Different Degrees of Formality

An Introduction to the Concept and a Demonstration of its Usefulness

Sebastian Böhne

A Thesis Presented for the Degree of
Doctor Rerum Naturalium
in Theoretical Computer Science

University of Potsdam
Faculty of Science
Institute of Computer Science

Potsdam, Germany
2019-01-17

Supervisor: Prof. Dr. Christoph Kreitz
Mentor: Prof. Dr. Christoph Benzmüller
Reviewers: Prof. Dr. Christoph Kreitz
Prof. Dr. Christoph Benzmüller
Prof. Dr. Cezar Ionescu

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-42379
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-423795

Abstract

In this thesis we introduce the concept of the degree of formality. It is directed against a dualistic point of view, which only distinguishes between formal and informal proofs. This dualistic attitude does not respect the differences between the argumentations classified as informal and it is unproductive because the individual potential of the respective argumentation styles cannot be appreciated and remains untapped.

This thesis has two parts. In the first of them we analyse the concept of the degree of formality (including a discussion about the respective benefits for each degree) while in the second we demonstrate its usefulness in three case studies. In the first case study we will repair Haskell B. Curry's view of mathematics, which incidentally is of great importance in the first part of this thesis, in light of the different degrees of formality. In the second case study we delineate how awareness of the different degrees of formality can be used to help students to learn how to prove. Third, we will show how the advantages of proofs of different degrees of formality can be combined by the development of so called tactics having a medium degree of formality. Together the three case studies show that the degrees of formality provide a convincing solution to the problem of untapped potential.

Contents

Chapter 1

Introduction

In science most problems, such as global warming, are easy to recognise but hard to solve. By contrast, in mathematics often to understand the problem is already a major part of it. Furthermore communication of mathematical problems usually requires that the recipient is mathematically sophisticated and sometimes even has some deep knowledge of the specific domain under consideration. This thesis is about a concept addressing a problem that pertains to mathematics but to informatics[1] and the philosophy of mathematics as well. Somewhat naively one might then expect that the effort to clarify the problem might be in between, which is indeed the case. In the following we will give four different examples of mathematical argumentation that will provide the context we need.

In figure 1.1 we can see two (quite good) students discussing their homework, which – incidentally – belongs to the domain of automata theory. The first student communicates an idea for proving that $\text{half}(L) := \{w \in \Sigma^* \mid \exists v \in \Sigma^*, |w| = |v| \wedge wv \in L\}$ is a regular language if L is. However, we are not interested in concrete content but in the form of argumentation at hand. Although regular languages are defined by the existence of suitable regular expressions the outlining student – without further explanation – chooses an automaton to start with in his argumentation. However, he does not specify which kind of automaton he is using (there are DFAs, NFAs, or ε-NFAs for instance). Then the student talks rather picturesquely about the automaton going forward and backward. Yet, mathematical objects do not move. There is only a so called transition function. Furthermore there is no constituent in any of the before mentioned kinds of automata that could be associated with the backward direction the student is adducing. The start "at both ends" leads to similar problems. Also, the student does not tell us what "both directions meet" means. Finally, the argumentation ends with the construction of

[1] In non-institutional contexts we avoid naming the respective discipline 'computer science' in this thesis.

Figure 1.1: Two (quite good) students discussing their homework.

the automaton without mentioning the latter's relation to the statement of the theorem. So what is characteristic in this argumentation are the imprecisions and omissions.

In the next figure we see a lecturer explaining some technical steps somewhere in the middle of some proof. We enter it when only some technical equation, in concreto $\xi = y^2 + y + y * z + x$, remains to be shown. There is some chain of equations indicated, starting with ξ and ending with $x + y * (z + 1) + y^2$. The lecturer points to the salient position of the latter term where something will be happening in the next step that concludes the proof. Note, however, that his finger together with the word 'here' are only vague hints. Note further that one obtains $x + (y * z + y * 1) + y^2$, not $y^2 + y + y * z + x$, as result after expanding. For showing that the latter two terms are indeed equal one would have to use the associativity law as well as the commutativity law several times.[2]

In figure 1.3 we see an already very technical looking textbook proof for the theorem $\langle x, y\rangle = \langle u, v\rangle \to x = u \wedge y = v$.[3] The involved formulae, all of which having a very basic structure, are the predominant part. They are accompanied by fragments of natural language roughly connecting them.

[2] Furthermore some convention is needed when parentheses can be omitted.

[3] The proof stems from [80, p. 230] while the idea to compare it with a much more formalised version is taken from [55, subsubsection 3.2.1.3].

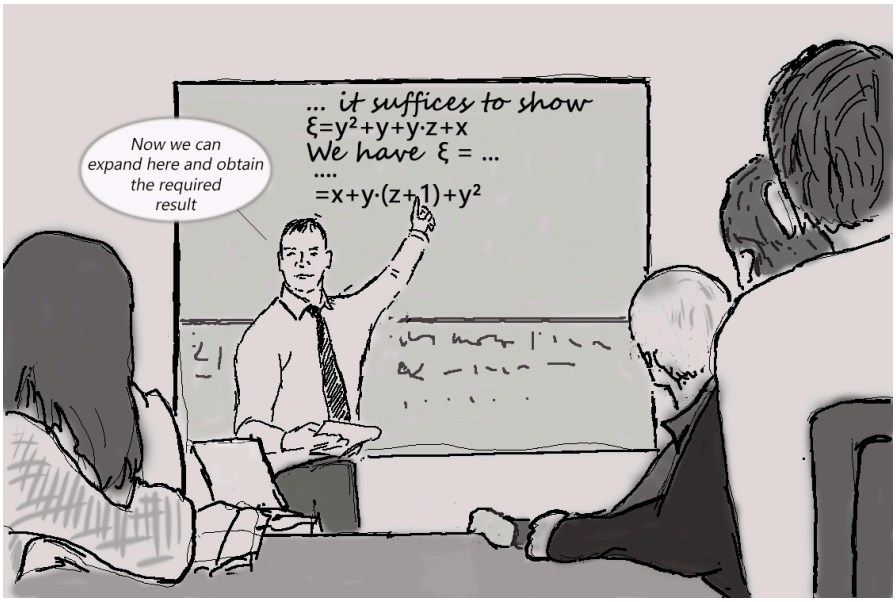

Figure 1.2: Lecturer explaining some technical steps somewhere in the middle of some proof.

Theorem: $\langle x, y\rangle = \langle u, v\rangle \to x = u \wedge y = v$.
Proof: "Assume $\langle x, y\rangle = \langle u, v\rangle$. Then $\{\{x\}, \{x, y\}\} = \{\{u\}, \{u, v\}\}$. Since $\{x\} \in \{\{x\}, \{x, y\}\}, \{x\} \in \{\{u\}, \{u, v\}\}$. Hence, $\{x\} = \{u\}$ or $\{x\} = \{u, v\}$. In either case, $x = u$. Now, $\{u, v\} \in \{\{u\}, \{u, v\}\}$; so, $\{u, v\} \in \{\{x\}, \{x, y\}\}$. Then, $\{u, v\} = \{x\}$ or $\{u, v\} = \{x, y\}$. Similarly, $\{x, y\} = \{u\}$ or $\{x, y\} = \{u, v\}$. If $\{u, v\} = \{x\}$ and $\{x, y\} = \{u\}$, then $x = y = u = v$; if not, $\{u, v\} = \{x, y\}$. Hence, $\{u, v\} = \{u, y\}$. So, if $v \neq u$, then $y = v$; if $v = u$, then $y = v$. Thus, in all cases, $y = v$."

Figure 1.3: A textbook proof of $\langle x, y\rangle = \langle u, v\rangle \to x = u \wedge y = v$.

1. $\langle x, y\rangle = \langle u, v\rangle \vdash \{\{x\}, \{x, y\}\} = \{\{u\}, \{u, v\}\}$ (Def)
2. $\vdash \{x\} \in \{\{x\}, \{x, y\}\}$ (H1)
3. $\vdash \{x\} \in \{\{x\}, \{x, y\}\} \rightarrow$
   $\{\{x\}, \{x, y\}\} = \{\{u\}, \{u, v\}\} \rightarrow \{x\} \in \{\{u\}, \{u, v\}\}$ (H4)
4. $\vdash \{\{x\}, \{x, y\}\} = \{\{u\}, \{u, v\}\} \rightarrow \{x\} \in \{\{u\}, \{u, v\}\}$ (MP, 2,3)
5. $\langle x, y\rangle = \langle u, v\rangle \vdash \{x\} \in \{\{u\}, \{u, v\}\}$ (MP, 1,4)
6. $\vdash \{x\} \in \{\{u\}, \{u, v\}\} \rightarrow (\{x\} = \{u\} \vee \{x\} = \{u, v\})$ (H3)
7. $\langle x, y\rangle = \langle u, v\rangle \vdash (\{x\} = \{u\} \vee \{x\} = \{u, v\})$ (MP, 5,6)

...

Figure 1.4: Excerpt from a detailed proof of $\langle x, y\rangle = \langle u, v\rangle \rightarrow x = u \wedge y = v$.

Let us contrast this with the beginning of a much more detailed proof for the same theorem (figure 1.4).[4] In full the proof contains 96 steps although it already uses a bunch of lemmata! For every step there is an explicit justification; a lemma (H1, H3, and H4 in this proof fragment[5]) or the modus ponens with the lines it is applied to.

Usually only the last example would be classified as formal while all previous ones would be called informal.[6] This dualistic attitude is problematic per se since the differences between the first and the third example, for instance, are greater than the ones between the third and the fourth example. Even worse, this division is unproductive because the individual potential of the respective argumentation styles cannot be appreciated. The alternative concept of the *degrees of formality* presented in this thesis is intended to solve this problem of untapped potential. With the help of the degrees of formality we will be able to underline the advantages of the different argumentation styles in such a clear manner that they can be combined to solve problems of different kinds.

This thesis has two parts. In the first of them we analyse the concept of the degree of formality (including a discussion about the respective benefits for each degree) while in the second we demonstrate its usefulness in three case studies.

[4] This proof is extracted from [55, p. 168-171]. In the original there are fragments of the proof of figure 1.3 on the left side of the numbers. We omitted them since we did not want to blur the impression of the different proof styles.

[5] The numeration is taken from [55, p. 168-171]. H2 will appear later in the proof.

[6] This rough distinction is even made in the philosophy of mathematics, in which terminology usually is very well-considered (see [34, 94, 108] for instance).

In concreto, the first part is structured as follows. We will start with a systematic overview of the concept of the degree of formality (chapter 2). We will not only consider different argumentation styles but also different system styles. Furthermore the overview comprises an introduction to the Coq proof assistant, a software for proving theorems in mathematics. In the third chapter we will present Haskell B. Curry's view of mathematics. This is done for four reasons. First, his treatment of logics[7] prepares our analogous treatment of formalisms, which in turn will help us to further clarify the concept of the degrees of formality (chapter 4). Second, Curry makes an important conceptional split that – third – justifies an attitude with pluralistic tendencies. Both points will help us in chapter 5, in which we will work out the benefits of the different degrees of formality step by step. Finally, Curry's view will be the subject of one of the three case studies in the second part of this thesis.

The second part, i. e. the demonstration of the usefulness of the concept of the degree of formality, begins – as already indicated – with a review of Curry's view of mathematics. This time, however, we will not consider the positive aspects but discuss why his view is flawed and how it can be repaired in light of the different degrees of formality (chapter 6). While this topic belongs to the philosophy of mathematics the following chapter is of a didactical nature. We will investigate how different degrees of formality can help students to learn how to prove in a textbook manner (chapter 7). The last case study can but does not have to be seen from a didactical point of view. We will combine the advantages of proofs of different degrees of formality by the development of tactics[8] having a medium degree of formality (chapter 8).[9] Although the idea is quite simple the cogency of the argument depends on elaborate technical evidence. At the end of the second part there will be a summary and conclusion.

This thesis also comprises a listing of most of the user defined tactics representing a logical or equational rule (appendix A) and a listing of all tactics of a medium degree of formality belonging to our approach as presented in chapter 8 (appendix B).

There are some things to be said about the style of this thesis. First, it is – at least in some aspects – intended to be close to spoken language. In particular '–' and ';' are used more often than usual. Then, besides the impersonal form (and except for the next paragraph) only the pluralis auctoris is used. This is in congruence with most works stemming from mathematics and informatics but it is uncommon in the philosophy of mathematics. Furthermore, in contrast to many modern texts (especially in the philosophy of mathematics) 'he' is used instead of

[7] It will become clear later on why we use 'logics' instead of just 'logic'.

[8] The notion of a tactic will be explained in section 2.6.

[9] A justification for the adjective medium will be given at the end of section 2.3.

'she' whenever the gender is unspecified. This is done for the sake of readability and the closeness to spoken language. Finally, there is a frequent use of the word 'contentual', which is a translation of the German word 'inhaltlich'. This word comprises everything except for the mere form or outer appearances of the matter under consideration. It refers to the substance or the core of the matter.

It would not have been possible for me to develop and complete this thesis without great support from others. First and foremost I would like to thank my supervisor Prof. Dr. Christoph Kreitz who supported me in so many ways. The regular meetings in which we reflected the progress of my work and discussed my documents were particularly helpful. Furthermore, I especially appreciate that he gave my much freedom regarding my research and that he constantly encouraged me to pursue the path I had chosen. In short, he did a fantastic job as a supervisor. Then, I would like to give special thanks to Tim Richter. Learning type theory or proof assistants with him has always been a great pleasure. Furthermore, without his technical support I would not have been able to realise even a single of my proof-assistant-related ideas. I would like to thank the whole research group of theoretical computer science, which are – apart from the two already mentioned – Nuria Brede, Mario Frank, and Dr. Eva Richter, for the fruitful discussions and their constructive feedback during and after my talks. Then I would like to thank Prof. Dr. Christoph Benzmüller for his willingness to be my mentor and for helping me whenever I asked for this help. There is a continuous course called the Cartesian Seminar at the University of Potsdam. I would like to thank Prof. Dr. Cezar Ionescu, Dr. Nicola Botta, and all other participants of this seminar for the methodological training and their lack of willingness to remain silent when they did not agree with my opinion. Many thanks also to PoGS for their 'Promotionscoaching' programme and to the interdisciplinary team of success arising from it, which helped me to solve all kinds of problems standing in my way when doing my PhD. Thanks to my father Klaus Böhne who provided the two pictures of this introduction and thanks to Katharina Leitner for her help concerning language aspects in different parts of this thesis, her willingness to lighten my external loads in the final phase of my PhD, and for her moral support. I would like to say thank you to my family and all those who supported me!

Part I

Analysis of the concept of the degrees of formality

Chapter 2

Systematic overview of the concept

2.1 Introduction

So far we have seen four argumentations (three of them also being proofs) of which we said that they have different degrees of formality. However, we did not clarify precisely what we mean by that. The purpose of this chapter is to rectify that omission.

In concreto we are going to present different aspects of formalisation that took place in (the history of) mathematics (section 2.2) and discuss what it is that makes some parts of modern mathematics more formal than others (section 2.3). We will enrich the discussion by an investigation of different formalisations having different degrees of formality. This will be done for the system complex of natural numbers (section 2.4) as well as for the proof and argumentation complex of commutativity of addition therein (section 2.5). At the close of this chapter we will introduce the Coq proof assistant and elaborate when and why the so called proof scripts in Coq can be seen as proofs having a high degree of formality (section 2.6).

2.2 What is being formalised? A historical review

Let us investigate chronologically which entities of mathematics have been formalised. We start with pre-Greek mathematics like that of the Egyptians or Babylonians. In that cultures real world objects like fields or pyramids (as buildings) were assigned geometrical forms, in this case rectangle and pyramid (in the geometrical sense), accompanied by length specifications. With the further help

of formulae new numbers could be computed, like the area of the rectangle or the volume of the pyramid. These numbers in turn could be assigned a specific real world value, such as the crop of a field or the needed building material.[1] So the pre-Greek mathematics formalised real world objects aiming for congruence with practice.[2]

By contrast, some of the ancient Greeks lost interest in practice, probably when slaves took over the Greeks' daily work.[3] So mathematics had to prove itself outside of practice. Without empirical feedback, however, the quality of reasoning had to become the measuring stick. Quality of reasoning in turn depends – as one factor – on its clarity. Therefore it is not surprising to find a formalisation of reasoning in the best known Greek mathematical achievement, i. e. Euclid's Elements [42]. At the beginning of this work's chapters so called definitions, postulates, and axioms are framed.[4] It follows a series of successive theorems. Each step in the respective proofs has an explicit justification, for instance a postulate, an axiom, or a previous result. In these justifications often the exact same wording is used, which means that there is some sort of syntactical form involved.[5] Admittedly, from a modern perspective not all arguments in Euclid's Elements are that accurate and hidden assumptions and even fallacies may be found. What is interesting for us, however, is that a formalisation of reasoning was tackled at all rather than its complete success.

The mathematics articulated in the elements is rather verbose while today we use formal symbols for variables, functions, and predicates like in the simple equation $n + m = l$. This can be seen as a formalisation of the appearance of mathematical objects. It fosters the symbolic calculation and enables the communication of complex formulae.

Albeit quite useful the use of symbols is a rather superficial formalisation step. By contrast, in the course of the arithmetisation of analysis mathematical entities themselves were formalised by reducing them to simpler and clearer ones:

[1] Morris Kline in [65, section 2.4] adduces further examples of the use of mathematics by the Egyptians at that time.

[2] That this and not mathematics per se was the true goal is confirmed by some formulae being wrong on purpose. The formula for computing the area of quadrangles, for instance, was a very rough approximation but good enough in the relevant cases (see [65, section 2.3]).

[3] This and the following consequence are discussed in [65, section 3.8]. The fragment 'some of' is an addition from us since Jens Høyrup in [62] argues that there were many Greeks doing mathematics like the Babylonians.

[4] These notions do not coincide with their modern usage.

[5] Let us consider two fragments of the first two proofs as instances. "Thus, CA and CB are each equal to AB. But things equal to the same thing are also equal to one another [C.N. 1]. Thus, CA is also equal to CB [...]." ([42, book 1, proposition 1]) "Thus, AL and BC are each equal to BG. But things equal to the same thing are also equal to one another [C.N. 1]. Thus, AL is also equal to BC [...]" ([42, book 1, proposition 2]).

infinitesimals were replaced by a formulation of propositions that used only finite numbers and quantifiers[6], complex numbers and imaginary numbers in particular were conceived as pairs of real numbers in the complex plane, and the real numbers themselves were explained by Dedekind cuts, i. e. via sets of rationals.

Such a reduction process cannot go on forever. There must be some primitive mathematical objects we cannot define with the help of other mathematical objects. Instead the primitive objects' behaviour can be made explicit by axioms and logical rules. For the sake of clarity and to avoid any confounding effect that may presuppose some intuition, these have to be formalised. This can be realised by the development of mathematical theories or by the development of theories of mathematics. A paragon of the former is David Hilbert's book 'Foundations of Geometry' [54] while the latter project is epitomised by Gottlob Frege's writings [45, 46, 47] as well as by Alfred N. Whitehead's and Betrand Russell's 'Principia Mathematica' [117].

2.3 Three adjusting parameters for degrees of formality

In the previous section we became acquainted with different manifestations of formalisation. Not all of them, however, can be deliberately influenced in modern mathematics. The restriction to the formalisation of real world objects is a first example of a low degree of formality one cannot restore. Furthermore, in general one cannot avoid the use of formulae or formal symbols since they have become an inherent part of mathematics. The same is true for the cumulative reasoning frame. One might refrain from (explicit) axioms but the theorems and argumentations have to build upon each other.

By contrast, – as we have seen in the introduction – reasoning itself, i. e. the argumentation style, can be varied entailing different degrees of formality. The same is true regarding the formalisation of systems; where we use the concept of a system as a lower degree generalisation of a theory comparable to the distinction between the concept of argumentation and that of proof.[7] Admittedly, one can

[6] One example is convergence. Instead of saying that a sequence (a_n) has limit a if for every infinite n the equation $a_n = a$ holds (or something similar), we can define: (a_n) has limit a if $\forall\, \varepsilon > 0\ \exists\, N \in \mathbb{N}\ \forall\, n \geq N\ (|a_n - a| < \varepsilon)$. We may write $\lim\limits_{n\to\infty} a_n = a$, but infinity does not occur in the definition.

[7] Stewart Shapiro defines "a system to be a collection of objects with certain relations among them." ([104, p. 259]) He then continues: "Define a *pattern* or *structure* to be the abstract form of a system, highlighting the interrelationships among the objects, and ignoring any features of them that do not affect how they relate to other objects in the system." ([104, p. 259], all emphases are in the original text) These definitions – we think – are more or less in accord with

prevent neither that there are ways to define the objects in the system by other mathematical objects lying outside the system nor that the behaviour of the objects can be characterised by elaborated mathematical theories about them. Yet, one does not have to care. Ignoring these achievements usually has no negative effects on ones mathematics. Instead of using such elaborated theories one may stick to vague systems having a lower degree of formality.

Let us now discuss means by which one can vary the degree of formality in both cases, argumentations and systems. The most obvious point with respect to argumentations is the *use of natural language*. Already our four examples in the introduction suggest a great variety concerning this matter. The verbal discussion of the students' homework is almost completely uttered in natural language. The latter is used to generate a mental representation in the mind and in fact diagrams etc. can be used as a supplement in (or sometimes even as an alternative for) such argumentations. The lecturer does not use too much natural language on the blackboard. However, natural language has still a pivotal role in the verbal explanation of what he is doing. In the textbook proof natural language is used in a minimal but still important way to structure the proof. The given example is somewhat extreme since other textbook proofs are more verbose. Yet, the tendency is right. The amount of natural language (or alternatives such as diagrams) decreases. In the last example then there is no natural language left.

The least formal systems are probably just representations in our brain. We apply such systems in a way that is similar to the unconscious application of grammar when we speak in our native language. We do not want to discuss here whether the representations are somehow given in natural language. Instead let us focus only on those systems that can be communicated somehow. For these it is the same as with argumentations: the use of natural language reduces the degree of formality. In particular, the most formal systems (or theories) do not use any natural language.

Another important point for the degree of formality in argumentation is *condensation*. That one proof is more condensed than another means that one step of the former usually corresponds to many steps of the latter. For instance, one step of a proof idea corresponds to several steps in a textbook proof and all the respective steps there correspond themselves to many steps in a proof that one would traditionally classify as formal (like the fourth example of the introduc-

our understanding of systems of a low respectively high degree of formality (see below for this terminology). The disadvantage of Shapiro's definitions is the dualistic attitude behind it. As we will see in this chapter this is not better than to only distinct between formal and informal proofs.

tion). Instead of saying that one argumentation is more condensed than another we sometimes alternatively state that the former has less *granularity* than the latter.

In case of systems the impact of condensation is inverted. The more formal a system is the more it will be condensed meaning that it will use only a (small) amount of axioms and logical rules; the salient point being the word 'only' not the word 'small', i. e. nothing else will be used implicitly. By contrast, implicit redundancies lessen the degree of formality. In particular a system of arithmetic in which natural numbers and reals are conceived of as some kind of intuitive primitives has a lesser degree of formality than one in which the reals are ultimately defined by the natural numbers; even when the respective concept of natural numbers does not refine. Another aspect that reduces formality in systems is the inclusion of vague associations like, for instance, the visual appearance of geometrical objects.

In case of argumentations there is one further aspect we want to emphasise, namely *frequency and precision of justifications*. By justifications we mean that parts of the overall argumentation that – if existent – explain the transition from one step to the succeeding one. If we do not offer justifications at all we do not have any proof (yet we may have a proof idea). However, even if we do offer justifications we can still do it in very different ways. We can give only short hints or be very precise. The former is what is usually done in textbook proofs. Some theorems are cited by their names or their statements as justifications for the most relevant steps but it is usually not stated in which way they are applied. In set theory, for instance, the Schröder-Bernstein theorem might be our justification when reasoning about cardinality, but we do not say explicitly with which sets we use it nor do we state which of the many formulations of the theorem we use. By contrast, in those proofs that are traditionally the only ones classified as formal we need to refer to a concrete formulation of the theorem (the H_i in the fourth example of the introduction) and – when appropriate – apply some values (the modus ponens with the respective lines in this example).

In principle any combination of the three adjusting parameters (two in the case of systems) is imaginable. However, most of the time they all point in the same direction. This in turn allows us to speak in a simplifying way of argumentations/proofs (and systems/theories) of a high, medium, or low degree of formality. As an abbreviation we will often just omit the 'of formality' or speak of high, medium, or low degree argumentations/proofs (and systems/theories).

Note that so far we discussed the concept of the degree of formality for argumentations and systems without explaining what a formalism is. After some preparation we will rectify this omission in chapter 4.

2.4 Natural number systems of different degrees of formality

Up to now our analysis was rather abstract. In this section we will concretise our considerations regarding the degree of systems by presenting different formalisations of natural numbers. We choose this domain as object of study since it is the most simple one we have that already allows to gain all relevant insides.[8]

The first possibility is to conceive the natural numbers as a part of another, more basic, system. In concreto this means that natural numbers and the relevant functions – like successor, addition, and multiplication – have to be defined by mathematical objects of the more basic system (of course without vicious circles). In set theory, for instance, this can be done by the Zermelo or Von Neumann implementation. Both define $0 := \emptyset$ and the successor as a class function on sets. In the concrete definition the two versions part: the Zermelo version defines $\mathrm{Suc}\, x := \{x\}$ and obtains $n = \underbrace{\{\ldots\{}_{n\text{-times}}\underbrace{\emptyset\}\ldots\}}_{n\text{-times}}$ while the Von Neumann version defines $\mathrm{Suc}\, x := x \cup \{x\}$ and obtains $n = \{0, \ldots, n-1\}$. Both can then define the set of natural numbers as the smallest inductive set, i. e. the smallest set containing $\emptyset$ that is closed under the successor operation.[9] In both variants addition and multiplication can then be defined recursively by set theoretical means.

Let us now turn our attention to systems for natural numbers that are conceived as autonomous. The variants of lowest degree cannot be written down but they are unconscious representations in our brain. They might comprise a lot of concrete natural numbers (and their names), as well as knowledge of how to count, add, multiply, divide etc.

Next, we consider a system of medium degree. It is characterised by the following axioms called Peano axioms:

1. 0 is a natural number.
2. There is a successor function S, mapping natural numbers to natural numbers.
3. S is an injection.
4. 0 is not the successor of any number.[10]
5. The induction principle: A set that contains 0 and with each natural number its successor contains all natural numbers.

[8] The same will be true for other following considerations, which is why arithmetic will serve as a running example in this thesis.

[9] The Existence of such a set is guaranteed by the axiom of infinity together with the possibility to intersect sets.

[10] The last axiom will ensure that all other numbers are indeed successors.

The system uses natural language but it does so in a precise manner. Furthermore some of the formulations can be understood as abbreviations for corresponding statements not using natural language and other formulations like 'set' or 'function' refer to technical notions. The system is axiomatised and the axioms are independent from each other. Yet, the apparent condensation level is deceptive. First of all, the whole logic part is omitted and second, there is some background theory needed explaining things like functions, injectivity, and sets.

Let us now watch a first and a second order formalisation of high degree in parallel by going through each single axiom above (with the formalisation of the first four axioms being the same in both cases).

1. The first attempt to state that 0 is a natural number might be something like $0 \in \mathbb{N}$. $\mathbb{N}$, however, is what we want to capture with the help of our axioms, we do not want to presuppose it. So we might want to state 0 as an axiom. This, of course, is silly, since 0 is not even a proposition. If our universe is $\mathbb{N}$ then all we have to guarantee is that we can state 0 as part of formulae, viz. 0 is a legal expression.[11] For the same reason we have to allow variables as expressions, too.

2. The situation regarding S is similar: we cannot state $S : \mathbb{N} \to \mathbb{N}$ or something like that. A function is something semantical, but we want to have a syntactical approach. The decisive property of a function as opposed to a relation is that for all inputs there is exactly one output. However, when we write $S(\sigma)$ for some input σ syntactically this already is its output. Therefore we only have to ensure that $S(\sigma)$ is an expression, whenever σ is.

3. That S is an injection is only an abbreviation of the following already formalised statement: $\forall\, n\, (S(n) = S(m) \to n = m)$.

4. We have a straightforward translation for 0 not being the successor of any number, namely $\neg\, \exists\, n\, (S(n) = 0)$.

5. At first glance the high degree formalisation of the induction principle seems to be straightforward, too: $\forall\, A\, (0 \in A \wedge \forall\, n \in \mathbb{N}\, (n \in A \to S(n) \in A) \to \forall\, n \in \mathbb{N}\, (n \in A))$. The salient point here is that we have to deal with two different kinds of quantification, one over natural numbers and one over sets of natural numbers.

   In second order logic this is not too much a problem. We conceive the sets of natural numbers as predicates being true for exactly those natural

[11] One could argue, of course, that this is not a proper translation of the above statement. Indeed, formalisation often causes losses with respect to some informal aspects.

numbers that are in the set. We obtain $\forall\,\mathbf{P}\,(\mathbf{P}(0) \wedge \forall\, n\,(\mathbf{P}(n) \to \mathbf{P}(S(n))) \to \forall\, n\,(\mathbf{P}\ n))$, whereby the bold print tells us that it is not a natural number we quantify over but a property.[12]

Unfortunately, things are not that easy in first order logic since we are not allowed to quantify over predicates. The solution is to have infinitely many axioms instead, one for each formula with exact one free variable, i. e. for all first order formulas ϕ with exact one free variable we have the following axiom: $\phi(0) \wedge \forall\, n\,(\phi(n) \to \phi(S(n))) \to \forall\, n\,(\phi(n))$. This is in a rigorous sense not a proper formalisation of the corresponding Peano axiom above. It is a formalisation of something different[13] – albeit similar. Nevertheless what we are constructing (we are not finished yet) is called Peano arithmetic.

6. Although we have translated every single axiom from the above version we are surprisingly not finished yet. High degree formalisations restrict our possibilities to define new functions; in our case addition and multiplication have to be definable or the theory must be expanded.

   In second order logic we can define addition and multiplication in the following sense: we find a formula $\phi(n, m, l)$ representing $n + m = l$ ($n * m = l$ respectively). Then we can take the former as definiens for the latter; $n+m = l :\leftrightarrow \phi(n, m, l)$ ($n * m = l :\leftrightarrow \psi(n, m, l)$ respectively). For addition we define $\phi(n, m, l) := \forall\,\mathbf{f}\,(\mathbf{f}(0) = m \wedge \forall\, k\,(\mathbf{f}(S(k)) = S(\mathbf{f}(k))) \to \mathbf{f}(n) = l)$. Similarly we can define $\psi(n, m, l) := \forall\,\mathbf{f}\,(\mathbf{f}(0) = 0 \wedge \forall\, k\,(\mathbf{f}(S(k)) = \mathbf{f}(k) + m) \to \mathbf{f}(n) = l)$.

   By contrast, if we restrict ourselves to first order formulas we are not able to define addition and multiplication anymore.[14] Instead we have to see '+' and '$*$' as term formation symbols, i. e. as signs, which when combined with terms result in a new term. So if t_1 and t_2 are terms then $t_1 + t_2$ as well as $t_1 * t_2$ should also be legal terms.

   This, however, does not explain how both operations work. Therefore we have to add the recursive "definitions" $\forall\, n\, m\,(S(n) + m = S(n + m))$ and $\forall\, n\, m\,(S(n) * m = n * m + m)$ to our axioms.

[12] If one is fastidious then indices on $\mathbf{P}$ are needed explaining if the quantification is over a function or predicate and what its arity is.

[13] In fact, the first order version will have nonstandard models while the second order version will be categoric. See [55, section 7.2] and [103, theorem 4.8] for further details.

[14] This is even true if we add addition to our theory and only have to define multiplication or vice versa. The reason is that without one of both operations the theory is known to be decidable while this is not the case for the theory including both, thanks to Gödel's incompleteness theorems.

7. There is a second type of axioms we have to add to the medium degree version in case of first order logic, namely the axioms of equality. In the medium degree version these are implicitly taken for granted (missing condensation). Here we have to state them explicitly. These axioms are:

   (a) reflexivity: $\forall\, x\, (x = x)$.

   (b) substitution: For each function symbol f (here S, $+$, and $*$) we have $\forall\, x\, y\, (x = y \to f(\ldots, x, \ldots) = f(\ldots, y, \ldots))$.

   (c) substitution for formulas: For each formulas ϕ, ϕ' where ϕ' results from ϕ by replacing some but not necessary all free occurrences of x by y we have $\forall\, x\, y\, (x = y \wedge \phi \to \phi')$.[15]

   In particular we can conclude symmetry and transitivity from these axioms.

   Again, in second order logic this supplement is not needed since we can simply define $x = y$ in the spirit of Leibniz as abbreviation of $\forall\, \mathbf{P}\, (P(x) \leftrightarrow P(y))$.

8. So far we are not able to deduce anything since logic is missing. Just like equality our logic is implicitly presupposed in the medium degree version. Even in otherwise rigorously formal systems the explication of logic is often left. However, if we are more pedantic, we have to add respective axioms, too. So let us add some arbitrary axiomatic system for first and second order logic, for instance those that can be found in [103, chapter 3.2].

Last, we will give a type theoretical version for the natural numbers. We will already use (and explain) some syntax of the Coq proof assistant, which we will present in the last section of this chapter. Furthermore we will use type theoretical notation from now on; in particular we write $f\ n$ instead of $f(n)$ for function applications and intend the quantifiers to bind as far as possible.

The type theoretical version for natural numbers (written in Coq style) uses the calculus of inductive constructions (see [89]) as background theory. The whole bunch of axioms of the medium degree version boils down to the following inductive definition:

`Inductive` $\mathbb{N} := 0 : \mathbb{N} \mid$ `Suc` $: \mathbb{N} \to \mathbb{N}$.[16]

[15] The last two are axiom schemata since we have different axioms for the different function symbols and different formulae.

[16] One may argue whether this rather creates a theory or whether it is rather an embedding into type theory (like the one we had for set theory before). That we do create new objects with the given definition militates in favour of the former classification while the use of other type theoretic constructs militates in favour of the latter.

First of all the definition says that 0 is a natural number and that Suc n is a natural number if n is. Implicitly included in this definition is the possibility to define functions over $\mathbb{N}$ via pattern matching, i. e. by case analysis. What is the value of the function on input 0 and how can the result be computed (possibly by using recursion) when the input is a successor? The possibility to define functions this way implies 0 being different from any Suc n and Suc to be injective.[17] In addition to that an induction principle is automatically created, which – of course – is the usual induction in case of the natural numbers: $\forall\ P : \mathbb{N} \to \text{Prop},\ P\ 0 \to (\forall\ n : \mathbb{N}, P\ n \to P\ (\text{Suc}\ n)) \to \forall\ n : \mathbb{N},\ P\ n$; where Prop is the type of propositions in Coq.

What about logic, equality, and the definability of addition and multiplication? Let us only mention here that there are ways to deal with logic and equality in type theory. So there is no additional work here. In case of addition we can define

```
Fixpoint addition (n : ℕ) : ℕ → ℕ :=
λ m : ℕ, match n with
 0 ⇒ m
| Suc n' ⇒ Suc (addition n' m).
```

The recursive input, here n, must be given as a parameter. More essential, however, is that we can call the recursive value addition $n'\ m$. Note that the use of the infix symbol + can be introduced as a notation after the definition.

Multiplication can be defined in the same way:

```
Fixpoint multiplication (n : ℕ) : ℕ → ℕ :=
λ m : ℕ, match n with
 0 ⇒ 0
| Suc n' ⇒ multiplication n' m + m.
```

The type theoretical version of natural numbers in Coq does not use natural language. It seems to be extraordinarily condensed; an impression that has to be relativised to a certain extent since we have to add the calculus of inductive constructions. Yet, the latter is itself a high degree system and so we can count the type theoretical version as high degree, too.

Let us sum up. We have seen multiple ways to formalise natural numbers and – on the basis of the use of natural language and condensation – we discussed the degree of formality for each resulting system. Furthermore it became clear that

[17] For the former we can define a function that maps 0 to True and Suc n to False for every n. There is a theorem, independent of natural numbers, that allows us to infer $f\ a = f\ b$ for any function (of the right type) if $a = b$. So from $0 = \text{Suc}\ n$ we could deduce True = False. However, it can be shown that True $\neq$ False, which in type theory is just an abbreviation for True = False $\to$ False. Hence $0 \neq \text{Suc}\ n$. To prove injectivity we define a function that maps 0 somewhere, let us say 0, and Suc n always to n and apply it to the equality Suc n = Suc m, obtaining $n = m$.

Theorem: $\forall\ n\ m : \mathbb{N},\ n + m = m + n$
Proof idea: Show that the definition is symmetric (meaning that addition could have been defined equivalently by recursion on the other parameter). Then, in the induction step extract the Suc, use the induction hypothesis, and draw the Suc to the other side.

Figure 2.1: Proof idea for the commutativity of addition.

high degree formalisation entails that one must care about additional issues like logic, equality and the definition of relevant constants, functions, and predicates. In particular, two systems of the same degree can be essentially different because these additional issues are treated entirely different.[18] Note that the non-treatment of the additional issues in medium degree versions is not necessarily a lapse but can be seen as focussing on the most relevant aspects.

2.5 Argumentations of different degrees validating the commutativity of addition

After we have seen how natural numbers can be formalised in or with different systems let us now have a look at argumentations – and proofs in particular – of different degrees of formality; all validating the same proposition, namely the commutativity of addition. In concreto we will delineate one proof idea, something between a proof idea and a medium degree proof, a medium degree proof, a Hilbert style proof, and a proof as λ-expression while a last variant will be postponed to the following section about the Coq proof assistant.

First, let us see what a proof idea for the commutativity of addition looks like (figure 2.1). One important aspect is the striking brevity. The whole argumentation is condensed into two sentences using predominantly natural language, which is the next point. However, one must be aware that there are higher degree formalisations for addition, induction etc. upholding this more informal version. A last point we want to mention is that there are not any justifications here. The argumentation reads like a recipe.

[18] An example of two non-essentially different systems for natural numbers would be our first order system compared with one where '0' is replaced by '$\emptyset$' or even '5'. Analogously we could replace our sign 'S' with 'Suc' or with '$'$', where in the last variation one would normally build a new term of 'σ' with 'σ'' instead of '$'\sigma$'. In general, however, it is not clear where to draw the line for essential difference.

Theorem: $\forall\ n\ m : \mathbb{N},\ n + m = m + n$.
Proofgumentation: It is easily shown by induction that the definition of addition is symmetric. We prove the actual theorem by induction over n. In the base clause we assume $m : \mathbb{N}$ arbitrary but fixed. We have $0 + m = m = m + 0$ thanks to the definition and the symmetry of it.
So let us assume now $n : \mathbb{N}$ arbitrary but fixed with $\forall\ m : \mathbb{N},\ n+m = m+n$. We have to show $\forall\ m : \mathbb{N},\ \text{Suc}\ n + m = m + \text{Suc}\ n$. So let m be a natural number, arbitrary but fixed. We have $\text{Suc}\ n + m = \text{Suc}\ (n + m) = \text{Suc}\ (m + n) = m + \text{Suc}\ n$ again thanks to the definition and symmetry of it as well as thanks to the induction hypothesis.

Figure 2.2: Proofgumentation for $\forall\ n\ m : \mathbb{N},\ n + m = m + n$.

Next, let us consider a form of argumentation that has no official name.[19] It lies between proof ideas and medium degree proofs (an example of the latter will be considered after this argumentation style). For the matter of brevity we dub it "proofgumentation". Our respective instance can be found in figure 2.2. What is essential for this argumentation style is its balance: some parts are still in the recipe like kind of argumentation while other parts (here the induction) are more elaborated. In the former parts our above analysis holds while for the latter the following analysis of medium degree proofs will be valid.

In medium degree proofs, like the one in figure 2.3, every part of them must be proven, not just some of them. Each proof (possibly including lemmata like in this case) consists of a mix of natural language and language free formulae. The former links the latter. Furthermore language structures the proof, including the weighing of the different parts of the proof.[20] Another important point is that medium degree proofs are much longer than, say, proof ideas. This is caused by the increased granularity (or by decreased condensation, which is the dual way to look at this): no gaps should be perceived in the reasoning. In addition to this, in a medium degree proof most steps need a justification, explaining why it is permitted to make the respective step. Yet, a short hint suffices at this level of argumentation. Instances of such justifications are given in figure 2.3; for example by the formulation 'thanks to the definition and the first lemma'.

In a high degree proof we have to prove the same lemmata as in the medium degree proof. However, for the reader to get an impression of such a high degree proof it suffices to presuppose these as lemma 1 (and lemma 2) and show only the base clause part of the proof (figure 2.4). Apart from the lemmata the proof

[19] In fact, argumentations of that kind are often referred to as proofs in textbooks but that is misleading.

[20] A typical example is the formulation 'Since xy, we obtain yz', which marks the 'Since'-part as easy and linearises the whole argumentation.

Theorem: $\forall\ n\ m : \mathbb{N},\ n + m = m + n$.

For proving this theorem we will need the symmetry of the definition. Therefore we are going to prove two lemmata first, one for the base and one for the recursive case.

Lemma: $\forall\ n : \mathbb{N}, n + 0 = n$.
Proof: We prove the theorem by induction over n. In the base clause we have $0 + 0 = 0$ by definition. So let n be a natural number, arbitrary but fixed with $n + 0 = n$. We have to show Suc $n + 0 =$ Suc n. But the left hand side is Suc $(n + 0) =$ Suc n due to the definition of the addition and the induction hypothesis. Qed.

Lemma: $\forall\ n\ m : \mathbb{N}, n +$ Suc $m =$ Suc $(n + m)$.
Proof: We prove the theorem by induction over n. In the induction basis we obtain $0 +$ Suc $m =$ Suc $m =$ Suc $(0 + m)$ by definition. So let n be a natural number, arbitrary but fixed with $\forall\ m : \mathbb{N}, n+$Suc $m =$ Suc $(n+m)$. We have to show $\forall\ m : \mathbb{N},$Suc $n +$ Suc $m =$ Suc (Suc $n + m$). Let m be an arbitrary but fixed natural number. It is Suc $n +$ Suc $m =$ Suc $(n +$ Suc $m) =$ Suc (Suc $(n + m)) =$ Suc (Suc $n + m$) thanks to the definition of addition and the induction hypothesis. Qed.

Proof of the theorem: We prove the actual theorem by induction over n. In the base clause we assume $m : \mathbb{N}$ arbitrary but fixed. We have $0+m = m = m + 0$ thanks to the definition and the first lemma. So let us assume now $n : \mathbb{N}$ arbitrary but fixed with $\forall\ m : \mathbb{N},\ n + m = m + n$. We have to show $\forall\ m : \mathbb{N},$ Suc $n+m = m+$Suc n. So let m be a natural number, arbitrary but fixed. We have Suc $n + m =$ Suc $(n + m) =$ Suc $(m + n) = m +$ Suc n thanks to the definition and the second lemma as well as thanks to the induction hypothesis. Qed.

Figure 2.3: Medium degree proof for $\forall\ n\ m : \mathbb{N},\ n + m = m + n$.

1. $\vdash (\forall\ m,\ 0 + m = m + 0) \rightarrow$
   $(\forall\ n,\ (\forall\ m,\ n + m = m + n) \rightarrow$
   $\forall\ m,\ \text{Suc}\ n + m = m + \text{Suc}\ n) \rightarrow$
   $\forall\ n\ m,\ n + m = m + n$ (S9)
2. $\vdash m = 0 + m \rightarrow m = m + 0 \rightarrow 0 + m = m + 0$ (S1)
3. $\vdash 0 + m = m \rightarrow m = 0 + m$ (PA2)
4. $\vdash (\forall\ n,\ 0 + n = n) \rightarrow 0 + m = m$ (A4)
5. $\vdash \forall\ n,\ 0 + n = n$ (S5)
6. $\vdash 0 + m = m$ (MP 4,5)
7. $\vdash m = 0 + m$ (MP 3,6)
8. $\vdash m = m + 0 \rightarrow 0 + m = m + 0$ (MP 2,7)
9. $\vdash m + 0 = m \rightarrow m = m + 0$ (PA2)
10. $\vdash (\forall\ n,\ n + 0 = n) \rightarrow m + 0 = m$ (A4)
11. $\vdash \forall\ n,\ n + 0 = n$ (Lemma 1)
12. $\vdash m + 0 = m$ (MP 10,11)
13. $\vdash m = m + 0$ (MP 9,12)
14. $\vdash 0 + m = m + 0$ (MP 8,13)
15. $\vdash \forall\ m,\ 0 + m = m + 0$ (G 14)
16. $\vdash (\forall\ n,\ (\forall\ m,\ n + m = m + n) \rightarrow$
    $\forall\ m,\ \text{Suc}\ n + m = m + \text{Suc}\ n) \rightarrow$
    $\forall\ n\ m,\ n + m = m + n$ (MP 1,15)

...

Figure 2.4: Base clause in the first order high degree proof of the commutativity of addition.

uses furthermore the axiom schemes S1, S5, S9, A4, the theorem scheme PA2 (the relevant instances of which are stated in the proof), and the inference rules MP (modus ponens) and G (generalisation).[21]

Obviously, the proof is much longer than the previous ones. It becomes clear that the steps considered as single in the medium degree proof are not so from this kind of proof's perspective. Every step be it ever so small has to be stated explicitly. Furthermore we see that there is no natural language left in the proof. Instead the proof offers exact justifications after every step; each of them being Modus Ponens, Generalisation, an axiom, or a theorem.

Last, we turn our attention to the proof that is just the λ-expression shown in figure 2.5. Although it is constructed with the Coq proof assistant one does not have to know anything about Coq to get the essential points here. Obviously, no natural language is used. Even without an understanding of the concrete steps it is clear that they have a high granularity. Yet, it is not as high as in the first order proof since we can conduct multiple applications of modus ponens at once. A further difference is that in the λ-proof the justifications are invisible. They are the matching of the input and the output types.

Let it be mentioned that it is possible to develop type theoretical proofs having a medium degree of formality. Examples of this can be found in [111]. One may discuss whether these are as understandable as their corresponding first or second order versions but we will not delve into this here.

Let us summarise. In this section we have seen a great variety of proofs for the same theorem. These differed significantly regarding the use of language, the condensation, and the frequency and precision of justifications. Proof scripts in Coq, which we omitted so far, require some preparation and will be important in some parts of this writing. Thus, we devote a separate section to it.

2.6 Proofs scripts as proofs

The aim of this section is to show how so called *proof scripts* written for the Coq proof assistant can be seen as proofs. However, before that we have to clarify what we mean by proof assistants in general and introduce Coq in particular. The way we use Coq looks somewhat different compared to the standard use and we will briefly discuss why. Furthermore when we come to our main aim, the proof scripts, we will also introduce a way of bringing such proof scripts to paper. This presentation form will be frequently used in chapter 8.

A proof assistant is a software that helps the human in one way or the other to create proofs. In contrast to a so called theorem prover this is not done completely

[21] The names correspond to [55, p.139], but the theorem itself is not proven there. We made two small changes by using n instead of x in the quantification of S9 and by quantifying S5.

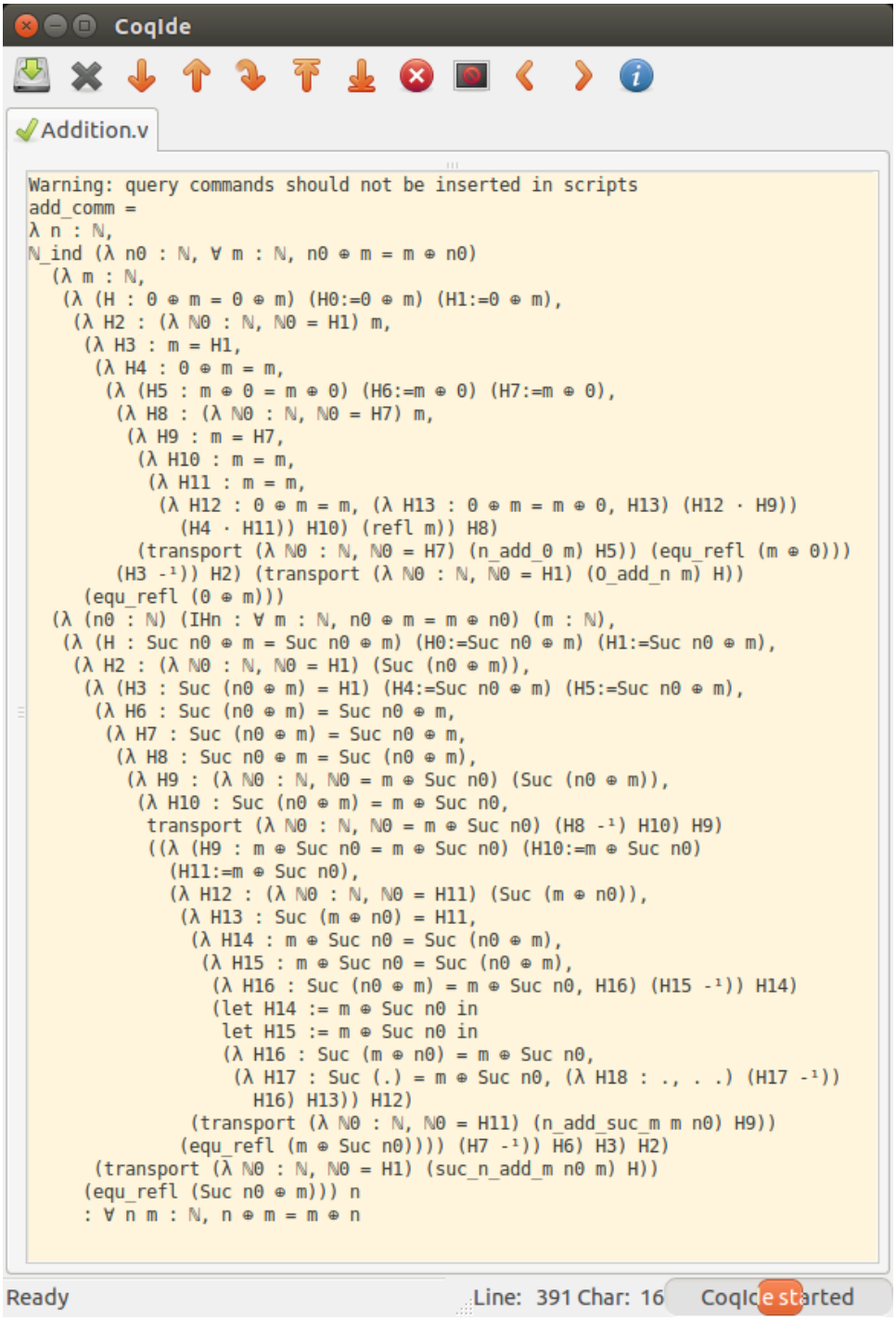

Figure 2.5: λ-expression as proof for the commutativity of addition.

automatic but in interaction with the human; where the latter uses a proof editor or some other interface for the interaction. Besides Coq (see [12, 26]) there are other proof assistants like Nuprl (see [3]), Isabelle (see [86]), Mizar (see [83, 118]), HOL (see [51, 52, 56]), or Agda (see [17]) having in part very different approaches. While classical Isabelle – like Coq – is based on proof scripts the attachment[22] Isabelle/Isar (see [116]), Mizar, as well as HOL allow to create proofs in a declarative proving style; i. e. the user is telling the system the steps of the proof while the system tries to infer justifications for that steps. The resulting proofs are of a synthetic nature and often resemble the ones created by humans alone. Nuprl uses a visual proof editor to create proofs interactively. A fourth, very different approach, which is realised in Agda, is to build proofs as functions with the possibility to leave so called holes for which the system infers the needed type.

The Coq proof assistant[23] has been developed in the late 1980's by researchers at the French Institut National de Recherche en Informatique et en Automatique (INRIA). Like other proof assistants, Coq implements a higher-order type theory; thus theorems in Coq are understood as types and the proofs for theorems as elements of the respective type. This implies that the latter are λ-expressions (as seen in the previous section) and so the background of Coq is closely interwoven with functional programming.

However, one feature of Coq is that the user does not have to know or even be aware of all these issues. Instead the user can develop a proof step by step by applying so called *tactics*. Every step the user asks Coq to execute is evaluated for its correctness and its effects are shown to him. As we will see in this section but especially in chapter 8 tactics in Coq are a very powerful and flexible tool. Furthermore, Coq is easily available and has a big and active community.

Let us now turn our attention to the proof scripts. These are sequences of tactics used to find a proof. Our first example is a simple proof script in the CoqIDE generating a proof of $A \wedge B \to B \wedge A$ (see figure 2.6). We start by stating in the left of the window that we want to prove an implication. The Coq-System processes and shows this by "greening" our code, a clear positive feedback. Had we started by trying to prove a conjunction instead we would have received an error-message at the bottom right of the window (see figure 2.7). In case we make a correct step the resulting proof situation is shown at the top right of the window (see again figure 2.6). What we know (or assume) can be seen above the line while the (sub)goal(s) are listed below. If there is more than one goal then above the line only the context of the first subgoal is shown.

[22] Isabelle is not a single proof assistant but a framework. See [115] for further information.

[23] The following introduction to Coq including the discussion of the tactics we used is taken in greater parts from the article [15]; where the concept of tactics will be introduced below.

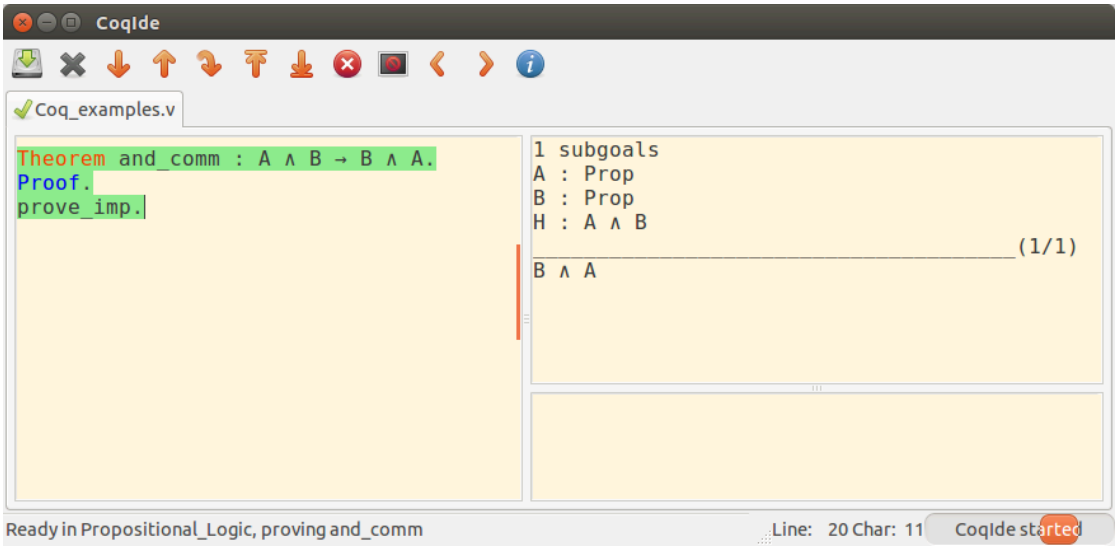

Figure 2.6: Proof script of $A \wedge B \rightarrow B \wedge A$ after the first step.

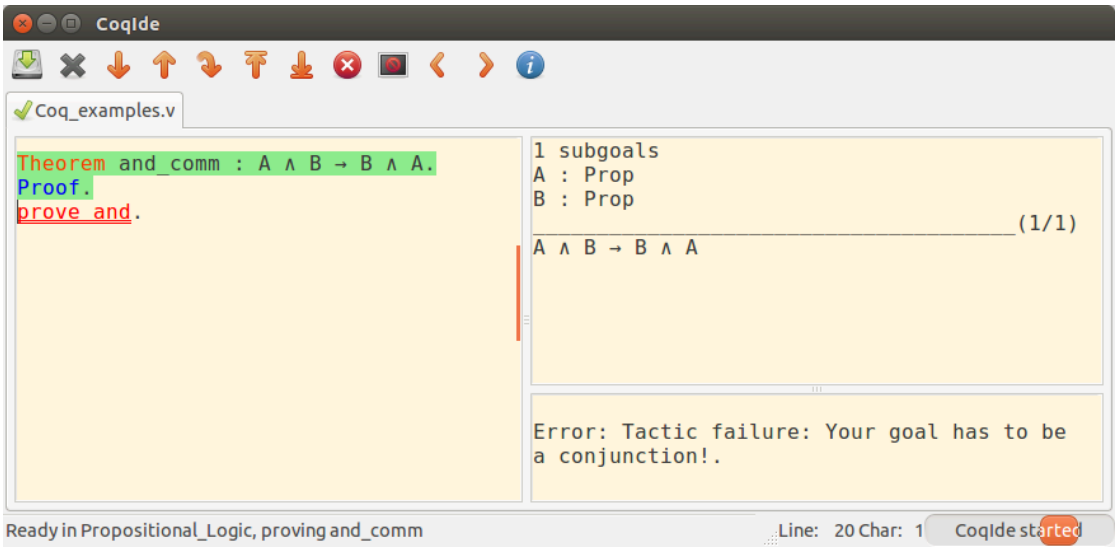

Figure 2.7: Proof script of $A \wedge B \rightarrow B \wedge A$ with a wrong beginning.

The remaining part of the proof script is given in figure 2.8. The screenshot shows the situation after step 3: $A \wedge B$ is already decomposed into A and B (via `use_and`) and we have applied the tactic `prove_and` to generate two subgoals, B and A. The '+' in the next line focusses on the first of them, i. e. after processing this symbol only the actual goal is shown. Both subproofs can be completed by using our assumptions.

Before we proceed with our analysis of proof scripts we have to mention that in Coq normally there are no tactics named `use_xyz` or `prove_xyz`. These tactics

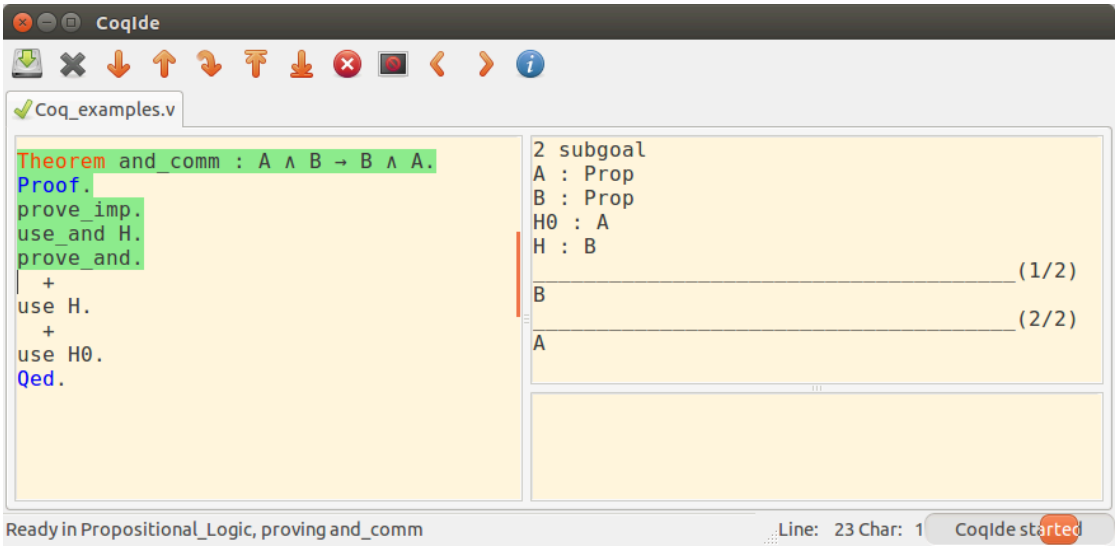

Figure 2.8: Complete proof script of $A \wedge B \to B \wedge A$.

were defined by us[24] and serve essentially as aliases for the logical and equational tactics Coq is providing. We use the former for the following reasons:

- Predefined Coq tactics have unstructured names and are therefore hard to remember; for tactic names such as `intro`, `case`, or `split` do not mention the relevant logical connectives and do not explain their relationship.
- The names of standard Coq tactics do not make an explicit distinction in the treatment of assumptions and goals. By contrast, our tactics are called `use_xyz` or `prove_xyz`.
- Predefined Coq tactics have unwanted features. For instance `split`, the tactic corresponding to `prove_and`, also works on goals like $A \to B \to A \wedge B$.

So we designed and named the tactics in a way that requires one to reflect on how to prove theorems in Coq.

So far we avoided to call the proof scripts themselves proofs. To see the subtle differences – and the reasons why we will call proof scripts proofs anyway – let us compare a theorem in type theory with a wardrobe. A proof, also called proof object, is a function – expressed as λ-term as seen in the previous section – explaining how the theorem is composed of different functions and applications. This is similar to the data we usually find next to the exhibit piece in a furniture store. Yet, for the analogy to work this data would have to be a complete description of the wardrobe. This would include not only the exact enumeration of its

[24] See Appendix A for a listing of most of the logical and equational tactics we use.

parts (doors, walls, knobs, mirrors etc.) with their respective properties (length specifications, material, colour etc.) but their exact arrangement (distances and angles of the respective parts), too. Furthermore, a complete description of the wardrobe would have to be done recursively for all its parts.

So a proof in type theory is an exact description of a product. By contrast, a proof script can be compared to an instruction for building a wardrobe. If every single tactic step is executed then the theorem (or the wardrobe respectively) will be constructed. However, requirement for this point of view is that each tactic has a clear specification of what to do. The logical tactics seen so far satisfy this criterion and the medium degree tactics we are going to present in chapter 8 will so, too, but so called auto tactics do not fulfil this requirement (see the next section for further discussion). In reality there is a big difference between a (complete) description of the wardrobe and the instruction for building it. However, in the mathematical world as well as in some parts of informatics resources are of no importance, which is why the difference between description and instruction fades away. Therefore the proof scripts can be conceived as proofs.

What about the degree of proof scripts? The tactics we use are named rather verbosely after the process they execute. Apart from that, there is no natural language involved. Whether the whole argumentation is rather condensed – in the sense that the number of steps is reduced – depends on the concrete tactics. The few we have already seen and the ones we will still see in this section are very elementary. They do not condense at all. The tactics do not only represent proving procedures but function as justifications as well: they give a reason why we are allowed to transition from the actual to the next proof situation. Their exactness can be conceived as the preciseness in the treatment of parameters. Since in this section all parameters are instantiated explicitly the proof scripts in this section can be classified as high degree. In chapter 8 we will see that not all proof scripts must be of a high degree.

Let us now come back to resources. We said that mathematics is resource independent but this is not the same as stating that mathematicians are resource independent. Therefore we should find a presentation of proof scripts suited to be used in text documents like this one. To understand what problems may arise let us first again consider the wardrobe first. In that case, a disadvantage of a pure instruction is that it does not tell us anything about the appearance of the product. Not until we executed some step we see its result. However, in reality we hardly drive to the furniture shop, buy a wardrobe, drive back, and construct the wardrobe only to see what it looks like. Instead there is a photo and/or an exhibit piece in advance helping us with our visualisation. Furthermore, since the individual steps of an instruction in reality are not that unambiguous, modern

instructions also or even exclusively use graphics showing the appearance of some of the intermediate products and highlighting the actual changes. Hence mistakes in the understanding or the construction can be recognised in time.

In the case of proof scripts we have no resource problem regarding execution if a computer is involved (computers are simply too fast for that to be of any relevance).[25] Nevertheless there are resource problems since we need to have the right software in the right version installed and the settings must be correct, too. Furthermore it would be highly preferable not to be committed to a computer at all. Then, however, the reader of a proof script is not informed about the actual proof situations (which is elsewise done by the Coq system) and mistakes will remain unrecognised for a long time and hence cannot be traced back in a good manner. Therefore proof scripts are assessed as unreadable by humans alone (see [85]). That is why we use equivalents to the graphics in the instructions. This time we use tables showing the proof situations. As in the case of instructions we will only show the changes after each step – instead of the whole proof situation – to contain the required place resources.

Consider figure 2.9 as an example. The pure proof script is given in the left column while in the right column in each row the proof situation after the respective step is listed. Above the line we find all actual hypotheses (the knowledge we have at our disposal) and below it the actual goal. If a hypothesis will not be changed for the remaining part of the subproof it is framed and will not be listed anymore. The induction hypothesis, for instance, is not changed in the subproof marked by $+_2$ and is therefore framed. Note that after those tactics that result in a branching no proof situation is given immediately. The respective proof situations are then shown after the respective begin of the subproof. One situation, which does not occur in this example, arises when we want to clear a hypothesis to reuse the name. It is only in this case that the tactic, namely `clear`, will not occur on the left hand side. Instead the respective hypothesis will be struck through in the step before to signalise that its name now can be reused.

Now let us conclude this section by discussing the contents of the proof. It starts by using induction over n which leads to two branches, induction begin and induction step. In the former one chooses m arbitrary but fixed and uses the definition of addition as presented in section 2.4 to reduce $0+m$ to m. Unfortunately, `fold` must be used afterwards to avoid that the definition of addition is written down on the right hand side of the equation.[26] Next, for the other side of the equation one needs a lemma stating $\forall\ n : \mathbb{N},\ n + 0 = n$. It is applied to the argument m obtaining $m + 0 = m$ and this equality is used in the goal. This leads to the new goal $m = m$, which can be concluded by the tactic `prove_equ`. The second

[25] At least this is true for the tactics we are considering within this thesis.

[26] Note that the feature to use notations is only implemented for `unfold` in Coq; not for `fold`.

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m : \mathbb{N},\ n + m = m + n$
prove_by_induction	
$+_1$	$\forall\ m : \mathbb{N},\ 0 + m = m + 0$
prove_all	$m : \mathbb{N}$ $0 + m = m + 0$
unfold "+"; fold addition	$m = m + 0$
use_equ (n_add_0 m)	$m = m$
prove_equ	subgoal proven
$+_2$	$n : \mathbb{N}$ $IHn : \forall\ m : \mathbb{N},\ n + m = m + n$ $\forall\ m : \mathbb{N},\ \text{Suc}\ n + m = m + \text{Suc}\ n$
prove_all	$m : \mathbb{N}$ $\text{Suc}\ n + m = m + \text{Suc}\ n$
unfold "+"; fold addition	$\text{Suc}\ (n + m) = m + \text{Suc}\ n$
use_equ (n_add_suc_m m n)	$\text{Suc}\ (n + m) = \text{Suc}\ (m + n)$
use_equ (IHn m)	$\text{Suc}\ (m + n) = \text{Suc}\ (m + n)$
prove_equ	Qed

Figure 2.9: High degree proof of $\forall\ n\ m : \mathbb{N},\ n + m = m + n$ in Coq.

subproof might be more difficult to find but – from a technical point of view – consists of the same kind of steps. Note that one has to use again a lemma; this time stating $\forall\ n\ m : \mathbb{N},\ n + \text{Suc}\ m = \text{Suc}\ (n + m)$.

2.7 Auto tactics and related work

For a better understanding of the related work presented in this section it will be helpful to discuss auto tactics first, which are a substantial ingredient in many of the corresponding approaches. We already mentioned that auto tactics do not fulfil clear specifications, a property we can take as definition for the former. One might object that each (auto) tactic has a specification in the sense that it does exactly what its code requires it to do, but we do not count this description as specification since it is not independent from the concrete code or process of the tactic.

Using auto tactics we obtain a formal proof, for instance a λ-term, in the same way this was true for non-auto tactics. From the proof script point of view, however, auto tactics – due to their lack of specifications – are no justifications. This speaks for a low degree of formality. Often the corresponding step size is very big but that does not need to be the case. If the auto tactic is too weak – be it so on purpose or not – the step size might be very small. Even when the auto tactic is very powerful the user might apply it only to very simple lemmata such that the degree of condensation remains low. Furthermore it should be mentioned that some auto tactics do not change the proof situation at all – usually returning an error –, when the actual goal could not be proven completely. So there is no guarantee for any progress. Altogether we see that proof scripts using auto tactics are definitely not of a high degree of formality due to their lack of justifications. Except for the language, which remains formal, such proof scripts may correspond to low degree proofs.[27] However, whether this is really the case depends on its use.[28] Last, let it be mentioned that major use of auto tactics is usually made by declarative proof assistants. This is understandable since the latter do not focus on justifications.

Let us now start with the discussion of related work. The aspect of granularity, which – as a reminder – is the same as condensation but seen from the other side, is the topic of Marvin Schiller in his PhD thesis [98]. To be more precise, Schiller wants to find the right criteria to determine what the perceived sizes of given steps in a proof are. He discusses that the perceived size of a step cannot be the sheer number of corresponding steps needed in what we call a proof of highest degree. So

[27] The existence of a formal proof like a λ-term or so may justify to use 'proof' here instead of 'argumentation'.

[28] See section 8.10 for further discussion.

respective investigations are needed. These are of an empirical nature and involve machine learning techniques. The knowledge of the granularity of steps is then used to support students' proof development and the teaching of proofs with the help of the computer. The respective "computer systems for teaching proofs need to allow students the freedom to produce various proofs at different granularities, but they also need to detect whether an individual solution is inappropriate." ([98, p. 47]) The realisation of the computer systems for teaching proofs is based on a declarative proof assistant using auto tactics. We should also mention that the idea of various degrees of formality is also present in Schiller's thesis: "mathematical language allows a multitude of syntactically different ways to express a solution to the same problem, which can be formulated in various styles ranging from scrutiny to vague descriptions." ([98, p. 8])

The role of granularity is also discussed by Ludovic Font, Philippe R. Richard, and Michel Gagnon [43] – this time as part of what they call didactical contract. They list some positive properties of proofs with a lower degree of granularity, for instance that the latter "[...] smoothen the flow of the resolution or improve its pedagogic efficiency." ([43, p. 39]) The issues of readability and accessibility are also addressed by them. The actual topic of their article, however, is the presentation of a tutor system, which we will discuss in the related work section of chapter 7.

In [119] Claus W. Zinn aims at building a verifier for medium degree proofs – mathematical proofs in his terminology. Since the latter involve a lot of natural language he has to analyse the role of the respective linguistic fragments used in the proofs and he has to find ways to translate them automatically and mathematically adequate into a machine readable language. For example, one kind of such linguistic fragments concerns variables: "[a]ny text understander needs to be aware of the different uses of variables in mathematical discourse and their respective linguistic realisations." ([119, p. 62]) In chapter 3 of Zinn's thesis we find an analysis of particular medium degree proofs, too, but this analysis focusses on convertibility to some corresponding formal proof only and so does not amount to a comparison between the different proof styles. Nevertheless the aspect of granularity is implicitly present: "constructing a maximally rigorous proof from Hardy & Wright's informal proof requires filling in many steps that they leave unverbalised. This is already true for the formal interpretation of the first proof sentence." ([119, p. 42])

Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable in [57] want to translate Nuprl proofs, which they are characterising as formal proofs, into proofs that mimic human reasoning. This sounds very promising. Yet, their Nuprl proof use auto tactics. As discussed above, the corresponding proofs are then not formal due to their lack of justification and – more important – there is no

guarantee for the right step sizes to simulate human reasoning. Therefore the true achievement of Holland-Minkley, Barzilay, and Constable is restricted to the aspect of translating the Nuprl proofs into a natural language argumentation. Regarding that point their work can be seen as reversed to Zinn's.

The remaining aspect, the frequency and precision of justifications, seems to be generally underestimated. At least we are not aware of any literature substantially addressing that subject.[29]

As discussed, proofs that are not of a high degree of formality are usually called informal. This already suggests to define or characterise the latter via the former using negation. A first alternative preferred by some authors is to characterise "informal proofs" by their properties instead. Fenner Tanswell does this, too, but he also gives – as a second alternative – a brief and direct characterisation: "informal proofs are proofs as they are written and produced in mathematical practice. They may make assumptions about the intended audience's background knowledge and ability to follow lines of reasoning, skip over tedious or routine steps, and make reference to semantic properties and properties of mathematical objects without stating these fully." ([108, pp. 295-296][30])

A very interesting idea is to represent different degrees in a single argumentation. Such an approach is pursued by Leslie Lamport in [72] with his concept of structured proofs.[31] These are nested proofs, in which some step can be explained by a nested series of more detailed steps. The reader then has the possibility to zoom into a more detailed level whenever necessary. Structured proofs mainly address granularity. Justifications are only written down explicitly at the respective end level, which – in our opinion – makes it hard to follow the proof without zooming at every place possible. By contrast, to be in agreement with our approach from a particular nesting level on there would have to be independent justifications instead. Last, we want to mention that the use of language is the same for all nesting levels.

The same idea of putting different degrees together into a single argumentation is even realised by an actual proof assistant, the ΩMEGA-System. It is presented in [105] by Jörg Siekmann, Christoph Benzmüller, and Serge Autexier. It realises proof planning approaches (see [19]), which in turn are inspired by corresponding

[29] In [39] and [40] Armin Fiedler deals with proofs having a list of alternative justifications for some of the steps. He orders these justifications with respect to the degree of their "abstraction", which sounds promising. However, it turns out that the latter is only a technical notion related to an extension of natural deduction. So all of Fiedler's justifications do not differ in their level of preciseness.

[30] A footnote has been omitted.

[31] This must not be confused with the structure faithful proofs we will introduce in section 7.5.

techniques in artificial intelligence. The user is supposed to start with a proof plan at a suitable degree, switching to a higher one if necessary, and to let the computer solve respective parts of high enough degrees automatically.

At the end let us remark that our use of low and high degree is inverted with respect to the one usually found in the literature. This is so because the latter use orients itself on programming languages while we orient on formality.

Chapter 3

Curry in a hurry

3.1 Introduction

Especially in the time of the foundational crisis of mathematics but still until today there is a controversy about the right foundations and methods of mathematics. Is mathematics based on set theory, type theory, category theory[1], or perhaps on logic alone? In any of the previous cases, which logic (or geometry in earlier times) is the right one? In particular, is the tertium non datur valid? At first glance this disunity looks like a danger: if everyone can do mathematics as he likes what value does mathematics have? Will *the* science be *a* science after all? It was this last question – with the former questions being in the background or explicitly mentioned – that Haskell B. Curry tried to answer positively in his writings [31, 29] in which he presented his view of mathematics as of the year 1939.[2] As a supplement to this view we take [30] into consideration, which was published in 1941 and which is in line with the before mentioned works.

Curry's solution to find a positive answer to the status of mathematics as a science is interesting for us due to multiple reasons. First, his treatment of logics will help us in the next chapter to gain more clarity about our concept of the degrees of formality. Next, Curry's approach in some sense is pluralistic fitting our pluralistic account of the degrees of formality. This motivates the search for the different benefits of the different argumentation styles (chapter 5). Furthermore, Curry provides a technique that will help in that chapter to characterise the purposes of argumentation. Finally, his position will serve us as object of study in the second part of this thesis.

[1] This can be a modern point of view only because category theory developed chronologically after the foundational crisis.

[2] The former was published in 1951 while the latter is a reprint of an article published in 1954. We will discuss that time discrepancies and the chronological relation of this two sources and the following one in section 3.6.

The concrete procedure of this chapter is as follows. We start with Curry's formal systems, which roughly correspond to our high degree ones, and discuss their role in saving the objectivity of mathematics to keep the latter's status as a science (section 3.2). It will – perhaps surprisingly – be these formal systems, too, which enable Curry's pluralistic attitude. In section 3.3 we will see why. We will present Curry's treatment of logic in section 3.4. Up to this point we will orient ourselves on [31, 29, 30]. However, there is previous and subsequent work of Curry. This will be discussed in section 3.5 in comparison to the three former sources. We will refer to this previous and subsequent work as the view of the early respectively later Curry.

3.2 Saving objectivity via formal systems

Curry's starting point in [31, chapter II] to maintain the status of mathematics as a science is to find a definition of mathematical truth that underlies an objective criterion; i. e. it must be possible to substantiate the truth of a mathematical statement objectively. Curry asserts that a "rigorous proof" is not a suitable candidate for a criterion of truth; for the concept of rigour itself is utterly vague.

So the question is how it is possible to define mathematical truth with a suitable, objective criterion for it. In [31, chapter II+III] Curry brings up three problem-solving approaches for discussion: realism, idealism and formalism. He discards realism, which for him is essentially mathematics of indigenous people because it misses the power for representing the actually practised mathematics. In particular the transfinite mathematics cannot be captured by such a realism. In the case of idealism, Curry distinguishes two kinds: platonism and intuitionism. For the criticism on platonism he refers to customary intuitionist's criticism. In addition intuitionism itself is rejected by him, too, because it requires a fundamental intuition that "has objective reality, at least in the sense that it is the same in all thinking beings." ([31, p. 6][3]) The problem with this fundamental intuition according to Curry is that the foundation for an associated objective criterion of truth has to be based on elementary philosophical hypotheses only ([31, p. 3]). Yet, the satisfaction of the postulate of a fundamental intuition is assessed by Curry as "an out and out postulate." ([31, p. 6][4])

According to Curry the correct problem solving approach for defining mathematical truth together with a suitable criterion for it is formalism. There are different variants of this school of thought but – as the name suggests – they all

[3] We have omitted the footnote at the end of the sentence. Curry enumerates further requirements, which we do not need for our argumentation.

[4] Again, we have omitted a footnote.

put great emphasis on what is usually called a formal system (and what roughly corresponds to our high degree systems). So let us have a look at what a formal system for Curry looks like:

> "A *formal system* [...] is defined by a set of conventions, which I shall call its *primitive frame*. These conventions are of three kinds, as follows: first, those which specify the objects of the system, which I call its *terms* [...]; second, those determining a set of propositions, which I shall call the *elementary propositions*, concerning these terms [...]; and third, those which specify which of the elementary propositions are true, that is, are theorems. These specifications have the form of recursive definitions. Thus the term-specifications give a list of primitive terms, or *tokens*, together with *operations* and *rules of term formation* which describe how further terms are to be constructed from the tokens. The specifications for elementary propositions consist of a list of *predicates* (properties, relations, and so on) together with *rules of proposition formation* for constructing elementary propositions by means of them. Finally, the specifications for theorems consist of a set of elementary propositions, called *axioms*, which are stated outright to be true; together with *rules of deduction* showing how further theorems are derived. As in recursive definitions generally a property of closure is understood, namely, that the specifications give all the entities recursively – for example, *all* the terms are obtained by the process described in the term specifications, and so on." ([30, p. 225-226][5])

It has to be mentioned that we cannot define our concept of the degrees of formality in the same precise way. This might explain why high degree systems and proofs are usually contrasted with all lower degree variants.

In our running example of Peano arithmetic, say in its first order version, Curry's definition of what formal systems are amounts to the following formal system (instance):

1. Terms:
   (a) Tokens: the constant 0 as well as an infinite list $[v_0, v_1, v_2, \ldots]$ of variables.
   (b) Operations: Suc, $+$ and $*$, $=$, $\wedge$, $\neg$, and $\forall$.

[5] All emphases are in the original text. A footnote has been omitted.

(c) Rules of term formation: there are two sorts of terms, real terms and formulae.

i. Real terms: 0 and all variables v_i are real terms and if t_1, t_2 are real terms so are Suc t_1, $t_1 + t_2$, and $t_1 * t_2$.

ii. Formulae: if t_1, t_2 are real terms $t_1 = t_2$ is a formula. If ϕ_1, ϕ_2 are formulae and v_i is a variable then $\phi_1 \wedge \phi_2$, $\neg\, \phi_1$, and $\forall\ v_i,\ \phi_1$ are formulae, too.

2. Elementary propositions:

(a) Predicates: only the unary $\vdash$.

(b) Rules of proposition formation: if ϕ is a formula then $\vdash \phi$ is a proposition.

3. Theorems:

(a) Axioms: elementary propositions of the form $\vdash \phi$ where ϕ is a non-logical axiom as stated in section 2.4 or where ϕ is a logical axiom in some arbitrary but fixed system for first order logic.

(b) Rules of deduction: this depends on the system of first order logic we take as our basis; probably Modus Ponens and Generalisation.

The salient point about the systems Curry calls formal is that they guarantee objectivity: when someone provides us with a proof for a theorem in such a system we – or even a computer – can check it line by line. In Curry's terminology a successful verification would establish the truth of the theorem. Yet, since this truth differs from other versions of truth that are external of the respective system under consideration we will speak of *provability* for the purpose of distinction.[6] [7] With this objective tool at hand Curry feels now entitled to define mathematics as the science of formal systems (see [31, chapter X][8]). So Curry seems to have saved mathematics' objectivity and therefore its status as a science.[9]

[6] Admittedly, this term itself is also not unproblematic, since we may want to reserve it for the unary predicate $\vdash$. Nevertheless it seems that the word 'provability' elicits the more suitable associations to Curry's view than the term 'truth'. Therefore we will call $\vdash$ the assertional sign instead.

[7] Another terminology (instead of provability) is given by Jonathan P. Seldin who speaks about "truth within a system or theory" ([100, p. V1:565]). This avoids the problem mentioned in the previous footnote but has the disadvantage of being more cumbersome in formulation.

[8] By "science of formal systems" he means not so much the work in formal systems but meta-theoretical work. The formal systems themselves become objects of consideration (see [31, chapter IX]). The formal systems are not seen as formal games but they mirror meaningful propositions (see [31, chapter X]).

[9] In chapter 6 we will discuss in which manner his definitions lead to problems. For now we want to concentrate on Curry's achievements.

3.3 Fruitful diversity in mathematics

Up to know we have shown (under reserve) how Curry's formal systems save the status of mathematics as a science but we promised more; namely that they also enable a pluralistic attitude. This should be surprising since provability does not leave room for interpretation and so at first glance seems to oppose any diversity. Yet, this is only one side of the coin. On the other side the lack of freedom inside a system allows a very liberal attitude towards the choice of such systems. In concreto Curry divides mathematical truth "into two stages" ([31, p. 60][10]): first provability of the corresponding formalised statement as already discussed and second *acceptability*[11] of the corresponding formal system itself.

Unlike provability acceptability is disputable. It is subjective and committed to a purpose. Curry exemplifies this for the formalised classical analysis: he argues for its justification in terms of success in the natural sciences, whereas he claims intuitionistic mathematics to be almost completely useless from that perspective. According to Curry intuitionistic mathematics is acceptable on other grounds instead; say because it delivers constructive proofs for metamathematical statements. The different systems do not contradict each other but serve different purposes. In the end the treatment of one system may foster progress in another one. Hence Curry establishes a view of mathematics that saves objectivity and endorses various appearances of mathematics at the same time:

> "This leads to my final plea – a plea for tolerance in matters of acceptability. Acceptability is not so much a matter of right and wrong as of choice of subject matter. Such a choice should be entirely free; and some difference of opinion is not only allowable, but desirable. We are often interested in systems for which the acceptability is hypothetical – even sometimes in those which, like non-desarguesian, non-archimedean, non-this-or-that geometries, are not acceptable for any known purpose. As mathematicians we should know to what sort of systems our theorems – if formalized – belong, but to exclude systems which fail to satisfy this or that criterion of acceptability is not in the interests of progress." [31, p. 64]

This – in our opinion – is the true achievement of Curry's solution. He manages to harmonise objectivity and subjectivity and so is able to justify pluralism in mathematics. We call this the essence of Curry's view of mathematics.

[10] In the remainder of this section we always refer to the corresponding chapter XI of that book.

[11] This time the terminology is adopted from Curry.

3.4 How Curry treats logic

Even in modern times there is an attitude shared by many authors that when a high degree system is defined often logic is not mentioned at all; assuming, let us say classical first order logic, implicitly. The background of this attitude is that these authors think of their respective logic as the only real one while all others are useless, foolish, or simply wrong.

By contrast, Curry denied that there is one correct logic for mathematics:

> "Let us distinguish two senses of <logic>. On the one hand logic is that branch of philosophy in which we discuss the nature and criteria of reasoning; in this sense let us call it logic (1). On the other hand in the study of logic (1) we may construct formal systems having an application therein; such systems and some others we often call ≪logics≫. We thus have two-valued, three valued, modal, Brouwerian, etc. ≪logics≫, some of which are connected with logic (1) only indirectly. The study of these systems I shall call logic (2).
>
> The first point regarding the connection of mathematics and logic is that mathematics is independent of logic (1). [...] [W]e are able to construct in logic (2) systems which stand more or less in contradiction with our intuitions of logic (1) – just as we can do the analogous thing in the case of geometry. Whether or not there are a priori principles of reasoning in logic (1), we at least do not need them for mathematics." ([31, p. 65-66][12])

The logics can be autonomous formal systems or can be part of some formal system.[13] This invites us to conceive of a mixed high degree system as a tuple consisting of a contentual and a logical part.[14] [15] In the next chapter this will be a good pointer for our treatment of what we will there call formalisms.

Finally, for Curry formal systems of logic are like all other formal systems. In particular they underlay the same considerations regarding acceptability:

> "There is no one system of logic which is acceptable a priori for every purpose under the sun. [...] When we have enough of such alternative logics we may realize that our religious faith which some

[12] We omitted a footnote. The original quotation marks were replaced in the right number by the smaller and greater sign to avoid confusion with our usage.

[13] See, for instance, examples 3 and 7 in [31, chapter V].

[14] Curry does not mention this division explicitly but it is in the air.

[15] This is an idealisation and therefore is not meant to imply the assertion that contentual and logical part can (always) be strictly separated. In fact, the later Curry – when discussing the logicists – stresses that it is not clear which contents should count as logical (see [32, section 1D]).

> people have in the classical logic is largely a matter of habit, and that the acceptability of systems of logic is essentially an empirical matter." [31, p. 68-69]

Hence all logics are a priori on the same level and may be worth considering. Curry has a pluralistic attitude towards mathematics and the domain of logic is no exception.

3.5 Earlier and later views of Curry

So far we focussed on Curry's presentations of his 1939 view. Let us now discuss important differences and other relevant aspects of Curry's earlier and later work. We refer to Jonathan P. Seldin's discussion of the earlier work (see [100, 101]). With respect to the later view of Curry we take two original sources into consideration: 'Foundations of mathematical Logic' ([32]), which was published in 1963 and is a mix of technical aspects and the philosophical view of Curry regarding mathematics of that time, and the article 'The Purposes of Logical Formalization' ([33]) written in 1968, which belongs to the philosophy of mathematics.

With respect to the early view one should know that Curry tried to deal with the technique of substitution in a more simple and clear manner than this was done in Principia mathematica (see [100] and [101, section 2.2]). The resulting project was combinatory logic, which in turn led him to the conviction that formal systems can create new things of real value (compare [33]). An important difference between the early Curry and the view presented in [29, 30, 31] is that the former only allowed one unary predicate for formal systems (named abstract system or abstract theory at that time[16]), namely the assertion (see [100]).[17]

Seldin stresses in [101, chapter 4] and [102] that Curry's view in [29, 31] is not mature. One difference Seldin is referring to is the following definition of the later Curry: "[M]athematics is the science of formal methods." ([32, p. 14]) This is broader than the former fixation on formal systems. In our terminology, it can also comprise mathematics of a lower degree of formality. Indeed, there are some points supporting that there is a substantial change regarding that matter. First of all, besides the philosophical logic and mathematical logics he discussed in [31, chapter 12] he now speaks of mathematical logic as the one that is usually applied in deductive reasoning and which was the only one used before logic was formalised (see [32, section 1A]). In addition it seems to be the logic of the metatheory between formal logics (see [32, section 1D]). He considers systems, called postulate systems, in which the non-logical axioms are formalised but for the results of which only

[16] Curry's terminology will be one subject of the related work section of chapter 6.
[17] Therefore in the later [32, chapter 2D1] these systems are called assertional.

the informal mathematical logic is used (see [32, subsection 1C1]). Instances of postulate systems are often algebraic theories like groups. Although we claimed that already in [31, 29, 30] Curry tries to justify pluralism in mathematics he is more explicit about it later: "[Formalism] admits the possibility of various kinds of mathematics existing side by side." ([32, p. 15])

So far all this fits our concept of different degrees of formality very well. However, we will argue later on for the respective benefits of systems and argumentations of different degrees of formality. By contrast, for Curry formalisation seems to be a one directional issue: the more formalisation the better. Other stages have no value on their own. When he considers stages of formalisation, for instance, something like the work of Frege and Russell is the first(!) step for him.[18] He explains this in a footnote: "One can of course distinguish a lot of earlier stages, but these do not concern us." ([32, p. 61]) A similar point is made by Curry in [33], in which he compares the transition from the 18th to the 19th century in calculus[19] with the development from the 19th century mathematics to that of the 20th century: "[...] the supposedly infallible logical intuitions of the nineteenth century were not actually so; it is necessary to refine our intuitions of set, function, proposition, etc. just as in the nineteenth century analysts found it necessary to refine the earlier intuitions of continuity and limit process. We can do this by introducing a higher degree of formalization." ([33, p. 362][20]) So the formalisation of one stage has to adjust the weaknesses of the previous one. This attitude towards mathematics of lower and higher degree is further confirmed by a listing of the advantages of higher degree formalisation that Curry gives: absence of irrelevant aspects appearing in the less formal versions, greater clarity, avoidance of ambiguity, creative power, easier to conduct (thought) experiments, and the possibility to find analogies between before unrelated concepts (see [33]). On the other hand the advantages of less formal versions are mentioned only implicitly. "One does not need the ultimate criterion [the highest degree of formalisation] any more than one needs to use primary standards every time one makes a physical measurement." ([33, p. 365]) "Again it is not a goal of logical formalization to abolish logical intuitions. The working mathematician reasons by his intuitions; let us hope that he will always do so." ([33, p. 365])

Whether Curry's view from 1939 to 1941 is mature or not, there is no major change later on regarding what we called the essence of it, namely the harmonisation of objectivity and subjectivity to justify pluralism in mathematics. Objectivity

[18] He calls these works 'formalised contensive theories'. Note that our use of the word 'contentual' is not congruent with Curry's use of the word 'contensive'.

[19] He should have included the 17th century also; the time where Leibniz and Newton were active.

[20] Note that the term 'higher degree of formalization' (instead of 'higher degree of formality') indicates a process (instead of an attribute), which has only one direction.

– or rigour, which amounts to the same thing – is constantly addressed as an issue in [32, 33]. In the latter source, for instance, he judges intuitive logic to be "not suitable as an ultimate criterion of rigorous proof." ([33, p. 361]) The concept of acceptability is further elaborated in [32, subsection 2B4]. There Curry introduces interpretations, which can be thought of as functions from theories to a contentual subject matter, and defines validity of a theory – in analogy to soundness – to mean that the interpretation of every provable statement in the theory is true. If for every true contentual statement every pre-image regarding the interpretation is provable Curry speaks of adequateness. Acceptability for him depends primarily on validity, which in turn depends on the contentual subject matter, while other criteria like simplicity, naturalness, aesthetics, or philosophical satisfaction are only secondary factors.[21] In addition to the treatment of objective aspects (via formal systems) and subjective ones (via acceptability) the later Curry still endorses pluralism in mathematics explicitly – as shown two paragraphs before. However, what is missing in his later work is a clear nexus between those three aspects. This might be due to the fact that only [31, 29] are systematic presentations of his view of mathematics while Curry's later work is not of this kind. Another explanation is that the necessity to justify pluralism in mathematics was already much lower in mathematical community in the sixties because the aftermath of the foundational crisis of mathematics had faded.

3.6 Discussion of the Literature and Related Work

Let us first explain the relationship between the different sources [31, 29, 30] presenting Curry's view of mathematics in 1939. In 1939 Curry wrote a systematic overview of his views of mathematics for the Journal of Unified Science. However, for publication he had to shorten it. All copies of the respective issue of the journal were destroyed during World War II. This is why in 1951 Curry published his views of 1939(!) again in [31] using the original, i. e. unshortened version. In that book he explicitly states that the text – except for little changes – was written in 1939 and does not represent his views of 1951 (see [31, p. V]). However, some copies of the conference version of his 1939 article re-emerged at some time and so the shortened article was – with minor corrections – republished in 1954. [29] is a 1983 reprint of that publication. [30] was published in 1941 to supplement the original 1939 article in the aspect of rigour.

The most extensive literature about Curry's philosophy of mathematics stems from Seldin, who even published together with Curry in the field of combinatory

[21] In chapter 6 we will argue that this is not liberal enough.

logic. [100] is a biography of Curry. [101] is a comparison of Curry's and Church's work, which comprises – apart from many technical aspects – some remarks on Curry's view of mathematics. The main source regarding this aspect, however, is [102]; an article, in which Seldin claims that Curry should be classified as structuralist.

Besides Seldin's work there is not much literature about Curry's view of mathematics. Thomas Bedürftig's and Roman Murawski's book [11], for instance, is about the philosophy of mathematics with 284 pages (without the appendix) but Curry's view is discussed in relation to Hilbert's on page 109 only. The main point discussed in [11] is Curry's attitude towards consistency proofs. Notably, three pages long is Stewart Shapiro's discussion of Curry's work in [104, section 6.5]. According to Shapiro the most important aspect of Curry's philosophy of mathematics is his metatheoretical point of view. In his entry 'Formalism in the Philosophy of Mathematics' in the Stanford Encyclopedia of Philosophy [114], in which one chapter is dedicated to Curry, Alan Weir comes to the same result. Although the available literature is a bit meagre regarding the discussion of Curry's work at least there is some discussion at all. Note that in the most important journal in the philosophy of mathematics 'Philosophia Mathematica' there is not even a single article about Curry's views of mathematics.[22]

There are two aspects of Curry's philosophy, usually regarded as important, we did not pay too much attention to. First, this is Curry's attitude towards consistency proofs, which – as already mentioned – is the aspect Bedürftig and Murawski in [11] are concentrating on. For Curry the consistency proofs were "neither necessary nor sufficient for acceptability." ([31, p. 61]) This is also stressed in [104, section 6.5]. The focus on consistency is understandable; for both sources want to explain the difference to Hilbert's formalism, which is mainly discussed. Shapiro adds that Curry's version of formalism is not susceptible to the Gödelian incompleteness theorems. Although mentioned in [100] for Seldin this aspect does not seem to be too important because he does not classify Curry as formalist and so does not need to demarcate Curry's against Hilbert's view. The same is true in our case.

The second aspect is metatheory, which – according to Shapiro ([104]) and Weir ([114]) – is the most important one in Curry's work. The latter stresses that the metamathematics becomes a contentful theory since it is about propositions with truth values. Seldin discusses metatheory many times, too. In [100] he stresses the importance of metatheory for Curry and sketches the project of Curry to formalise it. In [101, chapter 4] and [102] Seldin presents a very simple formal system for

[22] In fact, his name occurs only once in a philosophical context (in a review by Christopher Pincock). Instead the Curry-Howard correspondence (see [61]) is mentioned a few times. So – if at all – Curry is rather famous for his technical contributions.

counting (which originally stems from Curry) and states that it is possible to construct all of arithmetic and calculus as well as other important domains of mathematics as part of the metatheory of that system. Of course, the methods allowed in the metatheory need to be sufficiently strong for that endeavour. In [102] it is argued that Curry's methodological constraints on metatheory were based on pragmatism. The reason we do not stress metatheory explicitly is because it is only a system like any other in our view; having most of the times a lower degree than the formal systems it investigates as objects. This way of thinking metatheory will be important in chapter 6.

On the other hand our focus, the split of truth into the objective provability and the subjective acceptability enabling mathematics to be a pluralistic science, is not appreciated even remotely in the other sources. The split itself is mentioned by Seldin in [100] and [102][23], but there are no relevant conclusions drawn from it. Shapiro and Weir mention that Curry's view is anti-metaphysical (see [104, section 6.5], [114, chapter 6]). Like us they adduce Curry's statement that mathematics has "to be independent of any except the most rudimentary philosophical hypotheses" ([31, p. 3]). Neither of them, however, links this to provability and therefore to Curry's endeavour of saving objectivity in mathematics. Shapiro in [104, section 6.5] mentions Curry's acceptability, too. Yet, Shapiro discusses that aspect in isolation. There is no appreciation for the concept justifying and endorsing a variety of mathematical practices.

[23] In the 2011 article this split is compared to Rudolf Carnap's treatment of existence.

Chapter 4

Formalisms and logics are similar

4.1 Introduction

For a better understanding of a new concept it is often expedient to integrate it into some larger context or to compare it with another already more familiar concept. In this chapter we will conduct a mix of both approaches. In concreto, we will integrate the concept of the degree of formality into the wider and new concept of formalism. The latter in turn will be introduced in comparison to Curry's treatment of logics, as presented in section 3.4, first.

We will proceed as follows. First, we will discuss the combinability of logics and formalisms introducing the latter that way. The discussion will result in a formula that in turn allows us to re-characterise the degree of formality in terms of a component of that formula (all this in section 4.2). Then, we will see that – in some sense – there is a necessity to combine logics and formalisms with other systems (section 4.3). The subsequent chapter will address the monotony and the deductive nature of both concepts. After that we will show that they have a modelling character (section 4.5). Up to then we will have pointed out that logics and formalisms share many properties. In the last section of this chapter we will see that appreciation of logics leads to appreciation of higher degrees of formality and vice versa. Except for the work of Curry, which we have already discussed in detail, we are not aware of any related work and so there will be no respective section in this chapter.

4.2 Combinability, formalisms, and re-characterisation of the degrees of formality

Usually when we pick out two different high degree systems arbitrarily, for instance our first order version of Peano arithmetic and ZF, and put them together, we obtain only nonsense. In the above case Peano axioms enable us to conclude that every set is 0 – for which it is not clear why it should be the same as $\emptyset$ here – or of the form Suc x for some x. Due to set theory we are allowed to construct the set $\{0, \text{Suc } 0, \text{Suc } (\text{Suc } 0), \ldots\}$[1], which itself has to be 0 or a successor. So it contains itself, in contradiction to the ϵ-relation being well-founded in ZF.

In section 3.4 we have already seen that many high degree systems can be divided into a contentual and a logical part. This split invites us to combine the contentual part with logical parts of other systems and the logical part with contentual parts of other systems. So we might mix, for instance, the contentual part of Zermelo-Fraenkel set theory with classical or constructive logic or even second order logic at will as long as the two systems are compatible on the formal side.[2] [3] In the same way we can combine classical first order logic not only with Zermelo-Fraenkel set theory but also with, say, group theory or arithmetic. In contrast to the more general case of mixing arbitrary systems discussed in the previous paragraph in these cases we can never create total nonsense.

So far we spoke of the contentual part of a system in order to distinguish it from the logical part or form as we might have called it as well. Now, however, we have to distinguish the formalism from the remaining part, which therefore is also contentual. To avoid confusion we will from now on speak of the *subject-specific* part instead of the contentual part in the former case. The split Curry considered implicitly can be called *logical division*. It can then be expressed by type theoretical means as

$$\text{systems} = \text{subject-specific systems} \times \text{logic systems},$$

i. e. every system has a subject-specific and a logical part. Of course Curry only considered what we call high degree systems but there is no need for that restriction here.

[1] We use the existence of the set of natural numbers in ZF and the axiom schema of replacement for that step.

[2] This restriction will become clearer once we will have developed the complete division formula.

[3] Admittedly, inside a type theoretical setting some logic is fixed at a global level because of the Curry-Howard correspondence (see [61]), which binds the treatment of logic to the one of types. Nevertheless axioms can be added at will, for instance the tertium non datur. Furthermore, we can introduce each logic locally as a new datatype together with the respective axioms.

We now claim that a similar *formal division* is possible. To see this let us consider the subject-specific part of the high degree first order version of Peano arithmetic as presented in section 2.4. On the one hand we have a *contentual part*, meaning this time a collection of entities presenting the whole aspect of natural numbers. Ideally this part should not have any form. On the other hand we have the *formalism*. This itself consists of a tool to build systems, called the *kind of formalism*, and some instance produced by that tool. In our example the kind of formalism tells us that we get constants, function symbols, as well as predicates and it gives us a way to write down concrete axioms. The concrete instance gives us the constant symbol 0, the function symbols Suc, $+$ and $*$, the predicate symbol $=$, their respective arities as well as the concrete axioms (seen as symbolic strings).

The formal division is not specific to high degree systems. Our subject-specific part of the medium degree formalisation of Peano arithmetic, for instance, has the same contentual part since it does not depend on the form. The kind of formalism is some explicit frame allowing us to enumerate various axioms in a way that mixes natural and symbolic language. The concrete instance enumerates the five axioms using some notions like 0 and Suc and the defined concept of injectivity.

We can apply the formal division not only to the subject-specific part of systems but to the logical part – and therefore to the whole system – as well. The contentual part of the logical part of a system is a collection of true and false logical statements of that logic, ways to reason therein, etc. The kind of formalism is the same as in the subject-specific part and we have some concrete instance, telling us, for example, what the logical axioms (again seen as symbolic strings) look like.

Combining both, formal and logical division, and using type theoretical notation we can express the type of systems in the following way:

$$\begin{aligned}
\text{systems} &:= \sum_{c \,:\, \text{contents}} \text{formalisms } c \\
&:= \sum_{(s,l) \,:\, \text{subject-specific contents} \,\times\, \text{logical contents}} \; \sum_{k \,:\, \text{kinds of formalism}} \text{instances}(s,l,k) \\
&\cong \sum_{(s,l,k) \,:\, \text{subject-specific contents} \,\times\, \text{logical contents} \,\times\, \text{kinds of formalism}} \text{instances}(s,l,k)
\end{aligned}$$

For those not familiar with type theoretic notation: the above simply states that every (mathematical) system is a quadruple containing a subject-specific content, a logical content, a kind of formalism, and a concrete instance of that formalism, where the instance possibilities are limited by the choice of the other three components.

Although a system, in which the instance or even the kind of formalism itself is replaced by another one, is different from the original one, it is common practice to name both variants the same. One reason for this might be that we usually think of a system in a more informal way. On this lower degree the different variants may be confluent. So the necessity of differentiation disappears from that point of view.

Let us now discuss the re-characterisation of the concept of the degrees of formality for systems. We argued in section 2.3 that the use of natural language as well as condensation determine the degree of formality. Both of these adjusting parameters, as we called them, are in turn determined by the respective kind of formalism. By contrast, the instance of a formalism can only influence the sheer number of axioms, which is not the form of condensation we have in mind.[4] So the degree of formality of a system does depend on the kind of formalism alone, i. e. there is a "degree function" mapping every kind of formalism to some kind of value. In particular, we could speak of the degree of the kind of formalism instead of the degree of formality. Since the latter is more concise and intuitive, however, we will stick to that formulation.

In this section we focussed on systems, not on argumentations. The results, however, are transferable. An argumentation, too, can be seen as a quadruple of a subject related part, a logical part, a kind of formalism – also called argumentation (or proof) style –, and a concrete instance. Note that also in this case it will be the kind of formalism that decides the use of (natural) language and condensation. Furthermore it will determine the frequency and precision of justifications, too. Hence we can conceive the degree of formality in argumentations as the degree of the respective kind of formalism in the same way this could be done for systems.

In additon to that, it makes sense to evaluate the degree of an argumentation style (or of a proof style respectively) without any concrete argumentation (proof) at hand. We will use the same abbreviations as for systems and argumentations. In particular, we will speak of low, medium, or high degree argumentation (proof) styles (see chapter 7).

[4] Analogously, a proof does not become more formal only because it is more cumbersome and therefore, say, twice as long as another proof.

4.3 Necessity of combinations

So far we analysed how subject-specific contents, logical contents, and formalisms for them can be combined. In this section, we will discuss why these have to be combined. Let us first see what happens if we remove the logical part of a system like Peano arithmetic or ZF. Such a system $(s, \emptyset, k, i)$ has only subject-specific contents, for which there are no devices left to link them. Hence it atrophies to a collection of facts. Therefore we do not only have the possibility to connect subject-specific systems to the different logics but there is the necessity of such a combination, too – at least if the respective system should deliver new results.

The analogue situation with formalisms is even worse. A system without its logical part is useless but at least it is always clear what we are referring to. Yet, that is not the case if we leave out the formalism, obtaining a quadruple $(s, l, \emptyset, \emptyset)$. The contentual part of a system is an idealisation and we will never be able to see such a completely formalism-free entity. As an approximation it might help to imagine such systems as physical elements having such a short period of decay that one will never observe them in nature.

Next, we will show that the relation between the logical part and the subject-specific part is similar to the one between the formalism and the overall contentual part. Systems of the form $(\emptyset, l, k, i)$ can only derive tautologies. How narrow this restriction is depends on what we are willing to count as logic. Second order logic, for instance, is mightier in this sense than first order logic since functions and predicates are involved; enabling mathematical notions like Church numerals or sets (as characteristic functions). Permitting Frege's courses of value even makes logic too strong, resulting in inconsistencies.[5] However, even if we have a rather restricted view of logic it still strives to explain the world. Historically this desire can be traced back to Aristotle's time (see [7]) but even today we teach logics with the help of rain and wet streets. These examples do indicate that it is necessary at least from a psychological point of view to link logic with reality.

The situation with formalisms as isolated systems, i. e. with systems of the form $(\emptyset, \emptyset, k, i)$, is again even worse. Strictly speaking we must have $k = i = \emptyset$ in such cases since otherwise the instance i would have to depend on some content. However, we can weaken the premises by stating that we are just not interested in the contents, i. e. we conceive the respective systems as games with no relation to reality or other contents. This works well for high degree systems but the lower the degree of formality the less clear the situation becomes. From where, for instance, do the implicit assumptions stem or what should a proof idea without any relations

[5] A nice explanation of this exact inconsistency can be found in [70].

to contents look like? Leaving the problems pertaining to the existence aside, there is still another problem analogue to the one of logics without subject-specific contents just discussed above: without any content there is no emphasis on which theorems are important, aesthetic etc. In first order, for instance, we could add quantifiers where the quantified variables are never used, we could rename things arbitrarily, write everything in polish notation etc. and everything would be as good as the original version. In reality, however, humans and mathematicians in particular are not able to have such an attitude. In the case of formalisms we have the same problem as with logical systems: the temptation of contentual linking is irresistible.

Let us sum up. In this section we saw that logics must be combined with subject-specific contents – and vice versa – and that formalisms must be combined with the overall content – and vice versa – in an analogous way.

4.4 Monotony and deduction

When using logics we want to get from truth to truth using deduction. Furthermore in most logics, whenever some proposition has been proven to be true, it should – ideally – always keep this status.[6] The technical term of this is called monotony.

Let us now test whether formalisms satisfy both properties. If we have a high degree system together with a high degree proof style then the whole setup is about where to start from and how to gain new certain theorems with the help of the already proven ones. So both points above are obviously fulfilled in this case.

However, even in lower degree systems and proofs there is the same ambition since the reasoning frame is firmly established in nowadays' mathematics (compare section 2.3). Only the success and the expectation of it will change. In practice one might therefore be more moderate in pushing theorems forward this way; for the risk of an error in between increases significantly. Perhaps before the time Euclidean mathematics arose deduction has not been the aim. Yet, even there one might argue, that the respective formulae could be used to "deduce" new information about real life objects not considered so far. At least monotony was already part of the pre-Greek mathematics, since neither a formula itself nor the calculations done with it are lost by using the respective formula.

[6] Linear logic (see [49]) is a famous exception. In it a proposition is seen as a resource that is spent when it is used.

4.5 Modelling

In this section we will show that both, logics and formalisms, have a modelling character. By this we mean that some input is transferred to logics or formalisms, some results are found there, and these are transferred back to the original level.

Perhaps surprisingly, this can be seen best in the case of low degree systems and argumentations. In reality the task might be to paint some building. For calculation everything is transferred into geometrical forms with length specifications. After that formulas are applied to determine the required value. Finally, these results are transferred back into the real world, determining in this case the amount of paint needed. It is pretty much the same with physical formulas. Reality is modelled by concepts and corresponding values and units. Computation with such formulas delivers new results. Such results are transferred back into reality, for an example that some ball flies to a certain point.

In such modellings of reality we have no way to confirm our reasoning directly. Reality has no argument we could rely on. We can only falsify our modelling when its results contradict reality. Furthermore we cannot model all of reality: we can always ask questions about reality which are not answered by our low degree formalisms, e. g. what the most suitable colour for the building in the example above is.

Let us now turn our attention to higher degree systems and argumentations. Often they do not model reality in a direct way, but systems and proofs of a lower degree instead. The medium degree system for Peano arithmetic we presented, for instance, models our way of dealing with natural numbers on a less formal level, i. e. there are a few axioms that should mirror the more informal system. Inside the higher degree systems and argumentations we prove new theorems and transfer such results into the more informal language.

The indirect modelling, i. e. the modelling of lower degree systems and argumentations, allows us not only to falsify the higher degree ones but to ask for each higher degree argumentation whether it is correct with respect to the one of lesser degree. Verification only requires to show that every valid step in our higher degree argumentation is valid in our argumentation of lesser degree. By contrast to the modelling of reality, when modelling more informal systems and argumentations completeness is not absurd. Some of the higher degree systems with their corresponding argumentations can mirror every proposition of the more informal one; and even if this is not the case the question might be worth the consideration.

Next, let us consider logics. Like lower degree formalisms and argumentations they can model reality. In fact, all logical entities expressing that some statement follows from another – like implication or logical consequence for instance – are "a matter of the laws/regularities of nature." ([81, p. 10]) So experiences like "first it rained and then the street was wet" or "the sun started shining and it was

much warmer after that" over time were translated in "If it rains then the street will be wet" and "if the sun shines then it will be warmer".[7] Since both of the latter propositions share the same structure – namely that of a simple implication – we can translate both of them into $A \rightarrow B$. As soon as we have collected further factual data we can infer new theorems inside logic. If we, for instance, have in the street-example the additional factual information that it rains we have an implication $A \rightarrow B$ and an information A. So we can conclude B via modus ponens. We translate this result back to reality: "the street is wet". In contrast to this we do not have "it is warmer now" since we have no additional information about the sun shining. Although nowadays logics are mainly parts of pure mathematical systems this does not change the way of proceeding described above. The only difference is that we do not apply real life situations but formulae.

Since without logic we do not have any transitions to bridge between different real life situations, propositions etc. there is nothing like correctness for logic, i. e. there is no way to verify each logical step with the help of some extraneous non-logical device.[8] Falsifiability on the other hand – like in the physical models – is at our disposal.

To postulate completeness with respect to logic would essentially amount to assuming that each entity in the world is logical. This, of course, is a very extreme point of view. One way to come to such a conclusion might be to see everything as mathematical and everything mathematical as logical.

In this section we have seen that formalisms and logics are modelling tools. In both cases – like in physical models – falsifiability is present. Systems and argumentations of a higher degree of formality do have the additional feature to generate meaningful questions about correctness and completeness.

[7] Another way to state this is that post hoc observations became propter hoc via abstraction (see [107]).

[8] This of course does not mean that there is no correctness regarding higher degree logical systems representing less formal ones; where the latter is meant to include set theoretical semantics since the respective systems are in need of some logic behind them.

4.6 Appreciation of logics and the one of higher degrees of formality

Up to now we compared the properties of logics and formalisms. In this section we will see that and discuss why an appreciation of logics usually leads to an appreciation of a high degree of formality and vice versa.

That logic attracts a high degree is a historical fact: already the Aristotelian logic had a rather high degree of formality and the first high degree systems (for all of mathematics) were again logically motivated.

So why do we have this attraction? Logics are very general and abstract[9], which is why in contrast to, say, Euclidean geometry with its – although already abstracted – still more concrete objects, logics lack intuition. For instance, we would not be tempted to conclude that there is a unique straight line between to different straight lines, because points and lines are different. However, similar typing problems, like self-reference or treatment of existence[10], led to considerable confusion in case of logic.[11] In Aristotle's time this confusion culminated in the Sophist's challenge to be able to argue for every point of view by logical means. This provoked a reaction. Aristotle's answer was twofold:

1. He denied total generality by restricting (his) logic to legitimate concepts. This part was metaphysical and very unfruitful at best (see [7]).

2. With the syllogisms he gave rules of a rather high degree of formality for at least some logical inferences.[12]

The commitment of logic to high degree systems and argumentations did not change since then. Whether it be George Boole, Gottlob Frege, Bertrand Russell, or modern mathematicians, they all used (respectively use) high degree systems and argumentations for their logical work.

Before the time, in which formal methods start to become more popular, logic was not appreciated too much. In a passage of Goethes Faust (part I, verse 1908 et seqq.), for instance, Mephistopheles criticises logic because of the very small steps in its argumentation lines. According to him, intuitively obvious knowledge is reproduced through very cumbersome inference rules (at that time still the syllogisms). Nothing new, nothing of importance in particular, is developed in logic. All the formal rules are only obstacles. We do not want to discuss whether

[9] According to [81] thinking in a general and abstract manner was a precondition for logic to arise.

[10] We allude here to the proof of God's existence by Anselm of Canterbury.

[11] A similar argumentation can be found in [70, chapter 1].

[12] This is all we need for our argument. However, it seems as if his syllogisms were more ambitiously made to somehow mirror all valid arguments (see [106]).

Mephistopheles expresses the opinion of Goethe.[13] For us it suffices to know that there must have been contemporaries of Goethe arguing like that. Furthermore note that Goethe was not a mathematician but "only" very well-educated. In such a position knowledge of mathematics might have been one or two generations behind.

The situation regarding the appreciation of higher degrees of formality changed significantly at the latest in the beginning of the 19th century with Cauchy's work. Indeed with Boole's work it follows directly an important step in the history of logic. Admittedly, already Leibniz, living one and a half centuries before, with his calculus ratiocinator had comparable ideas to treat logic arithmetically but this exception confirms only our claim since Leibniz' inclination to formalisms with his characteristica universalis was also exceptional. Let us finally mention that the strive for finding a mathematical foundation "generated" new logicians like Frege and Russell, which in turn improved the formalisms as already discussed.

Our argumentation, however, is not merely historical. Nowadays most mathematicians are not too interested in foundational issues. They usually use a fixed logic in some implicit way but they do not really esteem it. In informatics on the other hand logic is much more popular than in mathematics.[14] One may argue that this is due to computers working with logic. Yet, this argument is not convincing since computers are working with electricity, too, but there is no overwhelming interest in electricity in informatics. Instead the salient point seems to be the practice of reasoning on a higher degree of formality, which is typical only for informatics but not for mathematics.

Altogether we have shown in this section that an appreciation of logical aspects leads to an appreciation of a high degree of formality and vice versa. This concludes our overall investigation of the relation of logics and formalisms. It showed us that both concepts are very close. Since we were able to integrate the concept of the degrees of formality into the one of formalisms this also helped to clarify the former even further.

[13] Paul Lorenzen in [75] argues that this is indeed the case.

[14] Jeremy Avigad, a philosopher of mathematics and a logician, said at a conference that he really likes to work with computer scientists since they do appreciate logic so much.

Chapter 5

Benefits of the different degrees of formality in argumentations

5.1 Introduction

Often proofs of a medium degree of formality are not admitted a value on their own. Instead they are seen as a tradeoff between the desirable high degree proofs and the resources, which do not suffice to always work on a high degree. As mentioned in section 3.5, Curry, for instance, did think about high degree proofs as a primary standard not being suitable for daily use. In this chapter we will argue that this point of view is mistaken, i. e. we will delineate why medium degree proofs have a great value on their own.

Paradoxically, it is just the essence of Curry's view of mathematics that will serve us as a guideline. Our endeavour to find the benefits of the different degrees of formality is analogous to his acknowledgement of different formal systems for different reasons. Furthermore his split into a subjective and an objective component of truth will serve as a model when we discuss the purposes of argumentations (see below).

We proceed as follows. In the next section we will discuss the purposes of argumentations mentioned in the last paragraph. What are they made for? What do mathematicians try to accomplish with them? In other writings that we will use to develop our answer, this topic is treated only in side remarks. In section 5.3 we will discuss how the three adjusting parameters of formalisation, use of natural language, degree of condensation, and frequency and precision of justifications, can contribute to the purposes. Based on this and as a kind of summary follows the same discussion for argumentations of the different degrees.

5.2 The purposes of argumentations

In this section we will first discuss some (side) remarks from other authors on the purposes of proofs[1] to ascertain more systematically the purposes of argumentations afterwards.

Let us start with Alan Robinson who in [94] enumerates tasks that a proof should fulfil. To this belong: to guarantee certainty, to provide understanding, to present ideas, to reveal the heart of the matter, and to create conviction. In addition to that he says that proofs should be simple, clear, elegant, and compelling. By contrast, high degree proofs – or formal proofs in his terminology – "seem always to be essentially hard to understand, and inevitably complicated." ([94, p. 269]) He continues by deploring what he calls the post-formalisation blues: "Formalization seems to hide the ideas or even to destroy them." ([94, p. 269]) This is illustrated by two problems, each having a very simple low degree solution – informal solution in his terminology – but only complicated high degree proofs that reveal no involved ideas. Robinson concludes that "informal proofs are the real stuff of mathematical understanding." ([94, p. 280])

In [90] Andrzej Pelc argues that (in general) there cannot be any sort of relation between medium degree proofs – simply proofs in his terminology – and high degree proofs – derivations in his terminology – that could explain why we trust the former ones. His reasoning is based on the thought that the resources needed for high degree proofs of an existing medium degree proof may exceed the ones the whole universe can offer. Pelc compares a mathematician with a biologist or trainer working with animals both of who do not explain the respective animal's behaviour with biochemical reactions in the animal's brain but who do their job on a more macroscopic level. For us the salient point is that Pelc's argumentation supports the view that medium degree proofs are valuable on their own; even with respect to matters of correctness. There is another line of argument leading to the same conclusion in [108] by Tanswell, whose characterisation of medium degree proofs – (informal) proofs in his terminology – we already mentioned in section 2.7. He reasons that there are too many very different formalisations of the same medium degree – informal – proof such that the latter cannot function as an indication for some higher degree proof. From that we draw the conclusion that correctness of a medium degree proof cannot depend on the correctness of some higher degree one alone.

The convincing role of medium degree proofs is stressed by John W. Dawson Jr. in [34]: "we shall take a proof to be an *informal* argument [medium degree argument in our terminology] whose purpose is to convince those who endeavor to follow it that a certain mathematical statement is true (and ideally, to explain

[1] Other authors do not care about argumentations that are no proofs.

why it is true)." ([34, p. 270][2]) So the convincing power is related to truth but it is also subjective in character: "[b]ecause standards of rigor have not remained constant, arguments that once were accepted as convincing may no longer be, while on the other hand, a rigorously correct proof may fail to be convincing to those who lack the requisite background or mathematical maturity. (And some results, such as the Jordan Curve Theorem, may appear so obvious that it requires mathematical sophistication even to understand the *need* for a proof.)" ([34, p. 271]) As in the case of Robinson, high degree proofs – formal proofs in his terminology – come off badly: "they are difficult to comprehend, and despite their rigor they are often *unconvincing*, because although they provide *verification* that a result follows logically from given premises, they may fail to convey *understanding* of *why* it does." ([34, p. 271]) Solomon Feferman in [38] emphasises that understanding every step of a proof and really understanding the proof as a whole (and on a higher level) are two very different animals. A proof should convince us that the theorem is true in a meaningful sense. Jody Azzouni adds "that ordinary rigorous [i. e. medium degree] mathematical proofs often (perhaps usually) confer certainty, a being convinced that the proof is right, in ways that *aren't* conveyed by the proof's formalized cousin." ([5, p. 249])

Obviously, all the authors discussed in the last three paragraphs would not agree that low and medium degree proofs are only secondary standards that are just easier to conduct. It would be nice, however, to have a characterisation of the proof's – or better argumentation's – purposes at hand that corroborates this point. The formula Proof = Guarantee + Explanation claimed by Robinson in [95] can be seen as such a characterisation. This excludes or at least degrades high degree proofs since – at least from Robinson's point of view – they are really bad regarding explanation. Medium degree proofs, by contrast, fulfil both aspects and are thus superior. However, we are not too happy about this formula. First, the equality sign is problematic since the right hand side is no real division of the left hand one but rather a list of tasks that can or must be satisfied.[3] Second, the property named 'explanation' seems to be too imprecise. There is a difference between an explanation of what is going on and an explanation why something works. The latter of both meanings is part of the convincing role of proofs, which even Robinson himself adduces. This role, however, is coupled with some concept of truth and that in turn should be rather connected with the guarantee aspect. So the convincing role cannot be classified in a convincing way, which is our last point of criticism.

[2] Emphases are adopted from the original here and in the remaining section.
[3] Compare this use of '+' with ours of '×' and 'Σ' in the previous chapter.

Instead we propose to characterise an argumentation by the following three main purposes:

1. Illustrate the idea(s): it should be clear what is happening in the whole argumentation and its respective parts; in particular orientation must be provided. Furthermore it must be clear what tools (in the widest sense of the word) are used and how they function.

2. Convince writer and reader: both should believe that the argumentation establishes the truth of the theorem under consideration and it must be clear for them why this should be the case. In particular one must not have the feeling that any step is missing.

3. Deliver objective[4] guarantee: independent from any human the argumentation should be a guarantee for the theorem such that everyone can rely on it.

In line with Robinson, Feferman, and Azzouni we think that there can be argumentations doing a fine job regarding objective guarantee that are nevertheless not convincing. Furthermore there can be very convincing argumentations that are objectively false. In Imre Lakatos' famous 'Proofs and Refutations' [71] many examples of that kind can be found. However, an anecdote of Lamport perhaps shows best that we can be convinced by a proof although the latter has no objective guarantee:

> "Some twenty years ago, I decided to write a proof of the Schroeder-Bernstein theorem for an introductory mathematics class. The simplest proof I could find was in Kelley's classic general topology text [...]. Since Kelley was writing for a more sophisticated audience, I had to add a great deal of explanation to his half-page proof. I had written five pages when I realized that Kelley's proof was wrong. Recently, I wanted to illustrate a lecture on my proof style with a convincing incorrect proof, so I turned to Kelley. I could find nothing wrong with his proof; it seemed obviously correct! Reading and rereading the proof convinced me that either my memory had failed, or else I was very stupid twenty years ago. Still, Kelley's proof was short and would serve as a nice example, so I started rewriting it as a structured proof. Within minutes, I rediscovered the error." ([72, p. 604-606][5] [6])

[4] In section 6.4 we will discuss the requirements for objectivity. For the moment the intuitive grasp of this concept suffices.

[5] We omitted a figure in between.

[6] See section 2.7 for a brief discussion of his structured proofs.

Before analysing the three main aspects of formalisation with respect to the purposes listed in the second last paragraph let us briefly mention that we can now define proofs in terms of argumentations in a relative elegant way: proofs are those argumentations that are doing well in convincing and/or in delivering objective guarantee.

5.3 Contributions of the different aspects of formalisation with respect to the purposes of argumentations

Natural language is a very important factor for all of human communication. In particular, it is used to convince other members of some social group; for in natural language one can tell a "story", which in turn can serve as a mental simulation of whatever is currently under discussion. The domain of mathematics is no exception regarding that point. Natural language helps to structure an argument, to accentuate some of its parts, and so to develop a "mathematical story".[7]

However, the inclination to heed stories makes humans susceptible to mistakes, too. Outside of mathematics, for instance, a human may be persuaded by another human with the help of a good story to buy something he does not need at all.[8] In mathematics, by contrast, – hopefully – there is no villain who deliberately deceives others. Nevertheless we might be misled by the coherence of a good mathematical story – even of our own one –, since neither it nor the natural language used to tell it are made to cover the subtleties involved in nowadays' mathematics. Hence the use of natural language can be a hindrance regarding objectivity. On the other hand natural language connects different concepts involved in the respective story, reinforcing the web of belief[9] and thus making it easier to spot mistakes in any of those concepts.[10] So natural language can increase and decrease the objective guarantee of arguments. The magnitude of both effects depends on the concrete argumentation.

To communicate ideas we use natural language but also other tools such as diagrams, gestures, or everyday items as well. Sometimes the latter are more suited to convey the salient point. Even if no external tools are involved the communication of ideas often leads to some kind of mental representation resembling some

[7] This point is also emphasised by Robinson in [94].

[8] See [2, chapter 3] for a description of phishing techniques based on story telling.

[9] Mentioning this concept is not meant to imply that we are completely in line with the book [91] of Willard v. O. Quine and Joseph S. Ullian having exactly this title.

[10] We will discuss a similar kind of argument in more detail in section 6.4.

external tool.[11] So it seems that natural language is sometimes not too useful in illustrating ideas directly but that it can be a good mediator instead.[12] It helps to bring our attention to the right point or tool. This can be done best when natural language is used in a concise manner.

Condensation as well as frequency and precision of justification are not less important but their strengths and weaknesses are more obvious such that we can be more concise about them. Clearly, condensation helps the illustration of an idea while justifications are only a hindrance regarding that aspect. The situation can be compared to that of proverbs, which must be to the point, too. Similarly clear is the situation regarding objective guarantee: the less the condensation, i. e. the smaller the single steps are, the greater is the level of objective guarantee. Every justification enhances that aspect; the more precise the better.

The less condensation and the more justification the better is not true with respect to the convincing power. In respective argumentations humans lose orientation of what is going on and of what the essential points are. Such argumentations are not more convincing than a story is fascinating which adduces every irrelevant detail. On the other hand a plot alone is no story and likewise a mathematical argumentation needs to go into some detail to be convincing. What exact level of condensation and justification one finds most convincing depends on the individual.

5.4 Benefits of the different degrees of formality

We have finished our preparations and can now evaluate which degrees of formality serve which purposes best. Let us start with argumentations of a low degree of formality. They can be characterised by a high degree of condensation, the concise use of natural language and/or other media, as well as minimal use or even the waiving of justifications. As discussed, all this increases the potential to present ideas. So low degree argumentations are made for just that aspect. Regarding the convincing power the level of condensation and justification are not optimal but they might still contribute to some extent. The use of natural language – even in its concise form – might help, too. We did not discuss the convincing power of other media but whatever the outcome would be it should not change the overall picture too much: low degree argumentations are an acceptable tool for convincing. By contrast, it is only the natural language that at least partially contributes objective guarantee. So low degree argumentations are not suitable to fulfil this task.

[11] Our first example in the introduction of this thesis was of that kind. The mental representation was an automaton there.

[12] Compare this with the idiom “A picture is worth a thousand words”.

Next, the argumentations of medium degree mainly use natural language. Furthermore they have a medium degree of condensation and justification. This is the perfect framework for being convincing. Although medium degree argumentations are too detailed and although it would be better to use natural language in a more concise way to focus on the most important aspects or to generate mental representations, medium degree argumentations can still convey ideas to some extent. The same is true with respect to objective guarantee; this time the reason being that medium degree argumentations are not detailed enough and that natural language does contribute and destroy at the same time. Altogether we see that medium degree argumentations are relatively balanced.

Finally, high degree argumentations have no or only little condensation, an abundance of justifications, and no natural language is used. So there are no influences spoiling objective guarantee. On the other hand any of the above characteristica hampers the illustration of ideas. The situation with respect to convincing power is not overwhelming, too. High degree argumentations are too detailed regarding that aspect but some contribution might nevertheless be made by them. Unfortunately, the waiving of natural language has a negative impact.

This analysis shows that neither of the degrees is the overall primary standard. Each of them is very good in fulfilling one task to the detriment of the other two. The amount of this detriment, however, is not the same in each case. We ascertained that medium degree styles are the most balanced ones. This could be a resource-independent reason why even nowadays medium degree proofs are the preferred style in mathematics.

Bear in mind that our division into low, medium, and high degree argumentations is meant to express tendencies. Although we would assign all textbook proofs a medium degree[13] this does not mean that there are no differences between them. The textbook proofs presented in [1], for instance, are a little bit closer to proof ideas while the one presented in figure 1.3 is a bit more formal than the average textbook proof.

[13] This must not be confused with the statement that every text that is written in textbooks after the word 'proof' is of medium degree.

Part II

Demonstration of the usefulness of the concept of the degrees of formality

Chapter 6

Three points of criticism regarding Curry's view of mathematics

6.1 Introduction

In this chapter we come back to Curry's view of mathematics; this time from a sceptic's point of view. First we will discuss three problems his view implies (section 6.2). Then – and that is the reason we deal with his view again – we will show that those problems vanish in light of the different degrees of formality (section 6.3). For this consequence to be of real value, however, we have to show also that the different degrees of formality do not destroy – as a side effect – what we asserted in chapter 3 to be the essence of Curry's view of mathematics. Since subjectivity and pluralistic tendencies are obviously not affected in a negative way we only have to exclude that objectivity is lost (section 6.4). We conclude this chapter with a related work section discussing briefly the awareness – the awareness of Curry in particular – regarding our three points of criticism. In addition to that we will mention two other points of criticism in that section.

6.2 The three points of criticism

Our first point of criticism is the most common one. Stated in our terminology it complains that all processes of mathematics standing outside the examination of high degree systems are not taken into account. That is mathematics could be only found where there is work in, on, or with high degree systems.[1] Historically such an understanding of mathematics is untenable, because up to the 19th century there was no system of a high degree of formality[2] and therefore – fol-

[1] Use, creation/improvement as well as metatheory of high degree systems is meant.
[2] The first high degree system is probably Frege's Begriffsschrift from 1879.

lowing Curry's definition – no mathematics at all. This, of course, is a conclusion most mathematicians and non-mathematicians would not accept. The situation becomes even worse when we remind ourselves that even nowadays' mathematics is more complex than Curry's definition of mathematics is able to cover[3], though the work with high degree systems has attained a decisive role.

Next, let us concentrate on acceptability. Curry cites it as reason why mathematicians deal with particular formal systems (in Curry's understanding of the word) and not with others. However, not every such system is examined because the one who does so, accepts it. On the contrary the goal of an analysis might be the refutation or questioning of the underlying system. Of course, in the case of success the result of such investigations could be an improved system which one would accept for reasons of acceptability, but ad interim one has worked within a system which one has assessed as false. An example for such a procedure is to be found, for instance, in Bertrand Russell's analysis of a proof of Georg Cantor – and therefore also of Cantor's set system – that there is no greatest number. Russell doubted that result since for him the number of all things in the world should be the greatest. The result of that investigation was Russell's antinomy leading to improved systems like that of Principia Mathematica.[4] However, the examination of such false systems cannot be explained by Curry's acceptability since the latter is coupled with some concept of validity (see section 3.5).

For the last point of criticism let us first have again a look at Curry's definition of what formal systems are. In chapter 3 we cited the definition but we did not mention an odd and vague addendum stating that all generalisations belong to the definition as well, as long as the definite character is not affected (see [31, chapter IV]); where definite character means that the recognition of terms and elementary theorems as well as proof verification is recursive[5] while theorems are recursively enumerable.[6]

Indeed Curry considers suitable alternative definitions of what formal systems are. One of them is to allow only syntactical formal systems which he calls calculi:

> "In a calculus it is explicitly stated that the objects we are dealing with are symbols. We start with a certain stock of symbols and with two kinds of rules for manipulating them. The first kind of rules, called *formation rules*, specifies recursively a certain set of *expressions* – that is, linear sets of symbols – which set I shall call *formulas*. The second

[3] This can be seen, for instance, in the analysis of the mathematical processes given by Michèle Friend in [48, chapter 5.5].

[4] See [96, chapter Principia Mathematica] for further details.

[5] A set is said to be recursive if there is a Turing machine deciding for each input in finite time whether it belongs to the respective set.

[6] A set is recursively enumerable if there is a Turing machine listing all elements of the set in a possibly infinite progress. There is no feedback that a particular element is not in the set.

> set of rules, called *transformation rules*, specifies a class of formulas which I shall call *assertible formulas*; the rules consist of first a definite list of formulas, called here *axiom formulas*, which are assertible, and second, rules determining recursively how further assertible formulas are to be constructed." ([30, p. 232][7])

Note that there is no reference to syntax or symbols in Curry's original definition of what formal systems are. A second alternative definition would be to accept only assertional formal systems, i. e. systems in which $\vdash$ is the only predicate. Our Peano system in section 3.2 is an instance of this more restricted variant.

Curry justifies this indeterminacy regarding his concept of formal system by citing the irrelevance of ontology in mathematics; but let us examine this point more carefully. It alludes to the treatment of mathematical objects, like natural numbers or real numbers for instance, in mathematics. It does not matter whether they are Zermelo- or Von Neumann ordinals (or whether they are cuts, Cauchy-sequences, or nested intervals respectively). What matters only is that the presentation satisfies the characteristic axioms, i. e. Peano axioms (or the axioms of an ordered field together with the existence of a supremum for nonempty, bounded above sets respectively). That the different variants satisfy the required axioms can be proven on a high degree level in the formalised Zermelo-Fraenkel set theory, for instance. However, this is the salient difference to the case of different definitions for formal systems; for we have no superior instance to prove the equivalence of all possibilities to define what formal systems are.[8]

Admittedly, Curry argues for the equivalence between his formal systems and calculi[9] and we can think of similar arguments for the comparison of other kinds of formal systems. Yet, first this requires infinitely many alternatives to consider and, second, the particular arguments are not on a high degree level. However, the arguments must be objective to justify the indeterminacy (between particular kinds of formal systems). Hence we face the initial problem for which Curry proposed formal systems to solve it.

So Curry's definition of what a formal system is together with the addendum is too indefinite to support his claim of ontological irrelevance. This in turn undermines objectivity in the way he wants to achieve it, namely via provability in formal systems. However, we cannot simply leave out the addendum since it would make mathematics too narrow.[10]

[7] The emphases are adopted from the original.

[8] This argumentation is influenced by an analogous one in Penelope Maddy's [76, section 3.2].

[9] See [30] for further details.

[10] Otherwise Curry would have seen no need for such an addendum.

6.3 The three problems in light of the different degrees of formality

If we translate Curry's definition of mathematics as science of formal systems into our terminology this amounts to stating that mathematics is the science of systems of a high degree of formality and the high degree proofs therein. Even if we take Curry's later and broader characterisation of mathematics as science of formal methods (see [32, chapter 1C3]) this still implies that high degree systems and proofs are the benchmark. It should be clear from the first part of this thesis, however, that this is not the the point of view we are espousing. At least medium degree systems and medium degree proofs must be included; for – first – these have the benefit of being the best balanced ones regarding the purposes of argumentation and – second – even today these are the ones most commonly used in mathematics. So the most narrow characterisation (or definition) of mathematics we can think of is as the engagement with systems and argumentations that are at least of medium degree of formality. The most liberal (but still sensible) characterisation of mathematics is as the engagement with all systems (perhaps unconsciously) and argumentations that mathematicians expect to be translatable into ones of at least medium degree. The translations of the orginal argumentations are then proofs that have great convincing power or that provide objective guarantee.

Either of both characterisations resonates with the history of mathematics since the limited possibilities with respect to formality of pre-Fregean mathematics can be taken into account. If one or both of the characterisations will – for some reason – turn out to be improper this should not be due to differences between the past, present or future but due to a general misconception.

With regard to the second point of criticism we can without difficulties allow for other motivations than acceptability to engage oneself in a system: we can conceive of the system under consideration as a mathematical object and do our analysis from the perspective of another (accepted) meta system. In case one wants to find a contradicition, for instance, it is an argumentation beginning with "Let us assume the system under consideration as correct. Then ...". Curry does not have such a meta system at his disposal since it is not formal in his understanding of the word.

Finally, we feel not in need to establish the irrelevance of what conception of formal system is used. By contrast, we will incessantly investigate systems representing the same content that will turn out to be not equivalent; especially when the systems are of a different degree of formality.

The above "solutions" of the three problems of Curry's view might be unsatisfactory (in particular in the third case) since we went around the problems

through generalisation. In fact, the difficult part is not to show that our broader view is not susceptible to the three problems related to Curry's view – which are related to narrowness of Curry's view –, but that we do not create new problems along the way. In concreto we must show that what we have claimed to be the essence of Curry's view on mathematics, namely that objectivity and subjectivity can be harmonised to justify pluralism in mathematics, is preserved. Obviously, it is only the objectivity part that is at risk of being disrupted by our generalisations. Therefore it will be addressed in the next section.

6.4 Is objectivity lost?

It is clear that many of the systems and proofs we are considering cannot be counted directly as completely objective. So if we expected all systems and all proofs to have a high or even the highest standard regarding objectivity then the answer to the section's question would be simply yes. Yet, there does not seem to be any good reason for such a requirement. Instead it suffices if (most of) mathematics can be lifted on demand with respect to objectivity. We claim that this can be done by formalisation; independent from any concrete choice of what counts as high degree system or proof and independent from any equivalence between the different variants. Furthermore we will argue that the medium and low degree systems and proofs are not the fifth wheel regarding objectivity but that they can contribute to even more objectivity in a way higher degree systems cannot.

To argue for both claims we must first clarify in more detail than in chapter 5 what makes a process[11] – and therefore an argument in particular – objective. Obviously, we need independence from human actions or thoughts unless these are not part of the process itself. The same is true regarding a particular time or space. The process should be repeatable and always have the same, unambiguous result.

With this characterisation of objectivity in mind we can now start to show both claims. Let us begin with the second claim that lower degree systems and proofs can contribute to objectivity. In the case of Egyptian geometry, for instance, the degree of formality was rather low but the mathematics was objective nevertheless. The reasons were the reality related contents instead of the formalisms. An ascertained formula could be proved by reality – and therefore independent from other humans etc. – again and again. The result was always the same and unambiguous. Either the calculated resources – like building material for a pyramid – were

[11] We avoid the word 'experiment' since this connotes a too narrow point of view for our purposes.

correct or not. By contrast, topics not related to reality in an obvious way, like infinity for instance, cannot be discussed objectively at a lower degree of formality in this manner.

Nevertheless, even modern proof ideas of contents that are not (obviously) related to reality can contribute to objectivity. This is so because recurring (schemes of) proof ideas can prove themselves in mathematical practice. Their application might be often successful, leading to comparable or even the same results, and the translation of such ideas to more formal versions might have been established as unproblematic. Proof ideas themselves are ambiguous but their outcome need not be. They are not independent from humans who have to understand them but there is usually a large mathematical community that scrutinises them. So their acceptance does at least not depend on a single human.

The case is similar regarding (schemes of) single steps of medium degree proofs. Instances of these might occur in different proofs for very different theorems. Since they are less condensed and of a more technical nature the single steps can be compared nevertheless. So the number of repetitions and the independence from content, humans etc. increases. In addition to that, due to their technical nature single steps are less ambiguous. Finally, the objectivity of the single steps is carried over to the whole medium degree proof.

Another reason why medium degree proofs contribute to objectivity is their combination of rather contentual and rather formal aspects. So there is double backup for the case that one of them turns out to be problematic. However, this insight does not only pertain to medium degree proofs per se but – on a higher level – to all combinations of mainly independent argumentations; especially when they are of a different degree. The susceptibility to errors can be compared to the probability of simultaneous occurrence of independent events in stochastics. This is – in our opinion – the proper reason why translations from one kind of formal system into another – and lower degree meta argumentations that these translations are correct – are expedient. If one of the variants to define formal systems turns out to be inappropriate this would – if the translations are correct – spread to all variants; which is improbable.

To say that low and medium degree systems and proofs contribute to objectivity does not imply that high degree systems are superfluous. In our view as in Curry's they are the most important guarantors of objectivity (claim one above). This is true because something like syntactical modifications belong to this world.[12] Our experience with them is not limited to single events but everyone – in particular a computer – can use and track syntactical modifications everywhere and every time. Furthermore the process of proving in a high degree system is unambiguous.

[12] Perhaps this is also true for non-syntactical formalistic modifications but we do not need to discuss that here.

It is clear which strings are given ab initio and how one may create new strings from these. All we need for recognising or writing a single character is space, say consisting of pixels that are filled or not and whose content can be erased or (re)filled.[13] Of course it is possible to fill only half a pixel such that its status becomes unclear. But that does not matter, for both writer and reader strive for unambiguity. Note that these considerations about the contribution of high degree systems to objectivity are not dependend on any special kind of formalism (apart from the restriction to syntax, which might not be necessary).

It is this tremendous contribution of high degree systems and proofs to objectivity that allows us to consider any contents even if they are completely odd. We simply do not have to rely on the contentual contribution to objectivity since the formalism does this job already well enough. So the strictness of formal systems is like a bottleneck. After passing it one enjoys the total freedom lying outside the bottle. If, by contrast, one does not pass this bottleneck contentual restrictions subsist. One is trapped, so to speak, in the body of the bottle.

Let us sum up. In this chapter we have shown that Curry's view bears three essential problems, which vanish in our more pluralistic alternative permitting systems and argumentations of different degrees of formality. We had to ensure, however, that our approach does not spoil the objectivity part in the essence of Curry's view of mathematics as a side effect. We did not deny the outstanding contributions of high degree systems and proofs with respect to objectivity. Instead we argued that low and medium degree systems and argumentations do not stay in conflict with higher degree versions but make their own contribution to objectivity. So objectivity is not lost; it is won! From a bird's eye perspective this shows that our concept of degrees of formality can help to solve problems belonging to the domain of the philosophy of mathematics.

6.5 Related Work

There are two other points of criticism on Curry's view we are aware of but which we did not cover in the previous sections. Let us start with one that we deem unjustified. In [11] and [114, chapter 6] the formal systems of Curry are erroneously described as (essentially) syntactic, which in turn is assessed by many mathematicians as a poor view of mathematics. Whether this latter attitude is justified or not, it canot be an adequate criticism of Curry's view since for Curry a syntactical system is just a special variant of what he called a representation of a formal system (see [31, chapter VIII]). He even states explicitly: "I do not agree that the only things which can be treated formally are symbols." ([31, p. 45])

[13] It is not by coincidence that this sounds similar to Alan Turing's [110, section 1]. His machines must be objective and concrete parts of this world, too.

Another – justified – point of criticism regarding Curry is his terminology. Weir in [114], for instance, criticises the word 'token' since tokens – according to him at least – must be finite. Seldin complains about the change in Curry's terminology (see [100, 102]). In [32, subsection 2S2] even Curry himself acknowledges the difficulties arising from the change of terminology. What in [31] is called 'term', for instance, is called 'entity' in previous works and 'ob' in later works. Next, in the earlier work he uses both, 'abstract system' or 'abstract theory', to designate what is called 'formal system' in [31], while in [32] systems are particular theories. As a last example for the change of terminology, Seldin reports in [102] that Curry changed the word 'metatheory' to 'epitheory' after he was criticised for the former word by Kleene because Curry hated discussions about words. Regarding the criticism of terminology we want to add that in [31] Curry divides truth into two stages: first provability and second acceptability (see [31, p. 60]). Yet, we see no reason that provability has to come chronologically before acceptability (neither vice versa). It is a restriction that can be omitted without consequences. Indeed, in [30, p. 241] Curry does not mention the word 'stage' but solely uses the word 'split'. A last issue of terminology pertains to the word 'formalism' itself. In [100] and [102] Seldin surmises that the word 'formalism' might have only been chosen since Curry was a PhD student under Hilbert. Having said that, the terminology is only a minor problem not related to any essential aspects of Curry's view. Therefore we did not list it above.

The first point of criticism we mentioned, i. e. that all processes of mathematics standing outside the examination of high degree systems are not taken into account, is not new. The related historical objection is presented by Shapiro in [104, p. 170] but according to Seldin it dates back at least to 1966 where it was orally uttered at a colloquium in Hannover, in which Seldin participated (see [102, p. 5]). However, either this criticism is even older or Seldin's explanation that this kind of criticism caused Curry to change his definition of mathematics in 1963 to the science of formal methods (see [32, p. 14] for the definition and [100] as well as [101, chapter 4] for the explanation) does not make sense. In fact, in [101] Seldin also says that the criticism is much older but he does not name a concrete year there. In [114], after the erroneous classification of Curry's view as syntactical, Weir supplements that mathematicians in practice prove things about mathematical objects, not about something like strings. Leaving the string aspect aside this is our first point of criticism but without the historical dimension. Curry's 1939 characterisation of mathematics as presented in [31, 29, 30] simply does not fit the mathematical practice, neither in the past nor in the present.

For Seldin, the criticism discussed in the previous paragraph is only valid for Curry's view in 1939 but not for the later one as presented in [32, 33]. Seldin himself criticises other sources for considering only [31, 29] and treating them

as mature work (see [101, chapter 4] and [102]).[14] It seems, however, that even Curry's later views cannot solve the problem. Admittedly, Curry's later view is more liberal considering different degrees of formality (or formalisation in his terminology) but the lower degree variants are not on a par with the higher degree ones (see section 3.5). In practice, however, the medium degree systems and argumentations are preferred. In chapter 5 we gave reasons for this preference that do not rely on resources. Therefore Curry's comparison of high degree formalisation with the primary standard for measuring in physics as discussed in section 3.5 is misleading.

We are not aware of any mentioning of the two other problems presented here. We already discussed in section 3.5 that Curry's attitude towards acceptability does not change in [32]. This indicates that there really was no criticism compelling him to change his opinion in that respect. In section 3.5 we also mentioned that Curry formalised the metatheory. This suggests that he himself was not totally convinced of his argumentations concerning the irrelevance of ontology of formal systems. However, since the formalisation could have been done in another kind of formal system, too, the formalisation of metatheory only leads to an infinite regress.[15]

[14] This criticism is conretely directed at Shapiro's [104]. The other sources we discussed, namely Bedürftig's and Murawski's [11] and Weir's [114], do not even cite [29] but only [31].

[15] This is the opposite stance to Shapiro's in [104, section 6.5], where he presumes the regress not to be vicious. However, Shapiro does not justify this assessment.

Chapter 7

Learning how to prove: from the Coq proof assistant to textbook style

7.1 Introduction

Most computer science students have difficulties with proving theorems. Since many of their solutions avoid formalisms or apply them in a wrong way (see [66]) it seems obvious that the formalisms are at the heart of the problem. In concreto we suspect the blending of formal and more informal aspects, which is typical for the textbook proofs the students are asked to develop, to be the main obstacle to learning how to prove. The formal aspects make a precise argumentation necessary while the informal ones are hiding this precision.

In accordance with the Cognitive Apprenticeship approach [23], which we already used in another course (see [68]), we try to render the strategies for precise argumentation visible. This can be realised most effectively in the case of high degree proofs. Due to their lack of natural language the suggestive character decreases and the students can concentrate on the true aspects of the proof. The high granularity as well as the high frequency and precision of justifications make the single steps replicable and the requirements for a correct step become clearer.

Apart from transparency, focussing on high degree proving has another – perhaps even more important – advantage: thanks to the symbiosis of high degree proof style and computers we can use Coq to provide the students with a clear and immediate feedback for every step they attempt.[1] So strategies will be not only visible but even evaluable all the time. This leads to almost perfect learning conditions. By contrast, the only individual feedback students would normally get

[1] Proof assistants like Nuprl or classical Isabelle share this advantage.

stems from the correction of their homework assignments. Such kind of corrections, however, can only evaluate the product, not the process. Early mistakes in the proof, then, render the remaining part moot. So there is no way to overestimate the benefits, which will arise due to the use of Coq in this phase of learning.

So far this militates in favour of teaching high degree proofs instead of the conventional teaching of textbook proofs. Nevertheless this would be no solution. Textbook proofs as an instance of medium degree proofs are not only very different from high degree ones but the former also have benefits that the latter are not sharing (see chapter 5). Therefore our overall approach is to start with teaching students how to prove at a high level of formality first and then to transfer their achieved proficiencies (and not just the proofs) to the textbook style. It is especially this latter step, in which our considerations of part I of this thesis will turn out to be very useful.

Before discussing our first experiences with our approach let us briefly mention that a declarative proof assistant would be no good alternative for the first step although working with it would be closer to creating typical textbook proofs; the reason being that students need to learn the single steps of the process of proving before creating proof products. This requires the clear (and immediate) feedback mentioned before. Declarative proof assistants like Isabelle/Isar, however, do not provide a clear, intensional feedback why some proof step is possible or wrong. They rather evaluate a declarative step as an idea that (already) works or not. This is a useful feedback for all those already mastering the technical aspects of proving but not for the students we are addressing. That is why we stick with Coq in the first step.

We tested the approach of the two steps – with the focus being mainly on the first one – the first time in a course, held in October 2016 at Universität Hamburg. Employing logic and inductive data types and in accordance with the previous paragraphs we had great success regarding the first step.[2] With respect to the second step of transferring back to normal textbook proofs we had a rather naive approach in that course: we gave a lot of assignments calling on the students to create a textbook proof out of a proof in Coq and vice versa. The results here were not nearly as convincing.[3]

So let us now discuss in which way the treatment of the second step might be improved. We think there are at least two appropriate ways to deal with the difficulties the students had with it[4]:

[2] The exam results are discussed in [15, section 2] while a more comprehensive evaluation of the course analysing also the amount of questions, working habits etc. of the students can be found in [67].

[3] See again [15, section 2] for further details.

[4] In [15], the relation of which to this chapter will be discussed below, we listed the focus on Coq proofs as a third alternative. With the background knowledge of the benefits of the different

1. Change Coq proofs to be (more) like normal textbook proofs.

2. Find some new way to bridge the gap between the ability to prove in Coq and in textbook style.

The idea of the first approach is to create *proof environments* for the different domains of mathematics, which allow students to prove in Coq in a way similar to the textbook proof style.[5] In principle such environments can be developed in Coq by an excessive use of medium degree tactics (and suitable libraries).[6] In the next chapter we will present such medium degree tactics for the domain of arithmetic.

In this chapter we will concentrate on a realisation of the second approach, which we already tested in a course in September 2017 at the University of Potsdam. The key idea for dealing with the second step (the bridging) is to reduce the degree of formality stepwise from Coq to textbook style. We will describe this approach in more detail in the next section. In sections 7.3 - 7.5 we will discuss the intermediate proof styles used for the stepwise reduction of the degree of formality. We call these line by line comments, weakened line by line comments, as well as structure faithful proofs. The subsequent section will be about relating the intermediate steps and the textbook proofs in teaching. Section 7.7 treats related work and section 7.8 provides summary and conclusion for this chapter.

There is already a separate article, namely [15], dealing with the same topic. This chapter is essentially a shortened version of it concentrating mainly on the conceptional parts.[7] The reader who is interested in the empirical data and the concrete procedure is referred to [15, sections 2,8] and [67]. Furthermore, for the first course there exists an open access video series of the lectures in German (see [14]).

---

degrees it is clear that this cannot be a solution. In the article, however, we argued against the focus on (conventional) Coq proofs by pointing to the difference between proving in a prepared file and the development of new projects.

[5] This does not necessarily include the use of natural language, which is not the most important point of the intended resemblance.

[6] For a real proof environment further minor aspects are relevant, which we do not want to discuss here.

[7] Other changes are due to the preparatory work of the previous chapters.

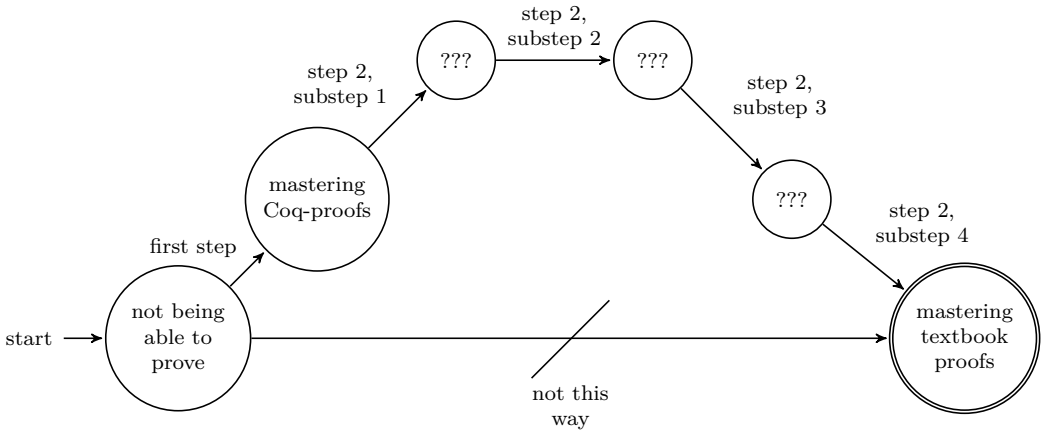

Figure 7.1: Second step divided into many approximately equidistant substeps.

7.2 The idea: stepwise reduction of the degree of formality

The basic idea to improve our teaching regarding the second step, i. e. the transferring to textbook proofs, is rather simple. We can compare this step with a stream students have to cross. Some of them can jump across very well and are able to do so without further assistance. However, most of them will need help. Let us say there is a big flat stone in the middle. This would make things much easier. By contrast, if there would be a stone only at the beginning or at the end this would not be of much help. So we have to find helping stones dividing the whole step into approximately equidistant substeps (figure 7.1). Such a scaffolding (and its later deinstallation) is in line with the Cognitive Apprenticeship approach (see [23, 68]).

What should the flat stones in the middle of the stream look like in the context of proving theorems? Our general approach is to teach students textbook proofs by teaching them proofs of a very high degree of formality first. So the idea to install platforms – which then are particular proof styles – that are intermediate in their degrees of formality is not far fetched. Yet, the concrete choice is a little bit fiddly. The proofs one can develop with proof assistants like Isabelle/Isar or Mizar might be considered as candidates. From the discussion in section 8.10 it will follow, however, that they are not suited. Ralph J. Backs pen and paper approach of structured derivations (see [6]) is product oriented and therefore does not fit our process oriented approach. Therefore we had to invent new suitable proof styles. Fortunately, we already detected three adjusting parameters for the degree of formality in section 2.3 we can orient ourselves on.

To begin with, we observed that students do not only face major problems in transferring from the formal language to natural language (and vice versa) but that at least some of them are even reluctant to use natural language. It must be made clear to all students that natural language is no unnecessary, imprecise adjunct. Therefore intermediate platforms should stress that natural language can be used in a very precise manner and that it has the power to structure arguments.

In the beginning of the second step students have been taught only to develop Coq proofs, in which every single modification of the proof situation is announced by an explicit justification. Since in textbook proofs the single steps are of a much coarser nature the intermediate platforms should increase the condensation cautiously. One way to do so is to eliminate all repetitive aspects first. Only when students have gotten used to this easy form of condensation they should be confronted with forms of condensation, in which information is really lost. As we will see, by increasing the degree of condensation we will be able to decrease the number of justifications and the precision of the remaining ones in a straightforward manner.

Two last points we want to mention are readability and feedback. In section 2.6 we already noted that proof scripts alone(!) are not readable at all. By contrast, textbook proofs use natural language to tell a mathematical story and are thus designed to be read (compare with section 5.3). So we should check for each platform whether the readability is increased. Next, – as discussed in the introduction – when working in Coq the students get a feedback for every single step they attempt. There is no feedback in this sense in the case of textbook proofs. Still, there are some possibilities regarding self-monitoring but these are hard to deal with. So it is desirable to create intermediate platforms, in which self-monitoring is rather straightforward to conduct. This way the feedback can be reduced stepwise together with the degree of formality.

7.3 A small step for mathematicians but a big one for learners: line by line comments

The first intermediate platform we introduce is *line by line commentation*. By this we mean adding comments to every line of an already existing Coq proof. These comments have to use natural language to describe precisely the modifications made in the proof situation at this step. This is illustrated in figure 7.2. To support this tasks students are provided with formulations for all the logical rules and the remaining tactics that can appear.

Please note that regarding language the sole focus is on precision. Hence (for now) we do not request the students to vary their formulations. Even more we

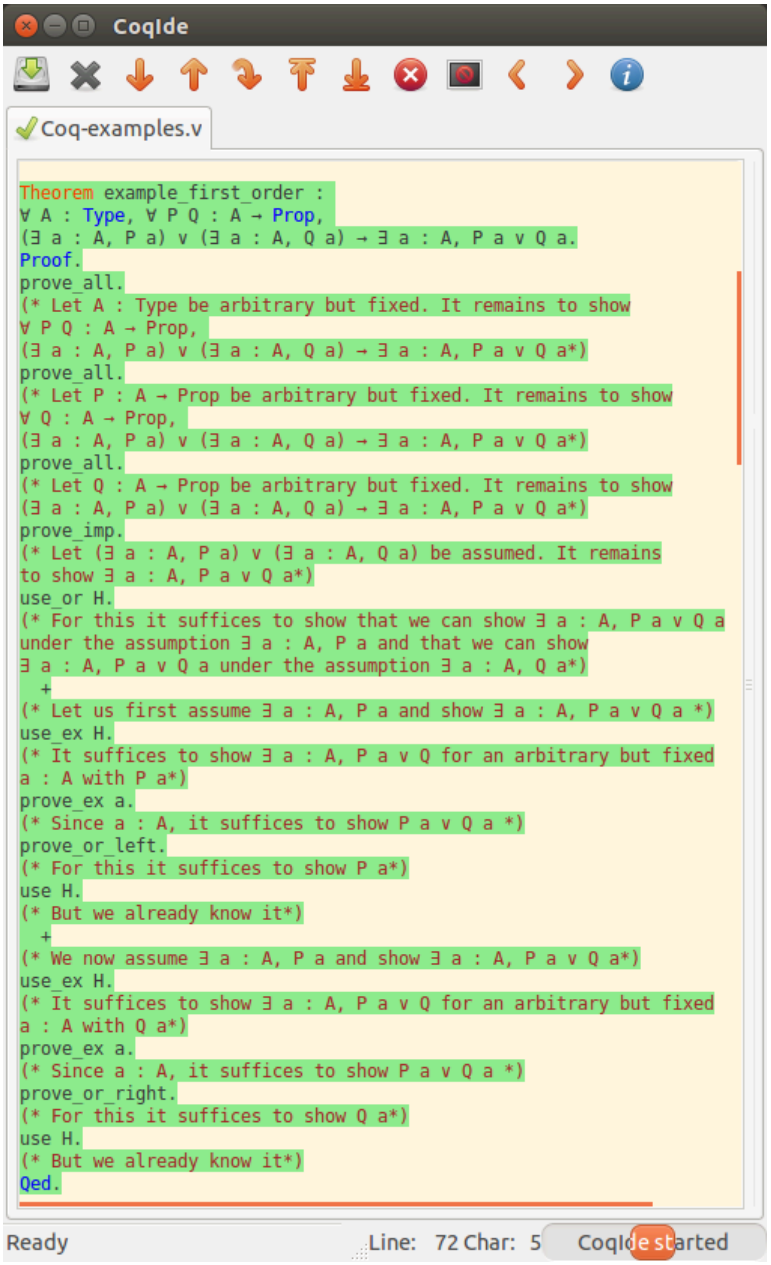

Figure 7.2: Proof exemplifying the line by line comments.

do not expect the students to use language in a structure giving way or an ideas mediating manner. Furthermore, at this stage we do not require any form of condensation, whatsoever. The logical justifications are implicit in the written assumptions and in the formulations explaining what remains to be shown. Theorem applications – if there are any – have still to be done in a rigorous way, instantiating one parameter at a time.

Another relevant point is that students need some time to write down comments. Since the writing process itself does not exhaust their mental resources, students will also reflect on what they are doing during their process. Furthermore they have to make explicit the bookkeeping that is usually done automatically by Coq. Hence they do not only have to handle every new situation for itself anymore, but they have to view their proof holistically. So every step becomes part of a whole and the different proof step situations become linked together. Hence we do foster the thinking about the Coq proof at hand and therefore line by line commentation is also a tentative step towards a bird's eye perspective. This perspective in turn should help the students to develop more readable proofs later on.

Obviously, we do not have any kind of automatic feedback for the comments. However, due to the given recommendations for formulations and the line by line comments of other proofs students can create their own feedback. By contrast, when proving a theorem in a textbook manner there are no such clear reference points for self-monitoring.

To sum up, we have created a Coq dependent proof style, which is between the Coq proof itself and textbook variants of it. Since it is the first of three intermediate platforms this style is still closer to the former than to the latter. Yet, this level already involves language and some transition to a bird's eye perspective. Furthermore the feedback is already weakened. That weakening can indeed be a step forward will be the subject of the next section.

7.4 Weakened line by line comments

By a *weakened line by line commentation* we mean a line by line commentation with a few important differences:

1. There are some situations where two or more steps of the same kind have to be condensed into a single step. For example the first four steps in figure 7.2 must be condensed to "Let A be a type, P and Q be propositional functions over A, and let furthermore $(\exists\ a : A,\ P\ a)\ \vee\ (\exists\ a : A,\ Q\ a)$ be assumed. We have to show $\exists\ a : A,\ P\ a\ \vee\ Q\ a$".

2. There is some textual smoothing required. The previous point already contains the formulations "be a type" and "propositional function over A". Furthermore from this level on many textual repetitions should be avoided.[8]

3. Trivial endings like "but we already know it", "the same are always equal", or "ex falso quodlibet" etc. are not commented anymore. Instead the comments for these are subsumed in the respective comments before (if there are any).

4. The comments have to be oriented at an intuitive understanding instead of the logical rules. In figure 7.2, for instance, we simplify the comment after "use_or" by "We make a case analysis over $(\exists\ a : A,\ P\ a) \vee (\exists\ a : A,\ Q\ a)$".

5. Parameters that can be made implicit must not be mentioned. Given the theorem `lequ_antisym` stating that $\forall\ n\ m : \mathbb{N},\ n \leq m \to m \leq n \to n = m$ and given two hypotheses $H : n \leq m$ and $H' : m \leq n$, for instance, the student should comment "Using theorem lequ_antisym applied to H and H' we obtain $n = m$" but not "Using theorem lequ_antisym applied to the natural numbers n and m as well as to the two hypotheses H and H' we obtain $n = m$".[9] [10]

6. In case of analogous branches in the proof only the first of them is permitted to be commented. The following ones have to be declared as analogous. Furthermore it has to be explained precisely what changes would have to be made if we had commented everything. In contrast to the proof in figure 7.2, for instance, a weakened line by line commentation would only write one comment for the second branch: "This case is analogous to the first one. $P\ a$ has to be replaced by $Q\ a$ everywhere except for the disjunction $P\ a \vee Q\ a$ and we have to prove the right side of the disjunction instead of the left one".

The use of language becomes more difficult at this stage. First of all the formulations must have a richer variety now. This in turn eliminates the possibility of a complete listing of all needed formulations, which was given to the students in the case of line by line commentation before. So students cannot simply replicate the suggested formulations from such a given list but have to modify the former formulations. In case of analogous branches, students even have to find formulations completely on their own. Finally, some of the formulations must condense

[8] In the course in Potsdam we did not require to avoid repetitions but it seems to be a suitable point.

[9] In the actual course this point was not taken into consideration.

[10] The direct application of multiple arguments for functions is the same as in the case of the universal quantifier; for both are functions from a type theoretical point of view.

several lines. So language starts to determine the structure of the proof. Yet, this is just the beginning and most parts are still given by the Coq proof. Furthermore language is still not responsible for the communication of ideas.

Despite the lack of ideas contained in them the weakened line by line comments become more readable. This is due to the more intuitive formulations and the omission of trivial endings. Furthermore the labelling of branches as analogous saves resources.

The latter point deserves further attention. When writing (or reading) line by line comments students should become frustrated by all the annoying repetitions. They should wish to condense these and this happens at this stage for the first time. This pertains to the analogous branches as well as to lines of the same kind. Yet, both kinds of condensation preserve information. They can be reverted and are therefore innocuous.

The omission of implicit parameters is a tentative step towards a more restrained handling of justifications. It fits the idea of avoiding repetitions at this stage and can therefore again be seen as the little brother of our efforts regarding condensation.

With respect to feedback comparison with canned formulations will not always be possible anymore. Instead students have to orient themselves on other weakened line by line comments or their former solutions for line by line comments. Although this does not sound like much it is still much more than in the case of textbook proofs. In the former "only" narrow deviations from a given standard have to be found.

The weakening of line by line comments has brought us closer to textbook proofs. While the main focus in the transition to line by line comments is on language the focus in the transition to weakened line by line comments is on condensation. Although we already made some tentative improvements regarding readability there is still a big gap between weakened line by line comments and textbook proofs. With the next intermediate platform we will reduce it.

7.5 Structure faithful proofs

The last of the three intermediate platforms is given by what we call *structure faithful proofs*. These are proofs in natural language that have the same structure as the corresponding Coq proofs. Besides the usual backward reasoning structure faithful proofs allow forward reasoning using equations and implications. Except for argumentations based on reflexivity, symmetry, and transitivity of equality or other equivalence relations every single step has to be mentioned to some extent. For every step a justification has to be stated but this should be a short hint

only. For instance, theorems have to be mentioned but not the arguments with which they are instantiated. Furthermore there is an important consequence of this definition: since structure faithful proofs do only depend on the structure of Coq proofs but not on the lines of code, they can be developed without having to write down a Coq proof.

Let us consider an example of a structure faithful proof (figure 7.3). What comes to our attention immediately is the explicit structure given by the symbols '+', '*', and '-'. Apart from that the proof looks similar to a textbook proof. Most of the justifications in this proof can be found on top of the equality signs. That justifications usually are only hints in this kind of proof can be seen, for example, in $Suc\ n \ominus Suc\ l \overset{\{suc\_n\_sub\_suc\_m\}}{=} n \ominus l$. Here, we express that the theorem `suc_n_sub_suc_m` is used but we do not state explicitly that n from the theorem is instantiated with n and m with l. This does not only pertain to equational reasoning: "[...] we have $n \ominus l = 0$ and therefore by equ_fct, $pred\ (n \ominus l) = pred\ 0$" is another example for this less precise kind of argumentation.

The reader should now have an impression of what a structure faithful proof looks like. So let us now start to analyse this kind of proof. Regarding language we are close to the goal now. In this respect the only remaining difference to textbook proofs is that language is not used to structure the proof per se. In particular we are still not able to give more weight to some of the branches. However, within the subproofs language already is used freely and therefore structures the argument. So the use of language is still a big hurdle for students.

The equational reasoning is a new way to condense a lot of steps. Instead of using different equalities step by step to modify hypotheses or the actual (sub)goal now different equations are put together while a justification is put above each equality sign. So far nothing spoils restorability. Yet, the justifications themselves are only hints now and so some information is really lost.

What sounds like a drawback is actually a feature since omitting details increases readability. The addition of forward elements helps to create a guiding thread, which again increases readability. Finally, the same is true for the flexible use of language. For instance, we can accentuate focussing on the essential parts of a subproof.

There is only a single rather external kind of feedback left. This is the structure the students can orient themselves on. They can see explicitly where they are in their proof and therefore it is easier to keep an overview of what is given at the moment and what is to do next. In addition to that in some parts of the proof students can orient themselves on the formulations used in (weakened) line by line comments. Yet, in most situations there is no unique or at least a standard formulation given to orient oneself on.

Theorem: $\forall\ n\ m : \mathbb{N},\ n \ominus m \neq 0 \vee n = m \rightarrow Suc\ n \ominus m = Suc\ (n \ominus m)$.
Proof: Let n, m be arbitrary but fixed natural numbers and let us assume $n \ominus m \neq 0 \vee n = m$. We have to prove $Suc\ n \ominus m = Suc\ (n \ominus m)$. We do this by case analysis of the disjunction.

+ We assume $n \ominus m \neq 0$ and show $Suc\ n \ominus m = Suc\ (n \ominus m)$. By the structure theorem of $\mathbb{N}$ we know $m = 0 \vee (\exists\ l : \mathbb{N},\ m = Suc\ l)$. So we can prove our goal by another case analysis.

  * We assume $m = 0$. Then we do have $Suc\ n \ominus m \overset{\{m\ =\ 0\}}{=} Suc\ n \ominus 0 \overset{\{n\_sub\_0\}}{=} Suc\ n$ on the left hand side and $Suc\ (n \ominus m) \overset{\{m\ =\ 0\}}{=} Suc\ (n \ominus 0) \overset{\{n\_sub\_0\}}{=} Suc\ n$ on the right hand side. So we have the same on both sides.
  * We assume $m = Suc\ l$ for some $l : \mathbb{N}$. This delivers $Suc\ n \ominus m \overset{\{m\ =\ Suc\ l\}}{=} Suc\ n \ominus Suc\ l \overset{\{suc\_n\_sub\_suc\_m\}}{=} n \ominus l$ on the left hand side. On the right hand side we have $Suc\ (n \ominus m) = Suc\ (n \ominus Suc\ l) \overset{\{n\_sub\_suc\_m\}}{=} Suc\ (pred\ (n \ominus l))$.
    - If we could prove $n \ominus l \neq 0$, we would have furthermore $Suc\ (pred\ (n \ominus l)) \overset{\{suc\_pred\_n\}}{=} n \ominus l$, such that left and right hand side would be equal.
    - So it remains to show that $n \ominus l \neq 0$. For this we assume $n \ominus l = 0$ and derive a contradiction. On the one hand we do have $0 \neq n \ominus m \overset{\{m=Suc\ l\}}{=} n \ominus Suc\ l \overset{\{n\_sub\_suc\_m\}}{=} pred\ (n \ominus l)$. On the other hand we have $n \ominus l = 0$ and therefore by equ_fct, $pred\ (n \ominus l) = pred\ 0 \overset{\{pred\_0\}}{=} 0$, the required contradiction.

+ We assume $n = m$ and show $Suc\ n \ominus m = Suc\ (n \ominus m)$. On the left hand side we have $Suc\ n \ominus m \overset{\{n=m\}}{=} Suc\ n \ominus n \overset{\{I\_add\_n\}}{=} (1 \oplus n) \ominus n \overset{\{\text{add\_comm}\}}{=} (n \oplus 1) \ominus n \overset{\{n\_add\_m\_sub\_n\}}{=} 1$. The right hand side delivers $Suc\ (n \ominus m) \overset{\{n=m\}}{=} Suc\ (n \ominus n) \overset{\{n\_sub\_n\}}{=} Suc\ 0 \overset{\{by\ Def\}}{=} 1$.
  q.e.d.

Figure 7.3: Example of a structure faithful proof.

We have now seen an intermediate platform with a considerable improvement in readability. Except for the explicit structure, with its forward elements and its reduced demand for precision, it reminds one already of textbook proofs. However, the explicit structure is still a big support since the requirement to prioritise different branches to make the proof more linear is not easy to fulfil.

In our comparison with the crossing of a river the next step leads to firm ground, namely the textbook proofs. They are what we intended to teach the students. Let it be mentioned, however, that in principle we do not have to stop at textbook proofs since they are not the least formal kind of argumentation. We can conceive them as only one further platform in the stream, the other end of which could be the proof ideas. In this case we would propose proofgumentations (see section 2.5) as a further intermediate platform between textbook proofs and proof ideas.

7.6 How to teach it?

Until now we have presented the different platforms (line by line comments, weakened line by line comments, and structure faithful proofs). In this section we will focus on the steps between these platforms; i.e. we try to answer how students can master some stage when the previous ones have already been learned.

In general we propose to spend most of the time with supervised training sessions where the students work on their own. Though autonomous working on assignments is definitely not a new idea it is of utmost importance in this case: the focus is not on contents but on methods and these can be learned best by applying them. The role of the lecturer should be limited to a brief introduction of the platform per se, including the presentation of solutions to some examples before the students start to work on their own. The lecturer should moderate the discussion of student's solutions only where appropriate.

So what should assignments for students look like? Of course, they should mirror the transitions. Therefore a lot of assignments require the students to take a solution of stage n and to transfer it to stage $n+1$. For a better grasp of the relation between two such stages students are asked to transfer a few solutions of stage $n+1$ back to stage n.

Yet, in the end students are not supposed to start with a Coq proof and transfer it step by step to textbook proofs but they should be able to develop the latter directly. So first, distance must be broadened, i. e. we recommend also assignments that require the transferring from stage n to stage m where neither n nor m is the immediate successor of the other. Naturally, the farther the treatment of an intermediate platform m is the farther the distance between n and m can be. In particular the ending point might be a textbook proof. Second, students must get

rid of the Coq crutch entirely. So we need assignments that do not presuppose any solution of a previous stage but instead ask the students to develop a proof of the respective stage from scratch.

However, in its character these assignments differ from the other ones since there is no given structure the students can orient themselves on. Therefore these kinds of exercises should be divided further gradually: the first assignments of this kind should be to prove theorems which are clear with respect to structure. For instance, theorems using the pumping lemma are good candidates here, whereas the latter assignments should be allowed to conceal this structure, like most theorems in fact do. Proving that a set is enumerable, for instance, belongs to that kind of assignment. In such cases students must be trained to find the structure (including the relevant types) to apply their transition skills from there on.

7.7 Discussion of related work

In visual appearance our different proof styles, including the Coq proofs, resemble the structured derivations presented by Back in [6]. For instance, we find indentations and justifications similar to ours. Particularly strong is the resemblance in the case of structure faithful proofs since structured derivations make use of equational reasoning, too, and the same kind of justifications (like transitivity) are left implicit. With us Back's approach shares the idea of transparency on the teaching and the learning side. However, there is no process of different proof styles in Back's case. Furthermore – as already mentioned – the main difference to our approach is that the whole idea of structured derivations is product-oriented, while we focus on the proof process.

There is a variety of approaches using proof assistants for teaching. Several discuss the poor performances of students in proving [58, 85, 93, 97]. Tobias Nipkow calls the student's creations in [85] LSD trip proofs. H. James Hoover and Piotr Rudnicki [58] stress that poor proofs can be found even in textbooks. We only want to add that the inability to create solutions at all might be the biggest problem.

Many authors agree that feedback is of utmost importance [20, 37, 58, 73, 84, 85, 93, 97, 99, 109, 113]. The most traditional view is that feedback assures us what (parts of) proofs are correct [73, 84, 93, 109]. Nipkow [85] stresses gamification aspects, namely that feedback is addictive and serves as a challenge that causes students to work a lot harder. Like us, he criticises that conventional homework corrections have too low a frequency. Wolfgang Schreiner, Alexander Brunhuemer, and Christoph Fürst [99] argue that feedback allows the students to try out different approaches ludically. Nathan C. Carter and Kenneth G. Monks [20] require a frequent, immediate, and clear feedback in the learning process. We would like to

point out that the last attribute cannot be accomplished in full if proof assistants employ an automatic justification mechanism, as this may permit steps too big or reject steps that should be fine but cannot be handled by the system. This point is conceded in articles on Mizar [58, 109]. On the other hand such proof assistants can sometimes be used to tell students whether their actual approach does work. The Sequent Calculus Trainer of Arno Ehle, Norbert Hundeshagen, and Martin Lange, for instance, uses auto tactics to generate feedback telling the student whether the actual (sub)goal can be proven, is invalid, or whether the system does not know yet (see [37]). Similarly, the Natural Deduction Assistant (NaDeA) of Jørgen Villadsen, Andreas H. Form, and Anders Schlichtkrull issues a warning when no proof can be found by auto tactics (see [113]).[11] What is completely new in our approach is the idea to reduce the amount of feedback during the course, which should cause students to mature and become independent of feedback eventually.

Many articles attach great importance to using a tool that does not require students to learn too much of the system before teaching the true contents of the course. While Nipkow [85] stresses that only a quarter of the time of a course was devoted to teaching Isabelle/HOL, all Mizar articles ([58, 109, 93]) concede that learning how to write Mizar proofs is challenging and requires much effort in courses. Others modify a standard system to avoid problems with the system itself. The web server ProofWeb [53] was developed to avoid installation issues and problems with different versions of Coq. The Papuq system [97] is a modification of Coq that doesn't require students to learn the original Coq tactics. Instead in every proof situation it shows a window with natural language suggestions how to proceed. Still others develop whole new systems designed to fit the didactical needs. Graham Leach-Krouse, for instance, presents the Carnap framework to bridge to real proof assistants [73]. It can be restricted to a few functionalities avoiding to overwhelm students in the beginning. Further functionalities can then be added later on. Font, Richard, and Gagnon, whose article [43] we already have started to discuss in section 2.7, present a tutor system called QED-Tutrix for the high school level. It provides three tabs corresponding to three aspects they emphasize in the process of finding a proof, namely exploration, construction, and redaction; where the last amounts to filling in the holes of a whole formal proof fitting the separate construction of the student. Most consequent are Carter and Monks in [20]. They introduce a system called Lurch that resembles a word processor and is easy to use even for non computer affine students. Further examples of new systems designed to fit the respective didactical needs can be found in [37, 99, 113]. In contrast to all

[11] In the case of wrong steps approaches like the last two can be enriched by a listing of countermodels, i. e. by a clear, extensional feedback.

those approaches, we did not modify the Coq proof assistant itself but only added tactics. We wanted and still intend to concentrate on the conceptional aspects of the course and for that the original CoqIDE is sufficient (see [67]).

Many approaches using proof assistants, like [37, 53, 58, 73, 113] for instance, are restricted to logics. Sakowicz and Chrząszcz [97] treat a simplified version of set theory in their course. Algebra and number theory is taught with Lurch on top of logic and set theory in [20]. Trybulec and Rudnicki [109] concentrate on relations while Nipkow [85] uses the Isabelle/HOL to teach semantics. We devoted the non logical part to data structures but could imagine treating other parts, especially relations, as well in future courses.

In recent times using proof script based proof assistants for teaching has fallen into disrepute. Instead it is endorsed to use declarative proof assistants (see section 2.6). Nipkow [85, section 3.3], for instance, criticises that proof script based proof assistants are not readable by humans, lack structure, do not make ideas visible, and – most importantly – have a relation to real proofs that can be compared to the relation between assemblers and normal programming languages. While the first point is true without additional bookkeeping or when auto tactics are used (see section 2.6), the second and third depend mostly on the person developing the proof, and the comparison in the fourth is flawed. The implicit argument here is that one does not have to learn assemblers first to learn programming and in the same way one does not have to learn proving in a proof script based proof assistant first. But procedural thinking is firmly fixed in human beings while static truths are not. Hence more students will have an intuitive grasp of programming than of proving. Thus students have to learn the basic principles of proving first such that they become second nature to them. Having said that, however, we do consider the use of declarative proof assistants as promising whenever the students have already learned how to prove.

Hoover and Rudnicki in [58] stress a method called structuring of proofs, which is conducted with the help of the Mizar-MSE proof checker. By structuring they mean the introduction of indented blocks having assumptions at the beginning, some steps in between, and a result at the end. Blocks are evaluated as a whole in the ongoing proof. An implication, for instance, is shown by such a block where the assumption is the premise and the result is the conclusion of the block. This is related to our structuring in Coq and the intermediate steps but comprises many more cases. All the examples given in [58, section A.5], for instance, do not require any structure in our case. This is because our approach is (initially) analytic while theirs is synthetic.

What is missing in all cited articles – except for Carter and Monks' [20], where students develop proofs with Latex in the end – is a treatment of the relation between the kind of proofs done in a proof assistant and the more informal textbook

proofs.[12] One reason is that the focus is often on machine readable proofs and not on transferring these to textbook proofs. In our opinion the difficulty and the value of transferring the skill of proving in a proof assistant to the skill of textbook proving is extremely underestimated.

7.8 Summary and conclusion

Most computer science students have difficulties with proving theorems. We have developed an approach in which students learn to prove in Coq first and start to develop textbook proofs only after that. In this chapter we focussed on teaching the transition to textbook proofs. The main idea was to find intermediate proof styles that reduce the degree of formality step by step. To do so we made use of the three adjusting parameters: language, condensation, and the frequency and precision of justifications. In addition, for each platform we checked whether the readability of the respective proofs increases and analysed the feedback possibilities.

We started with a concrete instance of high degree proofs, namely proofs in Coq, and defined new proof styles that depended on this. In concreto, we presented line by line comments, their weakened version, and structure faithful proofs. The first was mainly characterised by a leap in language while the second one was mainly characterised by a leap in condensation. In the transition to structure faithful proofs all three adjusting parameters played a major role leading to a big leap in readability. We also described types of assignments that can lead to success in textbook proving.

The results of our empirical investigations, as discussed in [67] and [15, sections 2,8], are quite promising. Students liked to work in Coq and got on well with the system, technically and contentually. Furthermore the data show that students do well with the intermediate platforms and the corresponding assignments. The results regarding the final, decisive transition to textbook proofs indicate a clear progress but not a panacea for all the problems related to the development of textbook proofs. The data suggest that one further improvement for the future could be to stress the formalisation of theorems even more in order to give the students a better understanding of the relation between formal and informal formulations of theorems. Regarding that aspect the approach presented by Patrick Jansson, Sólrún H. Einarsdóttir, and Cezar Ionescu in [63] might be particularly helpful.

From a bird's eye perspective our data indicate that – at least for computer science students – a high degree of formality is not the problem in learning textbook proofs (but rather the mix of formal and less formal elements) and that it is a good approach to start proving with a suitable proof assistant even if one only wants

[12] This has not to be confused with approaches focussing on automatic translations from machine proofs to textbook proofs.

to teach textbook proofs. We expect this to hold for students of mathematics as well. For a substantiation of this claim we aim at extending our empirical base to gain reliable data in the future.

In the introduction of this chapter we already discussed a different attempt to deal with the second step. The key idea is to "change" Coq via so called proof environments in a way that allows Coq proofs to be much closer to textbook proofs. In the next chapter we will discuss the core of such proof environments, namely the medium degree tactics.

Chapter 8

Simulating proofs of a medium degree of formality

8.1 Introduction

In chapter 5 we ascertained that an argumentation serves three purposes: to illustrate ideas, to convince writer and reader, and to deliver objective guarantee. We argued that low degree argumentations are best regarding the first, medium degree ones are best regarding the second, and high degree ones are best regarding the third point. Furthermore we found that medium degree proofs are the best allrounders. Still, as long as only humans are involved, it is clear that we have to focus on one of the three aspects to the detriment of the other ones. Instead it would be highly desirable to combine the advantages of different styles of argumentation to reach more than just one of the three aspects above in full. In this chapter we will show – or at least substantially indicate – that it is indeed possible to combine the advantages of medium and high degree proofs by simulating medium degree proofs with the help of the computer.

Before we consider the simulation itself let us briefly discuss what will be simulated. It is clear that this cannot be all proofs of a medium degree of formality in mathematics and so we have to focus on a special domain. Fortunately, we are not committed to any specific content and thus free to choose whatever serves best as proof of concept. When choosing a domain there are two extremes to avoid. On the one hand in logic even textbook proofs usually tend to be very formal and in that case there is no real difference between medium and high degree proofs. So (pure) logic is not suited for our kind of investigation. By contrast, most other domains are too complex. Their single reasoning steps in medium degree proofs already comprise whole bunches of steps of simpler domains or even whole algo-

rithms, changing the proof situation significantly.[1] To avoid both extremes we chose arithmetic as a suitable level to start with. Here, the difference between high and medium degree proofs is big but not too big. Furthermore other domains often are related to arithmetic anyway.

Apart from the domain of mathematics the single reasoning steps of a medium degree therein are relevant. To find precisely the correct or usual reasoning steps of – in our case – arithmetic would require a lot of empirical work.[2] Yet, this is not an empirical thesis. Instead our approach in this chapter should be understood as a proof of concept: whatever the real human reasoning of a medium degree of formality might be the reasoning we simulate should be close enough to see that it is possible to simulate the actual reasoning, too. The differences in both kinds of reasoning should be transferable to the simulation by changing the concrete implementation a little bit.

Now, what does our approach for simulation look like? The main idea is to use user defined tactics in Coq that represent the single reasoning steps of medium degree in arithmetic (or what we deem as such). On the one hand such tactics are beyond the scope of the basic tactic rules Coq is initially providing. On the other hand our self written tactics – in contrast to Coq's auto tactics – have still a precise specification of their behaviour, i. e. they still feel like an one step tactic. This distinguishes our approach from others (see section 8.11).

To really get at a proof of concept it does not suffice to consider just a single tactic. Instead our selection must be sufficiently broad to at least indicate that all of arithmetic can indeed be covered. In this thesis we will therefore discuss five tactics, namely `drop_identities`, `suc_pred_to_front`, `omit_parens`, `make_first`, and `drop` (sections 8.2 - 8.6), which together comprise a major part of the proof steps in the implementation of this approach (see [13]). All of the five tactics will be analysed in the following sequence of steps:

1. Presentation of the corresponding reasoning in medium degree proofs
2. Some illustration of how the respective tactic is used in Coq (contrasted with a proof without it)
3. Exposition of the implementation
4. Evaluation.

Except for `drop` all of these tactics are value preserving, i. e. they replace terms only with terms of equal value. For such tactics the implementation will always be

[1] In section 8.9 we will explain the latter problem in more detail.

[2] Such an investigation can orient itself, for instance, on Schiller's thesis [98], which we have discussed in section 2.7.

the same from a particular point on. The idea of that part of the implementation is given in section 8.7. All this together – we think – will be enough to grasp the essence of medium degree tactics in the case of arithmetic. A listing of all tactics, the five mentioned above and the fourteen remaining ones, can be found in appendix B.

In the main text we will continue by explaining where the formality in our approach "hides" (section 8.8). Next, we will give some outlook what simulation of such single reasoning steps might look like in case of more complicated domains like automata theory (section 8.9). After that, we go some step further and ask whether low degree proofs can be simulated with the help of the computer, too (section 8.10). At the end we provide summary and conclusion (section 8.12).

8.2 The tactic drop_identities

8.2.1 The corresponding reasoning in medium degree proofs

When speaking of dropping identities we mean identities in an algebraic sense: 0 is an identity with respect to + and 1 with respect to $*$. Identities usually appear in proofs when some special instances are applied to a general rule. For example the scalar multiplication of a vector with the unit vector in $\mathbb{N}^3$, $\begin{pmatrix} n \\ m \\ l \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, would be computed to $n * 1 + m * 0 + l * 0$.

What is interesting for us, however, is the seemingly harmless '$= n$', which will follow. We are not offered an argumentation like the following:

> $k * 0 = 0$ for all $k \in \mathbb{N}$ since 0 is an absorbing element with respect to multiplication. In particular we have $m * 0 = 0$ and $l * 0 = 0$. Furthermore, since 0 is an identity, we have $k + 0 = k$ for all k and so in particular $m * 0 + l * 0 = 0$. For the same reason we obtain $n * 1 + m * 0 + l * 0 = n * 1$. Since 1 is an identity with respect to multiplication we conclude $n * 1 + m * 0 + l * 0 = n$.

Note that the identity 0 in $m * 0$ and $l * 0$ is not dropped but m and l are. The name of the tactic is meant to comprise this behaviour.

Except for the natural language in between the latter argumentation would be a high degree version, while the simple '$= n$' is a medium degree representative. In medium degree proofs often identities are not even written down such that the whole calculation of the previous example reduces to $\begin{pmatrix} n \\ m \\ l \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = n$.

Next step in Coq	Proof situation
Initial situation	$\frac{}{\forall\ n\ m\ l : \mathbb{N}, n * 1 + m * 0 + l * 0 = n}$
prove_all_imp_star	$\frac{\boxed{n : \mathbb{N}}\quad \boxed{m : \mathbb{N}}\quad \boxed{l : \mathbb{N}}}{n * 1 + m * 0 + l * 0 = n}$
use_equ (n_mul_0 m)	$\frac{}{n * 1 + 0 + l * 0 = n}$
use_equ (n_mul_0 l)	$\frac{}{n * 1 + 0 + 0 = n}$
use_equ (n_add_0 0)	$\frac{}{n * 1 + 0 = n}$
use_equ (n_mul_1 n)	$\frac{}{n + 0 = n}$
use_equ (n_add_0 n)	$\frac{}{n = n}$
prove_equ	Qed

Figure 8.1: Proof of $\forall\ n\ m\ l : \mathbb{N}, n * 1 + m * 0 + l * 0 = n$ without the tactic drop_identities.

8.2.2 Treatment in Coq

Let us stick with the last example and prove the theorem $\forall\ n\ m\ l : \mathbb{N}, n * 1 + m * 0 + l * 0 = n$ in Coq as shown in figure 8.1. We use the form of representation discussed in section 2.6.

The first step in the proof is a conglomerate of logical rules. Its impact is that all universally quantified variables and all premises are assumed (the variables are arbitrary but fixed) and the remaining part has to be shown.

The following steps are all of the form `use_equ` p. In the first case, for instance, the theorem `n_mul_0`, stating $\forall\ n : \mathbb{N}, n * 0 = 0$ is first applied to the m in our context, yielding a proof of $m * 0 = 0$. Then, secondly, we use this equality via `use_equ`, substituting all occurrences of $m * 0$ by 0 in the goal. In this case $m * 0$ occurs only one time so there is only a single replacement. The remaining steps involving `n_add_0` ($\forall\ n : \mathbb{N}, n + 0 = n$) and `n_mul_1` ($\forall\ n : \mathbb{N}, n * 1 = n$) behave analogously. The proof can be finished with `prove_equ`.

By contrast, the proof we would like to have looks like shown in figure 8.2. Here all previous `use_equ` steps are condensed into a single application of the `drop_identities` tactic.

Next, we show how this tactic can be implemented.

Next step in Coq	Proof situation
Initial situation	$\frac{}{\forall\ n\ m\ l : \mathbb{N}, n * 1 + m * 0 + l * 0 = n}$
prove_all_imp_star	$\frac{\boxed{n : \mathbb{N}}\quad \boxed{m : \mathbb{N}}\quad \boxed{l : \mathbb{N}}}{n * 1 + m * 0 + l * 0 = n}$
drop_identities	$\frac{}{n = n}$
prove_equ	Qed

Figure 8.2: Proof of $\forall\ n\ m\ l : \mathbb{N}, n * 1 + m * 0 + l * 0 = n$ with the tactic drop_identities.

8.2.3 Implementation

All tactics that we are going to discuss in the main text operate on hypotheses and goals[3], both of which can be equations, negations thereof or inequalities. The tactic `drop_identities` now has the special property of being value preserving, i. e. the left and right hand side of an hypothesis or goal are always replaced by terms of equal value. For each tactic satisfying this property it suffices to develop a helping tactic named with the suffix '`equ_left_in`' that only operates on the left hand side of hypotheses that are equations. In section 8.7 we will explain why. For now let us concentrate on the implementation of the tactic `drop_identities_equ_left_in`.

The way to the tactic `drop_identities_equ_left_in` is cumulative in character. Let us start with the application of base cases for addition. This is realised by the following tactic:

```
Ltac drop_identities_add_equ_left_in_one_step_base_cases H :=
let T := type of H in lazymatch T with
  0 + ?n = ?m =>
    use_equ (O_add_n n) in H at 1 (* H : n = m *)
| ?n + 0 = ?m =>
    use_equ (n_add_0 n) in H at 1 (* H : n = m *)
end.
```
[4]

Here, `Ltac` is the key word that a tactic definition follows, the rather long `drop_identities_add_equ_left_in_one_step_base_cases` is the tactic name

[3] In the appendix B we also discuss tactics, which operate only on hypotheses or only on goals.

[4] In the actual code we use $\oplus$ instead of $+$ since the latter has a fixed meaning. The same will be true for $\otimes$ and $*$.

and `H` is the hypothesis, on the left hand side of which we want to work on. The part `let T := type of H in lazymatch T with` prepares the matching of the type of H with the following patterns.[5] The two different cases are separated by `|`. After `=>` a sequence of instructions follows. Between `(*` and `*)` stand comments. The tactic says that if the hypothesis has the form $0 + n = m$ then the theorem `0_add_n` applied with parameter n (stating $0 + n = n$) should be used in H at the first possible position. The latter addendum is necessary since else occurrences of $0 + n$ in m would also be replaced.[6] The treatment of the second base case is analogous. If there is no matching, i. e. if the hypothesis has any other form the tactic fails.

The above tactic cannot deal with an equation like $(0 + n) + m = l$ since $0 + n$ only occurs in a subterm. The solution is to work recursively:

```
Ltac drop_identities_add_equ_left_in_one_step H :=
(* Base cases *)
  drop_identities_add_equ_left_in_one_step_base_cases H
||
(* Recursive cases *)
( let T := type of H in lazymatch T with
    Suc ?n = ?m =>
      left_recursive_unary_op_in_equ
        drop_identities_add_equ_left_in_one_step H n
  | pred ?n = ?m =>
      left_recursive_unary_op_in_equ
        drop_identities_add_equ_left_in_one_step H n
  | ?n + ?m = ?l =>
      left_recursive_binary_op_in_equ
        drop_identities_add_equ_left_in_one_step H n m
  end).
```

The `||` means that the part before is executed if it does not fail and if it make some progress, i. e. the proof situation must change somehow. In our case the previous tactic is tried. If this is not successful the tactic works recursively with the subterms (where the terms have to be of the form Suc $n = m$, pred $n = m$, or $n + m = l$). Since this kind of recursive calling is a general approach and does not only belong to the `drop_identities` tactic we use general tactics, `left_recursive_unary_op_in_equ` and `left_recursive_binary_op_in_equ` to deal with it.

[5] The prefix `lazy` guarantees that the tactic will not try another pattern if the first possible pattern fails.

[6] In this special case this side effect would have no negative consequences because in the tactic `drop_identities` the right hand side will be changed anyway. However, it is better when the helping tactics, too, satisfy clear specifications.

These meta tactics are given by the following definitions:

```
Ltac left_recursive_unary_op_in_equ Tac_equ_left_in H n :=
let H' := fresh in
(* H : op n = m *)
fact_name (equ_refl n) H'; (* H' : n = n *)
Tac_equ_left_in H'; (* H' : n' = n if it doesn't fail *)
use_uqe H' in H at 1; clear H' (* H : op n' = m *).

Ltac left_recursive_binary_op_in_equ Tac_equ_left_in H n m :=
let H' := fresh in
let S := fresh in
(* H : n op m = l *)
(* n can be modified at least one step *)
( fact_name (equ_refl n) H'; (* H' : n = n *)
  Tac_equ_left_in H'; (* H' : n' = n if it doesn't fail *)
  use_uqe H' in H at 1; clear H' (* H : n' op m = l *))
||
(* m can be modified at least one step *)
( fact_name (equ_refl m) H'; (* H' : m = m *)
  Tac_equ_left_in H'; (* H' : m' = m if it doesn't fail *)
  set (S := n) in H at 1; (* H : S op m = l *)
  use_uqe H' in H at 1; clear H'; (* H : S op m' = l *)
  change S with n in H; clear S (* H : n op m' = l *)).
```

Note that both tactics expect another tactic, namely `Tac_equ_left_in`, as one of their inputs. Note further that we only call these tactics if H is of the form op $n = m$ or n op $m = l$. Both tactics use a fresh name for a hypothesis (H') to save intermediate results therein. The main trick we are using here as well as in many other cases is to add an equation of the form $n = n$ first and modify it on the left (or the right) hand side; here with the help of the input tactic, which is `drop_identities_add_equ_left_in_one_step` in our case. The new equality is then used in the original one after which the former is deleted by `clear`. In the case of a binary operation this can happen for both subterms. This is why in the tactic `left_recursive_binary_op_in_equ` it is first tested whether we can successfully modify the first subterm and only otherwise we try to change the second one. This is realised by the `||` in the definition above.

Since the tactic `drop_identities_equ_left_in` should drop all identities on the left hand side of the equation at once we need to repeat the preceding tactic `drop_identities_add_equ_left_in_one_step` as long as there is process to be made. Fortunately, there is a repeat instruction at our disposal when writing Coq tactics. So the former tactic can be simply implemented by:

```
Ltac drop_identities_add_equ_left_in H :=
repeat (drop_identities_add_equ_left_in_one_step H).
```

So far our tactic is restricted to addition. Extending it to subtraction and multiplication, however, is not that difficult.[7] First of all the base cases have to be extended. The respective transition from subtraction to multiplication, for instance, looks like this:

```
Ltac drop_identities_equ_left_in_one_step_base_cases H :=
(* The old base cases *)
  drop_identities_sub_equ_left_in_one_step_base_cases H
||
(* The new base cases *)
( let T := type of H in lazymatch T with
    1 * ?n = ?m =>
      use_equ (I_mul_n n) in H at 1 (* H : n = m *)
  | ?n * 1 = ?m =>
      use_equ (n_mul_1 n) in H at 1 (* H : n = m *)
  | 0 * ?n = ?m =>
      use_equ (O_mul_n n) in H at 1 (* H : 0 = m *)
  | ?n * 0 = ?m =>
      use_equ (n_mul_0 n) in H at 1 (* H : 0 = m *)
  end).
```

Hence the tactic tries first to apply one of the already implemented base cases and only otherwise deals with the following four cases.

`drop_identities_equ_left_in_one_step` is very similar to the already presented `drop_identities_add_equ_left_in_one_step`. Instead of trying to apply `drop_identities_add_equ_left_in_one_step_base_cases` we now try to apply `drop_identities_equ_left_in_one_step_base_cases`. Subtraction and multiplication are added to the recursive patterns and of course the recursive call has to be adjusted. The final tactic we are striving for, `drop_identities_equ_left_in`, is then again only a repetition of `drop_identities_equ_left_in_one_step`.

8.2.4 Evaluation

In this first case of a tactic simulating a class of medium degree steps there is not much to criticise: the tactic does in one step exactly what humans do in such situations. The only thing that cannot be simulated (this way) is the skipping in many medium degree proofs, when terms with identities are not even written down.

[7] This – of course – is only true because the previous implementation was optimised regarding the extensions.

8.3 The tactic suc_pred_to_front

8.3.1 The corresponding reasoning in medium degree proofs

There is a principled problem with the analysis of the treatment of the successor function (abbreviated by Suc) and the predecessor function (abbreviated by pred) in medium degree proofs since most often they are not used at all. Instead we find only $+1$ or -1. However, we cannot dismiss Suc and pred altogether because at least the former is needed as a foundation before we are able to define addition (or subtraction). So in our analysis of how Suc and pred "are used" in medium degree proofs we should pretend to see Suc where we actually see $+1$.

In a case, in which a term $(n+1)+(m+1)$ is given, for instance, one usually would immediately transform it into the equal $(n+m)+2$ whereby one would leave the parentheses implicit. Here '$+2$' replaces $+1+1$ and so can be conceived as a counting continuation. The analogy in terms of Suc would be Suc n + Suc m = Suc (Suc $(n+m)$).

Another example is an occurrence of something like $(n+1)-1$, which would be reduced to n. Written with Suc and pred the change would be from pred Suc n to n. What is special here is that pred and Suc eliminate each other. The same is true for $(n+1)-(m+1)$ or Suc n − Suc m respectively.

We might also be tempted to reduce $(n-1)+1$ to n but this would be a mistake concerning the natural numbers since it is wrong for $n = 0$. However, with the additional knowledge of $n \neq 0$ this would be correct and in the analogous version we would transform Suc pred n to n.

In the case of dropping identities even long terms could be handled in one sweep. By contrast, humans have to calculate terms like $((n+1)*(m-1)+1)*(m+1)$ step by step. Therefore the tactic `suc_pred_to_front` should also work successively.

8.3.2 Treatment in Coq

In the following we will analyse the proof of a statement in Coq that involves Suc, pred, addition, multiplication, and subtraction: $\forall\ n\ m\ l\ k : \mathbb{N}, n+m = l*k-k \rightarrow$ pred $(n +$ Suc $m) =$ pred $l * k$. This theorem might look a little bit artificial but it enables us to get the decisive points without too much irrelevant proof steps.

The proof without the use of the tactic `suc_pred_to_front` is shown in figure 8.3. After introducing the variables and the hypothesis we change the goal successively by applying equalities given by theorems representing special situations of Suc and pred. The first theorem, for instance, states $\forall\ n\ m : \mathbb{N}, n+$Suc $m =$ Suc $(n+m)$. This is applied to the concrete n and m of our proof and we obtain

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m\ l\ k : \mathbb{N}, n + m = l * k - k \rightarrow$ pred $(n +$ Suc $m)$ = pred $l * k$
prove_all_imp_star	$n : \mathbb{N}$, $m : \mathbb{N}$, $l : \mathbb{N}$, $k : \mathbb{N}$, $H : n + m = l * k - k$ / pred $(n +$ Suc $m)$ = pred $l * k$
use_equ (n_add_suc_m n m)	pred (Suc $(n + m)$) = pred $l * k$
use_equ (pred_suc_n $(n + m)$)	$n + m$ = pred $l * k$
use_equ (pred_n_mul_m l k)	$n + m = l * k - k$
use H	Qed

Figure 8.3: Proof of $\forall\ n\ m\ l\ k : \mathbb{N}, n+m = l*k-k \rightarrow$ pred $(n+$Suc $m)$ = pred $l*k$ without the tactic suc_pred_to_front.

an equality $n +$ Suc $m =$ Suc $(n + m)$ which is used to modify the goal. The next two steps are analogous. Finally, we can use our hypothesis to conclude the proof.

Let us compare this proof with the one using the tactic `suc_pred_to_front` as shown in figure 8.4. While we have the same beginning we now treat both sides of the equation simultaneously by using `suc_pred_to_front`, which incidentally changes the order of replacement. Since only one step at a time is executed on both sides, we need to run `suc_pred_to_front` twice.

The problem of the proof version not using the tactic `suc_pred_to_front` in comparison to the one using that tactic is not the length of the proof, i. e the condensation aspect; the difference in steps is only one here. Instead the different forms of justification make both proofs very different. In the first one we have to know (or find out) all the theorems we need to apply. Furthermore, having to provide all parameters explicitly is annoying. By contrast, in the second proof nothing like that has to be done. The call of `suc_pred_to_front` is rather a rough description of what is happening. Note that in the first proof the three justifications regarding the treatment of the successor and predecessor function are all different while in the corresponding steps of the second proof the two justifications are the same.

Next step in Coq	Proof situation
Initial situation	$\dfrac{}{\forall\ n\ m\ l\ k : \mathbb{N}, n + m = l * k - k \rightarrow \text{pred}\ (n + \text{Suc}\ m) = \text{pred}\ l * k}$
prove_all_imp_star	$\dfrac{\boxed{n : \mathbb{N}}\ \boxed{m : \mathbb{N}}\ \boxed{l : \mathbb{N}}\ \boxed{k : \mathbb{N}}\ \boxed{H : n + m = l * k - k}}{\text{pred}\ (n + \text{Suc}\ m) = \text{pred}\ l * k}$
suc_pred_to_front	$\dfrac{}{\text{pred}\ (\text{Suc}\ (n + m)) = l * k - k}$
suc_pred_to_front	$\dfrac{}{n + m = l * k - k}$
use H	Qed

Figure 8.4: Proof of $\forall\ n\ m\ l\ k : \mathbb{N}, n+m = l*k-k \rightarrow \text{pred}\ (n+\text{Suc}\ m) = \text{pred}\ l*k$ with the tactic suc_pred_to_front.

8.3.3 Implementation

Like `drop_identities` the tactic `suc_pred_to_front` is value preserving. So the implementation of the tactic `suc_pred_to_front_equ_left_in` needs to be elaborated. However, for reasons that will be discussed two paragraphs below this does not suffice this time.

As we have already seen, the arithmetical tactics are cumulative with respect to their restrictions. In case of `suc_pred_to_front_equ_left_in` we have the following restricted versions: one for successor and predecessor function only, one variant allowing also addition, one including subtraction as well, and finally `suc_pred_to_front_equ_left_in` itself, which also allows multiplication. We will only discuss the construction of the second, namely the first one dealing with addition, because all others are very similar in character and because all relevant implementation aspects appear in that variant.

We already indicated that the tactic `suc_pred_to_front` is somewhat exceptional. The reason for this is that it can be applied even in proof situation that require some additional information. Let us consider the modification of the term $n + \text{pred}\ m$ as an example. We want to shift the predecessor application to the front but this requires a cut stating $m \neq 0$. For avoiding unnecessary cuts, however, it is a good idea to concentrate on those cases first that allow a successful application without the use of cuts.

This is done by a helping tactic. We first consider its base cases:

```
Ltac suc_pred_to_front_add_equ_left_in_help_base_cases H :=
(* The old base cases without cut *)
  suc_pred_to_front_suc_pred_equ_left_in_help_base_cases H
||
(* The new base cases without cut *)
( let T := type of H in lazymatch T with
    Suc ?n + ?m = ?l =>
      use_equ (suc_n_add_m n m) in H at 1 (* H : Suc (n + m) = l *)
  | ?n + Suc ?m = ?l =>
      use_equ (n_add_suc_m n m) in H at 1 (* H : Suc (n + m) = l *)
  end).
```

The tactic comprises the base cases of its more restricted version that only considers the successor and predecessor function. Furthermore the patterns Suc $n+m = l$ and $n +$ Suc $m = l$ are added. In both cases a theorem can be applied allowing us to shift the successor application to the front.

Next, the tactic `suc_pred_to_front_add_equ_left_in_help` adds the recursive cases and – if possible – executes them, provided that no base case can be applied:

```
Ltac suc_pred_to_front_add_equ_left_in_help H :=
(* Base cases without cut *)
  suc_pred_to_front_add_equ_left_in_help_base_cases H
||
(* Recursive cases without cut *)
( let T := type of H in lazymatch T with
    Suc (?n) = ?m =>
      left_recursive_unary_op_in_equ
        suc_pred_to_front_add_equ_left_in_help H n
  | pred (?n) = ?m =>
      left_recursive_unary_op_in_equ
        suc_pred_to_front_add_equ_left_in_help H n
  | ?n + ?m = ?l =>
      left_recursive_binary_op_in_equ
        suc_pred_to_front_add_equ_left_in_help H n m
  end).
```

Now we turn our attention to those situations, in which cuts need to be included. Again we start with the base cases:

```
Ltac suc_pred_to_front_add_equ_left_in_base_cases H :=
let H' := fresh in
(* The old base cases with cut *)
  suc_pred_to_front_suc_pred_equ_left_in_base_cases H
||
(* The new base cases with cut *)
( let T := type of H in lazymatch T with
    pred ?n + ?m = ?l =>
      assert (n ≠ 0);
      [ (* ⊢ n ≠ 0 *)
        idtac
      | (* H' : n ≠ 0 *)
        use_equ (pred_n_add_m H' m) in H at 1; clear H'
          (* H : pred (n + m) = l *)]
  | ?n + pred ?m = ?l =>
      assert (m ≠ 0);
      [ (* ⊢ m ≠ 0 *)
        idtac
      | (* H' : m ≠ 0 *)
        use_equ (n_add_pred_m n H') in H at 1; clear H'
          (* H : pred (n + m) = l *)]
  end).
```

As in the tactic implementation of the cutless part the new base cases with cut comprise the old base cases with cut. To shift the predecessor function to the front we need the additional information $n \neq 0$ (or $m \neq 0$ respectively). This requirement is conveyed to the user with the help of the assert, which generates two proof tasks. The treatment of the first – requiring it to be different from 0 – is stated before the `|` while the treatment of the second one – being the old goal with the first subgoal as additional hypothesis – is described afterwards. Note that `idtac`, the identity tactic, does not change the respective proof situation. So in such situations the user has to prove $n \neq 0$ (or $m \neq 0$ respectively) first and then the predecessor application is shifted automatically to the front.

The implementation of the tactic `suc_pred_to_front_add_equ_left_in` comprises the helping tactic and is – apart from that – essentially the same as in the case of the latter tactic but with the new base cases (and an error message).

```
Ltac suc_pred_to_front_add_equ_left_in H :=
(* The case that it is possible to shift without a cut *)
  suc_pred_to_front_add_equ_left_in_help H
||
```

```
  (* Base cases with cut *)
(   suc_pred_to_front_add_equ_left_in_base_cases H
  ||
  (* Recursive cases with cut *)
  ( let T := type of H in lazymatch T with
      Suc (?n) = ?m =>
        left_recursive_unary_op_in_equ
          suc_pred_to_front_add_equ_left_in H n
    | pred (?n) = ?m =>
        left_recursive_unary_op_in_equ
          suc_pred_to_front_add_equ_left_in H n
    | ?n + ?m = ?l =>
        left_recursive_binary_op_in_equ
          suc_pred_to_front_add_equ_left_in H n m
    end))
||
  fail "suc_pred_to_front_add_equ_left_in H:
        No Suc or pred can be shifted to the front on the left hand
        side".
```

Let us now briefly discuss the particular difficulties that the cuts are causing when lifting to `suc_pred_to_front`. In a situation like $n + \text{pred } m \leq l + \text{Suc } k$, for instance, an application of the tactic should yield $n + \text{pred } m \leq \text{Suc } (l + k)$ – without any cuts. More generally, the tactic should introduce new subgoals only if otherwise no process on both sides was possible. So first it must be tried out whether the tactic `suc_pred_to_front_add_equ_left_in_help` can be applied on both sides of the respective hypothesis or goal. If this is not the case the same has to be done for the left hand side only. If this does not work, neither, it is tried on the right hand side.[8] Only if all these attempts fail the tactic `suc_pred_to_front_add_equ_left_in` is applied.

8.3.4 Evaluation

The tactic `suc_pred_to_front` is special since it does not simulate an actual reasoning of medium degree proofs but attempts to simulate a reasoning that itself simulates the treatment of $+1$ or -1 in terms. The reasoning about $+1$ and -1 in medium degree proofs usually separates the concrete forward and backward countings from the variables and combines them to one single number; usually being put at the end. By contrast, the simulation of this in terms of Suc and pred

[8] What exactly is meant by applying an `equ_left_in`-tactic on the left or the right hand side of a hypothesis or goal that does not have to be an equation will be discussed in section 8.7.

brings the applications to the front. However, apart from this the treatment is analogous. In particular the different applications of Suc and pred can erase each other such that the equivalent to a concrete number, a term beginning only with applications of Suc or only with applications of pred, remains.

Our tactic is able to simulate the simulation of the treatment of $+1$ and -1 without problems. Since there are situations where it is difficult to see the outcome of bringing Suc and pred to front immediately we decided to let the tactic only do one change at a time – but on both sides of the equation. A drawback of this decision is that there are situations, too, where we can see the outcome of bringing Suc and pred to front immediately. This is an example, in which empirical investigations (with the corresponding version of $+1$ and -1) would have to be done to justify or deny this successive behaviour. The salient point, however, is that any adjustment – if necessary – would be possible.

8.4 The tactic omit_parens

8.4.1 The corresponding reasoning in medium degree proofs

In mathematics, addition, multiplication etc. are defined as binary operations written in infix notation. One consequence is that in principle any calculation with three or more arguments has to use parentheses for clarification. For instance one has to write $n+(m+l)$ instead of just $n+m+l$. Due to the associativity law, however, the former term and $(n+m)+l$ result in the same value. As a consequence in such cases parentheses can be omitted. In order to avoid further parentheses there is a binding hierarchy introduced between the different operations. So most of the time only those parentheses that one cannot avoid with these two kinds of conventions remain.[9] The respective omissions are usually done in one sweep. Often the whole situation before the omission of parentheses is not even written down.

In high degree systems and in Coq in particular there is a problem with this attitude. This is not due to the binding hierarchies, which can be introduced without any problems, but due to the "do-not-care conventions" corresponding to associativity. Fortunately, there is some workaround: one can define some normal form with respect to parentheses and leave the parentheses in that case implicit. Indeed, this is what we do in our tactic `omit_parens`. We bring terms into their respective normal form.

[9] Sometimes unnecessary parentheses are used to emphasize that terms belong together; for instance because the respective construct is replaced in the next step or one needs a special form to apply a theorem etc.

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m\ l\ k\ j : \mathbb{N},$ $(((n * m) * l + (n + m) + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
prove_all_imp_star	$\boxed{n : \mathbb{N}}$ $\boxed{m : \mathbb{N}}$ $\boxed{l : \mathbb{N}}$ $\boxed{k : \mathbb{N}}$ $\boxed{j : \mathbb{N}}$ $(((n * m) * l + (n + m) + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
use_equ (mul_assoc $n\ m\ l$)	$((n * m * l + (n + m) + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
use_equ (add_assoc $n\ m\ l$)	$((n * m * l + n + m + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
use_equ (mul_assoc $(n * m * l + n + m + l)\ k\ j$)	$(n * m * l + n + m + l) * k * j$ $= (n * m * l + n + m + l) * k * j$
prove_equ	Qed

Figure 8.5: Proof of $\forall\ n\ m\ l\ k\ j : \mathbb{N}, (((n * m) * l + (n + m) + l) * k) * j = (n * m * l + n + m + l) * k * j$ without the tactic omit_parens.

8.4.2 Treatment in Coq

Let us see how we can omit all unnecessary parentheses in the term $(((n * m) * l + (n + m) + l) * k) * j$ (figure 8.5). After the usual introducing of variables we have to change the left side of the equation into the right one. We do this by successively applying the associativity of addition (once) and multiplication (twice). Furthermore we have to state each time exactly with which parameters the associativity laws have to be applied. This is especially cumbersome in the last case since the first term is already very long.

Bear in mind that our goal of omitting parentheses feels rather simple. So the tactic `omit_parens` should make the proof simple, too. The three steps applying associativity laws should become one (condensation) and no mentioning of the parameters should be necessary (name of process as justification suffices). Indeed this is the case, as shown in figure 8.6.

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m\ l\ k\ j : \mathbb{N},$ $(((n * m) * l + (n + m) + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
prove_all_imp_star	$\boxed{n : \mathbb{N}}$ $\boxed{m : \mathbb{N}}$ $\boxed{l : \mathbb{N}}$ $\boxed{k : \mathbb{N}}$ $\boxed{j : \mathbb{N}}$ $(((n * m) * l + (n + m) + l) * k) * j$ $= (n * m * l + n + m + l) * k * j$
omit_parens	$(n * m * l + n + m + l) * k * j$ $= (n * m * l + n + m + l) * k * j$
prove_equ	Qed

Figure 8.6: Proof of $\forall\ n\ m\ l\ k\ j : \mathbb{N}, (((n * m) * l + (n + m) + l) * k) * j = (n * m * l + n + m + l) * k * j$ with the tactic omit_parens.

8.4.3 Implementation

The implementation of `omit_parens` is simple. It is a value preserving tactic (with no cut issues) such that it suffices to care about the implementation of the helping tactic `omit_parens_equ_left_in`. The idea is to implement a tactic first that omits exactly one pair of parentheses – and fails if this is not possible. The true tactic is then just repetition of the one step version.

For the same reasons as in the previous implementations we skip the restriction to addition and focus on the multiplication part. Once more, the one step tactic separates the treatment of the base cases:

```
Ltac omit_parens_equ_left_in_one_step_base_cases H :=
(* The old base cases *)
  omit_parens_add_equ_left_in_one_step_base_cases H
||
(* The new base cases *)
( let T := type of H in lazymatch T with
    (?n * ?m) * ?l = ?k =>
      use_equ (mul_assoc n m l) in H at 1 (* H : n * m * l = k *)
  end).
```

The old base case is the same as the one added here but with addition instead of multiplication.

If a term does not have the overall form for omitting parentheses one of its subterms might have it. So again we need a possibility to forward the job to the subterms. This can be done canonically already resulting in our one step version:

```
Ltac omit_parens_equ_left_in_one_step H :=
(* Base cases *)
  omit_parens_equ_left_in_one_step_base_cases H
||
(* Recursive cases *)
( let T := type of H in lazymatch T with
    Suc ?n = ?m =>
      left_recursive_unary_op_in_equ
        omit_parens_equ_left_in_one_step H n
  | pred ?n = ?m =>
      left_recursive_unary_op_in_equ
        omit_parens_equ_left_in_one_step H n
  | ?n + ?m = ?l =>
      left_recursive_binary_op_in_equ
        omit_parens_equ_left_in_one_step H n m
  | ?n * ?m = ?l =>
      left_recursive_binary_op_in_equ
        omit_parens_equ_left_in_one_step H n m
  end).
```

If it is possible to omit a pair of parentheses at all then the above tactic omits exactly one such pair. Hence by repetition we must reach a term having no omittable parentheses left.

```
Ltac omit_parens_equ_left_in H :=
repeat (omit_parens_equ_left_in_one_step H).
```

8.4.4 Evaluation

At first glance the tactic `omit_parens` seems to work perfectly. Like in the medium degree reasoning all omittable parentheses are in fact omitted. This is done in one step, which – as already discussed in the case of `drop_identities` – is the best we can expect. Furthermore, we do not have to add any unnecessary information; it suffices to name the process that should happen.

Yet, one should not forget that the parentheses do still exist inside the system. If, for instance, one wants to put j in $(n * m * l + n + m + l) * k * j$ to the front

this is not directly possible via the use of commutativity. The reason is that the system sees $(n*m*l+n+m+l)*k*j$ as $(n*m*l+n+m+l)*(k*j)$ instead of $((n*m*l+n+m+l)*k)*j$. If we take care of this problem in the implementation of other tactics as well, however, then the "illusion" can be maintained.

8.5 The tactic make_first

8.5.1 The corresponding reasoning in medium degree proofs

Like the associativity law for operators permits one to omit parentheses the commutativity law allows one to change the position of terms applied to the respective operations. For instance, $(n+m)*(l+k)$ can be replaced by $(k+l)*(n+m)$. Again, the guiding principle is that commutations do not matter, i. e. a do-not-care attitude.

As already discussed in the case of omitting parentheses the computer is not able to work with such an attitude. Regarding parentheses we found a normal form to disguise this problem. Yet, in the case of commutativity there is no appropriate normal form. Of course there are some conventions, for instance to write $42+n*n*m+n*73$ as $m*n^2+73*n+42$, but these only work in special contexts stating with which kind of objects one is dealing (in that case polynomials over n). Sometimes there is no such context and the user would be quite surprised if the term at hand was ordered in a way that is not comprehensible for him.

8.5.2 Treatment in Coq

As we discussed in the last subsection, in general it is not a good idea to bring terms in some polynomial normal form. Nevertheless – if required – it must be possible to make such changes somehow. So let us consider how the term $9+6*n+2*n*n+3*n$ can be transformed into $2*n*n+9*n+9$. This seems to be pretty easy; yet, the proof without `make_first` is a little bit tedious (see figure 8.7).

The first difficulty is to bring the two relevant subterms, $6*n$ and $3*n$, together. The idea is to move the disruptive $2*n*n$ one position more to the left. This can be done by using the associativity law backwards first. For this, of course, we have to realise that there is a subterm $6*n+2*n*n+3*n$, which is implicitly $6*n+(2*n*n+3*n)$. After the backward use of the associativity law we obtain $(6*n+2*n*n)+3*n$, in which we can change $6*n$ and $2*n*n$ due to the commutativity of addition. Using associativity a second time brings $6*n$ and $3*n$ together. The next phase is to calculate $6+3$. For this we use distributivity

Next step in Coq	Proof situation
Initial situation	$\forall\ n : \mathbb{N}, 9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$
prove_all	$n : \mathbb{N}$ $9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$
use_uqe (add_assoc $(6 * n)\ (2 * n * n)\ (3 * n)$)	$9 + (6 * n + 2 * n * n) + 3 * n = 2 * n * n + 9 * n + 9$
use_equ (add_comm $(6 * n)\ (2 * n * n)$)	$9 + (2 * n * n + 6 * n) + 3 * n = 2 * n * n + 9 * n + 9$
use_equ (add_assoc $(2 * n * n)\ (6 * n)\ (3 * n)$)	$9 + 2 * n * n + 6 * n + 3 * n = 2 * n * n + 9 * n + 9$
use_uqe (add_mul_distr 6 3 n)	$9 + 2 * n * n + (6 + 3) * n = 2 * n * n + 9 * n + 9$
unfold "+" at 3	$9 + 2 * n * n + 9 * n = 2 * n * n + 9 * n + 9$
use_equ (add_comm $9\ (2 * n * n + 9 * n)$)	$(2 * n * n + 9 * n) + 9 = 2 * n * n + 9 * n + 9$
use_equ (add_assoc $(2 * n * n)\ (9 * n)\ 9$)	$2 * n * n + 9 * n + 9 = 2 * n * n + 9 * n + 9$
prove_equ	Qed

Figure 8.7: Proof of $\forall\ n : \mathbb{N}, 9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$ without the tactic make_first.

backwards to factorise n out. The unfold rule calculates $6+3$ as 9. After reaching the same terms on both sides we have to bring them in the same order. This is again done by using commutativity and associativity.

The corresponding proof using `make_first` (see figure 8.8) is more condensed and has simpler justifications but is a little bit tricky, too: instead of moving the terms we are interested in, we are moving the uninteresting terms away. In this case there is only $2 * n * n$ but in general we have to move every term between the two terms of interest as well as all the terms behind both interesting terms.[10] The use of distributivity and the calculation is the same as in the previous proof. The last task, which is typical, is reordering. Once more, the special case we are considering is simple since we only need to bring $9 * n$ to the front. If we do not want to change the term behind the equality sign we can use `make_first` $(9 * n)$ `at` 1 followed by `make_first` $(2 * n * n)$ `at` 1 instead, where the number behind the 'at' says that we only want to change the respective occurrence.

8.5.3 Implementation idea

So far we have discussed the implementation of each tactic in much technical detail and the reader should have gained some grasp of how to do the technical realisations. Therefore we can switch to the bird's eye perspective from now on and focus on the implementation ideas instead.

The tactic `make_first_equ_left_in` receives a term as input and shifts the first of its occurrences on the left hand side of the hypothesis – where this is possible – to the front. The main idea is to separate the search for such an occurrence and the shifting.

To do so we need a tactic that checks whether the first occurrence of a term is firstable. This statement needs some clarification. By checking we mean a tactic that is `idtac` in case the condition is fulfilled and that otherwise fails. Next, we call an occurrence of a term (on the left hand side of a hypothesis) firstable if all ancestors of this occurrence in the term tree are the successor function, the predecessor function, addition, or multiplication.[11] Furthermore we require that the occurrence is not a proper part of a number instance: we must prevent that shifting 4 to the front in $n + 5 + 4$ yields $5 + n + 4$ since 5 is just an abbreviation for Suc 4.

In the next step we can extend the tactic checking firstability to one that marks the first firstable occurrence of the term and fails if there is no such occurrence.

[10] The latter could be done by one single step via `make_first` since these terms are all in one parenthesis. Yet, we have to spend one `omit_parens` afterwards if we want to avoid parentheses in the end.

[11] An extension to arbitrary unary operations is worth considering while arbitrary binary operations cannot be considered since they do not need to be commutative.

Next step in Coq	Proof situation
Initial situation	$\forall\ n : \mathbb{N}, 9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$
prove_all	$\boxed{n : \mathbb{N}}$ $9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$
make_first $(2 * n * n)$	$2 * n * n + 9 + 6 * n + 3 * n = 2 * n * n + 9 * n + 9$
use_uqe (add_mul_distr 6 3 n)	$2 * n * n + 9 + (6 + 3) * n = 2 * n * n + 9 * n + 9$
unfold "+" at 3	$2 * n * n + 9 + 9 * n = 2 * n * n + 9 * n + 9$
make_first $(9 * n)$	$9 * n + 2 * n * n + 9 = 9 * n + 2 * n * n + 9$
prove_equ	Qed

Figure 8.8: Proof of $\forall\ n : \mathbb{N}, 9 + 6 * n + 2 * n * n + 3 * n = 2 * n * n + 9 * n + 9$ with the tactic make_first.

This is realised by going through all occurrences successively until a firstable has been found. Since there does not seem to be a good way for counting indices[12] we use a technique we call *immunisation* to go through: we always check the first occurrence and substitute it by a new fixed variable if the respective term is not firstable. So after immunisation the old second occurrence becomes the new first one. If we reach a firstable occurrence we replace it by another new variable and all previous substitutions, indicating failures, are changed back. So there is only one substitution variable left (if there is at least one firstable occurrence), which we shift to the front afterwards (see below) and only then change back.

So how does the shifting work when we have the guarantee that the term under discussion occurs at most once?[13] We have a base case stating that nothing needs to be done if the term to be shifted is the whole left hand side of the respective hypothesis. If the left hand side is a successor or predecessor application then we

[12] The position numbers are of an ML type for natural numbers, not the one of Coq.

[13] The tactic cannot expect exactly one occurrence since it has to work with the subterms, too, which in general do not contain the immunisation variable under discussion.

can simply create a new hypothesis for the input and work on the left hand side. For shifting n to the front in the term Suc $(m + n)$, for instance, we create a new hypothesis stating $m + n = m + n$ and shift n to the front on the left hand side of this hypothesis before we apply the result to the original hypothesis.

Things are not that simple in the case of addition and multiplication. If we want to shift n to the front on the left hand side of a hypothesis of the form m op $l = k$ we first construct a new hypothesis $l = l$ and try to shift n to the front on the left hand side of this; the result being saved in the new hypothesis as $l' = l$ for some l'. Now we have to check if n is at the front of l'. This requires an extra tactic since the n could be nested as the example $(n + j) * i = 42$ illustrates. Now, if n is in front of l' we want to change m op $l = k$ to m op $l' = k$[14] and then commute m and the beginning of l' containing the n. If l' is itself of the form j op i, however, we have to use the associativity law twice to keep the order of the noninvolved terms – as far as possible – and to avoid the creation of new parentheses.

If n is not in front of l' a similar procedure – but without the commutation – is done for m instead of l. If n is not in m neither, the whole tactic behaves like `idtac` to enable recursive calls. By contrast, the overall tactic of this subsection, `make_first_equ_left_in`, would fail in such situations since the tactic trying to immunise the first firstable occurrence of n, which is applied before, would fail.

8.5.4 Evaluation

The tactic `make_first` is further away from the corresponding reasoning in proofs of a medium degree than the other tactics considered so far. We cannot just point to two subterms and commute them (if legal). Instead our tactic allows to bring single terms to the front (if legal). This enables us to bring the subterms in our desired order but this may require many steps; so condensation cannot be guaranteed. Sometimes there are ways to make the new arrangement in fewer steps but to see these is a lot easier for a user realising the implicit parentheses. Furthermore the handling is often counterintuitive: we have to work with the terms we are not interested in for the moment to bring the interesting terms together.

Nevertheless the handling of commutativity with the help of `make_first` is more adequate than without. The justification can at least concentrate on the terms that are brought to the front and the user does not have to know the commutativity and associativity law (as well as the names thereof) nor the implicit parentheses to find some way of rearrangement.

[14] This is unproblematic since n is in front of l' and hence occurs already in l. So n cannot occur in m since otherwise it would occur at least twice in total, contradicting our assumption.

A rather easy improvement would be a tactic with two (or more) terms as inputs that shifts both terms to the end such that these are in one pair of parentheses. A more challenging betterment would be to somehow track the position of the two terms (for instance via a path in the syntax tree) and commute only these two positions via automatic use of associativity and commutativity laws.

However, perhaps the most important purpose of `make_first` is not its direct use for commutations but its preparative character for the `drop` tactic, which we are going to consider next.

8.6 The tactic drop

8.6.1 The corresponding reasoning in medium degree proofs

When developing proofs of a medium degree of formality we often reach situations, in which it suffices to show some arithmetical (in)equation or negation thereof. Let us say, for instance, that $n + m^2 + k^2 + 2 * l * m + l^2 \leq 3 * k^2 + (l + m)^2 + n$ remains to be proven. In such cases we simply drop n and k^2 and show $m^2 + 2 * l * m + l^2 \leq 2 * k^2 + (l + m)^2$ instead. If we use the first binomial formula we can drop $m^2 + 2 * l * m + l^2$ and reach $0 \leq 2 * k^2$. This is trivially true for natural numbers but it also holds for integers, rationals, and reals.

Sometimes we want to drop a term in a hypothesis we already have at our disposal. In $n + m = m + l$ this is unproblematic and by dropping m we obtain $n = l$; whereas in case of multiplication such dropping is only possible for $m \neq 0$.

Although the treatment of goals and hypotheses looks similar, on a high degree of formality it is quite different. In the former case the terms that seem to be dropped are actually added to the new goal (as sums or factors) and then the terms are placed in the order of the original goal. In the latter case, by contrast, we need to use cancellation laws like $n + m = n + l \rightarrow m = l$. If negations are involved the treatment of goals and hypotheses is exchanged.

8.6.2 Treatment in Coq

In order to see how the dropping of terms works in Coq let us consider the statement $\forall\ n\ m\ l\ k\ j : \mathbb{N},\ m \neq 0 \rightarrow n * m * l = m * k \rightarrow n * j * l = j * k$. This example is well suited since we will need the reasoning for dropping terms in a hypothesis as well as in the goal.

The proof without the use of the tactic `drop` (and without the use of other medium degree tactics) is shown in figure 8.9. The essential part of the proof begins by bringing j to the front (steps 3-5). We already know that this could

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m\ l\ k\ j : \mathbb{N}, m \neq 0 \to n * m * l = m * k \to n * j * l = j * k$
prove_all_imp_star	$\boxed{n : \mathbb{N}}$ $\boxed{m : \mathbb{N}}$ $\boxed{l : \mathbb{N}}$ $\boxed{k : \mathbb{N}}$ $\boxed{j : \mathbb{N}}$ $\boxed{H : m \neq 0}$ $H0 : n * m * l = m * k \vdash n * j * l = j * k$
use_uqe (mul_assoc $n\ j\ l$)	$H0 : n * m * l = m * k \vdash (n * j) * l = j * k$
use_equ (mul_comm $n\ j$)	$H0 : n * m * l = m * k \vdash (j * n) * l = j * k$
use_equ (mul_assoc $j\ n\ l$)	$H0 : n * m * l = m * k \vdash j * n * l = j * k$
cut $(n * l = k)$	
$+_1$	$\boxed{H0 : n * m * l = m * k} \vdash n * l = k \to j * n * l = j * k$
prove_imp	$\boxed{H1 : n * l = k} \vdash j * n * l = j * k$
use_equ $H1$	$\vdash j * k = j * k$
prove_equ	$+_1$ completed
$+_2$	$H0 : n * m * l = m * k \vdash n * l = k$
use_uqe (mul_assoc $n\ m\ l$) in $H0$	$H0 : (n * m) * l = m * k \vdash n * l = k$
use_equ (mul_comm $n\ m$) in $H0$	$H0 : (m * n) * l = m * k \vdash n * l = k$
use_equ (mul_assoc $m\ n\ l$) in $H0$	$\boxed{H0 : m * n * l = m * k} \vdash n * l = k$
use (n_mul_m_equ_n_mul_l $H\ H0$)	Qed

Figure 8.9: Proof of $\forall\ n\ m\ l\ k\ j : \mathbb{N}, m \neq 0 \to n * m * l = m * k \to n * j * l = j * k$ without the tactic drop.

Next step in Coq	Proof situation
Initial situation	$\forall\ n\ m\ l\ k\ j : \mathbb{N}, m \neq 0 \to n * m * l = m * k \to n * j * l = j * k$
prove_all_imp_star	$\boxed{n : \mathbb{N}}$ $\boxed{m : \mathbb{N}}$ $\boxed{l : \mathbb{N}}$ $\boxed{k : \mathbb{N}}$ $\boxed{j : \mathbb{N}}$ $\boxed{H : m \neq 0}$ $H0 : n * m * l = m * k$ ——— $n * j * l = j * k$
drop j	$H0 : n * m * l = m * k$ ——— $n * l = k$
drop m in $H0$	
$+_1$	$H0 : n * m * l = m * k$ ——— $m \neq 0$
use H	$+_1$ completed
$+_2$	$\boxed{H0 : n * l = k}$ ——— $n * l = k$
use $H0$	Qed

Figure 8.10: Proof of $\forall\ n\ m\ l\ k\ j : \mathbb{N}, m = 0 \to n * m * l = m * k \to n * j * l = j * k$ with the tactic drop.

have been done in one step using the tactic `make_first`. Next, we show that $n * l = k$ suffices to prove the original goal.[15] This claim is proven in the subpart of the proof, marked by '$+_1$', by using the equality. It remains to show $n * l = k$ (subpart '$+_2$'). Here again, the first three steps are only the long version of bringing m to the front in $H0$. As the proof illustrates this is the expensive part while the remainder is just an application of the theorem `n_mul_m_equ_n_mul_l`.

The second proof (see figure 8.10) is more condensed and the justifications are much simpler. The only two essential steps are the dropping of j and m (the last one in $H0$). The first dropping is unproblematic, while the second one requires the further information of $m \neq 0$, which is why we have to branch. Note that we do not have to be aware of any of the problems regarding parentheses since these are already managed by the tactic.

[15] With other tactics than `drop` allowed we could have used here `prove_by_components_equ`, too (see appendix B).

8.6.3 Implementation idea

The basic idea for the implementation of `drop` is to use the tactic `make_first` – or to be more precise, a variant thereof – to shift the terms to be dropped to the front of both sides of the respective hypothesis or goal. This separation permits the respective terms to be eliminated by the use of suitable lemmata.

For the first of the two steps we need to modify the classical `make_first` a little bit. In $n+(m*l) = 42+l$, for instance, both occurrences of l can be shifted to the front but only the latter one can be modified to become the first operand of the outermost operation. So we need a tactic "predicate" `is_droppable` analogous to `is_firstable`. Technically, the former differs from the latter inasmuch as there is no nesting with the successor and predecessor function as well as no mix of addition and multiplication.

With `is_droppable` at hand the immunisation of the first droppable can be done just in the same way we immunised the first firstable. Now we can apply the tactic `make_first_equ_left_in_at_most_one_occurrence` since it works for every firstable term and so for every droppable term in particular.

For the second step, the elimination, all the different cases that may appear after shifting have to be taken into account. Let us consider two of them when the dropping is applied to a hypothesis.

```
...
  | n * ?k = n * ?j =>
      assert (n ≠ 0);
      [ (* ⊢ n ≠ 0 *)
        idtac
      | (* H' : n ≠ 0 *)
        cut (k = j);
        [ clear H H';
          prove_imp_name H (* H : k = j *)
        | use (n_mul_m_equ_n_mul_l H' H)]]
  | n ≠ n * ?k =>
      cut (1 ≠ k);
      [ clear H;
        prove_imp_name H (* H : 1 ≠ k *)
      | prove_imp_name H'; (* H' : 1 = k, ⊢ False *)
        mul_equ n in H'; (* H' : 1 * n = k * n *)
        use_equ (I_mul_n n) in H' at 1; (* H' : n = k * n *)
        use_equ (mul_comm k n) in H'; (* H' : n = n * k *)
        use_False (H H')]
...
```

In the first of both cases we need to know that $n \neq 0$. The assert construct requires a proof of this statement first, which is left via `idtac` to the user. In the second branch – where $n \neq 0$ is now assumed – we cut with $k = j$ to replace the original hypothesis by this new version ($n \neq 0$ is also cleared). The second branch of the second branch, which requires to prove $k = j$, is not visible for the user since it is closed by application of a suitable theorem using $n \neq 0$ and the original hypothesis. The second case is simpler since no additional precondition must be fulfilled. Again we use a cut to replace the original hypothesis. This happens in the first branch. The second is closed and therefore invisible to the user. Note here that $1 \neq k$ is defined as $1 = k \rightarrow$ False, which is why `prove_imp_name` can be used.

The tactic `drop` is not value preserving. So there is no automatic lifting to a goal version or inequalities. Hence for the overall tactic all of the following combining possibilities have to be considered: only addition or also multiplication, one or two operands on both sides, hypothesis or goal, and equation, negation thereof, the less equal, or the less than relation. Furthermore we need a variant of the tactic allowing to point at concrete positions where the corresponding term is supposed to be dropped. Fortunately, at least in the latter case further inflation can be avoided: the main tactics can be enriched by a parameter `is_at_at_version` while the true tactics are then realised as tactic notations:

```
Ltac drop_equ_in_meta make_first_droppable_equ_left_in_version
  make_first_droppable_already_executed is_at_at_version n H m l :=
let H' := fresh in let H'' := fresh in
lazymatch make_first_droppable_already_executed with
  true =>
    idtac
    (* This is helpful because it enables the recursive call
    below[16] *)
| false =>
    lazymatch is_at_at_version with
      true =>
        equ_in_at_at make_first_droppable_equ_left_in_version
          n H m l false
        (* If droppable, the mth n on the left and the lth n on the
        right hand side are shifted to the front *)
```

[16]The case $n * k \neq n$ can be treated by referring to the $n \neq n * k$ case.

```
    | false =>
        equu_in make_first_droppable_equ_left_in_version n H false
        (* The first droppable terms on both sides are shifted to
        the front. If one side has no droppable term it fails *)
    end
end;
...[17]

Tactic Notation "drop_equ" constr(n) "in" hyp(H) :=
drop_equ_in_meta
  make_first_droppable_equ_left_in false false n H 0 0.

Tactic Notation "drop_equ" constr(n) "in" hyp(H)
  "at" integer(m) integer(l) :=
drop_equ_in_meta make_first_droppable_equ_left_in
  false true n H m l.[18]
```

8.6.4 Evaluation

The tactic `drop` does a good job in simulating the respective reasoning step: the same steps are performed and the justification is as simple as this is possible in Coq since we cannot give no justification at all. Although the tactic does essentially build on `make_first` it does not share the same disadvantages: we can directly work with the terms we are interested in and – except for verifications of the form $n \neq 0$ for some n – we always only need one step for one dropping. In particular, there is no need to take advantage of invisible parentheses.

Nevertheless we can think of some improvements. So far the tactic is not able to handle situations like $n+m \leq l+2*n$, in which we would like to reach $m \leq l+n$ by dropping n. Furthermore a term must occur in a contiguous way to be dropped: in $n+5+m = (n+m)+l$ and in $n+m+5 = (n+m)+l$, for instance, $n+m$ cannot be dropped by the tactic yet.

Let us end the investigation of the `drop` tactic with a didactical remark. The implementation of the tactic needs consideration of many cases but in the application this can hardly be perceived. In other words, one can drop terms in the right way without understanding the reasons behind it. Perhaps this can be a part of an explanation why arithmetically trained pupils in school experience difficulties when they are facing proofs.

[17] At this point the elimination part as discussed above starts; in this case for hypotheses involving multiplication but with a restriction to equalities and negations thereof.

[18] We changed the position of some parts of the code due to layout reasons.

8.7 The implementation for the lifting of value preserving tactics

Up to now in the implementation of value preserving tactics we considered only versions for modifying the left hand side of hypotheses that are equations. In the given examples, however, the tactics were used to modify whole hypotheses and goals. Furthermore, there is no reason why such tactics should only work for equations and indeed they can be applied to negations of equations and inequations, too. In this section we discuss how tactics operating only on the left hand side of equations can be lifted canonically to the corresponding actual tactics.[19]

The key idea is the following. If we have some hypothesis or goal of the form $n\ R\ m$, with R being a binary propositional function, we create two new equalities, namely $n = n$ and $m = m$ as new hypotheses, modify their left hand sides, and use their equalities backward in the original hypothesis or goal. For the hypothesis case this is realised by the following meta tactic:

```
Ltac equ_in_modifications Tac_equ_left_in n H m l :=
let H' := fresh in let H'' := fresh in
let S := fresh in
(* H : m R l *)
first
[ fact_name (equ_refl m) H'; (* H' : m = m *)
  Tac_equ_left_in n H'; (* H' : m' = m *)
  fact_name (equ_refl l) H''; (* H'' : l = l *)
  try (Tac_equ_left_in n H'') (* H'' : l' = l *)
| fact_name (equ_refl m) H'; (* H' : m = m *)
  try (Tac_equ_left_in n H'); (* H' : m' = m *)
  fact_name (equ_refl l) H''; (* H'' : l = l *)
  Tac_equ_left_in n H'' (* H'' : l' = l *)];
set (S := m) in H at 1; (* H : S R l *)
use_uqe H'' in H; clear H''; (* H : S R l' *)
change S with m in H; clear S; (* H : m R l' *)
use_uqe H' in H at 1; clear H' (* H : m' R l' *).
```

The input `Tac_equ_left_in` is the respective tactic version operating on the left hand side only, n is the input of that tactic (if there is no such input a pseudo tactic for operating on the left hand side is previously defined, which ignores that input and hence behaves like the one without the additional parameter), and H is the hypothesis we want to modify stating that $m\ R\ l$. The tactic must be applicable at least to one side (possibly without changes). This is realised by

[19] As far as we know this technique is unprecedented (see section 8.11).

two alternatives given in the `first` construct. The first requires the left hand side tactic to be applicable to m while the second does this for l. In each of the alternatives the application to the other side is tried, too, but if that fails there are simply no changes made by `equ_in_modifications`. After the execution of one of the alternatives the old m and l are replaced by m' and l'. This requires some immunisation.

Next, there is a cumulative series of meta tactics called `equ_in`, `lequ_in`, and `less_in`, which conduct the `equ_in_modifications` tactic for all relevant relations. Since they are all constructed in the same way it suffices to consider the implementation of the last one:

```
Ltac less_in Tac_equ_left_in n H exactly_one_step :=
let H' := fresh in
let T := type of H in lazymatch T with
  ?m = ?l =>
    equ_in Tac_equ_left_in n H exactly_one_step
| ?m ≠ ?l =>
    equ_in Tac_equ_left_in n H exactly_one_step
| ?m ≤ ?l =>
    lequ_in Tac_equ_left_in n H exactly_one_step
| ?m < ?l =>
    fact_name H H'; (* H' : m < l *)
    equ_in_modifications Tac_equ_left_in n H m l; (* H : m' < l' *)
    change_happened_or_no_change_necessary H H' exactly_one_step;
    clear H'
| _ =>
    fail "The hypothesis must be of the form ... = ..., ... ≠ ...,
          ... ≤ ..., or ... < ..."
end.
```

The tactic `change_happened_or_no_change_necessary` checks whether a change is requested (the last parameter `exactly_one_step` is true) and – if so – whether H after the modification differs from its copy before the modifications.

The lifting of a concrete tactic is now realised by tactic notations. Let us consider the lifting of `drop_identities_lequ` as an example.

```
Tactic Notation "drop_identities_lequ" "in" hyp(H) :=
lequ_in drop_identities_equ_left_in_pseudo 0 H false.

Tactic Notation "drop_identities_lequ" :=
lequ drop_identities_equ_left_in_pseudo 0 false.
```

As already mentioned pseudo tactics like `drop_identities_equ_left_in_pseudo` are implemented to behave like the corresponding original left hand side tactic but with one additional input, here 0, without effect. The pseudo tactic has the right number of input arguments and hence can be applied within the `lequ_in` and `lequ` tactic. The `false` states that no changes are needed (since `drop_identities` is not a one step tactic).

Let it be briefly mentioned that there are variants of all the meta tactics permitting modifications of a particular occurrence or two particular occurrences of the term under discussion. This is again realised by immunisations. The tactics we want to use can then again be defined as tactic notations:

```
Tactic Notation "make_first_equ" constr(n)
  "at" integer(m) integer(l) :=
equ_at_at make_first_equ_left_in n m l true.
```

8.8 Where the higher degree of formalism hides

In sections 8.2-8.6 we presented some examples showing that reasoning in medium degree proofs can be simulated (most times adequately) by tactics in Coq. This implies that Coq proofs using such tactics should have advantages very close to the ones of medium degree proofs. Nevertheless there are no concessions regarding the guarantee aspect of formal proofs whatsoever. As it is the case with any other completed proof in Coq in the end a completely formal proof object has been constructed. It does not involve tactics but consists only of λ-expressions and references to applied theorems. For any theorem `example_theorem` this proof object can be made explicit by typing `Print example_theorem` in Coq. So we have a big win with respect to the illustration of ideas and with respect to the convincing power but no loss in objective guarantee. How is this possible?

Pattern matching is the key. It explicitly provides the context humans would subconsciously infer. However, prerequisite for this is that the programmer of the tactic considered all possible (and impossible) contexts, in which the tactic can(not) be applied, in advance. For instance in the case of the pre-version `drop_identities_add_equ_left_in` the possible contexts (with pattern variables n, m, and l) are $0 + n = m$, $n + 0 = m$, as well as Suc n, pred n, and $n + m = l$ if the first two are not applicable.

Now, the context allows the machine to follow a precise sequence of instructions using the context variables. Such sequences mainly consist of logical and equational rules, other tactics, the application of theorems, as well as immunisations. Some instructions may generate new situations not uniquely determined by the context

at hand. In such cases new contexts are needed and the programmer has to take care to anticipate them all. Technically this results in nested pattern matchings. Of course, for each nested context there is again a precise sequence of instructions.

So the answer to the question above is that we have an automatic transition from the medium degree version of a proof in Coq to one of higher degree, which is finally converted into a formal proof object. The former transition is possible because of

1. Coq internal procedures that allow us to use pattern matching
2. preparation work of the programmer anticipating all contexts and implementing their respective treatment.

8.9 Simulating medium degree reasoning in more complicated domains – a prospect

So far we focussed on arithmetic in our simulation of medium degree reasoning. This domain was chosen because it is already rich enough to gain decisive insights regarding the simulation techniques while it is not too complicated to start with. Let us now outline what simulation of more complicated domains may look like. We chose automata theory for this as an example since the simulation of medium degree reasoning here does create problems that are not due to problems in formalising automata theory per se.[20] In fact, automata theory has already been formalised; even in Coq (see chapter 9 for further details).

In arithmetic it sufficed to develop tactics modifying single hypotheses or goals. By contrast, in automata theory we often do not only change a single statement but a whole system of statements at once. The reason behind this behaviour is that in automata theory complete algorithms – instead of just single theorems – are applied.

Let us take a look at an example. If we define regular languages to be the languages described by regular expressions, a textbook proof of a medium degree of formality that regular languages are closed under complement could look like the one shown in figure 8.11; where 'RE', 'DFA', and 'NFA' stand for 'regular expression', 'deterministic finite automaton', and 'nondeterministic finite automaton' respectively.[21]

[20] In calculus, for instance, we would have the formalisation issue of how to formalise the real numbers.

[21] See [59] for further information about these concepts.

Theorem : For all regular languages L the complement $\overline{L}$ is also regular.
Proof : If L is regular then there exists a regular expression R with $L(R) = L$. We apply the transformation that generates an ε-NFA N with $L(N) = L(R) = L$. Now we convert this into a DFA $D = (Q, \Sigma, \delta, q_0, F)$ with $L(D) = L(N) = L$. We define the DFA $D' := (Q, \Sigma, \delta, q_0, Q\backslash F)$ and obtain $L(D') = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in Q\backslash F\} = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \notin F\} = \Sigma^*\backslash\{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\} = \Sigma^*\backslash L(D) = \overline{L(D)} = \overline{L}$. We can conceive the DFA D' as an ε-NFA N' with $L(N') = L(D') = \overline{L}$. Next we use the procedure for eliminating states to reach an RE R' with $L(R') = L(N') = \overline{L}$. Hence $\overline{L}$ is regular if L is. Qed.

Figure 8.11: Textbook proof of the closure of regular languages under complements.

What is interesting for us in this proof are the transformations, for instance "the transformation that generates an ε-NFA N with $L(N) = L(R) = L$". This statement does not only explain that there is an algorithm for converting REs into ε-NFAs but applies implicitly a theorem stating the preservation of the original language. Therefore in our simulation a system of statements (R is an RE and $L = L(R)$) has to be replaced by a new system (N is an ε-NFA and $L = L(N)$) in a single step.

Let us now peruse this idea by comparing two Coq proofs for the closure of regular languages under complements. First, we will discuss a corresponding proof not using medium degree tactics (figure 8.10). Then, we will see what a proof with such tactics may look like (figure 8.11).

Up to and including the first `use_ex` nothing of particular interest happens. At that point then we want to switch to a DFA representation[22], which is realised in three steps. First we use the function `from_RE_to_DFA` and a theorem stating that this transition is correct, i. e. language preserving. Next, we call the DFA created by the function D in analogy to the textbook proof given above. The final substep is to relate L to D, which allows us to delete the now uninteresting relation of L to R.

The next three steps are devoted to the construction of the complement DFA and its incorporation into the current proof situation. Again we use a transition function, this time called `complement_DFA` and a theorem guaranteeing its correctness. As in the previous conglomerate of substeps we introduce a variable, here D', as a shortcut in analogy to the textbook proof. Finally, we use our previous results to relate L to the most actual representation; this time D'.

[22] A path through an ε-DFAs would bring no additional insights. The two single paths can be composed into a new single one.

Next step in Coq	Proof situation
Initial situation	$\forall\ L : \text{Lang}, \text{is\_regular}\ L \rightarrow \text{is\_regular}\ !L$
prove_all_imp_star	$\boxed{L : \text{Lang}}$, $H : \text{is\_regular}\ L$ ⊢ $\text{is\_regular}\ !L$
unfold is_regular in H	$H : \exists\ R : \text{RE}, \text{lang\_RE}\ R = L$ ⊢ $\text{is\_regular}\ !L$
use_ex H	$\boxed{R : \text{RE}}$, $H : \text{lang\_RE}\ R = L$ ⊢ $\text{is\_regular}\ !L$
fact (from_RE_to_DFA_correct R)	$H : \text{lang\_RE}\ R = L$, $H0 : \text{lang\_RE}\ R = \text{lang\_DFA}\ (\text{from\_RE\_to\_DFA}\ R)$ ⊢ $\text{is\_regular}\ !L$
set ($D :=$ from_RE_to_DFA R) in $H0$	$H : \text{lang\_RE}\ R = L$, $\boxed{D := \text{from\_RE\_to\_DFA}\ R : \text{DFA}}$, $H0 : \text{lang\_RE}\ R = \text{lang\_DFA}\ D$ ⊢ $\text{is\_regular}\ !L$
use_equ H in $H0$	~~$H : \text{lang\_RE}\ R = L$~~, $H0 : L = \text{lang\_DFA}\ D$ ⊢ $\text{is\_regular}\ !L$
fact (compl_DFA_correct D)	$H0 : L = \text{lang\_DFA}\ D$, $H : \text{lang\_DFA}\ D = !(\text{lang\_DFA}\ !\text{DFA}D)$ ⊢ $\text{is\_regular}\ !L$
set ($D' :=$!DFA D) in H	$H0 : L = \text{lang\_DFA}\ D$, $\boxed{D' := !\text{DFA}\ D : \text{DFA}}$, $H : \text{lang\_DFA}\ D = !(\text{lang\_DFA}\ D')$ ⊢ $\text{is\_regular}\ !L$
use_uqe $H0$ in H	~~$H0 : L = \text{lang\_DFA}\ D$~~, $H : L = !(\text{lang\_DFA}\ D')$ ⊢ $\text{is\_regular}\ !L$
fact (from_DFA_to_RE_correct D')	$H : L = !(\text{lang\_DFA}\ D')$, $H0 : \text{lang\_DFA}\ D' = \text{lang\_RE}\ (\text{from\_DFA\_to\_RE}\ D')$ ⊢ $\text{is\_regular}\ !L$

Next step in Coq	Proof situation
set ($R' :=$ from_DFA_to_RE D') in $H0$	$H : L = \,!$(lang_DFA D') $\boxed{R' :=$ from_DFA_to_RE $D' : \text{RE}}$ $H0 :$ lang_DFA $D' =$ lang_RE R' is_regular $!L$
use_equ $H0$ in H	~~$H0 :$ lang_DFA $D' =$ lang_RE R'~~ $H : \; L = !$(lang_RE R') is_regular $!L$
fact (equ_fct compl_lang H)	~~$H : L = \,!$(lang_RE R')~~ $H0 : \; !L = !(!$(lang_RE R')) is_regular $!L$
use_equ (compl_compl_lang (lang_RE R')) in $H0$	$\boxed{H0 : \; !L =$ lang_RE $R'}$ is_regular $!L$
unfold is_regular	$\exists \; R0 : \text{RE}, \text{lang\_RE } R0 = !L$
prove_ex R'	lang_RE $R' = !L$
switch	$!L =$ lang_RE R'
use $H0$	Qed

Figure 8.10: Proof for the closure of regular languages under complements in Coq not using medium degree tactics.

The path back to regular expressions is similar to the forward aggregations of steps. After that, however, we have to cope with an additional problem. So far our gained insights are formulated in terms of L; but our goal is stated in terms of $!L$, the complement of L. To bridge the gap both sides of the equation $L = !$(lang_RE R') have to be complemented. Furthermore we need a theorem stating that double complementation boils down to the original language.

Only the last four steps are dedicated to the goal. We unfold the definition of regularity of a language to show that R' is the RE we are searching for.

Working on this level is irksome. The steps managing the transformation of the RE to the DFA belong together and constitute only one real change. The same is true for the backward direction and the steps for construction and incorporation of the complement DFA. Furthermore in the latter case the conversion regarding the complement language should be included. In figure 8.11 we show what such a proof would look like.

Next step in Coq	Proof situation
Initial situation	$\forall\ L : \text{Lang}, \text{is\_regular}\ L \to \text{is\_regular}\ !L$
prove_all_imp_star	$\boxed{L : \text{Lang}}$ $\text{is\_regular}\ L$ $\text{is\_regular}\ !L$
unfold is_regular in H	$H : \exists\ R : \text{RE}, \text{lang\_RE}\ R = L$ $\text{is\_regular}\ !L$
use_ex H	~~$R : \text{RE}$~~ $H : \text{lang\_RE}\ R = L$ $\text{is\_regular}\ !L$
from_RE_to_DFA R	~~$D : \text{DFA}$~~ $H : \text{lang\_DFA}\ D = L$ $\text{is\_regular}\ !L$
make_complement_DFA D	~~$D' : \text{DFA}$~~ $H : \text{lang\_DFA}\ D' = !L$ $\text{is\_regular}\ !L$
from_DFA_to_RE D'	$\boxed{R' : \text{RE}}$ $\boxed{H : \text{lang\_RE}\ R' = !L}$ $\text{is\_regular}\ !L$
unfold is_regular	$\exists\ R : \text{RE}, \text{lang\_RE}\ R = !L$
prove_ex R'	$\text{lang\_RE}\ R' = !L$
use H	Qed

Figure 8.11: Proof for the closure of regular languages under complements in Coq using medium degree tactics.

Except for one `switch`, starting and ending of the Coq proof with medium degree tactics is the same as in the first version without such tactics. So the essential changes are in between. Obviously the second proof is condensed here significantly: it has only three steps left in this part. Each of these steps corresponds to a series of substeps in the previous proof. The first transition tactic, for instance, does not only replace R with a DFA D. It replaces what depends on R in the actual proof situation with a corresponding statement involving D, too. In this case lang_RE R has to be replaced by lang_DFA D. In case of the complementation tactic for DFAs it is the complementation of the equation lang_DFA $D = L$ that has to be conducted automatically. Note that the proof with the medium degree tactics also simplifies the justifications and makes them procedural in character.

The realisation of such medium degree tactics is anything but trivial. Instead of a single hypothesis or just the goal the whole actual proof situation has to be considered. Furthermore there are many relevant situations ((in)equality of languages, change of components like Q or F etc.) that have to be anticipated. However, the advantages are so obvious that developing such tactics should be worth the effort.

8.10 The simulation of low degree argumentations

So far we discussed the simulation of medium degree reasoning with the help of tactics to acquire the benefits of medium degree proofs (and systems) together with objective guarantee. A natural question reads as follows: can we extend the results of that project to argumentations having a low degree of formality? Such a goal sounds very ambitious but it only reflects a very natural interaction between user and computer: the former is the creator of ideas while the latter calculates that everything is fine.

First of all, we have to recognise that the required extension cannot be a continuation of our approach since tactics like the ones we have discussed so far are not suited for this job. Anticipating of and working with concrete contexts is essential for their usage. In low degree argumentations, however, the condensation is too big and the proof situations to imprecise to keep track of the contexts in advance.

A more appropriate candidate for the simulation of low degree argumentations might be auto tactics, since they do not need to anticipate any contexts. Instead they are based on heuristic strategies to autonomously search for solutions without further interference by humans. In chapter 2, however, we already discussed that the auto tactics are black boxes that do not have to be suited to simulate low degree reasoning steps and that – even if they are suited to do so – humans can use them in their proofs for medium or high degree steps nevertheless. So auto tactics cannot guarantee low degree argumentations but can they at least enable such reasoning?

Different proof assistants use auto tactics in different ways. In Coq there are auto tactics used to prove the actual goal or to modify it or the actual hypotheses. Some auto tactics like `auto` are general while others like `omega` are restricted to particular domains (in that case arithmetic). Usually, the auto tactics in Coq can be found in proofs side by side with those tactics that represent basic logical rules. Yet, this crude mix of low and high degree reasoning is no option when simulating low degree reasoning. The only way to circumvent this problem would be proof

scripts that only contain auto tactics and cuts. In that case the proof scripts would degenerate to declarative proofs. This might be fine but why then not use Isabelle/Isar or ACL2 instead, which are intended to work in this declarative fashion?

However, even these alternatives are not completely satisfying. First of all, the statements of theorems and — in the case of Isabelle/Isar – intermediate proof situations have to be written down in a high degree way. Second, it is far from clear that the respective auto tactics can handle the condensation level of low degree argumentations. In particular this may be the case when a rather simple idea turns out to be technically demanding. Finally, to simulate a low degree reasoning the machine must construct a proof object guaranteeing the success of the approach. Yet, it is a guarantee for the coherence of the argumentation that is needed instead.

8.11 Related work

Our approach to simulate medium degree reasoning is based on the tactic language L_{tac}, which is introduced by David Delahaye in [35]. Delahaye's motivation is to define smaller tactics directly in Coq, which before his work had to be done with the help of the programming language OCaml. He assesses the definition of tactics in Coq via the L_{tac} language as a middle way between Coq and programming languages. Of course, this reminds us of a medium degree of formality. Delahaye mentions advantages of proof scripts using tactics defined by the L_{tac} language, namely that they are more compact, more simple, more readable, and more maintainable (see [35]). This again, sounds a little bit like a characterisation of a medium degree of formality. He even states that pattern matching helps to handle the proof process, which in turn was our key explanation in section 8.8 for tactics being able to simulate medium degree reasoning. However, Delahaye applies his small tactics in concrete situations and even directly in proofs in the same way a lambda term is used as an anonymous function. So he uses the L_{tac} tactics rather as ad hoc devices. The kind of tactics we presented in this chapter is probably not what he had in mind.

In [64], an article in which a semantical foundation is given to the L_{tac} language, Wojciech Jedynak, Malgorzata Biernacka, and Dariusz Biernacki make some remarks in the same spirit as Delahaye. They asses atomic tactics, which are the ones corresponding to a single rule, as impractical due to their high level of granularity. Therefore the authors want to reason about Coq scripts using L_{tac} tactics. The tactics they employ, however, are just ad hoc ones trying to prevent repetitions for the user in practical situations. In particular, there are only aggregates of single steps but there is no consideration of tactics representing reasoning units.

With regard to the idea of using tactics to simulate medium degree reasoning most close to our approach is the work of Thomas Braibant and Damien Pous [18]. In this article tactics are used to manage parentheses, order, and identities. According to the authors proof scripts without such tactics are too verbose and to painful to write. On the other hand the authors state explicitly that they want small rewrite steps instead of a complete decision of a proposition, i. e. they want rewrite terms in an appropriate way instead of a black box solution of the complete actual goal. This sounds like medium degree of formality. There are further similarities to our approach. The authors do not want to change the kernel of Coq and welcome tactics that are written directly in Coq using the L_{tac} language. However, more than half of their code is still written in OCaml. Braibant and Pous use a presentation of proof scripts that involves the actual proof situation and is thus similar to ours. Furthermore they strive for simple usability as we do, too. From our point of view one disadvantage of Braibant's and Pous' approach is that the tactics comprise too many different kinds of reasoning steps. Furthermore the effects of rewriting are sometimes not predictable. So they cannot provide clear control. The biggest problem of their approach, however, is that it can only be applied to all value preserving tactics at best. Tactics like `drop` cannot be implemented by rewriting techniques alone and in domains other than arithmetic most reasoning steps – we expect – will not be value preserving. Having said that there are some advantages of their approach as opposed to ours. First of all, their approach is already implemented not only for natural numbers but more generally for algebraic structures. Second, their rewriting even works in (nested) arguments of arbitrary functions. In our case this is true only for the standard operations successor, predecessor, addition, subtraction, and multiplication. Finally, their realisation of the tactics works on a syntactical level. This should allow to reduce the extent of the tactics in favour of Coq theorems about such syntactical terms and their manipulations.

It should be noted that Schiller, whose PhD thesis we already discussed in section 2.7, makes a general point of criticism by which our approach is affected in particular: "The question of appropriate granularity is deferred to the author of the tactics." ([98, pp. 95-96]) We reply to this (or similar) criticism that our granularity level is indeed fixed for each tactic but that it is the same for all of the arithmetical tactics; and furthermore, that the level itself is not arbitrary but aims at corresponding to the steps actually conducted in most of (good) mathematics. We already said in section 2.7 that we appreciate the idea to bring different degrees of granularity and – more general – of formality into one proof but we doubt that an aggregation of elementary steps via auto tactics instead of reasoning units is the right way to pursue this goal.

As already mentioned in section 7.1, for each domain our approach of simulating medium degree reasoning can be seen as (the core of) a respective proof environment. Besides the more essential tactics this comprises a library in each case, too. So it makes sense to compare our proof environment for arithmetic with other Coq libraries and projects. In the standard library of the Coq proof assistant [28] natural numbers occur in two ways. First, there is a module Init, which is loaded every time Coq is started. In that module there are the two submodules Nat and Peano. In the former several functions are defined but no theorems are given. The latter contains some theorems but also new definitions. Second, there is a module named Arith (which has to be loaded), in which the submodule PeanoNat is the most relevant containing some definitions and many theorems. Both modules are very different in nature with respect to our proof environment. Already the selection of contents indicates that but the salient difference is the attitude towards proofs. There are no proofs to be found in [28] but one has to take a look at the source code [24] to find them. It turns out that only Coq standard tactics are used, i. e. one finds only a mix of basic rules and auto tactics. The auto tactics of an arithmetical nature are called ring, omega, field, and fourier. They are developed to solve arithmetical goals automatically in one step (see [12, subsections 7.4.1-7.4.4] for further explanation). All four tactics are implemented in OCaml. There is a file named Tactics.v where tactics are defined inside of Coq but all of them are neither of a proof environmental nature nor arithmetical.

What about libraries and projects outside the standard library? In [74] different math projects in Coq are listed.[23] For evaluating these projects with respect to similarities with our proof environment in the following we often refer to the respective source code; because we are interested rather in the methodology than the concrete contents. The mathematical components project [78] is about algebra and group theory. For their proofs they use Ssreflect[24], a very technically oriented variant of proving in Coq that is far away from our proof environment philosophy. C-CoRN [25], ForMath [44] and math-classes [77] are related projects (see [27]) trying to formalise different domains of mathematics in an efficient way. C-CoRN and math-classes use standard Coq tactics while ForMath uses Ssreflect. In C-CoRN there are some further tactics defined but these do not resemble ours at all. In math-classes natural numbers are seen as instances of type classes, the use of which can be seen as the essence of that project. The HoTT [60] library pursues a category theoretical approach using standard tactics with some modifications. Natural numbers are consequently conceived as a category. In the HoTT library there are tactics for rewriting modulo associativity showing once again that a medium degree treatment of parentheses is needed. In Gappa [79] and Flocq [16] exact real

[23] We discuss most but not all projects listed there.

[24] See [50] for further information.

arithmetic and floating point arithmetic are investigated. Both approaches are of a numerical nature and so target efficiency. Russell O'Connor formalised the Gödel-Rosser Incompleteness theorem in Coq (see [87] for the code and [88, Part I] for a delineation of this approach). He simply uses the standard Coq tactics and the Arith module of the standard library of Coq.

Closest to our approach regarding the concrete implementation of the environment for natural numbers is a part of the UniMath project [112]. First of all, a lot of lemmata are implemented in that part, many of them being similar or even equal to the ones we have proven. Next, there are many tactics for the handling of natural numbers. Yet, most of them seem to be auxiliary tactics such that in the end only five types of tactics remain. Three of them are rather simple and correspond to our tactics `basic_fact`, `contradiction`, and `exception` (see section B). Another type of tactic allows to permute three terms and is thus an alternative solution to dealing with commutativity and associativity. Very interesting is the last type of tactic, which allows – if possible – to regroup a goal such that a given term is then a subterm of the goal. This could be the basis for simulating the consideration of certain parts of a term in other tactics. However, there is a salient difference to our approach. The tactics in the UniMath project remain on a technical level. They do not replace the non-logical standard Coq tactics in the proofs of the theorems[25]; and much less are they built to be used in a proof environment by non Coq experts.

As a last point we want to mention that we believe our lifting technique (see section 8.7) – though not far to seek – to be unprecedented. The technique can be seen as one of rewriting and so relevant literature might be found there. However, all of the rewriting topics we are aware of are very different from ours in character. For instance Jacek Chrząszcz and Daria Walukiewicz-Chrząszcz present a vision in [22] that amounts to bringing all relevant rewriting rules directly into the definition of a function (like addition for an example). Another technique called generalized rewriting is presented in [10] by David A. Basin. In that article rewriting is extended to arbitrary relations (of course under some side conditions).[26] Such a possibility can help when dealing with integers (or rationals) of the same value but which are not equal in the sense of Coq.[27]

[25] In fact, they do not seem to be used at all.

[26] Another important aspect of that article is the automation of goals that are generated when one tries to rewrite something in Nuprl. Since this problem and any way to solve it are idiosyncratic to the Nuprl system we do not elaborate on this further.

[27] There are no quotient types in Coq.

8.12 Summary and conclusion

In this chapter we presented a way for using self written tactics in Coq to simulate medium degree reasoning. The goal of this simulation was to reach the same (or nearly the same) magnitude as medium degree proofs regarding the emphasis on ideas and regarding the convincing power while the objective guarantee should soar to maximum. Since we worked in Coq the last requirement was fulfilled automatically while success in the first two aspects depended on the quality of the simulation.

The developed tactics are – most of the time – able to condense a bunch of single high degree steps into one medium degree step. Since the tactics use pattern matching to infer the actual contexts it is not necessary to give them much input. Instead their inputs are few in number, intuitive, and usually rather short. The tactics are named after the process they execute. Taken together the last observations show that the applications of these tactics are similar to the justifications of medium degree proofs.[28]

Summarised two of the three most important aspects of formalisms, condensation and justification, can be simulated quite well by the given tactics. Therefore we can expect our simulation to have an emphasis on ideas and a convincing power that is at least close to real medium degree proofs. The third adjusting parameter, the partial use of natural language, has not been addressed so far. Its realisation is independent to our approach.[29]

So the simulation of single reasoning steps in the field of arithmetic is a clear success. Yet, it is only a part of the greater picture. We aim at transferring our approach to a more complicated case more or less on a par with most other fields of mathematics: automata theory. Success with this endeavour would point to a general feasibility of our approach.

[28] As discussed, something we cannot simulate is when medium degree proofs give no justification at all for some reasoning.

[29] The use of natural language is addressed, for instance, by Zinn [119] and Fiedler [39, 40].

Chapter 9

Summary and conclusion

In the first part of this thesis we introduced and analysed the concept of the degree of formality; for systems as well as for argumentations. We gave a lot of examples of different degrees for both, beginning in the introduction and then again in sections 2.4 (systems of Peano arithmetic) and 2.5 (argumentations for the commutativity of natural numbers). The discussion of the examples in chapter 2 already used a previous analysis, in which we found the use of natural language, condensation, and frequency and precision of justifications[1] as adjusting parameters for the degree of formality. Furthermore, in chapter 2 we pointed out that proof scripts of a particular kind can be seen as proofs.

Next, we discussed the work of Curry, which turned out to be decisive in the subsequent chapters. First, his treatment of logics as independent systems invited us to think of a system as consisting of a contentual and a logical part. We called that the logical division. In chapter 4 we extended the logical division by a formal one such that systems become a quadruple (s, l, k, i) where s is the subject-specific content, l is the logical content, k is the kind of formalism, and i is the concrete formal instance. We noticed that the degree of formality can be seen as the value of a degree function taking kinds of formalism as arguments. This and the resemblance between logics and formalisms, which we demonstrated in the following sections, helped us to further localise the concept of the degree of formality.

Second, Curry's split into an objective and a subjective criterion for truth influenced our proceeding in chapter 5 in two ways: it motivated our search for the benefits of the different degrees of formality and led to the concrete listing of the purposes of argumentations. Regarding the former aspect, one should bear in mind that Curry's division had led him to ask what formal systems are acceptable for what reasons, which is very similar to our question of the different benefits

[1] The last one, of course, is for argumentations only.

of the different degrees of formality. With respect to the purposes note that the convincing power can roughly be seen as an equivalent to acceptability and the objective guarantee as one to provability. However, the illustration of ideas and the names of the identified purposes were rather influenced by the other literature we discussed in chapter 5. Once the question of the benefits was asked and the catalogue of purposes was at hand the answer could be found via an analysis of the ingredients, i. e. via checking the benefits of the three adjusting parameters. It became apparent that low degree argumentations are the best regarding ideas, that medium degree proofs are the most convincing, and that high degree proofs are the best with respect to objective guarantee. An even more important result, however, was that medium degree proofs are the most balanced argumentations, i. e. in contrast especially to high degree systems their performance in their non special purposes of argumentations is still OK.

In the second part of the thesis we demonstrated the usefulness of the concept of the degree of formality. First, we showed that the extended pluralistic view is able to avoid three points of criticism that can be directed at Curry's view of mathematics. The critical part, however, was to show that the essence of Curry's position, i. e. the harmonisation of objectivity and subjectivity to justify pluralism in mathematics, is not lost by admitting systems and argumentations of all degrees of formality. We argued that lower degree argumentations are not futile or even harmful with respect to the objectivity aspect but that they contribute further and partially in a manner higher degrees cannot. In particular we were able to justify Curry's use of meta argumentations to show different kinds of formalisms to be equivalent.

Our next application example was of a didactical nature. We wanted to teach the students how to prove. Since we assumed that difficulties with proving are not due to the formal aspects but due to a mix of formal and informal aspects we had an unusual approach: we used the Coq proof assistant to teach students how to develop high degree proofs first. The idea then was that the students should manage the transition to textbook proving by stepwise reduction of the degree of formality. To find suitable in between proof styles we used our knowledge of the adjusting parameters. The transition to line by line comments was mainly characterised by a leap in natural language, the one to the weakened line by line comments by condensation while the transition to structure faithful proofs was characterised by all three adjusting parameters. First results indicate that our approach for learning how to prove is a significant help for many – but not all – students.[2]

[2] To avoid confusion we want to mention again that the success of our approach can hardly be compared to any of those approaches that are not about medium degree proofs. Regarding high degree proving in Coq the exam results were close to 100% of the maximum, clearly indicating that this is not the problematic part.

Finally, we showed – as a proof of concept[3] – that it is possible to simulate medium degree proofs (except for the natural language aspect) with the help of user defined tactics in Coq. In our view this simulation possibility amounts to the core of future proof environments and makes our approach very distinct from a mere library. Regarding the latter point note that the most time consuming aspects of the development of (the core of) the proof environment in arithmetic were the initial training with Coq, the development of a suitable style for implementing the tactics, and finding such an arrangement of the theorems that their respective proofs consist almost entirely of applications of tactics already implemented at the respective place. By contrast, to prove all the theorems of the corresponding library (somehow) was not too big a challenge.

So we have found three examples of very different kinds from different areas, in which the concept of the degree of formality and our corresponding analysis thereof turned out to be of great help. By contrast, without such a concept, i. e. with a dualistic attitude of formal and informal proofs (or argumentations) as described in the introduction of this thesis, we would not only have been unable to find solutions to these three examples but we would not even have recognised the examples as problems that are in need of a solution: Curry's pluralistic view of mathematics and so any attempt to improve it would have been of no interest to us, we would have seen no need to transfer from Coq proofs to "the" informal proof style, and every thought of a proof environment would have been restricted to auto tactics, which are not suitable for that job.[4] All this is clearly enough to demonstrate the usefulness of the concept of the degree of formality.

However, the true result of this thesis is neither the usefulness of any of the shown case studies nor the analysis we did before but the demonstration that seeing the scientific world through the eyes of the degrees of formality, is a very helpful attitude. Who assumes it can spot and solve problems (from very different areas) hidden so far, gaining something that might vary significantly from case to case.

Also for us there is no reason to stop with the three case studies. One of our future research directions will again pertain to the philosophy of mathematics. We want to investigate the usage of lower degree systems and argumentations where corresponding higher ones already exist and where the latter were once needed to solve problems of the earlier lower degree versions. In concreto, we

[3] This restriction has two dimensions. First, other tactics or other tactic behaviour might be more suitable for the simulation of medium degree proofs. Future empirical investigations can close this gap. Second, we treated only the domain of arithmetic so far, which we argued to be a suitable choice to begin with.

[4] Otherwise there would already be proof environments used by most mathematicians.

want to compare this situation with what is called second nature, meaning rituals, attitudes of morality being independent of biology[5] etc., which all evolved by third nature, i. e. by rational considerations.

In the didactical field we want to collect more data to improve our evaluation of our approach for teaching students how to prove. To reach more of the students we plan a video series (in English) for an introduction to Coq in the way we use it. Furthermore we want to gradually test out at what age Coq in the way we use it is introduced best. Will it also help soon-to-be freshmen or even pupils? Finally, in the long run we want, of course, to use our then existing proof environment for automata theory (see below) in teaching.

Most of our future work, however, will pertain to the area of medium degree tactics. First of all, there are many minor projects. Lifting our approach for the natural numbers to integers and rational numbers is the first of them. Then we want to test our medium degree tactics for different basic but famous theorems and in different fields like the theory of primes or initial segments of natural numbers. Next, we want to extend the present medium degree tactics by ones for concrete calculations, dealing with different representations for the same value like in case of Suc n and $n + 1$. Furthermore it would be of much help to have a tool that automatically generates TEX-code for the proof scripts in the way we presented them.

The main step in the near future will be the development of (the core of) a proof environment for automata theory, i. e. the development of suitable medium degree tactics. As in the case of arithmetic this must not be confused with a mere formalisation, which for automata theory already began in 1986 (see [69]) and which has already been developed in Coq (see [41, 4, 82]) and the Ssreflect extension of it (see [36]).[6] Of how much help these and other existing implementations will be is difficult to assess in advance. Regarding the proof environment we have already indicated that the simulation of medium degree reasoning in automata theory will be more complex than in the case of arithmetic. Yet it might be simpler to implement than other domains of mathematics; for automata theory is constructive – and therefore fits Coq's logic – and operates on structures, which can be represented easily by the datatypes in Coq.

On the very long run we want to develop a proof environment for ZF set theory and the meta-argumentations for it. The latter would, for instance, enable one to formalise forcing.[7] We guess that problems here might be due to the different

[5] Other attitudes of morality pertain to first nature instead.

[6] Even substantial parts of the theory of context-free languages have already been formalised (see [8, 9]) and again this has also been done in Coq, too (see [92]).

[7] An gentle introduction to forcing is provided by Timothy Chow in [21].

levels of reasoning, the different logic, and the different concepts of functions.[8] In particular there must be some kind of simulation implemented in or outside of Coq such that the reasoning in and about ZF will nevertheless look like the normal reasoning in Coq.

[8] For instance, a function f in ZF with domain x and range y is itself of type set, not of type $x \to y$.

Part III

Appendix

Appendix A

Listing of logical and equational tactic rules

Logical tactics are tactics that deal with logical connectives independently from any logic-free content. Equational tactics deal with equations or negations thereof and are also content-independent. The logical and equational tactics we use in this thesis and in the corresponding code differ from the built-in ones Coq is providing.[1] Therefore in this appendix we list (most of) the logical and equational tactics implemented by us. To be more precise, we discuss the behaviour of each such tactic when it is applied together with the expected input. In each case a first hint of the respective behaviour is already given by the tactic's name (for instance `use_and`). Furthermore, we state the exact result of applying the tactic, provided that the preconditions that we have stated before are satisfied.[2] With respect to the results we only refer to the changes while everything else will stay the same. If there is a close link to some built-in tactic of Coq this will be also mentioned.

Let us briefly discuss the kind of tactics we are not considering in this appendix. First, we do not list tactics, like `use_true` and `prove_or`, that we only created to catch particular user mistakes. Second, we do not discuss the '`_name`'-variations like `prove_imp_name`, which exist for some tactics that generate a new hypothesis. In the respective variant the new name is given explicitly. Tactics that have restricted scopes like `use_equ_left` are also not considered. Next, we do not list fixed finite repetitions of tactics like `prove_or9` or `use_all3`. Furthermore, we do not list variations that are developed to prevent unnecessary hypotheses: the tactics `use_and_first`, `use_all_and_clear`, and `clear_all` are instances of that kind. Then, we do not list tactics having an arithmetical extension like `basic_fact` or `contradiction` since these are listed in the subsequent appendix. We do not list

[1] There are only two built-in tactics we use in the proofs, namely cut and assert.

[2] If the preconditions are not satisfied the respective application of the tactic will in most cases lead to a suitable, user defined error message but we will not discuss that here.

tactics belonging to path induction since those tactics are not relevant regarding the topics of this thesis. Finally, we do not list tactics that are only supposed to be used in the definition of other tactics.

- `use` H:
  - Preconditions: H has to be a hypothesis or theorem. The type of H must be the goal.
  - Result: The actual goal is proven.
  - Relation to built-in tactics: The tactic is a synonym for `exact`.
- `use_and` H:
  - Preconditions: H has to be a hypothesis and the type of it must be a conjunction.
  - Result: The first conjunct is the type of a fresh hypothesis H' while the second conjunct is the new type of H.
  - Relation to built-in tactics: The implementation uses `elim`, which is more general.
- `use_and_star` H:
  - Preconditions: H must be a hypothesis and its type must be a proposition.
  - Result: If the type of H has the form $P_1 \wedge P_2 \wedge \ldots P_n$ (where P_n is not a conjunction) then $n-1$ new hypotheses for the first $n-1$ conjuncts are generated while the last conjunct is the new type of H.
- `use_or` H:
  - Preconditions: H has to be a hypothesis or theorem. The type of it must be a disjunction.[3]
  - Result: The proof task is replaced by two new ones having the same goal. In the first of them H states the first disjunct of the original type of H while in the second H states the second disjunct.
  - Relation to built-in tactics: The essential part of the implementation uses `cases`.

[3] We do not only permit to use $\vee$ but also the constructive or, which is denoted by $+$.

- `use_or_star` H:
  - Preconditions: H has to be a hypothesis or theorem and its type must be a proposition.
  - Result: If the type of H is of the form $P_1 \vee ... \vee P_n$ (where P_n is not a disjunction) the proof task is replaced by n new ones all having the same goal. In the mth proof task the type of the hypothesis H is replaced by the P_m.
- `use_imp` H H':
  - Preconditions: H has to be a hypothesis having an implication $P \to Q$ as its type where P and Q have to be propositions.[4] H' has to be a hypothesis or theorem of type P.
  - Result: The type of H is replaced by Q.
  - Relation to built-in tactics: The implementation uses `apply`.
- `use_iff` H:
  - Preconditions: H has to be a hypothesis and the type of it must be a logical equivalence.
  - Result: The forward direction is the type of a fresh hypothesis H' while the backward direction is the new type of H.
- `use_False` H:
  - Preconditions: H has to be a hypothesis or theorem. The type of H has to be False.
  - Result: The actual goal is proven.
- `use_not` H H':
  - Preconditions: H has to be a hypothesis while H' has to be a hypothesis or theorem. The type of the former has to be the negation of the type of the latter.
  - Result: H is replaced by False.

[4] The addendum after the 'where' is necessary since '$\to$' – in contrast to $\wedge$, $\vee$, $\neg$, and $\leftrightarrow$ – is not defined exclusively between propositions.

- `prove_and`:
  - Preconditions: The goal must be a conjunction.
  - Result: The two conjuncts have to be shown separately.
  - Relation to built-in tactics: The tactic uses essentially `split`, which is more general.
- `prove_and_star`:
  - Preconditions: The goal must be a proposition.
  - Result: If the goal is of the form $P_1 \wedge P_2 \wedge \ldots \wedge P_n$ (where P_n is no conjunction) n new goals, one for each conjunct, are created.
- `prove_or_left`:
  - Preconditions: The goal must be a disjunction.
  - Result: The goal is replaced by its left hand side.
  - Relation to built-in tactics: The tactic executes the tactic `left`.
- `prove_or_right`:
  - Preconditions: The goal must be a disjunction.
  - Result: The goal is replaced by its right hand side.
  - Relation to built-in tactics: The tactic executes the tactic `right`.
- `prove_imp`:
  - Preconditions: The goal must be an implication between propositions.[5]
  - Result: The goal is replaced by the conclusion of the implication while the premise is the type of a new hypothesis.
  - Relation to built-in tactics: The tactic essentially executes `intro`.
- `prove_iff`:
  - Preconditions: The goal must be an equivalence.
  - Result: The goal is replaced by two new ones, the first being the forward and the second being the backward direction.

[5] The reason for the addendum 'between propositions' is the same as in the case of `use_imp`.

- `prove_True`:
  - Preconditions: The goal must be True.
  - Result: The goal is proven.
- `prove_not`:
  - Preconditions: The goal must be a negation.
  - Result: There is a new hypothesis stating the interior of the negation while the goal is replaced by False.
- `use_all` H t[6]:
  - Preconditions: H has to be a hypothesis of the form $\forall\ x : T,\ P$ while the type of t must be T. If H quantifies over the type Type then the type of t is also allowed to be Set.[7]
  - Result: A new hypothesis is generated stating $P[t/x]$.
- `use_ex` H:
  - Preconditions: H has to be a hypothesis or theorem of the form $\exists\ x : T,\ P$.
  - Result: A new arbitrary but fixed variable $x' : T$ is introduced while H is changed into $P[x'/x]$.
  - Relation to built-in tactics: The tactic uses `elim` for the decisive part of the implementation.
- `prove_all`:
  - Preconditions: The goal must have the form $\forall\ x : T,\ P$.
  - Result: A new arbitrary but fixed variable $x' : T$ is introduced while the goal is changed into $P[x'/x]$.
  - Relation to built-in tactics: The tactic is essentially `intro`.

[6] This tactic is implemented for didactical purposes. In our implementation we prefer to use fact (see below).

[7] Type, Set, and Prop are special types in Coq.

- `prove_ex` t:
  - Preconditions: The goal must have the form $\exists\ x : T,\ P$ while the type of t must be T.
  - Result: A new arbitrary but fixed variable $x' : T$ is introduced while the goal is changed into $P[x'/x]$.
  - Relation to built-in tactics: The tactic is essentially `exists`.
- `prove_all_imp_star`:
  - Preconditions: The goal must be a proposition.
  - Result: If the goal is of the form $\forall\ x_1 : P_1,\ \ldots,\ \forall\ x_n : P_n,\ Q$[8] (where Q is not a universal quantification) new arbitrary but fixed variables $x'_i : P_i$ are introduced while the goal is changed into $Q[x'_1/x_1, \ldots, x'_n/x_n]$.
  - Relation to built-in tactics: The tactic behaves similar to `intros`.
- `fact` H:
  - Preconditions: H must be a lemma or a hypothesis.
  - Result: A new hypothesis stating the type of H is created.
  - Relation to built-in tactics: The implementation makes use of the tactic `generalize`.
- `prove_by_induction`:
  - Preconditions: The goal must have the form $\forall\ x : T,\ P$ where T has to be an inductive type.
  - Result: In accordance with the T_ind function Coq is providing new hypotheses and goals are generated.
  - Relation to built-in tactics: The tactic is a restricted version of the `induction` tactic provided by Coq.

[8] Note that every implication is also a universal quantification.

- `prove_by_structure`:
  - Preconditions: The goal must have the form $\forall\ x : T,\ P$ where T has to be an inductive type.
  - Result: For each constructor in the definition of T a new proof task is generated. In each task and for each input of the constructor there are arbitrary but fixed variables. In addition, there is an arbitrary but fixed variable x' of type T for each constructor. The goal in each of the respective tasks is $P[x'/x]$.
  - Relation to built-in tactics: The tactic is a restricted version of the `destruct` tactic provided by Coq.
- `use_equ` H (in H') (at n)[9]:
  - Preconditions: H must be a theorem or hypothesis of the form $a = b$ while in the variant with H' the latter must be a hypothesis. If the place n is given as further input there must be n occurrences of a in the goal (or in the hypothesis H' respectively) and the nth occurrence must be replaceable.[10]
  - Result: In the version without 'at' all replaceable occurrences of a in the goal (or in the hypothesis H' respectively) are replaced by b. In the at-version only the nth occurrence is replaced that way.
  - Relation to built-in tactics: The tactic is a restricted version of the `rewrite` tactic provided by Coq. However, it does not use the latter tactic in its implementation.
- `use_uqe` H (in H') (at n):
  - Preconditions: H must be a theorem or hypothesis of the form $a = b$ while in the variant with H' the latter must be a hypothesis. If the place n is given as further input there must be n occurrences of b in the goal (or in the hypothesis H' respectively) and the nth occurrence must be replaceable.
  - Result: In the version without 'at' all replaceable occurrences of b in the goal (or in the hypothesis H' respectively) are replaced by a. In the at-version only the nth occurrence is replaced that way.
  - Relation to built-in tactics: The tactic is a restricted version of the `rewrite` tactic written with a backward arrow. However, it does not use the latter tactic in its implementation.

[9] Here and in the following parantheses express optional arguments.
[10] There can be problems with dependent types which is why this addendum is necessary.

- `prove_equ`:
  - Preconditions: The goal must be of the form $a = a$.
  - Result: The subgoal is proven.
  - Relation to built-in tactics: The tactic behaves like the `reflexivity` tactic Coq is providing but it does not use the latter tactic in its implementation.
- `switch` (in H):
  - Preconditions: The goal (or H respectively) must be an equation or a negation thereof. In the version with H the latter needs to be a hypothesis.
  - Result: The sides of the goal (or of the hypothesis H respectively) are switched.
- `begin_work_on_term` t (in H):
  - Preconditions: t must be a term occurring in the goal (or the hypothesis H respectively).
  - Result: The term t is immunised in the goal (or in the hypothesis H respectively) by some variable S. A new hypothesis is generated stating $S = t$.
- `end_work_on_term` t in (H) H':
  - Preconditions: In the goal (or in H respectively) there must occur some immunisation variable S and H' must be of the form $S = t'$.
  - Result: All occurrences of S in the goal (or of the hypothesis H respectively) are replaced by t'. The immunisation variable and the hypothesis H' are deleted.

Appendix B

Listing of medium degree tactics in arithmetic

In the following we list all the medium degree tactics we implemented so far. For every tactic we give a rough description at first.[1] A precise formulation of the preconditions follows. Finally, we state the exact behaviour of the tactic when the preconditions are satisfied. Note that – as in the case of the logical and equational tactics – we do not give any specifications in case the preconditions are not satisfied.

Before we start we have to explain respectively introduce some terminological aspects. First, – as in the case of the equational tactics – parentheses after the name of the tactic state the interior to be optional. n, m etc. are always natural numbers. We call $=$, $\neq$, $\leq$, and $<$ *basic relations*. $=$, $\leq$, and $<$ are also called *positive basic relations*. The type of a hypothesis or a goal that has the form $n\ R\ m$ for some (positive) basic relation R is called *(positively) basic*. In the former case the hypothesis itself is then called a (positively) basic hypothesis. Furthermore, we call the operations $+$, $*$, $-$, Suc, and pred *basic*, too. Next, a subterm of a term is called *reachable* by a set of operations if the subterm coincides with the term or if the latter is an application of one operation of the set such that the subterm is reachable with respect to one of the operands. If a term is reachable by the set of basic operations we call the term *basically reachable*. When we speak of the theorems we are providing we mean the theorems proven in our natural numbers library.[2] If R is a basic relation then $R_{\mathbb{B}} \in \{=_{\mathbb{B}}, \neq_{\mathbb{B}}, \leq_{\mathbb{B}}, <_{\mathbb{B}}\}$ is meant to be the corresponding boolean relation. Finally, for the sake of readability we will often speak of the equation of H, the left hand side of H etc. where we actually mean the equation of the type of H, the left hand side of the type H etc. The respective context will leave no room for misunderstandings.

[1] In contrast to the case of logical and equational tactics the names of the arithmetical tactics cannot be suggestive enough anymore to fulfil this job.

[2] Bear in mind that the medium degree tactics can be seen as the core of a proof environment, which in particular comprises a library.

- `trans` H H': The first and second statement are transitively linked.
  - Preconditions: H, H' are positively basic hypotheses or theorems such that the right hand side of H coincides with the left hand side of H'.
  - Result: A new hypothesis H'' is generated relating the left hand side of H to the right hand side of H'. If $<$ is the basic relation in H or H' then it is also the basic relation in H''. Otherwise, if $\leq$ is the basic relation in H or in H' then this is the case in H'', too. If H and H' are equations then H'' is also an equation.
- `basic_fact`: Completes a proof when a known basic fact remains to be proven.
  - Preconditions: The actual goal is an atomic proposition or a negation thereof. Furthermore the goal must be an instance of a theorem we are providing that has the form $P_1 \dots P_n G'$ where G' is an atomic formula or a negation thereof and where each P_i is either a universal quantification over natural numbers or an implication in which the premise is a restriction excluding finitely many natural numbers (like $n \neq 0$ or $n \geq 2$).
  - Result: The actual goal is replaced by subgoals for all restrictions the P_i are representing. If there is no restriction the actual goal is proven.
- `contradiction`[3] H: Completes a proof when a contradiction is reached.
  - Preconditions: H is an atomic proposition or a negation thereof. If H is a negation there must be a theorem we are providing of the form $P_1 \dots P_n H'$ where H' is an atomic formula, where each P_i is either a universal quantification over natural numbers or an implication in which the premise is a restriction excluding finitely many natural numbers, and where H is the negation of an instance of H'. If H is an atomic proposition it is the same except for the circumstance that H' has to be a negation and that H is an instance of the interior of H'.
  - Result: The actual goal is replaced by subgoals for all restrictions the P_i are representing. If there is no restriction the actual goal is proven.

[3] The tactic is actually called `my_contradiction` in the code due to a name conflict with a built-in tactic.

- `change_to_bool` (in H): Basic relations are converted into their corresponding boolean versions.
  - Preconditions: The actual goal (or the hypothesis H respectively) must be basic.
  - Result: If the actual goal (or the hypothesis H) is of the form $n\ R\ m$ for $R \in \{=, \leq, <\}$ then it is changed into $(n\ R_{\mathbb{B}}\ m) = \text{true}$.[4] If the goal (or the hypothesis H) has the form $n \neq m$ then it is changed into $(n =_{\mathbb{B}} m) = \text{false}$.
- `change_to_prop` (in H): The boolean versions of basic relations are converted into their corresponding basic relations.
  - Preconditions: The actual goal (or the hypothesis H respectively) must be of the form $(n\ R_{\mathbb{B}}\ m) = \text{true}$, where $R \in \{=, \leq, <\}$, or it must be of the form $(n =_{\mathbb{B}} m) = \text{false}$.
  - Result: If the actual goal (or the hypothesis H) is of the first form it is changed into $n\ R\ m$. If it is of the second form it is replaced by $n \neq m$.
- `prove_by_components_equ`: Decomposes equality with respect to the involved operation.
  - Preconditions: The goal must be of the form n op $m = n'$ op m' where op is a binary basic operation.
  - Result: The actual goal is replaced by two new ones, namely $n = n'$ and $m = m'$.
- `suc_pred_to_front` (in H): A Suc or pred is brought one step to the front.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic. At least at one side of the goal (or the hypothesis H respectively) it must be possible to extract a Suc or pred, directly or in some basic subterm. This may require to fulfil some further constraint.
  - Result: If there is the possibility to bring a Suc or pred to the front (including the elimination cases) without a further constraint on both sides of the goal (or the hypothesis H respectively) this is done. If this is only possible on one side this is done there and the other side stays the same. If this is not achievable, neither, the tactic brings Suc or pred to the front on the left or the right hand side, the left being prioritised, creating a new subgoal for the cut that was used. If

[4] In the actual code $\mathbb{B}$ is not a subscript character.

there are multiple possibilities to bring a Suc or pred to the front the following prioritisation is used: first the outermost possibility, then the left subterm, then the right one. If the same term facilitates multiple shiftings Suc is preferred before pred and the first operand before the second one.

- `drop_identities` (in H): Drops the identities and multiplications with 0.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic.
  - Result: In the terms on the left and the right hand side of the goal (or the hypothesis H respectively) all basically reachable 0-summands, 0-subtrahends, 1-factors, and multiplications containing a 0 are dropped. Furthermore, subtractions with a 0-minuend are evaluated to 0. Apart from all that the terms stay the same.
- `omit_parens` (in H): All unnecessary parentheses are dropped.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic.
  - Result: In the terms on the left and the right hand side of the goal (or the hypothesis H respectively) all parentheses in basically reachable subterms are – if possible – omitted. Apart from that the terms stay the same.
- `make_first` n (in H) (at m (l)): The term n is shifted to the front.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic. Furthermore, when no 'at' is used there must be a firstable occurrence of n, i. e. an occurrence which is reachable by the basic operations Suc, pred, +, and *, on at least one side of the goal (or of the hypothesis H respectively). If the single at-variant is used the mth occurrence in the goal (or the hypothesis H respectively) must be firstable. In case of the double at-variant the mth occurrence on the left hand side and the lth occurrence on the right hand side must be firstable.
  - Result: When no 'at' is used the first firstable occurrence of n is shifted to the front on both sides if possible. If there is a side on which there is no firstable n this side stays the same. In the single at-version the mth occurrence of the goal (or the hypothesis H respectively) is shifted to the front on the respective side while the other side does not change. In case of the double at-variant the mth occurrence on the left hand side and

the lth occurrence on the right hand side are shifted to the front. In all variants the order and the parentheses of the remaining terms survive as far as this is possible. Furthermore, all shifted n are isolated as much as possible, i. e. they become the whole first outermost operand if possible, the whole first outermost operand of the first outermost operand if this is not possible etc.

- `add` n (in H): The term n is added as a summand at the end.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic.
  - Result: An n is added as a summand at the end of the left and the right hand side of the goal (of the hypothesis H respectively). Note however, that no calculations are done.
- `sub` n in H: The term n is subtracted at the end.
  - Preconditions: The hypothesis H must be of the form $m\ R\ l$ with $R \in \{=, \leq, <\}$.
  - Result: H is changed into $m - n\ R\ l - n$ but in case of $R = <$ a new subgoal, namely $n \leq m$, is also generated.
- `sub_from` n in H: The type of the hypothesis H is subtracted from n.
  - Preconditions: The hypothesis H must be of the form $m\ R\ l$ with $R \in \{=, \leq, <\}$.
  - Result: If R is the equality relation then H becomes $n - m\ =\ n - l$. If $R = \leq$ then H is changed into $n - m \geq n - l$. In the last case ($R = <$) the side condition $m < n$ has to be shown first before H is replaced by $n - m > n - l$.
- `mul` n (in H): The term n is added as a factor at the end.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic.
  - Result: An n is added as a factor at the end of the left and the right hand side of the goal (of the hypothesis H respectively). However, in the following variants $n \neq 0$ has to be proven before that: the goal is of the form $m = l$ or $m \leq l$ (H is of the form $m \neq l$ or $m < l$ respectively).

- `drop` n (in H) (at m l): The term n is dropped on both sides.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic. When no 'at' is used there must be a droppable occurrence of n, i. e. an occurrence which is reachable by addition only or by multiplication only, on both sides of the goal (or of the hypothesis H respectively). Furthermore, the first droppable occurrences of the respective sides must be either both reachable by addition or both reachable by multiplication. In the at-version the mth occurrence on the left hand side and the lth occurrence on the right hand side must be both reachable by addition or both reachable by multiplication.
  - Result: When no at is used the first droppable occurrence of n is dropped on both sides. In the at-version the mth occurrence on the left hand side and the lth one on the right hand side are dropped. In all variants, the order and the parentheses of the remaining terms survive. Furthermore, in all variants a side consisting only of n is seen as addition with 0 or multiplication by 1, depending on the outermost operation of the other side (if both sides are n the additional view is chosen). In the following variants the additional information $n \neq 0$ is needed and has to be shown by the user first: dropping n in the goal being a negation of an equation between multiplications or being a $<$-inequality between multiplications (or dropping n in the hypothesis H being an equality between multiplications or being a $\leq$-inequality between multiplications respectively). Again, the above cases are meant to imply sides that are just n.
- `exception` H: The hypothesis H is replaced by the finitely many possibilities that remain to prevent H from becoming contradictory.
  - Preconditions: The hypothesis H must be basic. Furthermore, there must be a theorem in the library we are providing that has the form $\forall\ n\ m\ l\ \ldots : \mathbb{N},\ H' \to E$ where H' is a basic formula and where H is an instance of the theorem without the conclusion E while the latter expresses finitely many possibilities of concrete numbers (as $n = 0$, $n = 0 \vee n = 5$, or $n > 2$ for instance).
  - Result: The hypothesis H is replaced by the instance of E that corresponds to the way in which H is an instance of the anterior part of the whole theorem.

- `apply_reciprocal` (in H): Identical terms occurring as summands in the minuend and subtrahend of a subtraction are dropped.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic.
  - Result: For each of both sides that have a subtraction as outermost operation all summands appearing in the minuend and in the subtrahend as well are dropped. In case of multiple occurrences the first one is chosen. If afterwards there are no summands left in the subtrahend only the minuend remains; and if this is the case for the minuend the whole term is evaluated as 0 (including cases of the form $n - n$).
- `expand` (in H) (at n): One of the distributivity laws is applied.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic. In the variant without 'at' at least on one side there must be a subterm of the form $m * (l + k)$, $m * (l - k)$, $(m + l) * k$, or $(m - l) * k$ that is reachable by addition, subtraction, and multiplication. In the at-variant there must be n such subterms in total and with no double countings in the goal (or the hypothesis H respectively).
  - Result: In the variant without 'at', on both sides the first suitable subterm – if existent – is expanded. There are no further changes. In the at-version the nth possibility to expand (counted in total and without doublings) is expanded. On the respective sides the outermost expanding possibilities are counted first, then the ones of the first operand, and finally the ones of the second operand. In case there are two possibilities in a (sub)term with respect to expansion the first one appearing in the listing above is chosen.
- `factorise` n (in H) at m (l): One of the distributivity laws is applied backwardly.
  - Preconditions: The goal (or the hypothesis H respectively) must be basic. In the double at-version the mth and lth summand must appear on the same side of the goal (or the hypothesis H respectively) and comprise an n as factor (not necessarily as outermost operand but n must be reachable in the mth and lth summand by multiplication only); where all summands of the whole goal (or of the whole hypothesis H respectively) that we would achieve after omitting all unnecessary parentheses are counted ($(n + 5 * (l + m)) + 3$, for instance, has three summands in this counting). If the single at-version with $m = 1$ is

used the left or the right hand side of the goal (or of the hypothesis H respectively) must be a subtraction in which n occurs as a factor in the minuend and in the subtrahend. If $m = 2$ both sides must fulfil this requirement.

– Result: In the double at-version the involved side of the goal (or of the hypothesis H respectively) is replaced by an addition in which the first summand is of the form $n * (k + j)$ where k and j are the mth and the lth summand without the first occurrence of n as a factor (counted like the summands above); and in which the second summand is the old term but without the mth and the lth summand. If the original term of the involved side only consists of two summands there is no second summand in the replacing term. In the single at-version with $m = 1$ – if factorising n is possible – the left hand side is replaced by a multiplication in which n is the first factor and the second factor is the original term but without the first n in minuend and subtrahend. If this is not possible or if $m = 2$ this is done for the right hand side.

C List of Figures

Appendix D

Bibliography

[1] Martin Aigner & Günter M. Ziegler (1998): *Proofs from The Book*, 2009 edition. ISBN: 978-3-642-00855-9.

[2] George A. Akerlof & Robert J. Shiller (2015): *Phishing for Phools – The Economics of Manipulation and Deception*. ISBN: 978-0-691-16831-9.

[3] Stuart F. Allen, Mark Bickford, Robert L. Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo & E. Moran (2006): *Innovations in computational type theory using Nuprl*. *J. Applied Logic* 4(4), pp. 428–469, DOI: 10.1016/j.jal.2005.10.005.

[4] Marco Almeida, Nelma Moreira & Rogério Reis (2009): *Testing the Equivalence of Regular Languages*. In Jürgen Dassow, Giovanni Pighizzini & Bianca Truthe, editors: *Proceedings Eleventh International Workshop on Descriptional Complexity of Formal Systems (DCFS 2009)*, *EPTCS* 3, pp. 47–57, DOI: 10.4204/EPTCS.3.4.

[5] Jody Azzouni (2013): *The Relationship of Derivations in Artificial Languages to Ordinary Rigorous Mathematical Proof*. *Philosophia Mathematica* 21(2), pp. 247–254, DOI: 10.1093/philmat/nkt007.

[6] Ralph-Johan Back (2010): *Structured derivations: a unified proof style for teaching mathematics*. *Formal Aspects of Computing* 22(5), pp. 629–661, DOI: 10.1007/s00165-009-0136-5.

[7] Nimrod Bar-Am (2008): *Extensionalism – The Revolution in Logic*. DOI: 10.1007/978-1-4020-8168-2.

[8] Aditi Barthwal & Michael Norrish (2010): *A Formalisation of the Normal Forms of Context-Free Grammars in HOL4*. In Anuj Dawar & Helmut Veith, editors: *Computer Science Logic, 24th International Workshop (CSL 2010, Proceedings)*, pp. 95–109, DOI: 10.1007/978-3-642-15205-4_11.

[9] Aditi Barthwal & Michael Norrish (2014): *A mechanisation of some context-free language theory in HOL4*. *Journal of Computer and System Sciences* 80(2), pp. 346–362, DOI: 10.1016/j.jcss.2013.05.003.

[10] David A. Basin (1994): *Generalized Rewriting in Type Theory*. *Elektronische Informationsverarbeitung und Kybernetik* 30(5/6), pp. 249–259.

[11] Thomas Bedürftig & Roman Murawski (2010): *Philosophie der Mathematik*. ISBN: 978-3-11-019093-9.

[12] Yves Bertot & Pierre Castéran (2004): *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. DOI: 10.1007/978-3-662-07964-5.

[13] Sebastian Böhne: *Implementation of the Naturals Project*. Available at `https://www.cs.uni-potsdam.de/~sboehne/Naturals_Project.tar.gz`.

[14] Sebastian Böhne: *Mathematisches Argumentieren und Beweisen in Coq*. Available at `https://lecture2go.uni-hamburg.de/l2go/-/get/v/19876`.

[15] Sebastian Böhne & Christoph Kreitz (2017): *Learning how to Prove: From the Coq Proof Assistant to Textbook Style*. In Pedro Quaresma & Walther Neuper, editors: *Proceedings 6th International Workshop on Theorem proving components for Educational software (ThEdu '17)*, pp. 1–18, DOI: 10.4204/EPTCS.267.1.

[16] Sylvie Boldo & Guillaume Melquiond: *Flocq*.
Available at `http://flocq.gforge.inria.fr/`.

[17] Ana Bove, Peter Dybjer & Ulf Norell (2009): *A Brief Overview of Agda - A Functional Language with Dependent Types*. In Stefan Berghofer, Tobias Nipkow, Christian Urban & Makarius Wenzel, editors: *Theorem Proving in Higher Order Logics, 22nd International Conference (TPHOLs 2009, Proceedings)*, pp. 73–78, DOI: 10.1007/978-3-642-03359-9_6.

[18] Thomas Braibant & Damien Pous (2011): *Tactics for Reasoning Modulo AC in Coq*. In Jean-Pierre Jouannaud & Zhong Shao, editors: *Certified Programs and Proofs - First International Conference (CPP 2011, Proceedings)*, pp. 167–182, DOI: 10.1007/978-3-642-25379-9_14.

[19] Alan Bundy (1996): *Proof Planning.* In Brian Drabble, editor: *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS 1996)*, pp. 261–267, ISBN: 978-0-929280-97-4.

[20] Nathan C. Carter & Kenneth G. Monks: *Using the Proof-Checking Word Processor Lurch to Teach Proof-Writing.* Available at `https://pdfs.semanticscholar.org/9679/6462f47348a3d81326e706f6d334e5eda0c2.pdf`

[21] Timothy Y. Chow (2007): *A beginner's guide to forcing.* 2008 edition. Available at `https://arxiv.org/abs/0712.1320`.

[22] Jacek Chrząszcz & Daria Walukiewicz-Chrząszcz (2007): *Towards Rewriting in Coq.* In Hubert Comon-Lundh, Claude Kirchner & Hélène Kirchner, editors: *Rewriting, Computation and Proof, Essays Dedicated to Jean-Pierre Jouannaud on the Occasion of His 60th Birthday*, pp. 113–131, DOI: 10.1007/978-3-540-73147-4_6.

[23] Allan Collins, John S. Brown & Ann Holum: *Cognitive Apprenticeship: Making thinking visible.* Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.8616&rep=rep1&type=pdf`. The article was originally published in 1991 in American educator 15(3).

[24] *Coq 8.8.0 – Source code.*
Available at `https://github.com/coq/coq/releases/tag/V8.8.0`.

[25] *The Coq Constructive Repository at Nijmegen.*
Available at `https://github.com/c-corn/corn`.

[26] *The Coq Proof Assistant.* `https://coq.inria.fr`.

[27] *Coq Repository at Nijmegen.* `http://corn.cs.ru.nl`.

[28] *The Coq Standard Library.*
Available at `https://coq.inria.fr/distrib/current/stdlib`.

[29] Haskell B. Curry (1939): *Remarks on the definition and nature of mathematics.* In Paul Benacerraf & Hilary Putnam, editors: *Philosophy of mathematics – Selected readings*, 1983 edition, pp. 202–206, ISBN: 978-0-521-22796-4. Reprint of a reprint with minor corrections of an article published in Journal of Unified Science 9, pp. 164-169.

[30] Haskell B. Curry (1941): *Some Aspects of the Problem of Mathematical Rigor. Bulletin of the American Mathematical Society* 47, pp. 221–241.

[31] Haskell B. Curry (1951): *Outlines of a Formalist Philosophy of Mathematics*, 1958 edition.

[32] Haskell B. Curry (1963): *Foundations of Mathematical Logic*, 1977 edition, ISBN: 978-0-486-63462-3.

[33] Haskell B. Curry (1968): *The Purposes of Logical Formalization*. *Logique Et Analyse* 11(43), pp. 357–366.

[34] John W. Dawson, Jr (2006): *Why Do Mathematicians Re-prove Theorems?* *Philosophia Mathematica* 14(3), pp. 269–286, DOI: 10.1093/philmat/nkl009.

[35] David Delahaye (2000): *A Tactic Language for the System Coq*. In Michel Parigot & Andrei Voronkov, editors: *Logic for Programming and Automated Reasoning, 7th International Conference (LPAR 2000, Proceedings)*, pp. 85–95, DOI: 10.1007/3-540-44404-1_7.

[36] Christian Doczkal, Jan-Oliver Kaiser & Gert Smolka (2013): *A Constructive Theory of Regular Languages in Coq*. In Georges Gonthier & Michael Norrish, editors: *Certified Programs and Proofs - Third International Conference (CPP 2013, Proceedings)*, pp. 82–97, DOI: 10.1007/978-3-319-03545-1_6.

[37] Arno Ehle, Norbert Hundeshagen & Martin Lange (2017): *The Sequent Calculus Trainer with Automated Reasoning - Helping Students to Find Proofs*. In Pedro Quaresma & Walther Neuper, editors: *Proceedings 6th International Workshop on Theorem proving components for Educational software (ThEdu '17)*, pp. 19–37, DOI: 10.4204/EPTCS.267.2.

[38] Solomon Feferman (2012): *And so on. . . : reasoning with infinite diagrams*. *Synthese* 186(1), pp. 371–386, DOI: 10.1007/s11229-011-9985-6.

[39] Armin Fiedler (2001): *Dialog-driven Adaptation of Explanations of Proofs*. In Bernhard Nebel, editor: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pp. 1295–1300. Available at `http://ijcai.org/proceedings/2001-1`.

[40] Armin Fiedler (2001): *User-adaptive proof explanation*. Ph.D. thesis, Saarland University, Saarbrücken, Germany. Available at `http://scidok.sulb.uni-saarland.de/volltexte/2004/182/index.html`.

[41] Jean-Christophe Filliâtre (1997): *Finite Automata Theory in Coq: A constructive proof of Kleene's theorem*. Technical Report, Research Report 97–04, LIP-ENS Lyon.

[42] Richard Fitzpatrick, editor (2007): *Euclid's Elements of Geometry*, 2008 edition. Available at
`http://farside.ph.utexas.edu/Books/Euclid/Elements.pdf`.

[43] Ludovic Font, Philippe R. Richard & Michel Gagnon (2017): *Improving QED-Tutrix by Automating the Generation of Proofs*. In Pedro Quaresma & Walther Neuper, editors: *Proceedings 6th International Workshop on Theorem proving components for Educational software (ThEdu '17)*, pp. 38–58, DOI: 10.4204/EPTCS.267.3.

[44] *ForMath: Formalisation of Mathematics*. Available at
`http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/ForMath`.

[45] Gottlob Frege (1879): *Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought*. In Jean Van Heijenoort, editor: *From Frege to Gödel – A source book in mathematical logic*, 2002 edition, ISBN: 978-0-674-32449-7. The German subtitle of the 1879 version is *Eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*.

[46] Gottlob Frege (1884): *The Foundations of Arithmetic – A logico-mathematical enquiry into the concept of natural number*, 1960 edition. The German title of the 1884 version is *Grundlagen der Arithmetik – Eine logisch-mathematische Untersuchung über den Begriff der Zahl*.

[47] Gottlob Frege (1893 (und 1903)): *The Basic Laws of Arithmetic – Exposition of the System*, 1982 edition. ISBN: 978-0-520-04761-7. The original German version has two volumes. The German title is *Grundgesetze der Arithmetik – begriffsschriftlich abgeleitet*.

[48] Michèle Friend (2014): *Pluralism in Mathematics: A New Position in Philosophy of Mathematics*. DOI: 10.1007/978-94-007-7058-4.

[49] Jean-Yves Girard (1987): *Linear Logic*. *Theoretical Computer Science* 50, pp. 1–102, DOI: 10.1016/0304-3975(87)90045-4.

[50] Georges Gonthier & Roux Stéphane Le (2009): *An Ssreflect Tutorial*. Technical Report RT-0367, INRIA.
Available at `https://hal.inria.fr/inria-00407778/document`.

[51] John Harrison (2016): *The HOL Light System – REFERENCE*. Available at `http://www.cl.cam.ac.uk/~jrh13/hol-light/reference.pdf`.

[52] John Harrison (2017): *HOL Light Tutorial*. Available at
`http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial.pdf`.

[53] Maxim Hendriks, Cezary Kaliszyk, Femke Van Raamsdonk & Freek Wiedijk (2010): *Teaching logic using a state-of-the-art proof assistant*. *Acta Didactica Napocensia* 3(2), pp. 35–48.
Available at `https://eric.ed.gov/?id=EJ1056118`.

[54] David Hilbert (1899): *The Foundations of Geometry*, 2005 edition. Ebook #17384 (for the 1950 version). The German title of the 1899 version is *Grundlagen der Geometrie*.

[55] Dirk W. Hoffmann (2011): *Grenzen der Mathematik – Eine Reise durch die Kerngebiete der mathematischen Logik*, 2013 edition.
ISBN: 978-3-827-42559-1.

[56] *HOL – Interactive Theorem Prover*. `https://hol-theorem-prover.org`.

[57] Amanda M. Holland-Minkley, Regina Barzilay & Robert L. Constable (1999): *Verbalization of High-Level Formal Proofs*. In Jim Hendler & Devika Subramanian, editors: *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI 99)*, pp. 277–284. Available at `https://www.aaai.org/Papers/AAAI/1999/AAAI99-041.pdf`.

[58] H. James Hoover & Piotr Rudnicki (1996): *Teaching freshman logic with MIZAR-MSE*. In: *Workshop on Teaching Logic and Reasoning in an Illogical World*. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.4770&rep=rep1&type=pdf`.

[59] John E. Hopcroft, Rajeev Motwani & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*, 2001 edition. ISBN: 978-0-201-44124-6. The two editions are different in character. Motwani was no author of the first one.

[60] *HoTT-library*. Available at `https://github.com/HoTT/HoTT`.

[61] William A Howard (1980): *The formulae-as-types notion of construction*. In: *To H.B. Curry: essays on combinatory logic, lambda calculus and formalism*, pp. 479–490, ISBN: 978-0-12-349050-6. A 2017 version written in Latex is available at `http://www.dcc.fc.up.pt/~acm/howard2.pdf`.

[62] Jens Høyrup (2005): *Tertium Non Datur – On Reasoning Styles in Early Mathematics*. In Paolo Mancosu, Klaus F. Jørgensen & Stig A. Pedersen, editors: *Visualization, Explanation and Reasoning Styles in Mathematics*, ISBN: 978-1-4020-3334-6.

[63] Patrik Jansson, Sólrún H. Einarsdóttir & Cezar Ionescu: *Examples and Results from a BSc-level Course on Domain Specific Languages of Mathematics.* This article is not published yet. The most actual version is of the year 2018.

[64] Wojciech Jedynak, Malgorzata Biernacka & Dariusz Biernacki (2013): *An Operational Foundation for the Tactic Language of Coq.* In Ricardo Peña & Tom Schrijvers, editors: *15th International Symposium on Principles and Practice of Declarative Programming (PPDP '13)*, pp. 25–36, DOI: 10.1145/2505879.2505890.

[65] Morris Kline (1972): *Mathematical Thought - From Ancient to Modern Times*, 1990 edition. ISBN: 978-0-19-506135-2.

[66] Maria Knobelsdorf & Christiane Frede (2016): *Analyzing Student Practices in Theory of Computation in Light of Distributed Cognition Theory.* In Judy Sheard, Josh Tenenberg, Donald Chinn & Brian Dorn, editors: *Proceedings of the 2016 ACM Conference on International Computing Education Research (ICER 2016)*, pp. 73–81, DOI: 10.1145/2960310.2960331.

[67] Maria Knobelsdorf, Christiane Frede, Sebastian Böhne & Christoph Kreitz (2017): *Theorem Provers as a Learning Tool in Theory of Computation.* In Josh Tenenberg, Donald Chinn, Judy Sheard & Lauri Malmi, editors: *Proceedings of the 2017 ACM Conference on International Computing Education Research (ICER 2017)*, pp. 83–92, DOI: 10.1145/3105726.3106184.

[68] Maria Knobelsdorf, Christoph Kreitz & Sebastian Böhne (2014): *Teaching Theoretical Computer Science using a Cognitive Apprenticeship Approach.* In J. D. Dougherty, Kris Nagel, Adrienne Decker & Kurt Eiselt, editors: *The 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*, pp. 67–72, DOI: 10.1145/2538862.2538944.

[69] Christoph Kreitz (1986): *Constructive Automata Theory Implemented with the Nuprl Proof Development System.* Cornell University. Department of Computer Science, Ithaca.
Available at `https://ecommons.cornell.edu/handle/1813/6619`.

[70] Twan D. L. Laan (1997): *The evolution of type theory in logic and mathematics.* PhD thesis, Technische Universiteit Eindhoven, DOI: 10.6100/IR498552.

[71] Imre Lakatos (1976): *Proofs and refutations – The logic of mathematical discovery*, 1983 edition. ISBN: 978-0-521-29038-8. The book was first published after Lakatos' death. John Worall and Elie Zahar are the editors.

[72] Leslie Lamport (1995): *How to Write a Proof. The American Mathematical Monthly* 102(7), pp. 600–608.
Available at `http://www.jstor.org/stable/2974556`.

[73] Graham Leach-Krouse (2017): *Carnap: An Open Framework for Formal Reasoning in the Browser.* In: *Proceedings 6th International Workshop on Theorem proving components for Educational software (ThEdu '17)*, pp. 70–88, DOI: 10.4204/EPTCS.267.5.

[74] *List of Coq Math Projects.*
`https://github.com/coq/coq/wiki/List-of-Coq-Math-Projects`.

[75] Paul Lorenzen (1968): *Collegium Logicum.* In: *Methodisches Denken*, 1980 edition, pp. 7–23, ISBN: 978-3-518-07673-6.

[76] Penelope Maddy (1990): *Realism in Mathematics*, 2003 edition. ISBN: 978-0-19-824035-8.

[77] *math-classes – A library of abstract interfaces for mathematical structures in Coq.* Available at `https://github.com/math-classes/math-classes`.

[78] *Mathematical components.*
Available at `https://github.com/math-comp/math-comp`.

[79] Guillaume Melquiond & Erik Martin-Dorel: *GAPPA.*
Available at `https://gforge.inria.fr/projects/gappa/`.

[80] Elliott Mendelson (1964): *Introduction to Mathematical Logic*, 1997 edition. ISBN: 978-0-412-80830-2.

[81] Julius Moravcsik (2004): *Logic Before Aristotle: Development or Birth?* In Dov M. Gabbay & John Woods, editors: *Handbook of the History of Logic – Volume I Greek, Indian and Arabic Logic*, ISBN: 978-0-444-50466-1.

[82] Nelma Moreira, David Pereira & Simão Melo de Sousa (2012): *Deciding Regular Expressions (In-)Equivalence in Coq.* In Wolfram Kahl & Timothy G. Griffin, editors: *Relational and Algebraic Methods in Computer Science - 13th International Conference (RAMiCS 2012, Proceedings)*, pp. 98–113, DOI: 10.1007/978-3-642-33314-9_7.

[83] Michal Muzalewski (1993): *An Outline of PC Mizar.* Available at
`https://pdfs.semanticscholar.org/b819/df93026a857740e20fc5f0b75b1fea3b305a.pdf`.

[84] Julien Narboux (2005): *Toward the use of a proof assistant to teach mathematics*. Available at `https://hal.inria.fr/inria-00495952/document`. The Seventh International Conference on Technology in Mathematics Teaching (ICTMT7) <inria-00495952>.

[85] Tobias Nipkow (2012): *Teaching Semantics with a Proof Assistant: No More LSD Trip Proofs*. In Viktor Kuncak & Andrey Rybalchenko, editors: *Verification, Model Checking, and Abstract Interpretation - 13th International Conference (VMCAI 2012, Proceedings)*, pp. 24–38, DOI: 10.1007/978-3-642-27940-9_3.

[86] Tobias Nipkow, Lawrence C. Paulson & Markus Wenzel (2002): *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*. DOI: 10.1007/3-540-45949-9.

[87] Russell O'Connor (2005): *The Gödel-Rosser 1st incompleteness theorem*. Available at `http://r6.ca/Goedel20050512.tar.gz`.

[88] Russell O'Connor (2009): *Incompleteness and Completeness – Formalizing Logic and Analysis in Type Theory*. PhD thesis, Radboud Universiteit Nijmegen. Available at `http://r6.ca/thesis.rev.fluorine.pdf`.

[89] Christine Paulin-Mohring (2015): *Introduction to the Calculus of Inductive Constructions*. In Bruno Woltzenlogel-Paleo & David Delahaye, editors: *All about Proofs, Proofs for all*, pp. 116–133, ISBN: 978-1-84890-166-7.

[90] Andrzej Pelc (2009): *Why Do We Believe Theorems? Philosophia Mathematica* 17(1), pp. 84–94, DOI: 10.1093/philmat/nkn030.

[91] Willard v. O. Quine & Joseph S. Ullian (1970): *The Web of Belief*, 1978 edition. ISBN: 978-0-07-553609-3.

[92] Marcus V. Ramos (2016): *Formalization of Context-Free Language Theory*. Ph.D. thesis, Universidade Federal de Pernambuco (Recife). Available at `http://www.marcusramos.com.br/univasf/tese.pdf`.

[93] Krzysztof Retel & Anna Zalewska (2005): *Mizar as a Tool for Teaching Mathematics. Mechanized Mathematics and Its Applications* 4(1), pp. 35–42. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.151.6028&rep=rep1&type=pdf#page=35`.

[94] Alan Robinson (1991): *Formal and Informal Proofs*. In Robert S. Boyer, editor: *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pp. 267–282, ISBN: 978-0-7923-1409-7.

[95] Alan Robinson (2000): *Proof = guarantee + explanation*. In Steffen Hölldobler, editor: *Intellectics and Computational Logic – Papers in Honor of Wolfgang Bibel*, pp. 277 – 294, DOI: 10.1007/978-94-015-9383-0.

[96] Bertrand Russell (1951): *The Autobiography of Bertrand Russell – 1872-1914*, 1967 edition. ISBN: 978-0-671-20358-0.

[97] Jakub Sakowicz & Jacek Chrząszcz (2007): *Papuq: a Coq assistant*. *Proceedings of PATE* 7, pp. 79–96. Available at `http://www.cs.ru.nl/~herman/PUBS/proceedingsPATE.pdf#page=79`.

[98] Marvin R. G. Schiller (2010): *Granularity Analysis for Tutoring Mathematical Proofs*. PhD thesis, Universität des Saarlandes.

[99] Wolfgang Schreiner, Alexander Brunhuemer & Christoph Fürst (2017): *Teaching the Formalization of Mathematical Theories and Algorithms via the Automatic Checking of Finite Models*. In Pedro Quaresma & Walther Neuper, editors: *Proceedings 6th International Workshop on Theorem proving components for Educational software (ThEdu '17)*, pp. 120–139, DOI: 10.4204/EPTCS.267.8.

[100] Jonathan P. Seldin (2005): *Curry, Haskell Brooks (1900-82)*. In John R. Shook, editor: *The Dictionary of Modern American Philosophers – Volumes 1,2,3 and 4*, pp. V1:562–567, ISBN: 978-1-84371-037-0.

[101] Jonathan P. Seldin (2009): *The Logic of Church and Curry*. In Dov M. Gabbay & John Woods, editors: *Logic from Russell to Church*, *Handbook of the History of Logic* 5, pp. 819–873, DOI: 10.1016/S1874-5857(09)70019-6.

[102] Jonathan P Seldin (2011): *Curry's Formalism as Structuralism*. *Logica Universalis* 5(1), pp. 91–100, DOI: 10.1007/s11787-011-0028-3.

[103] Stewart Shapiro (1991): *Foundations without Foundationalism – A Case for Second-Order Logic*. ISBN: 978-0-19-853391-7.

[104] Stewart Shapiro (2000): *Thinking About Mathematics – The Philosophy of Mathematics*. ISBN: 978-0-19-289306-2.

[105] Jörg H. Siekmann, Christoph Benzmüller & Serge Autexier (2006): *Computer supported mathematics with ΩMEGA*. *J. Applied Logic* 4(4), pp. 533–559, DOI: 10.1016/j.jal.2005.10.008.

[106] Robin Smith (2018): *Aristotle's Logic*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, spring 2018 edition. Available at `https://plato.stanford.edu/archives/spr2018/entries/aristotle-logic`.

[107] Bruno Snell (1946): *The Discovery of the Mind: The Greek Origins of European Thought*, 1953 edition. The German title of the 1946 version is *Die Entdeckung des Geistes – Studien zur Entstehung des europäischen Denkens bei den Griechen*.

[108] Fenner Tanswell (2015): *A Problem with the Dependence of Informal Proofs on Formal Proofs*. *Philosophia Mathematica* 23(3), pp. 295–310, DOI: 10.1093/philmat/nkv008.

[109] Andrzej Trybulec & Peter Rudnicki (1993): *Using Mizar in Computer Aided Instruction of Mathematics*. In: *Norvegian-French Conference of CAI in Mathematics*. Available at `http://mizar.uwb.edu.pl/project/oslo.pdf`.

[110] Alan M. Turing (1937): *On Computable Numbers, With an Application to the Entscheidungsproblem. Proceedings of the London mathematical society* 2(1), pp. 230–265. Available at `http://www.cs.unibo.it/~martini/CS/Turing_Paper_1936.pdf`.

[111] The Univalent Foundations Program (2013): *Homotopy Type Theory – Univalent Foundations of Mathematics*. Available at `https://homotopytypetheory.org/book`.

[112] *Univalent Mathematics Coq files*. Available at `https://github.com/UniMath/UniMath/tree/master/UniMath/Foundations`.

[113] Jørgen Villadsen, Andreas Halkjær From & Anders Schlichtkrull (2017): *Natural Deduction and the Isabelle Proof Assistant. Electronic Proceedings in Theoretical Computer Science (EPTCS)* 267, pp. 140–155, DOI: 10.4204/EPTCS.267.9.

[114] Alan Weir (2011): *Formalism in the Philosophy of Mathematics*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, 2015 edition. Available at `https://plato.stanford.edu/archives/spr2015/entries/formalism-mathematics/`. The first and this edition differ substantially.

[115] Makarius Wenzel, Lawrence C. Paulson & Tobias Nipkow (2008): *The Isabelle Framework*. In Otmane Aït Mohamed, César A. Muñoz & Sofiène Tahar, editors: *Theorem Proving in Higher Order Logics, 21st International Conference (TPHOLs 2008, Proceedings)*, pp. 33–38, DOI: 10.1007/978-3-540-71067-7_7.

[116] Makarius Wenzel et al. (2017): *The Isabelle/Isar Reference Manual*. Available at `http://128.232.0.20/research/hvg/Isabelle/dist/Isabelle/doc/isar-ref.pdf`.

[117] Alfred N. Whitehead & Bertrand Russell (1910): *Principia Mathematica*.

[118] Freek Wiedijk (1999): *Mizar: An Impression*.
Available at `http://www.cs.ru.nl/F.Wiedijk/mizar/mizarintro.pdf`.

[119] Claus W. Zinn (2004): *Understanding Informal Mathematical Discourse*. PhD thesis, Friedrich-Alexander Universität Erlangen-Nürnberg.