dcg_lexicon([to,divide],verb,transitive~plus..mode~to-infinitive,divide).
dcg_lexicon([divide],verb,Features,divide) :-
        Features = transitive~plus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).
dcg_lexicon([intersects],verb,transitive~plus..mode~finite..number~singular,
            intersect).
dcg_lexicon([to,intersect],verb,transitive~plus..mode~to-infinitive,intersect).
dcg_lexicon([intersect],verb,Features,intersect) :-
        Features = transitive~plus,
        ( Features = mode~infinitive; Features = number~plural..mode~finite ).

dcg_lexicon([is],verb,transitive~copula..mode~finite..number~singular,be).
dcg_lexicon([are],verb,transitive~copula..mode~finite..number~plural,be).
dcg_lexicon([be],verb,transitive~copula..mode~infinitive,be).
dcg_lexicon([to,be],verb,transitive~copula..mode~to-infinitive,be).

% "to be the center of" as a predicate that does not trigger presuppositions:
dcg_lexicon([is, the, center, of],verb,
            transitive~plus..mode~finite..number~singular,center).
dcg_lexicon([to, be, the, center, of],verb,
            transitive~plus..mode~to-infinitive,center).
dcg_lexicon([are, centers, of],verb,
            transitive~plus..mode~finite..number~plural,center).
dcg_lexicon([be, the, center, of],verb,
            transitive~plus..mode~infinitive,center).


dcg_lexicon([in],preposition,alt_copulas~[],grouped~no,'\\in').
dcg_lexicon([on,the,same,side,of],preposition,alt_copulas~[lie],grouped~1,
            on_the_same_side_of).
dcg_lexicon([on],preposition,alt_copulas~[lie],grouped~no,on).
dcg_lexicon([inside],preposition,alt_copulas~[lie],grouped~no,inside).
dcg_lexicon([through],preposition,alt_copulas~[go],grouped~no,through).
dcg_lexicon([between],preposition,alt_copulas~[lie],grouped~2,between).


%---------------------------------------------
% Specifiers, quantifiers and sentence initials
%---------------------------------------------
```

```
dcg_lexicon([no],specifier,specifier_type~negative..empty~no).

dcg_lexicon([every],specifier,
            specifier_type~universal..number~singular..empty~no).
dcg_lexicon([all],specifier,
            specifier_type~universal..number~plural..empty~no).

dcg_lexicon([precisely,one],specifier,
            specifier_type~one..number~singular..empty~no).

dcg_lexicon([the],specifier,
            specifier_type~definite..empty~no..number~singular).

dcg_lexicon([a],specifier,
            specifier_type~indefinite..number~singular..empty~no).
dcg_lexicon([an],specifier,
            specifier_type~indefinite..number~singular..empty~no).
dcg_lexicon([some],specifier,specifier_type~indefinite..empty~no).
dcg_lexicon([],specifier,
            specifier_type~indefinite..number~plural..empty~yes).


dcg_lexicon([for,every],quantifier,typ~universal..number~singular).
dcg_lexicon([for,all],quantifier,typ~universal..number~plural).

dcg_lexicon([there,is],quantifier,
            typ~existential..mode~finite..number~singular).
dcg_lexicon([there,are],quantifier,
            typ~existential..mode~finite..number~plural).
dcg_lexicon([there,exists],quantifier,
            typ~existential..mode~finite..number~singular).
dcg_lexicon([there,exist],quantifier,
            typ~existential..mode~finite..number~plural).
dcg_lexicon([there,be],quantifier,typ~existential..mode~infinitive).
dcg_lexicon([there,exist],quantifier,typ~existential..mode~infinitive).
dcg_lexicon([there,to,be],quantifier,typ~existential..mode~to-infinitive).
dcg_lexicon([there,to,exist],quantifier,typ~existential..mode~to-infinitive).

dcg_lexicon([there,is,at,most,one],quantifier,
            typ~at_most_one..mode~finite..number~singular).
dcg_lexicon([there,be,at,most,one],quantifier,
            typ~at_most_one..mode~infinite..number~singular).
dcg_lexicon([there,to,be,at,most,one],quantifier,
            typ~at_most_one..mode~to-infinite..number~singular).
```

```
dcg_lexicon([there,is,precisely,one],quantifier,
            typ~precisely_one..mode~finite..number~singular).
dcg_lexicon([there,be,precisely,one],quantifier,
            typ~precisely_one..mode~infinite..number~singular).
dcg_lexicon([there,to,be,precisely,one],quantifier,
            typ~precisely_one..mode~to-infinite..number~singular).

dcg_lexicon([it,is,false],sentence_init,typ~negative..mode~finite).
dcg_lexicon([it,be,false],sentence_init,typ~negative..mode~infinitive).
dcg_lexicon([it,to,be,false],sentence_init,
            typ~negative..mode~to-infinitive).
dcg_lexicon([it,is,not,the,case],sentence_init,
            typ~negative..mode~finite).
dcg_lexicon([it,not,be,the,case],sentence_init,
            typ~negative..mode~infinitive).
dcg_lexicon([it,not,to,be,the,case],sentence_init,
            typ~negative..mode~to-infinitive).
dcg_lexicon([it,is,the,case],sentence_init,
            typ~affirmative..mode~finite).
dcg_lexicon([it,be,the,case],sentence_init,
            typ~affirmative..mode~infinitive).
dcg_lexicon([it,to,be,the,case],sentence_init,
            typ~affirmative..mode~to-infinitive).


%---------
% Triggers
%---------

dcg_lexicon([then],trigger,typ~assertion..comma~no).

dcg_lexicon(X,trigger,typ~assertion..comma~Comma):-
        ( Type = all; Type = ie; Type = conjunction; Type = conseq ),
        dcg_lexicon(X,trigger,typ~Type..comma~Comma).

dcg_lexicon(X,trigger,typ~conseq_conjunct..comma~Comma):-
        ( Type = all; Type = conseq ),
        dcg_lexicon(X,trigger,typ~Type..comma~Comma),
        \+ X = [].

dcg_lexicon([hence],trigger,typ~all..comma~optional).
dcg_lexicon([therefore],trigger,typ~all..comma~optional).
dcg_lexicon([recall,that],trigger,typ~all..comma~no).
dcg_lexicon([now,recall,that],trigger,typ~all..comma~no).
dcg_lexicon([now,observe,that],trigger,typ~all..comma~no).
```

```
dcg_lexicon([now],trigger,typ~all..comma~optional).
dcg_lexicon([now,this,implies,that],trigger,typ~all..comma~no).
dcg_lexicon([now,this,in,turn,implies,that],trigger,typ~all..comma~no).
dcg_lexicon([now,this,implies],trigger,typ~all..comma~no).
dcg_lexicon([now,this,in,turn,implies],trigger,typ~all..comma~no).
dcg_lexicon([so],trigger,typ~all..comma~no).

dcg_lexicon([clearly],trigger,typ~conseq..comma~optional).
dcg_lexicon([trivially],trigger,typ~conseq..comma~optional).
dcg_lexicon([obviously],trigger,typ~conseq..comma~optional).
dcg_lexicon([in,particular],trigger,typ~conseq..comma~optional).
dcg_lexicon([observe,that],trigger,typ~conseq..comma~no).
dcg_lexicon([furthermore],trigger,typ~conseq..comma~optional).
dcg_lexicon([this,implies,that],trigger,typ~conseq..comma~no).
dcg_lexicon([this,in,turn,implies,that],trigger,typ~conseq..comma~no).
dcg_lexicon([this,implies],trigger,typ~conseq..comma~no).
dcg_lexicon([this,in,turn,implies],trigger,typ~conseq..comma~no).
dcg_lexicon([finally],trigger,typ~conseq..comma~optional).
dcg_lexicon([also],trigger,typ~conseq..comma~optional).
dcg_lexicon([],trigger,typ~conseq..comma~no).

dcg_lexicon(['i.e.'],trigger,typ~ie..comma~no).
dcg_lexicon([so],trigger,typ~ie..comma~no).

dcg_lexicon([and],trigger,typ~conjunction..comma~no).
dcg_lexicon([but],trigger,typ~conjunction..comma~no).

dcg_lexicon([','],trigger,typ~conjunction_or_comma..comma~no).
dcg_lexicon(X,trigger,typ~conjunction_or_comma..comma~Comma) :-
        dcg_lexicon(X,trigger,typ~conjunction..comma~Comma).


dcg_lexicon([we,have],trigger,typ~formula..mode~Mode..comma~no) :-
        \+ Mode = infinitive.
dcg_lexicon([we,get],trigger,typ~formula..mode~finite..comma~no).
dcg_lexicon([],trigger,typ~formula..comma~no).

dcg_lexicon([assume],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([suppose],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([assume,that],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([suppose,that],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([assume,for,a,contradiction,that],
            trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([now,assume],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([now,suppose],trigger,typ~ass..mode~finite..comma~no).
```

```
dcg_lexicon([now,assume,that],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([now,suppose,that],trigger,typ~ass..mode~finite..comma~no).
dcg_lexicon([now,assume,for,a,contradiction,that],
            trigger,typ~ass..mode~finite..comma~no).

dcg_lexicon([let],trigger,typ~ass..mode~infinitive..comma~no).
dcg_lexicon([now,let],trigger,typ~ass..mode~infinitive..comma~no).

dcg_lexicon([consider],trigger,typ~ass..mode~to-infinitive..comma~no).
dcg_lexicon([now,consider],trigger,typ~ass..mode~to-infinitive..comma~no).

dcg_lexicon([consider],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([consider,arbitrary],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([fix],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([fix,arbitrary],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([now,consider],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([now,consider,arbitrary],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([now,fix],trigger,typ~variable_declaration..comma~no).
dcg_lexicon([now,fix,arbitrary],trigger,typ~variable_declaration..comma~no).


dcg_lexicon([thus],trigger,typ~ass_closing..comma~no).

dcg_lexicon([in,all,cases],trigger,typ~case_closing..comma~optional).
dcg_lexicon([in,both,cases],trigger,typ~case_closing..comma~optional).

% enumerations
dcg_lexicon(['A'],enumeration,typ~capital).
dcg_lexicon(['B'],enumeration,typ~capital).
dcg_lexicon(['C'],enumeration,typ~capital).
dcg_lexicon(['D'],enumeration,typ~capital).
dcg_lexicon(['E'],enumeration,typ~capital).
dcg_lexicon(['a'],enumeration,typ~latin).
dcg_lexicon(['b'],enumeration,typ~latin).
dcg_lexicon(['c'],enumeration,typ~latin).
dcg_lexicon(['d'],enumeration,typ~latin).
dcg_lexicon(['e'],enumeration,typ~latin).
dcg_lexicon(['1'],enumeration,typ~number).
dcg_lexicon(['2'],enumeration,typ~number).
dcg_lexicon(['3'],enumeration,typ~number).
dcg_lexicon(['4'],enumeration,typ~number).
dcg_lexicon(['5'],enumeration,typ~number).
dcg_lexicon(['i'],enumeration,typ~roman).
dcg_lexicon(['ii'],enumeration,typ~roman).
dcg_lexicon(['iii'],enumeration,typ~roman).
```

```
dcg_lexicon(['iv'],enumeration,typ~roman).
dcg_lexicon(['v'],enumeration,typ~roman).
dcg_lexicon([_],enumeration,typ~unknown).
```

A.3 Quantterm grammar

The formal quantterm grammar presented below also shows how the notational types of its components and the name of a circumfix function are determined. This information is important for the `better_reading` algorithm defined below.

```
%% quantterm(+Accessibles,-NotationalType)

quantterm(Acc,[classical|NT]) -->
    quantterm(Acc,NT),
    ['('],
    variable_list(Acc),
    [')'].

quantterm(Acc,NT) -->
    variable(Acc),
    quantterm(Acc,[infix|NT]),
    variable(Acc).

quantterm(Acc,NT) -->
    quantterm(Acc,[prefix|NT]),
    variable(Acc).

quantterm(Acc,NT) -->
    variable(Acc),
    quantterm(Acc,[suffix|NT]).

quantterm(Acc,NT) -->
    new_circumfix_term(Acc,Name,[circumfix|NT]).

quantterm(_,_) -->
    new_variable.

new_variable(Tree) -->
    [Var],
    {
    atom(Var),
    Var \= '(',
    Var \= ')',
    Var \= ',',
    Var \= '^',
```

```
    Var \= '{',
    Var \= '}',
    Var \= '\mapsto'
    }.

new_circumfix_term(Acc,[S|NameTail],_) -->
    [S],
    circumfix_term_tail(Acc,NameTail).

circumfix_term_tail(Acc,Name) -->
    new_circumfix_term(Acc,Name,_).

circumfix_term_tail(Acc,[[arg]|Name]) -->
    variable(Acc,VarTree),
    new_circumfix_term(Acc,Name,_).

circumfix_term_tail(_,[]) -->
    [].

variable(Acc) -->
    [Var],
    {
    member(Var,Acc)
    }.

variable_list(Acc) -->
    variable(Acc),
    [','],
    variable_list(Acc).

variable_list(Acc) -->
    variable(Acc).
```

As explained in section 7.4.6, a quantterm often has multiple readings, and there is an algorithm for determining which one is the best reading. What follows is the Prolog code that formally defines this algorithm, which was only sketched in section 7.4.6. The quantterm readings are presented as feature structures with the features `nt` (notational type), `name` (name of a circumfix function) and `head` (the head of a quantterm consisting of a function applied to some arguments is just that function without its arguments).

```
%%  better_reading(+Reading1,+Reading2,-BetterReading)
%
%   BetterReading is instantiated to either Reading1 or Reading2 depending
%   on which one of them is to be preferred.

better_reading(Reading1,Reading2,Reading1) :-
```

```
    Reading1 = nt~[NT|_],
    \+ NT == circumfix,
    Reading2 = nt~[Circumfix|_],
    Circumfix == circumfix.

better_reading(Reading1,Reading2,Reading2) :-
    Reading2 = nt~[NT|_],
    \+ NT == circumfix,
    Reading1 = nt~[Circumfix|_],
    Circumfix == circumfix.

% Two circumfix readings are compared by comparing their names:
better_reading(Reading1,Reading2,BetterReading) :-
    Reading1 = nt~[Circumfix1|_]..name~Name1,
    Circumfix1 == circumfix,
    Reading2 = nt~[Circumfix2|_]..name~Name2,
    Circumfix2 == circumfix,
    ( name_better(Name1,Name2) ->
        BetterReading = Reading1
        ;
        ( name_better(Name2,Name1) ->
            BetterReading = Reading2
        )
    ).

% infix readings are always prefered over prefix and suffix readings:
better_reading(Reading1,Reading2,Reading1) :-
    Reading1 = nt~[Infix|_],
    Infix == infix,
    Reading2 = nt~[PrefixOrSuffix|_],
    ( PrefixOrSuffix == prefix ; PrefixOrSuffix == suffix ).

better_reading(Reading1,Reading2,Reading2) :-
    Reading2 = nt~[Infix|_],
    Infix == infix,
    Reading1 = nt~[PrefixOrSuffix|_],
    ( PrefixOrSuffix == prefix ; PrefixOrSuffix == suffix ).

% The following rule identifies the prefered reading of a quantterm of the
% form f'(x): Here we want ' to be a suffix function making f' classical
% rather than ' being a  classical function making '(x) a suffix. This is
% generalised to the rule that a reading that had "classical" in the second
% position of the notational type list is prefered over one that doesn't.
better_reading(Reading1,Reading2,Reading1) :-
    Reading1 = nt~[_,NT1|_],
```

```
    NT1 == classical,
    Reading2 = nt~[_,NT2|_],
    \+ var(NT2),
    NT2 \= classical.

better_reading(Reading1,Reading2,Reading2) :-
    Reading2 = nt~[_,NT1|_],
    NT1 == classical,
    Reading1 = nt~[_,NT2|_],
    \+ var(NT2),
    NT2 \= classical.

% When none of the above rules decides which reading is better, we
% recursively check which head function is prefered by those rules.
better_reading(Reading1,Reading2,BetterReading) :-
    Reading1 = head~Head1,
    \+ var(Head1),
    Reading2 = head~Head2,
    \+ var(Head2),
    better_reading(Head1,Head2,BetterHead),
    ( BetterHead = Head1 ->
        BetterReading = Reading1
        ;
        BetterReading = Reading2
    ).

% Finally, if none of the above rules decides which reading is better,
% we call Reading1 better. (The only known case where this is needed is
% when an infix reading is the best reading, but a prefix and a suffix
% reading are compared before comparing any of the two to the infix
% reading. No matter which of the two readings is called better, it will
% in the end be worse than the infix reading.)
better_reading(Reading1,_,Reading1).

% A circumfix name with an [arg] in the position where another circumfix
% name has a symbol is prefered:
name_better(Name1,Name2) :-
    make_args_to_vars(Name1,Name1WithVars),
    make_args_to_vars(Name2,Name2WithVars),
    \+ Name1WithVars == Name2WithVars,
    subsumes(Name1WithVars,Name2WithVars).

make_args_to_vars([[arg]|TailIn],[_|TailOut]) :-
    !,
    make_args_to_vars(TailIn,TailOut).
```

```
make_args_to_vars([Head|TailIn],[Head|TailOut]) :-
    make_args_to_vars(TailIn,TailOut).

make_args_to_vars([],[]).
```

A.4 Term grammar

Below we describe the grammar semi-formally by first listing (in a formal DCG-notation) a list of simplified grammar rules that any term must obey and then providing an informally described list of additional constraints that any term must satisfy in order to be actually parsed by the formula grammar. The constituent "term" used in the DCG-rules below, has two features: One is the notational type (a list of basic notational types), and the other one describes the complexity of the term. We distinguish four complexities:

1. simple: Any term that either is bracketed or has a classical or circumfix function as its head.
2. prefix_simple: Any term consisting of a prefix or quantifier function and its argument(s).
3. semisimple: Any term consisting of a suffix function and its argument.
4. complex: Any term consisting of an infix function and its arguments.

When we use CamelCase names in the position of the complexity feature, these describe semi-formally which complexities are allowed at that place (e.g. "NotSimple" means any complexity apart from "simple" is allowed).

SIMPLIFIED GRAMMAR RULES:

```
term(NT,simple) --> term([classical|NT],_), ['('], term_list, [')'].

term(NT,semisimple) --> term(_,SimpleOrSemisimple), term([suffix|NT],simple).

term(NT,prefix_simple) --> term([prefix|NT],_), term(_,_).

term(NT,prefix_simple) --> term([quantifier|NT],_), variable_list, term(_,_).

term(NT,complex) --> term(_,_), term([infix|NT],semisimple), term(_,_).

term(NT,simple) --> cimrcumfix_term(NT,_).
```

```
term(NT,simple) --> ['('], term(NT,NotSimple), [')'].

term(NT,simple) --> variable(NT).

-- -- -- --

term_list --> term(_,_), [','], term_list.

term_list --> term(_,_).

-- -- -- --

variable_list --> quantified_variable, [','], variable_list.

variable_list --> quantified_variable

-- -- -- --

quantified_variable --> [_].

-- -- -- --

variable([infix]) --> [\rightarrow].

variable([infix]) --> [\leftrightarrow].

variable([infix]) --> [\wedge].

variable([infix]) --> [\vee].

variable([prefix]) --> [\neg].

variable([quantifier]) --> [\forall].

variable([quantifier]) --> [\exists].

variable([infix]) --> [=].

variable([infix]) --> [\neq].

variable(_) --> simple_variable, variable_tail.

-- -- -- --

simple_variable([]) --> [_].
```

```
-- -- -- --

variable_tail --> [].

variable_tail --> [(v], digits, [)v].

-- -- -- --

digits --> digit, digits.

digits --> digit.

-- -- -- --

digit --> [0]; [1]; [2]; [3]; [4]; [5]; [6]; [7]; [8]; [9].

-- -- -- --

For every accessible variable V of type NT, we add a rule of the
following form to the grammar:

variable(NT) --> V.

-- -- -- --

For every accessible circumfix function of type NT and with name
[Sym11,...,Sym1n1,[arg],Sym21,...,Sym2n2,[arg],...,[arg],Symm1,...,Symmnm]
adds a rule of the following form to the grammar:

circumfix_term(NT) --> [Sym11], ..., [Sym1n1], term(_,_), [Sym21], ..., [Sym2n2],
                       term(_,_), ..., term(_,_), [Symm1], ..., [Symmnm].

-- -- -- --

ADDITIONAL CONSTRAINTS:

1. Types:

    * Every variable and circumfix function has a type. In the first
    five rules, the type of the function term has to be of the form
    [t1,...,tn]=>t, the types of the arguments have to be of the form
    t1,...,tn, and the type of the resulting term is t. The type of
    a circumfix function also has to be of the form [t1,...,tn]=>t,
    and the arguments in the added circumfix_term rules have to be of
```

```
    type t1,...,tn; the circumfix_term then gets type t.

    * We may not use the "variable(_) --> simple_variable, variable_tail."
    rule to parse a variable of type o. When we parse a variable V
    according to this rule, we add a rule of the following form to the
    grammar (where the variable in this added rule must be of the same
    type as the V we just parsed):

    variable(_) --> V.

    * A term of type var(t,X) is parsed as a quantified_variable. For
    parsing any term of type X-t' later on, we add the following
    additional rule to the grammar (where V is the term parsed as
    quantified_variable, the type of this variable is t and NT has to
    take the same value whenever this rule is used):

    variable(NT) --> V.

The predefined variables have the following types:
        * \rightarrow, \leftrightarrow, \wedge and \vee: [o,o]=>o
        * \neg: [o]=>o
        * \forall: [var(_,X),X-o]=>o
        * \exists: [var(_,X),X-o]=>o
        * =: [T,T]=>o (i.e. its two arguments must be of the same type)
        * \neq: [_,_]=>o ( i.e. its arguments may be of different type)

2. Priorities in the case of ambiguous variables:
We distinguish different kinds of variables:
        * Predefined logical variables parsed by one of the first nine
        variable rules in the grammar.
        * Variables parsed according to a rule added when parsing a
        quantified_variable (see the third part of constraint 1), i.e.
        bound variables.
        * Variables parsed according to a rule of the form
        "variable([]) --> V." added according to the second part of
        constraint 1, i.e. variables implicitely introduced earlier in
        the formula and reused at this point.
        * Accessible variables whose antecedent is in the same sentence
        as the formula that is being parsed.
        * Accessible variables whose antecedent is before the sentence
        of the formula being parsed.
        * Variables parsed according to the
        "variable(_) --> simple_variable, variable_tail." rule , i.e.
        implicitely introduced new variables.
```

When trying to parse a variable, we always first try to parse it according to a variable kind higher up in the above list before trying the kinds lower down in the list. Once a variable has been parsed in one way, it may no longer be parsed in such a way as to be of a kind that is mentioned later in the above list than the kind that it has already been assigned. (This means for example that if "x" is accessible and we parse "\exists x x+x=x", then all instances of "x" in this formula are bound by the existential quantifier; none of the instances of "x" refers to the accessible variable.)

A non-empty variable_tail may only be parsed after a simple_variable, if that simple_variable followed by (v, a term of type i and)v cannot be parsed as a term without the use of non-empty variable tails. (This means for example that x_1 canot be used as a variable name, if we have defined a two-place function (x,y) |--> x_y for individuals x, y.)

3. Operator priorities (OP) have to be obeyed:
   * The OP of a complex term is the OP of its head function. The left argument of an infix function must have OP less than or equal to the OP of the infix function, and the function's right argument must have OP strictly less than the functions OP.

   * The operator priorities of +, -, \rightarrow and \leftrightarrow are 3; all other operator priorities are 2.

   * Prefix functions are treated as if they had operator priority 2.5: After a prefix function and after a quantifier and its variable list, there must be a term with OP =< 2. And the left argument of an infix function may not be prefix_simple.

4. Special treatment of formulae (i.e. terms of type o):
   * Atomic formulae are generally treated like simple terms with OP 0.

   * A complex argument to a prefix, suffix or infix function may only be of type o if the corresponding argument type of the function was predefined to be of type o. For example, "a = b \neq c" may not be parsed as "(a = b) \neq c", even though "\neq"'s first argument may in general be of any type, because "a = b" is complex and of type o and "\neq"'s first argument was not predefined to be of type o.

   * Infix relation symbols (i.e. function symbols with type of the form [_,_]=>o) may be used for chained formulae, e.g. t1 = t2 = t3 = t4. In this case the tree we produce for the formula is the same as if the formula had been t1 = t2 \wedge t2 = t3 \wedge t3 = t4.

```
5. Notational types other than "classical" have to be predefined:
    * In the second to fifth rule, as well as in the
    "variable(_) --> simple_variable, variable_tail." rule, the sytactic
    type of a term may not be instantiated to prefix, quantifier,
    suffix or circumfix and may only be instantiated to infix if
    it is a prefered infix function symbol (\cdot, +, -, *, ., \circ,
    /, \in, <, >, \leq, \geq); for example, the requirement of the
    final term to have "suffix" as notational type in the second rule
    means that this notational type must already be in the term when parsing it
    and may not be attached to the term afterwards. (In practice, this
    constraint means that when you are quantifying over a function, this
    function may be used with classical notational type or, if a prefered
    infix function symbol is used, with infix notational type, but not
    with prefix, suffix or quantifier notational type. So "\exists f f(a)=0"
    and "\exists * x*x=x" are allowed, but "\exists z xzx=x",
    "\exists f fa=0" and "\exists g ag=0" (with "z" read as an infix,
    "f" as a prefix and "g" as a suffix function symbol) are not allowed.)

    * If the notational type of an infix function symbol is instantiated
    to "infix", then the arguments of this infix function may not be
    complex formulae. (Thus "\exists * x*x=x" may not be read as
    "\exists * x*(x=x)".)

6. The variables parsed by variable_list must be distinct.
```

Appendix B

Chapter 1 of Landau's *Grundlagen* in the Naproche CNL

This appendix contains the reformulation of the first chapter of Landau's *Grundlagen der Analysis* in the Controlled Natural Language of Naproche. This reformulation can be parsed by Naproche 0.52. The proof checking has some limitation: Up to theorem 8, the limitations are only due to the fact that premise selection is currently not supported. Additionally, some problems in the implementation of the proof checking of proofs by cases cause problems in the proof of theorem 9.

The reformulation is based on a previous reformulation of the first chapter of Landau's *Grundlagen* for the CNL of Naproche 0.3 and 0.4. This previous text was joint work by Merlin Carl, Daniel Kühlwein and this thesis' author. The previous text was in a number of points less faithful to the original than the current text:

- No talk about sets was possible: The induction axiom (Axiom 5) could not be formulated and was replaced by a proof-by-induction principle included in the system. The proofs by induction thus also avoided talk about sets.
- Since there was only one domain of discourse (namely the natural numbers, and not also set of natural numbers as in the original text and the current reformulation), there was no need for the predicate "natural number" to be used. The sentence "Small latin letters will stand throughout for natural numbers." that is now used in a similar way as a corresponding sentence in the original text ("Kleine lateinische Buchstaben bedeuten in diesem Buch, wenn nichts anderes gesagt wird, durchweg natürliche Zahlen.") was hence not needed (and would not have been accepted by the CNLs of Naproche 0.3 and 0.4).
- Quantification over functions was not possible, so theorem 4 could not be stated as in the original text and in this reformulation. (The proof of theorem 4 without talk about sets would not have worked at any rate.

Instead, the CNL allowed for recursive definitions, which were treatde by the proof-checking module in a special way.)

- It was not possible to omit the multiplication sign.
- Since there were no inbuilt operator priorities additional brackets has to be added in formulae containing both multiplication and addition.

All adaptations made to that previous reformulation for making the text more faithful to the original were made by this thesis' author.

For readability, the text is presented here not as the LaTeX code that actually serves as input to the Naproche system, but in the typeset form that LaTeX produces out of this LaTeX code.

Assume that there is a set of objects called natural numbers.

Small Latin letters will stand throughout for natural numbers.

Axiom 1: 1 is a natural number.

Axiom 2: For every x, there is a natural number x'.

Axiom 3: For every x, $x' \neq 1$.

Axiom 4: If $x' = y'$, then $x = y$.

Axiom 5: Suppose $\mathfrak{M}$ is a set of natural numbers satisfying the following properties:
Property 1: 1 belongs to $\mathfrak{M}$.
Property 2: If x belongs to $\mathfrak{M}$, then x' belongs to $\mathfrak{M}$.
Then $\mathfrak{M}$ contains all natural numbers.

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

Theorem 2: For all x $x' \neq x$.
Proof:
Let $\mathfrak{M}$ be the set of x such that $x' \neq x$.
By Axiom 1 and axiom 3, $1' \neq 1$, i.e. 1 belongs to $\mathfrak{M}$.
If x belongs to $\mathfrak{M}$, then $x' \neq x$, i.e. by Theorem 1 $(x')' \neq x'$, i.e. x' belongs to $\mathfrak{M}$.
By Axiom 5 $\mathfrak{M}$ contains all natural numbers, i.e. for every x $x' \neq x$. Qed.

Theorem 3: If $x \neq 1$ then there is a u such that $x = u'$.
Proof:
Let $\mathfrak{M}$ be the set of x such that $x = 1$ or there is a u such that $x = u'$.
1 belongs to $\mathfrak{M}$.

Suppose x belongs to $\mathfrak{M}$. Now if $u = x$ then $x' = u'$. So x' belongs to $\mathfrak{M}$.
Thus by Axiom 5, $\mathfrak{M}$ contains all natural numbers. Hence for every x such that $x \neq 1$, there is a u such that $x = u'$. Qed.

Theorem 4: There is precisely one function $x, y \mapsto x + y$ such that for all x, y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$.
Proof:
A) Fix x. Suppose that there are functions $y \mapsto a_y$ and $y \mapsto b_y$ such that $a_1 = x'$ and $b_1 = x'$ and for all y, $a_{y'} = (a_y)'$ and $b_{y'} = (b_y)'$.
Let $\mathfrak{M}$ be the set of y such that $a_y = b_y$.
$a_1 = x' = b_1$, so 1 belongs to $\mathfrak{M}$.
If y belongs to $\mathfrak{M}$, then $a_y = b_y$, i.e. by Axiom 2 $(a_y)' = (b_y)'$, i.e. $a_{y'} = (a_y)' = (b_y)' = b_{y'}$, i.e. y' belongs to $\mathfrak{M}$.
So $\mathfrak{M}$ contains all natural numbers. Thus for all y, $a_y = b_y$.
Thus there is at most one function $y \mapsto x + y$ such that $x + 1 = x'$ and for all y, $x + y' = (x + y)'$.
B) Now let $\mathfrak{M}$ be the set of x such that there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$.
Suppose $x = 1$. Define $x + y$ to be y'. Then $x + 1 = 1' = x'$, and for all y,$x + y' = (y')' = (x + y)'$. Thus 1 belongs to $\mathfrak{M}$.
Let x belong to $\mathfrak{M}$. Then there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$. For defining $+$ at x', define $x' + y$ to be $(x + y)'$.
Then $x' + 1 = (x + 1)' = (x')'$ and for all y, $x' + y' = (x + y')' = ((x + y)')' = (x' + y)'$.
So x' belongs to $\mathfrak{M}$.
Thus $\mathfrak{M}$ contains all x. So for every x, there is a function $y \mapsto x + y$ such that for all y, $x + y$ is a natural number and $x + 1 = x'$ and $x + y' = (x + y)'$. Qed.

Theorem 5: For all x, y, z, $(x + y) + z = x + (y + z)$.
Proof:
Fix x, y. Let $\mathfrak{M}$ be the set of z such that $(x + y) + z = x + (y + z)$.
A) $(x + y) + 1 = (x + y)' = x + y' = x + (y + 1)$, so 1 belongs to $\mathfrak{M}$.
B) Let z belong to $\mathfrak{M}$. Then $(x + y) + z = x + (y + z)$, so $(x + y) + z' = ((x + y) + z)' = (x + (y + z))' = x + (y + z)' = x + (y + z')$, so z' belongs to $\mathfrak{M}$.
Thus $\mathfrak{M}$ contains all z. Qed.

Lemma 4a: For all y, $1 + y = y'$.
Proof:
Let $\mathfrak{M}$ be the set of y such that $1 + y = y'$.
By Theorem 4, $1 + 1 = 1'$, so 1 belongs to $\mathfrak{M}$.
Let y belong to $\mathfrak{M}$. Then $1 + y = y'$. Then by Theorem 4, $1 + y' = (1 + y)'$. So $1 + y' = (y')'$. So y' belongs to $\mathfrak{M}$.
Thus $\mathfrak{M}$ contains all y. Therefore for all y $1 + y = y'$. Qed.

Lemma 4b: For all x,y, $x' + y = (x + y)'$.
Proof:
Fix x. Let $\mathfrak{M}$ be the set of y such that $x' + y = (x + y)'$. Then by Theorem 4 $x' + 1 = (x')' = (x + 1)'$, so 1 belongs to $\mathfrak{M}$.
Let y belong to $\mathfrak{M}$. Then $x' + y = (x + y)'$. Then by Theorem 4 $x' + y' =$

$(x' + y)' = ((x + y)')' = (x + y')'$. So y' belongs to $\mathfrak{M}$.
Thus for all y $x' + y = (x + y)'$. Qed.

Theorem 6: For all y, x, $x + y = y + x$.
Proof:
Fix y. Let $\mathfrak{M}$ be the set of x such that $x + y = y + x$.
A) $y + 1 = y'$ and by lemma 4a $1 + y = y'$, so $1 + y = y + 1$ and 1 belongs to $\mathfrak{M}$.
B) If x belongs to $\mathfrak{M}$, then $x + y = y + x$, so $(x + y)' = (y + x)' = y + x'$.
By lemma 4b $x' + y = (x + y)'$, so $x' + y = y + x'$, so x' belongs to $\mathfrak{M}$.
Thus for all x $x + y = y + x$. Qed.

Theorem 7: For all x, y, $y \neq x + y$.
Proof:
Fix x. Let $\mathfrak{M}$ be the set of y such that $y \neq x + y$.
A) $1 \neq x'$, i.e. $1 \neq x + 1$, so 1 belongs to $\mathfrak{M}$.
B) If y belongs to $\mathfrak{M}$, then $y \neq x + y$, so $y' \neq (x + y)'$, i.e. $y' \neq x + y'$, so y' belongs to $\mathfrak{M}$. Thus for all y $y \neq x + y$. Qed.
Theorem 8: If $y \neq z$, then for all x $x + y \neq x + z$.
Proof:
Assume $y \neq z$. Let $\mathfrak{M}$ be the set of x such that $x + y \neq x + z$.
A) $y' \neq z'$, i.e. $1 + y \neq 1 + z$, so 1 belongs to $\mathfrak{M}$.
B) If x belongs to $\mathfrak{M}$, then $(x + y)' \neq (x + z)'$, i.e. $x' + y \neq x' + z$, so x' belongs to $\mathfrak{M}$.
Thus for all x $x + y \neq x + z$. Qed.

Theorem 9: Fix x, y. Then precisely one of the following cases holds:
Case 1: $x = y$.
Case 2: There is a u such that $x = y + u$.
Case 3: There is a v such that $y = x + v$.
Proof:
A) Case 1 and case 2 are inconsistent and case 1 and case 3 are inconsistent. Suppose case 2 and case 3 hold. Then $x = y + u = (x + v) + u = x + (v + u) = (v + u) + x$.
Contradiction. Thus case 2 and case 3 are inconsistent. So at most one of case 1, case 2 and case 3 holds.
B) Fix x. Let $\mathfrak{M}$ be the set of y such that precisely one of case 1, case 2 and case 3 holds.
I) If $y = 1$, then by Theorem 3 $x = 1 = y$ or $x = u' = 1 + u = y + u$.
Thus 1 belongs to $\mathfrak{M}$.
II) Let y belong to $\mathfrak{M}$. Then there are three cases:
Case 1: $x = y$.
Then $y' = y + 1 = x + 1$, i.e. y' belongs to $\mathfrak{M}$.
Case 2: $x = y + u$.
If $u = 1$, then $x = y + 1 = y'$, i.e. y' belongs to $\mathfrak{M}$.
If $u \neq 1$, then $u = w' = 1 + w$, so $x = y + (1 + w) = (y + 1) + w = y' + w$. So y' belongs to $\mathfrak{M}$.
Case 3: $y = x + v$.
Then $y' = (x + v)' = x + v'$, i.e. y' belongs to $\mathfrak{M}$.
So in all cases y' belongs to $\mathfrak{M}$.
Thus for all y, case 1 or case 2 or case 3 holds. Qed.

Definition 2:
Define $x > y$ iff there is a u such that $x = y + u$.

Definition 3:
Define $x < y$ iff there is a v such that $y = x + v$.

Theorem 10: Let x, y be given. Then precisely one of the following cases holds:
Case 1: $x = y$.
Case 2: $x > y$.
Case 3: $x < y$.
Proof: By Theorem 9, definition 2 and definition 3. Qed.

Theorem 11: $x > y$ implies $y < x$.
Proof: For all x,y, we have $x > y$ iff there is a u such that $x = y + u$. Furthermore, we have $y < x$ iff there is a u such that $x = y + u$. So for all x, y, $x > y$ implies $y < x$. Qed.

Theorem 12: $x < y$ implies $y > x$.
Proof: We have $x < y$ iff there is a v such that $y = x + v$. Furthermore, we have $y > x$ iff there is a v such that $y = x + v$. So $x < y$ implies $y > x$. Qed.

Definition 4:
Define $x \geq y$ iff $x > y$ or $x = y$.

Definition 5:
Define $x \leq y$ iff $x < y$ or $x = y$.

Theorem 13: $x \geq y$ implies $y \leq x$.
Proof:
By Theorem 11. Qed.

Theorem 14: $x \leq y$ implies $y \geq x$.
Proof:
By Theorem 12. Qed.

Theorem 15: If $x < y$ and $y < z$ then $x < z$.
Proof: Assume $x < y$ and $y < z$. Then there is a v such that $y = x + v$. Furthermore, there is a u such that $z = y + u$. Then $z = (x + v) + u = x + (v + u)$. So there is a w such that $z = x + w$. So $x < z$. Qed.

Theorem 16: Let x, y, z be given. If $x \leq y$ and $y < z$ or $x < y$ and $y \leq z$ then $x < z$.
Proof:
By Theorem 15. Qed.

Theorem 17: If $x \leq y$ and $y \leq z$ then $x \leq z$.
Proof:
By Theorem 16. Qed.

Theorem 18: For all x,y, $x + y > x$.
Proof: For all x,y we have $x + y = x + y$. Qed.

Theorem 19: Let x, y, z be given. Then $x > y$ implies $x + z > y + z$, $x = y$ implies $x + z = y + z$ and $x < y$ implies $x + z < y + z$.
Proof:
Let z be given.
If $x > y$, then $x = y + u$, so $x + z = (y + u) + z = (u + y) + z = u + (y + z) = (y + z) + u$, i.e. $x + z > y + z$.
If $x = y$ then clearly $x + z = y + z$.
If $x < y$, then $y > x$, i.e. $y + z > x + z$, i.e. $x + z < y + z$. Qed.

Theorem 20: Let x, y, z be given. Then $x+z > y+z$ implies $x > y$, $x+z = y+z$ implies $x = y$ and $x + z < y + z$ implies $x < y$.
Proof:
By Theorem 19. Qed.

Theorem 21: If $x > y$ and $z > u$ then $x + z > y + u$.
Proof:
Assume $x > y$ and $z > u$. Then by Theorem 19 $x + z > y + z$. Then $y + z = z + y > u + y = y + u$. So $x + z > y + u$. Qed.

Theorem 22: Let x, y, z, u be given. If $x \geq y$ and $z > u$ or $x > y$ and $z \geq u$ then $x + z > y + u$.
Proof:
By Theorem 19 and theorem 21. Qed.

Theorem 23: If $x \geq y$ and $z \geq u$ then $x + z \geq y + u$.
Proof:
Trivial. Qed.

Theorem 24: For all x, we have $x \geq 1$.
Proof:
Fix x. Then $x = 1$ or $x = u' = u + 1 > 1$. Qed.

Theorem 25: $y > x$ implies $y \geq x + 1$.
Proof:
Assume $y > x$. Then $y = x + u$. $u \geq 1$, i.e. $y \geq x + 1$. Qed.

Theorem 26: $y < x + 1$ implies $y \leq x$.
Proof:
Assume for a contradiction that $y < x + 1$ and $\neg y \leq x$. Then $y > x$. So by Theorem 25 $y \geq x + 1$. Contradiction. Qed.

Theorem 28: There is a function $x, y \mapsto x \cdot y$ such that for all x, y, $x \cdot y$ is a natural number and $x \cdot 1 = x$ and $x \cdot y' = (x \cdot y) + x$.
Note: Instead of $x \cdot y$ we also write xy.
Proof:
A) Fix x. Suppose that there are functions $y \mapsto a_y$ and $y \mapsto b_y$ such that $a_1 = x'$ and $b_1 = x'$ and for all y, $a_{y'} = (a_y) + x$ and $b_{y'} = (b_y) + x$.

Let $\mathfrak{M}$ be the set of y such that $a_y = b_y$.
$a_1 = x = b_1$, so 1 belongs to $\mathfrak{M}$.
If y belongs to $\mathfrak{M}$, then $a_y = b_y$, i.e. $a_{y'} = (a_y) + x = (b_y) + x = b_{y'}$, i.e. y' belongs to $\mathfrak{M}$. So $\mathfrak{M}$ contains all natural numbers. Thus for all y, $a_y = b_y$.
Thus there is at most one function $y \mapsto x \cdot y$ such that $x \cdot 1 = x$ and for all y, $xy' = xy + x$.
B) Now let $\mathfrak{M}$ be the set of x such that there is a function $y \mapsto x \cdot y$ such that $x \cdot 1 = x$ and for all y, $xy' = xy + x$.
Suppose $x = 1$. Define $x \cdot y$ to be y. Then $x \cdot 1 = 1 = x$ and $xy' = y' = y + 1 = xy + x$. Thus 1 belongs to $\mathfrak{M}$.
Let x belong to $\mathfrak{M}$. Then there is a function $y \mapsto x \cdot y$ such that $x \cdot 1 = x$ and for all y, $xy' = xy + x$. For defining $\cdot$ at x' define $x' \cdot y$ to be $(xy) + y'$.
Then $x' \cdot 1 = x \cdot 1 + 1 = x + 1 = x'$ and for all y, $x'y' = xy' + y' = (xy + x) + y'$ and $(xy + x) + y' = xy + (x + y') = xy + (x + y)'$ and $xy + (x + y)' = xy + (x' + y) = xy + (y + x')$ and $xy + (y + x') = (xy + y) + x' = x'y + x'$.
So x' belongs to $\mathfrak{M}$.
Thus $\mathfrak{M}$ contains all x. Qed.

Lemma 28a: For all y, $1 \cdot y = y$.
Proof:
Let $\mathfrak{M}$ be the set of y such that $1 \cdot y = y'$.
By Theorem 28, $1 \cdot 1 = 1$, so 1 belongs to $\mathfrak{M}$.
Let y belong to $\mathfrak{M}$. Then $1 \cdot y = y$. Then by Theorem 28, $1 \cdot y' = (1 \cdot y) + 1 = y + 1 = y'$. So y' belongs to $\mathfrak{M}$.
Thus for all y $1 \cdot y = y$. Qed.

Lemma 28b: For all x,y, $x'y = xy + y$.
Proof:
Fix x. Let $\mathfrak{M}$ be the set of y such that $x' + y = (x + y)'$. Then by Theorem 28 $x' \cdot 1 = x' = (x \cdot 1)' = (x \cdot 1) + 1$, so 1 belongs to $\mathfrak{M}$.
Let y belong to $\mathfrak{M}$. Then $x'y = xy + y$. Then by Theorem 28 $x'y' = x'y + x' = (xy + y) + x'$ and $(xy + y) + x' = xy + (y + x') = xy + (x' + y)$ and $xy + (x' + y) = xy + (x + y)' = xy + (x + y')$ and $xy + (x + y') = (xy + x) + y' = (xy') + y'$. So y' belongs to $\mathfrak{M}$.
Thus for all y $x'y = xy + y$. Qed.

Theorem 29: For all x, y, $xy = yx$.
Proof:
Fix y. Let $\mathfrak{M}$ be the set of x such that $xy = yx$.
I) $y \cdot 1 = y$, and by lemma 28a, $1 \cdot y = y$, so $y \cdot 1 = 1 \cdot y$. Hence 1 belongs to $\mathfrak{M}$.
II) Suppose x belongs to $\mathfrak{M}$. Then $xy = yx$, i.e. $xy + y = yx + y = yx'$. By lemma 28b, $x'y = xy + y$, so $x'y = yx'$, i.e. x' belongs to $\mathfrak{M}$.
Thus for all x $xy = yx$. Qed.

Theorem 30: For all x, y, z, $x(y + z) = xy + xz$.
Proof:
Fix x, y. $x(y + 1) = xy' = xy + x = xy + (x \cdot 1)$.
Now suppose $x(y + z) = xy + xz$. Then $x(y + z') = x((y + z)') = (x(y + z)) + x$ and $(x(y + z)) + x = (xy + (xz)) + x$ and $(xy + (xz)) + x = xy + (xz + x) = xy + xz'$.
Thus by induction, for all z $x(y + z) = xy + xz$. Qed.

Theorem 31: For all x, y, z, $(xy)z = x(yz)$.
Proof:
Fix x, y. Then $(xy) \cdot 1 = xy = x(y \cdot 1)$.
Now suppose $(xy)z = x(yz)$. Then by Theorem 30, $(xy)z' = ((xy)z) + (xy) = (x(yz)) + (xy)$ and $(x(yz)) + (xy) = x((yz) + y) = x(yz')$.
Thus by induction, for all z $(xy)z = x(yz)$. Qed.

Theorem 32: For all z, $x > y$ implies $xz > yz$, $x = y$ implies $xz = yz$ and $x < y$ implies $xz < yz$.
Proof:
Let z be given.
If $x > y$, then $x = y + u$, i.e. $xz = (y + u)z = (yz) + (uz) > yz$.
If $x = y$, then clearly $xz = yz$.
If $x < y$, then $y > x$, i.e. $yz > xz$, i.e. $xz < yz$. Qed.

Theorem 33: $xz > yz$ implies $x > y$, $xz = yz$ implies $x = y$ and $xz < yz$ implies $x < y$.
Proof:
By Theorem 32 and theorem 10. Qed.

Theorem 34: If $x > y$ and $z > u$, then $xz > yu$.
Proof:
Suppose $x > y$ and $z > u$. By Theorem 32, $xz > yz$ and $yz = zy > uy = yu$, i.e. $xz > yu$. Qed.

Theorem 35: For all x, y, z, u, if $x \geq y$, $z > u$ or $x > y$, $z \geq u$, then $xz > yu$.
Proof:
By Theorem 32 and theorem 34. Qed.

Theorem 36: If $x \geq y$ and $z \geq u$, then $xz \geq yu$.
Proof:
By Theorem 35. Qed.

Appendix C

Differences between the presented theory and the implementation

In this appendix we briefly discuss the main differences between the theory described in this thesis and what is implemented in the the Naproche system, version 0.52.

C.1 Proof Representation Structures

As already mentioned in the introduction, the main difference between the presented theory and the implementation is that where we used the formalism *PTL* in the presented theory, we used *Proof Representation Structures* (*PRSs*) in the implementation. Just as *PTL* is an extension of Dynamic Predicate Logic, PRSs are an extension of *Discourse Representation Structure* (*DRSs*) (see (see Kamp & Reyle, 1993)). And just as Dynamic Predicate Logic, DRSs were developed for modelling the dynamic nature of natural language quantification.

So in the implemented system, the Naproche CNL input is translated into a PRS, and the proof checking algorithm is defined on a PRS input.

We now describe the syntax and semantics of PRSs,[1] at the same time comparing PRSs to *PTL*.

A PRS has five constituents: An identification number, a list of discourse referents, a list of mathematical referents, a list of textual referents and an ordered list of conditions[2]. Similar to DRSs, we can display PRSs as "boxes" (Figure C.1).

Mathematical referents are the parse trees of the terms and formulae which appear in the text. As in DRSs, discourse referents are used to identify objects in the domain of the discourse. The discourse referents correspond directly to *PTL* variables. Mathematical referents do not have a direct correspondent in

[1] The description of PRS syntax is partly taken over from Cramer, Fisseni, et al. (2010).

[2] The order of the conditions in a PRS reflects the argument structure of a proof and is relevant to the PRS semantics. This was in part inspired by Asher's SDRT (Asher, 1993)

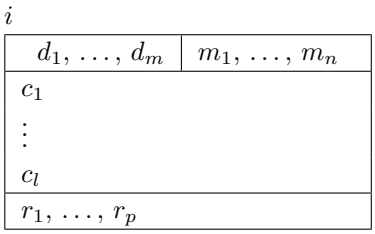

Figure C.1: A PRS with identification number i, discourse referents $d_1, ..., d_n$, mathematical referents $m_1, ..m_k$, conditions $c_1, ..., c_l$ and textual referents $r_1, ..r_p$.

PTL: They are used to keep track of the link between discourse referents and certain symbols in the input text, in order to implement the anaphoric resolution of symbolic expressions in the Naproche-CNL-to-PRS translation.

The textual referents correspond to the IDs in *PTL* syntax, i.e. their function is to model intratextual references.

PRSs have identification numbers, so that we can refer to them from other points in the discourse. The textual referents indicate the intratextual and intertextual references.

Just as in the case of DRSs, PRSs and PRS conditions are defined recursively: Let $A, B, B_1, \ldots, B_n$ be PRSs, $d, d_1, \ldots, d_n$ discourse referents, t a parse tree of a term of formula, T a list of term or formula parse trees, Id a PRS ID and ϑ a theorem type ("theorem" or "lemma"). Then

- for any n-ary predicate p (expressed by an adjectives, noun, verb or preposition in the Naproche CNL), predicate$(d_1, \ldots, d_n, p)$ is a condition, expressing that the tuple $(d_1, \ldots, d_n)$ satisfies the predicate p;
- holds(t, T) is a condition, representing the claim that the formula whose possible parse trees are listed in T and whose actual parse tree is t is true;
- math_id(d, t, T) is a condition which links the discourse referent d to a symbolic term, whose possible parse trees are listed in T and whose actual parse tree is t;
- A is a condition;
- $\neg A$ is a condition, representing a negation;
- $A \Rightarrow B$ is a condition, representing an assumption (A) and the set of claims made inside the scope of this assumption (B);
- $A \Leftrightarrow B$ is a condition, representing a logical equivalence;
- $A \Leftarrow B$ is a condition, representing a material implication in the reversed order;
- $A \vee B$ is a condition, representing an inclusive disjunction;

- $>< (A_1, \ldots, A_n)$ is a condition, representing an exclusive disjunction, i.e. the claim that precisely one of $A_1, \ldots, A_n$ holds;
- $<> (A_1, \ldots, A_n)$ is a condition, representing the claim that at most one of $A_1, \ldots, A_n$ holds;
- the(d, A) is a condition, representing a definite description;
- static(A) is a condition, representing an assertion with static quantifiers (i.e. the discourse referents introduced in A cannot bind discourse referent outside static(A); this corresponds to a formula of the form $\Diamond\varphi$ in PTL);
- at_most_one(d, A) is a condition, representing a quantification of the form "at most one object has a given property";
- holds_prs(Id) is a condition, representing the truth of the PRS with ID Id.
- theorem(ϑ, A, B) is a condition, representing a theorem or lemma and its proof;
- *contradiction* is a condition, representing a contradiction.

Furthermore, there are three PRS conditions that are used for the preliminary translation of the CNL input to PRSs, but which get eliminated by the plural interpretation algorithm (in the first two cases) or by algorithm for interpreting cataphoric meta-NPs (in the third case):

- plural_dref$(d, [d_1, \ldots, d_n])$, which links a plural discourse referent d to a list of discourse referent $[d_1, \ldots, d_n]$ (this corresponds to the plural variables with a list of PTL terms as subscript in the extension of PTL introduced for the plural interpretation algorithm in section 7.6.4).
- plural(n, A), corresponding to formulae of the form $plural(x, \varphi)$ in the PTL extension of section 7.6.4.
- followings, used for preliminarily translating cataphoric meta-NPs.

The semantics of the various PRS conditions is already alluded to in the above list of possible PRS conditions. In order to precisely define the semantics of PRSs, we define a translation from PRSs to PTL texts. Given that we have defined a semantics for PTL texts in section 5.2.2, such a translation fixes the semantics of PRSs. The three temporary PRS conditions do not need to be given a semantics.

Before we define this translation, we need to point out that in PRSs there is no distinction analogous to the distinction between $\wedge$ and & in PTL: The only way to express a conjunction in PRS is by including more than one PRS condition in a PRS; in that case the PRS conditions will be interpreted as conjuncted. Recall that in the model-theoretic definition of PTL semantics, $\wedge$ and & were at any rate indistinguishable. In the proof checking they were treated differently, and the concatenation of PRS condition is treated in the same way as &. But recall that in PTL, & may only appear in certain positions, since it may not appear in PTL formulae. In order to ensure that the PTL texts in the translation sketched below are always well-formed, we therefore use $\wedge$ rather than & for translating the concatenation of PRS conditions.

A future version of the Naproche system should implement the distinction that *PTL* makes between $\wedge$ and & in the PRS formalism. But at any rate it is not a big problem that this distinction is so far not implemented. Sentences where is would make a difference are at any rate considered bad style in the language of mathematics. One example of such a sentence is (1):

(1) Some prime p divides N, and A contains $p^2 - 1$.

In the Naproche-CNL-to-*PTL* translation, (1) gets translated as $\exists p\ \mathit{divide}(p, N) \wedge p^2 - 1 \in A$. Proof-checking this sentence involves checking a proof obligation with conjecture $\exists p \neq u\ (\mathit{divide}(p, N) \wedge p^2 - 1 \in A)$ (note the different bracketing in the *PTL* formula and this conjecture). If we had translated (1) by $\exists p\ \mathit{divide}(p, N)\ \&\ p^2 - 1 \in A$ instead, proof-checking the sentence would involve checking two proof obligations: First one with conjecture $\exists p \neq u\ \mathit{divide}(p, N)$, and secondly a proof obligation with $p \neq u$ and $\mathit{divide}(p, N)$ as additional premises and $p^2 - 1 \in A$ as conjecture. In the second case, the proof-checking could only be successful if for every prime divisor p of N, $p^2 - 1 \in A$, whereas in the first case it is sufficient if for some prime divisor p of N, $p^2 - 1 \in A$. It is probably this ambiguity which makes mathematicians usually avoid sentences like (1).

For defining the PRS-to-*PTL* translation, we identify the supply of discourse referents with the supply of *PTL* variables and the supply of textual references with the supply of IDs in *PTL* syntax. Now the *PTL* translation $\mathbf{t}(A)$ of a PRS A with discourse referents $d_1, \ldots, d_n$, conditions $c_1, \ldots, c_l$ and textual referents $r_1, \ldots, r_p$ is $\exists d_1 \ldots \exists d_n\ \mathit{ref}(\langle r_1, \ldots, r_p \rangle, \mathbf{t}(c_1) \wedge \ldots \wedge \mathbf{t}(c_n))$, where $\mathbf{t}(c_i)$ is the *PTL* translation of the PRS condition c_i. If $p = 0$, i.e. A contains no textual referents, $\mathbf{t}(A)$ is $\exists d_1 \ldots \exists d_n\ (\mathbf{t}(c_1) \wedge \ldots \wedge \mathbf{t}(c_n))$ instead. The translation of PRS conditions to *PTL* is shown in the following table:[3]

$\mathrm{predicate}(d_1, \ldots, d_n, p)$	$p(d_1, \ldots, d_n)$
$\mathrm{holds}(t, T)$	t
$\mathrm{math\_id}(d, t, T)$	$\top$
A	$\mathbf{t}(A)$
$\neg A$	$\neg\mathbf{t}(A)$
$A \Rightarrow B$	$\mathbf{t}(A) \rightarrow \mathbf{t}(B)$
$A \Leftrightarrow B$	$\Diamond(\mathbf{t}(A) \rightarrow \mathbf{t}(B)) \wedge \Diamond(\mathbf{t}(B) \rightarrow \mathbf{t}(A))$
$A \Leftarrow B$	$\Diamond(\mathbf{t}(B) \rightarrow \mathbf{t}(A))$
$A \vee B$	$\mathbf{t}(A) \vee \mathbf{t}(B)$
$>< (A_1, A_2)$	$(\mathbf{t}(A_1) \vee \mathbf{t}(A_2)) \wedge \neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_2))$
$>< (A_1, A_2, A_3)$	$(\mathbf{t}(A_1) \vee \mathbf{t}(A_2) \vee \mathbf{t}(A_3)) \wedge \neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_2)) \wedge$ $\neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_3)) \wedge \neg(\Diamond\mathbf{t}(A_2) \wedge \Diamond\mathbf{t}(A_3))$
$\vdots$	$\vdots$
$<> (A_1, A_2)$	$\neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_2))$
$<> (A_1, A_2, A_3)$	$\neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_2)) \wedge \neg(\Diamond\mathbf{t}(A_1) \wedge \Diamond\mathbf{t}(A_3)) \wedge$ $\neg(\Diamond\mathbf{t}(A_2) \wedge \Diamond\mathbf{t}(A_3))$
$\vdots$	$\vdots$

[3]Here we identify parse trees of symbolic terms and formulae with their *PTL* translations as specified in section 7.5.2, and natural language predicated with their *PTL* translations as specified in section 7.5.1.

the(d, A)	$\iota d\ \mathbf{t}(A)$
static(A)	$\Diamond\mathbf{t}(A)$
at_most_one(d, A)	$\neg(\exists d\ \exists d'\ (\mathbf{t}(A) \wedge \mathbf{t}(A)\frac{d'}{d}))$
holds_prs(Id)	$\mathbf{t}(A)$, where A is the PRS with ID *Id*.
theorem(ϑ, A, B)	$thm(\vartheta, \mathbf{t}(A), \mathbf{t}(B))$

The Naproche CNL input text is translated into one big PRS. This PRS contains *inter alia* one sub-PRS for every statement of the input text. But note that it is not the case that the Naproche-CNL-to-PRS translation is completed for the total input text before the proof checking starts. Instead, the Naproche-CNL-to-PRS translation and the proof checking alternate: The input text is processed sentence by sentence; after translating a statement (i.e. a content-full sentence) to a PRS, the PRS of this statement is proof-checked before the Naproche-CNL-to-PRS translation is continued.

This alternation is done for enabling the usage of presupposition fulfilment as a criterion for disambiguation symbolic expressions as described in section 7.4.4: When a symbolic expression has more than one reading, the PRS constructed will contain PRS conditions of the form holds(t, T) and/or math_id(d, t, T) with the list T of possible readings containing more than one element. In the position of t there will be an uninstantiated Prolog variable, which after the disambiguation gets instantiated to the chosen reading. The order of possible readings in T already indicates which reading should be preferred in case not precisely one of the readings fulfils its presuppositions. During the proof checking, the presuppositions of each of the reading in T are checked in turn. Once the presuppositions of one reading are fulfilled, t is instantiated to that reading. If for no reading in T the presuppositions can be fulfilled, t is instantiated to the first element of T.

Similarly to the accessibility relation usually defined on DRSs, one can define an accessibility relation on PRSs which specifies for each PRS conditions in a PRS (possibly embedded into a larger PRS) which discourse referents and which math_id conditions are accessible from that PRS condition.

Additionally to the five constituents of PRSs mentioned above, the implementation contains three further PRS constituents, which only serve the goal of simplifying the implementation: Two constituents keep track of which discourse referents and math_id conditions are accessible at the beginning and at the end of the PRS. The final constituent is a list of links between discourse referents and PRS conditions which keep track of the fact that a certain PRS condition results from the same indefinite noun phrase as a given discourse referent. This information is used in the implementation of the algorithm for interpreting bi-implications and reversed implications described in section 7.5.9.

C.2 Background theory

In the theory presented in this thesis, *CMTN* was used as a mathematical background theory. The function-theoretic part of *CMTN* made it possible to treat implicit function introduction in a way that does not involve the paradoxes of

unrestricted function comprehension discussed in section 3.3, but is nevertheless more flexible than a type-theoretic approach. Furthermore, the set-theoretic part of *CMTN* ensured that the natural linguistic constructions for talking about sets (e.g. "the set of x such that ...") can be used to model usual set-theoretic constructions found in mathematical texts up to the strength of what is possible in *ZFC*. On the other hand, in order to make good usage of this background theory, the input text has to contain the technical term "limited" corresponding to the symbol L of *CMTN* in the right positions; given that *CMTN* is a theory developed in the course of the work conducted for this thesis and not a "natural" theory used by ordinary mathematicians, the obligation to sensibly use the technical term "limited" in the input text can be seen as an unnatural element of the system.

In the actual system, on the other hand, we have not implemented *CMTN*. For the consistent treatment of functions, the type system that is used for disambiguating symbolic expressions (see section 7.4.2) is imposed throughout on all terms. In other words, the paradoxes of unrestricted function comprehension are actually avoided in a type-theoretic way. The set-theoretic paradoxes on the other hand are not avoided in a type-theoretic way. Instead, the only set theory built into the system are implicit applications of the Axiom of Separation and the Axiom of Extensionality in the proof checking of presuppositions requiring the existence and uniqueness of a set with certain elements: For a PRS of the form $\text{the}(d, A)$, for which A gets translated to a *PL* formula of the form $\text{set}(d) \wedge \forall x\ (x \in d \leftrightarrow \Psi(x))$, we replace the two proof obligations with conjectures (2) and (3), which would normally have to be checked for checking the existence and uniqueness presuppositions of $\text{the}(d, A)$, by a single proof obligation with conjecture (4):

(2) $\exists d\ (\text{set}(d) \wedge \forall x\ (x \in d \leftrightarrow \Psi(x)))$

(3) $\forall d, d'\ (\text{set}(d) \wedge \forall x\ (x \in d \leftrightarrow \Psi(x)) \wedge \text{set}(d') \wedge \forall x\ (x \in d' \leftrightarrow \Psi(x)) \rightarrow d = d')$

(4) $\exists d\ (\text{set}(d) \wedge \forall x\ (\Psi(x) \rightarrow x \in d))$

This minimal implicit set theory is enough for the first chapter of Landau's *Grundlagen der Analysis*.

No theory of tuples or natural numbers is included in Naproche 0.52.

C.3 Quantifier restriction

In the definition of the proof checking algorithm in chapter 6, we used restricted quantifiers of the form $\exists x \neq u$ and $\forall x \neq u$ for translating the *PTL* quantifier $\exists x$ to *PL*. Of course, PRSs with discourse referents in the discourse referent slot should similarly be translated to *PL* using quantifiers thus restricted by "$\neq u$". But in the current implementation, they are translated using quantifiers not restricted in this way.

In most cases in which we use quantification in the language of mathematics, we restrict the domain of quantification using some expression corresponding to an atomic formula. For example, we write things like "for every natural number n such that ..." or "there is some $x \in A$ such that ...", where the domain of quantification is restricted by "natural_number(n)" and "$x \in A$" respectively.

In such cases, the fact that we do not restrict quantifiers by "$\neq u$" cannot cause any problems, because the atomic formula can at any rate not be satisfied by the undefinedness object u. So when we do implement the missing quantifier restriction in the Naproche system, it is desirable to make it explicit only in the few cases in which the quantification is not already restricted in such a way by an atomic formula, since otherwise the *PL* formulae used as conjectures and premises in the proof obligations will be complicated in an unnecessary manner.

Appendix D

Concise manual of the Naproche system

In this appendix we briefly explain how to install and use the Naproche system, version 0.52.

D.1 System requirements[1]

The Naproche system only works on Linux systems. The tested and recommended distribution is Ubuntu 10.04 or newer. For smooth operation at least 1GB of RAM is required and a reasonably new processor (from 2008 or later) is recommended. The Java Runtime Environment (JRE) has to be installed on the system.

D.2 Download and installation[2]

The installation package for Naproche 0.52 can be found on the website of the Naproche project, `http://www.naproche.net`. There are separate versions for 32 bit systems (`naproche052-i686.tgz`) and for 64 bit systems (`naproche052-i686.tgz`).

For installing the system, you first need to unpack the installation package. Next you need to add a line of the form

```
check_src('/home/naproche').
```

to the `user.pl` file in the Naproche base directory, where `/home/naproche` should be replaced by the absolute path of the Naproche base directory on your system.

If the system does not work as described below, please refer to the *Troubleshooting* section of the README file located in the base directory.

[1] This section is largely based on the README file of the Naproche system, which was written mainly by Julian Schlöder and partially by the author of this thesis.

[2] This section is partly based on the README file of the Naproche system written by Julian Schlöder and the author of this thesis.

D.3 Usage of the Naproche system

In order to start the GUI (graphical user interface) of the Naproche system, you need to run the shell script `naproche.sh` located in the Naproche base directory. `naproche.sh` will provide debugging output (partially in German, though) if run from a shell.

The GUI provides two text fields: The large text field above is for the input text (written in the Naproche CNL described in chapter 7); the smaller text field below provides some feedback to the user.

Having input an input text into the above text field, you can press the `Check` button for letting the system parse and proof-check the input text. The input text will get coloured during the runtime of the system:

- Sentences that do not trigger any proof obligations are coloured grey.
- Sentences all of whose proof obligations are successfully checked by the Automated Theorem Prover (ATP) are coloured green.
- If at least one of the proof obligations triggered by a sentence cannot be successfully checked by the ATP, the sentence will be coloured red.
- Additionally, sentences may be coloured orange: This indicated that even though all of the sentence's proof obligations were successfully checked by the ATP, some of them were checked without usage of the conjecture of the proof obligation, which means that the premises were inconsistent. If sentences are coloured orange outside proofs by contradiction, this may indicate that the axioms stated in the input text are inconsistent.

The parsing and proof-checking functions like an incremental parser: One can add text to an already checked text or modify an already checked text, and the parsing and proof checking will restart from the first sentence that is different from the previously checked text. When editing an already checked text, all sentences starting from the first modified sentence are coloured blue.

Below the `Check` button is a `Show PRS` button for displaying the Proof Representation Structure (PRS) that represents the content of the input text in the default browser of the system.

When one saves a checked Naproche text using the `Save` function in the `File` menu, the system does not only save the input text, but also all data produced by the proof checking. Hence one can make use of the incremental parsing and checking across sessions. Using the `Open` function in the `File` menu, one can also open the example texts found in the `examples` directory.

Between the two text fields, there is a button which shows whether the *debug mode* is switched on or off. When starting the interface, it is switched off. Switching it on provides two further buttons, which are mainly of use for debugging, but which may also be of interest for an inquisitive user: The *Prolog-Input* button shows the tokenized and preprocessed version of the input text that is used by the formal grammar described in appendix A. The `Clear Tmp` button deletes the data used for the incremental incremental parsing and proof checking, thus making it possible to recheck a text from the beginning.

When a text not adhering to the rules of the Naproche CNL is entered, the system gives an error message and marks the first word of the input text that could not be parsed. When rechecking a text after correcting such a parsing

error, a bug in the GUI implementation causes the error message to be displayed again in the lower text field, even if the parsing and proof checking is now successful. In order to make an error message not appear again, one currently has to restart the system.[3]

It is also possible to see the proof obligations that are given to the ATP. For this, one has to open the `tmp` directory in the base directory. Note that the sentences of the input text are consecutively numbered. To avoid manual counting in the case of a long input text, you can open the `text.nap` file for seeing the tokenized and preprocessed input, where you can easily read of the sentence number of any from the input text. For every sentence that triggered at least one proof obligation, there is a subdirectory in `tmp` named according to the sentence number. For every proof obligation triggered by the sentence, the sentence directory contains four files with the file endings `.input`, `.output`, `.proofsummary` and `.result` and the same file name in front of these endings. The `.input` file contains the proof obligation in TPTP syntax (see Sutcliffe & Suttner, 1998). The `.output` file contains the output of the prover, and the `.proofsummary` file contains data extracted from the `.output` file by an external program called *Proof Summary* (see Sutcliffe, 2009). Finally, the `.result` file lists the results of the proof checking relevant for the Naproche system: After the absolute path of the corresponding `.input` file, it lists two Booleans separated by a semicolon. The first indicates whether the proof obligation was successfully checked by the ATP. The second indicates whether the the premises were shown to be inconsistent. So what corresponds to the green colour in the GUI is the sequence `true;false`.

One can also use Naproche from the command line. For this, start a shell and ensure that you are located in the base directory of Naproche. Enter `swipl` for starting SWI Prolog. Now enter `['naproche.qlf'].` for compiling the Naproche code. Now you can run Naproche by using the ternary Prolog predicate `naproche`. The first argument should be the input text as a Prolog atom, i.e. placed between two apostrophes. The second and third arguments should be entered as uninstantiated Prolog variables (e.g. capital letters), as they are output arguments. Here is an example of how to run Naproche on a short two-sentence input text:

```
naproche('Let $a=b$. Then $b=a$.',X,Y).
```

After executing this query, Prolog will return the following instantiations of the output arguments:

```
X = [sentence(1, [let, math([a, =, b])]),
sentence(2, [then, math([b, =, a])])],
Y = complete_text .
```

The first output argument is the tokenized and preprocessed version of the input text. The second output argument indicates whether the input text is a complete text: Because the parsing and proof-checking can be performed incrementally (also when using Naproche from the command line), it can make sense to enter an incomplete text, e.g. a theorem followed by an incomplete proof not yet ended with the keyword "Qed" that marks the end of a proof in the

[3]Since the implementation of the GUI was delegated to a member no longer active in the Naproche project, the debugging of the GUI has been paused.

Naproche CNL. In this case, the second output argument would be instantiated to `incomplete_proof`. But is as in the above example the input text is a complete Naproche CNL text, it will be instantiated to `complete_text`.

Note that LaTeX commands including backslashes, e.g. `\neq`, have to be entered with a double backslash, e.g. `\\neq`. when using Naproche from the command line. When a long input text is entered directly on the command line, Prolog will throw an error indicating that the input line is too long. In order to avoid this, one has to save the input text in aseparate file, say `a.txt`, and then read in this file from Prolog and call the `naproche` predicate as follows:

```
read_file_to_codes('a.txt',C,[]),atom_chars(A,C),naproche(A,X,Y).
```

For seeing whether the proof checking was successful when using Naproche from the command line, one has to look into the `.result` files in the sentence directories. Note that when using Naproche from the command line, the sentence directories are found in a directory called `output_folder` instead of in the directory `tmp`.

For displaying the PRS that represents the content of the input text, you need to execute the Prolog query `make_super_prs.`, which will produce a `prs.html` file in the `tmp` directory, which you can open with any modern web browser.

References

Abramsky, S., Artemov, S., Shore, R. A., & Troelstra, A. S. (1999). *Categorical logic and type theory.* Amsterdam: Elsevier.

Ackermann, W. (1956). Zur Axiomatik der Mengenlehre. *Mathematische Annalen*, *131*, 336–345.

Arndt, D. (2009). *Semantik und Korrektheit von Prolog-Programmen im Naproche-Projekt.* Diploma thesis, Universität Bonn.

Asher, N. (1993). *Reference to abstract objects in discourse.* Dordrecht, Netherlands: Kluwer Academic Publishers.

Avigad, J., Dean, E., & Mumma, J. (2009). A formal system for Euclid's Elements. *Review of Symbolic Logic*.

Banakh, T., & Zarichnyy, I. (2008). The coarse classification of homogeneous ultra-metric spaces. *Preprint*. (arXiv:0801.2132)

Bancerek, G. (2012). *MML query – statistics.* Available from `http://mmlquery.mizar.org/mmlquery/fillin.php?filledfilename=statistics.mqt&argument=number+1&version=4.181.1147`

Blackburn, P., & Bos, J. (2005). *Representation and inference for natural language. A first course in computational semantics.* Stanford: CSLI Publications.

Blackburn, P., Bos, J., & Striegnitz, K. (2006). *Learn Prolog now!* (Vol. 7). London: College Publications.

Bonk, M. (1992). On the second part of Hilbert's fifth problem. *Mathematische Zeitschrift*, *210*(1).

Brachman, R. J., & Levesque, H. J. (2004). *Knowledge representation and reasoning.* San Francisco: Morgan Kaufmann Publishers.

Bundy, A. (1988). The use of explicit proof plans to guide inductive proofs. In R. Lusk & R. Overbeek (Eds.), *Proceedings of the 9th conference on automated deduction.* (pp. 111–120). Springer.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, *5*, 56–68.

Clark, P., Harrison, P., Murray, W. R., & Thomson, J. (2010). Naturalness vs. predictability: A key debate in controlled languages. In N. E. Fuchs (Ed.), *Controlled natural language, LNAI 5972* (pp. 65–81). Springer.

Cohen, A. (2002). Genericity. *Linguistische Berichte*, *10*, 59–89.

Cohen, G. L. (2003). Derived sequences. *Journal of integer sequences*, *6*.

Connell, E. H. (1999). *Elements of abstract and linear algebra.* Available from `http://www.math.miami.edu/~ec/book/author.html`

Covington, M. A. (1994a). *Natural language processing for Prolog programmers.* Englewood Cliffs, NJ: Prentice Hall.

Covington, M. A. (1994b). *Research report AI-1994-06. GULP 3.1: An extension of Prolog for unification-based grammar.*

Covington, M. A. (2007, September). *Addendum to research report AI-1994-06.* Available from `http://www.ai.uga.edu/mc/ai199406.pdf`

Cramer, M. (2011). Komputlingvoscienca kaj logika analizado de matematikaj tekstoj. In P. Baláž & K. Nosková (Eds.), *Modernaj teknologioj por Esperanto – Prelegkolekto el KAEST 2010* (pp. 60–69). Espero.

Cramer, M. (2012). Implicit dynamic function introduction and its connections to the foundations of mathematics. In O. Prosorov (Ed.), *Proceedings of the International interdisciplinary conference on Philosophy, Mathematics, Linguistics: Aspects of interaction (PhML 2012)* (pp. 35–42).

Cramer, M., Fisseni, B., Koepke, P., Kühlwein, D., Schröder, B., & Veldman, J. (2010). The Naproche project – controlled natural language proof checking of mathematical texts. In N. E. Fuchs (Ed.), *Controlled natural language, LNAI 5972* (pp. 170–186). Springer.

Cramer, M., Koepke, P., Kühlwein, D., & Schröder, B. (2010). Premise selection in the Naproche system. In J. Giesl & R. Hähnle (Eds.), *Automated reasoning, LNAI 6173* (pp. 434–440). Springer.

Cramer, M., Koepke, P., & Schröder, B. (2011). Parsing and disambiguation of symbolic mathematics in the Naproche system. In J. Davenport, W. Farmer, F. Rabe, & J. Urban (Eds.), *Intelligent computer mathematics, LNAI 6824* (pp. 180–195). Springer.

Cramer, M., Kühlwein, D., & Schröder, B. (2010). Presupposition projection and accommodation in mathematical texts. In M. Pinkal, I. Rehbein, S. Schulte im Walde, & A. Storrer (Eds.), *Semantic approaches in natural language processing: Proceedings of the Conference on Natural Language Processing 2010 (KONVENS)* (pp. 29–36). Universaar.

Cramer, M., & Schröder, B. (2012). Interpreting plurals in the Naproche CNL. In M. Rosner & N. E. Fuchs (Eds.), *Controlled natural language, LNAI 7175* (pp. 43–52). Springer.

Damljanović, D. (2010). Towards portable controlled natural languages for querying ontologies. In N. E. Fuchs (Ed.), *CNL 2009, Pre-proceedings of the workshop on controlled natural language.*

de Bruijn, N. G. (1968). *Automath, a language for mathematics* (Tech. Rep.). Eindhoven University of Technology. (T.H.-Report 68-Wsk-05)

de Bruijn, N. G. (1994). Reflections on Automath. In R. P. Nederpelt, J. H. Geuvers, & R. C. de Vrijer (Eds.), *Selected papers on Automath* (pp. 201–228). Amsterdam: North-Holland.

Dedekind, R. (1872). *Stetigkeit und irrationale Zahlen.* Braunschweig: Vieweg.

Dedekind, R. (1888). *Was sind und was sollen die Zahlen?* Braunschweig: Vieweg.

Ebbinghaus, H.-D., Flum, J., & Thomas, W. (2007). *Einführung in die mathematische Logik.* Heidelberg: Spektrum.

Enderton, H. (1972). *A mathematical introduction to logic.* San Diego, CA: Academic Press.

Ferreirós, J. (2001). The road to modern logic – an interpretation. *The Bulletin of Symbolic Logic*, 441–484.

Fisseni, B. (2003). *Die Entwicklung einer Annotationssprache für natürlichsprachlich formulierte mathematische Beweise.* Magister thesis, Universität Bonn.

Frege, G. (1879). *Begriffsschrift. Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle a. S.: Louis Nebert.

Frege, G. (1884). *Die Grundlagen der Arithmetik: eine logisch-mathematische Untersuchung über den Begriff der Zahl*. Breslau: W. Koebner.

Frege, G. (1893). *Grundgesetze der Arithmetik*. Jena: Hermann Pohle.

Fuchs, N. E., Höfler, S., Kaljurand, K., Rinaldi, F., & Schneider, G. (2005). Attempto Controlled English: A knowledge representation language readable by humans and machines. In N. Eisinger & J. Maluszynski (Eds.), *Reasoning web, first international summer school 2005, LNCS 3564* (pp. 213–250). Springer.

Ganesalingam, M. (2009). *The language of mathematics*. PhD thesis, University of Cambridge.

Geach, P. T. (1962). *Reference and generality. an examination of some medieval and modern theories*. Ithaca, NY: Cornell University Press.

Gentzen, G. (1934/35). Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, *39*, 176–210, 405–431.

Groenendijk, J., & Stokhof, M. (1991). Dynamic predicate logic. *Linguistics and Philosophy*, *14*(1), 39–100.

Hales, T. (2005). Introduction to the Flyspeck project. *Mathematics, Algorithms, Proofs*.

Hardy, G. H., & Wright, E. M. (1960). *An introduction to the theory of numbers* (4th ed.). Oxford: Oxford University Press.

Hatcher, A. (2002). *Algebraic topology*. Cambridge: Cambridge University Press.

Heim, I. (1983). On the projection problem for presuppositions. *Second Annual West Coast Conference on Formal Linguistics*, 114–126.

Heuser, H. (1991). *Lehrbuch der Analysis, Teil 2* (6th ed.). Stuttgart: B.G. Teubner.

Hilbert, D. (1899). Die Grundlagen der Geometrie. In E. Wiechert & D. Hilbert (Eds.), *Festschrift zur Feier der Enthüllung des Gauss-Weber-Denkmals* (pp. 3–92). Leipzig: Teubner.

Hrbacek, K., & Jech, T. (1999). *Introduction to set theory* (3rd ed.). New York: Marcel Dekker.

Humayoun, M. (2012). *Developing the system MathNat for automatic formalization of mathematical texts*. PhD thesis, Université de Grenoble.

Jaśkowski, S. (1934). On the rules of suppositions in formal logic. *Studia Logica*, *1*.

Kadmon, N. (2001). *Formal pragmatics*. Oxford: Wiley-Blackwell.

Kaljurand, K. (2009). Paraphrasing controlled English texts. In N. E. Fuchs (Ed.), *Pre-proceedings of the workshop on controlled natural language (cnl 2009)*.

Kamareddine, F., Laan, T., & Nederpelt, R. P. (2004). *A modern perspective on type theory – from its origins until today*. Dordrecht, Netherlands: Kluwer Academic Publishers.

Kamp, H., & Reyle, U. (1993). *From discourse to logic: Introduction to model-theoretic semantics of natural languge*. Dordrecht, Netherlands: Kluwer Academic Publishers.

Klein, W., & Stutterheim, C. von. (1987). Quaestio und referentielle bewegung in erzählungen. *Linguistische Berichte*.

Koepke, P. (2006). *Naproche, version 3.0 – documented source code.* Available from `http://naproche.net/downloads/2006/Naproche_0.1.pl`

Koepke, P., & Koerwien, M. (2008). The theory of sets of ordinals. *Preprint.* (arXiv:math/0502265v1)

Kolev, N. (2008). *Generating Proof Representation Structures for the Project NAPROCHE.* Master's thesis, University of Bonn.

Kühlwein, D. (2009). *A calculus for Proof Representation Structures.* Diploma thesis, Universität Bonn.

Lackenby, M. (2008). *Topology and groups* [Lecture Notes]. Available from `http://people.maths.ox.ac.uk/lackenby/tg050908.pdf`

Landau, E. (1930). *Grundlagen der Analysis.* Leipzig: Akademische Verlagsgesellschaft.

LeVeque, W. J. (1962). *Elementary theory of numbers.* Ontario: Addison-Wesley Publishing Company.

Levinson, S. C. (1983). *Pragmatics.* Cambridge: Cambridge University Press.

Lévy, A. (1959). On Ackermann's set theory. *Journal of Symbolic Logic*, *24*(2), 154–166.

Lévy, A., & Vaught, R. (1961). Principles of partial reflection in the set theories of Zermelo and Ackermann. *Pacific Journal of Mathematics*, *11*(3), 1045–1062.

Link, G. (1991). Plural. In A. von Stechow & D. Wunderlich (Eds.), *Semantik / semantics. Ein internationales Handbuch zeitgenössischer Forschung* (pp. 418–440). Berlin: de Gruyter.

Lyaletski, A., & Verchinine, K. (2010). Evidence algorithm and system for automated deduction: A retrospective view. In S. Autexier et al. (Eds.), *Intelligent computer mathematics, LNAI 6167* (pp. 411–426). Springer.

Lyaletski, A., Verchinine, K., & Paskevich, A. (2008a). *System for Automated Deduction (examples).* Available from `http://nevidal.org/help-txt.en.html#examples`

Lyaletski, A., Verchinine, K., & Paskevich, A. (2008b). *System for Automated Deduction (web interface).* Available from `http://nevidal.org/cgi-bin/sad.cgi?ty=txt`

Martin-Löf, P. (1984). *Intuitionistic type theory.* Naples: Bibliopolis.

Matuszewski, R., & Rudnicki, P. (2005). Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, *4*, 3–24.

Montague, R. (1961). Fraenkel's addition to the axioms of Zermelo. In Y. Bar-Hillel, E. I. J. Poznanski, M. O. Rabin, & A. Robinson (Eds.), *Essays on the foundations of mathematics* (pp. 91–114). Jerusalem: Magnes Press.

Moschovakis, J. (2010). Intuitionistic logic. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy.* The Metaphysics Research Lab. Available from `http://plato.stanford.edu/archives/sum2010/entries/logic-intuitionistic/`

Naumowicz, A., & Korniłowicz, A. (2009). A brief overview of Mizar. In S. Berghofer, T. Nipkow, C. Urban, & M. Wenzel (Eds.), *Theorem proving in higher order logics* (pp. 67–73). Springer.

Pace, G., & Rosner, M. (2010). A controlled language for the specification of contracts. In N. E. Fuchs (Ed.), *Controlled natural language, LNAI 5972* (pp. 226–245). Springer.

Pasch, M. (1882). *Vorlesungen über neuere Geometrie.* Leipzig: Teubner.

Paskevych, A. (2007). *Méthodes de formalisation des connaissances et des raisonnements mathématiques : aspects appliqués et théoriques.* PhD thesis, Université Paris XII.

Pereira, F. C. N., & Warren, D. H. D. (1980). Definite clause grammars for language analysis—a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 231–278.

Ranta, A. (1994). Type theory and the informal language of mathematics. In H. Barendregt & T. Nipkow (Eds.), *Types for proofs and programs, LNCS 806* (pp. 352–365). Springer.

Ranta, A. (1995). Syntactic categories in the language of mathematics. In P. Dybjer, B. Nordström, & J. Smith (Eds.), *Types for proofs and programs, LNCS 996* (pp. 162–182). Springer.

Ranta, A. (1996). Context-relative syntactic categories and the formalization of mathematical text. In S. Berardi & M. Coppo (Eds.), *Types for proofs and programs, LNCS 1158* (pp. 231–248). Springer.

Ranta, A. (1997a). Structure grammaticales dans le français mathématique I. *Mathématiques, Informatique et Sciences Humaines*, 5–56.

Ranta, A. (1997b). Structure grammaticales dans le français mathématique II (suite et fin). *Mathématiques, Informatique et Sciences Humaines*(139), 5–36.

Reck, E. (2011). Dedekind's contributions to the foundations of mathematics. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy.* The Metaphysics Research Lab. Available from `http://plato.stanford.edu/archives/fall2011/entries/dedekind-foundations/`

Reinhardt, W. (1970). Ackermann's set theory equals ZF. *Annals of Mathematical Logic*, *2*, 189–249.

Russell, B. (1959). *My philosophical development.* London: George Allen and Unwin.

Schulz, S. (2004). System abstract: E 0.81. In D. Basin & M. Rusinowitch (Eds.), *Proceedings of the 2nd IJCAR, Cork, Ireland.* Springer. Available from `http://www4.informatik.tu-muenchen.de/~schulz/E/E.html`

Schwitter, R. (2010). Controlled natural languages for knowledge representation. In C.-R. Huang & D. Jurafsky (Eds.), *Proceedings of the 23rd international conference on computational linguistics (coling 2010)* (pp. 1113–1121). Association for Computational Linguistics.

Simon, D. L. (1990). *Checking number theory proofs in natural language.* PhD thesis, University of Texas at Austin.

Steinhardt, F. (trans., 1951). *Foundations of analysis.* Bronx, NY: By E. Landau. Chealsea Publishing Company.

Sutcliffe, G. (2009). The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, *43*(4), 337–362.

Sutcliffe, G., & Suttner, C. (1998). The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, *21*(2), 177–203.

Torretti, R. (2010). Nineteenth century geometry. In E. N. Zalta (Ed.), *The Stanford encyclopedia of philosophy.* The Metaphysics Research Lab. Available from `http://plato.stanford.edu/archives/sum2010/entries/geometry-19th/`

Trench, W. (2003). *Introduction to real analysis.* Upper Saddle River, NJ: Prentice Hall.

van der Hoeven, J. (2011). *TeXmacs.* Available from `http://www.texmacs.org/` (retrieved December 14, 2012)

Velleman, D. J. (2006). Variable declarations in natural deduction. *Annals of Pure and Applied Logic*, *144*, 133–146.

Volkert, K. (1988). *Geschichte der Analysis.* Mannheim: Wissenschaftsverlag.

White, C., & Schwitter, R. (2009). An update on PENG Light. In L. Pizzato & R. Schwitter (Eds.), *Proceedings of ALTA 2009* (pp. 80–88).

Whitehead, A. N., & Russell, B. (1910, 1912, 1913). *Principia mathematica.* Cambridge: Cambridge University Press.

Wiedijk, F. (2008). *The seventeen provers of the world.* Berlin: Springer.

Wiedijk, F. (2009). Formalizing Arrow's theorem. *Sādhanā*, *34*, 193–220.

Wielemaker, J., Schrijvers, T., Triska, M., & Lager, T. (2012). SWI-Prolog. *Theory and Practice of Logic Programming*, *12*(1-2), 67–96.

Wolfenstein, S. (1969). *Introduction to linear algebra and differential equations.* San Francisco: Holden-Day.

Zinn, C. (2004). *Understanding Informal Mathematical Discourse.* PhD thesis, Friedrich-Alexander-Universität Erlangen Nürnberg.

Zittermann, S. (2011). *Entwicklung des Naproche-Proof-State-Datentyps.* Master's thesis, Fachhochschule Köln.

Index of symbols

Symbolic expressions are listed in the order of their first occurrence in the thesis. Note that combined symbolic-textual expressions like "Φ-map" are listed in the general index and not in this index of symbols.

Index

Combined symbolic-textual expressions like "Φ-map" are listed in this index according to their usual verbalization, e.g. "Φ-map" under "Phi-map".