Textbook Proofs Meet Formal Logic — The Problem of Underspecification and Granularity

Serge Autexier and Armin Fiedler

Saarland University & German Research Centre for Artificial Intelligence (DFKI GmbH), Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany,
{autexier,afiedler}@ags.uni-sb.de

Abstract. Unlike computer algebra systems, automated theorem provers have not yet achieved considerable recognition and relevance in mathematical practice. A significant shortcoming of mathematical proof assistance systems is that they require the fully formal representation of mathematical content, whereas in mathematical practice an informal, natural-language-like representation where obvious parts are omitted is common. We aim to support mathematical paper writing by integrating a scientific text editor and mathematical assistance systems such that mathematical derivations authored by human beings in a mathematical document can be automatically checked. To this end, we first define a calculus-independent representation language for formal mathematics that allows for underspecified parts. Then we provide two systems of rules that check if a proof is correct and at an acceptable level of granularity. These checks are done by decomposing the proof into basic steps that are then passed on to proof assistance systems for formal verification. We illustrate our approach using an example textbook proof.

1 Introduction

Unlike computer algebra systems (CASs), mathematical proof assistance systems have not yet achieved considerable recognition and relevance in mathematical practice. Clearly, the functionalities and strengths of these systems are generally not sufficiently developed to attract mathematicians on the edge of research. For applications in e-learning and engineering contexts their capabilities are often sufficient, though. However, current systems suffer from several major drawbacks. First, instead of supporting the language the mathematician is used to, most systems impose their own formal language on the user and require a machine-oriented formalization of the mathematical content to allow for powerful automatic inference capabilities. As a result, the line of reasoning is often unnatural and obscured. Next, the proofs are at a level of excruciating detail spelling out many logically necessary steps, which a human would nevertheless consider trivial or obvious. Thus, the proofs are often illegible and incomprehensible. Finally, the acceptance of mathematical assistant systems would be increased by integrating them with scientific WYSIWYG text editors. Indeed,

current word processors regularly employ spell checkers to check the correct spelling of the words and sometimes grammar checkers to check the correct application of the grammar rules in the sentences. Our aim is to support the practice of mathematical paper writing in the scientific text editor by employing proof assistance systems that provide definitions, lemmas and theorems from mathematical databases and automatically check the derivations spelled out in a mathematical document. The vision is to achieve the possibility to verify mathematical documents fully automatically. We envision a scientific text editor that allows the author to write semantically annotated mathematical content. The semantic annotations can then be exploited to generate a formal representation of the mathematical content (cf. Autexier *et al.* [2]), which allows for further automatic processing. The first step towards this end is to provide the formal language that can represent human-authored mathematical content.

Since the 1960ies, the AUTOMATH project [9] has been addressing the problem of developing a formal language with a natural-language-like syntax that allows both for the exact formalization of mathematical content and for the easy reading and authoring of the documents by mathematicians [8]. Whereas the original AUTOMATH language is very mechanical and thus tedious to author, its derivatives Mathematical Vernacular [4], Weak Type Theory [10] and MathLang [7] are close to a natural language. Since the 1970ies, the Mizar[1] project aims at supporting mathematical publications by means of a formal language that allows for automatic consistency checks of documents and for references to other articles published in the same formal language. A similar, more recent approach is taken in Isabelle/Isar [11], where proofs can be entered in a formal language of mathematics, which are readable for both human and machine (in fact, the Isar language is very similar to Mizar's language [14]). In another approach in the same tradition, Abel and colleagues [1] present a formal language for first-order intuitionistic logic used in a tutorial system for intuitionistic logic. The student writes proofs directly in this language, which are then automatically checked using a system of proof checking rules for intuitionistic logic.

A more sophisticated approach with respect to human readability is taken in the grammatical framework [12], a formalism based on a typed λ-calculus that allows for the definition of context-free grammars for fragments of natural language. However, only simple linguistic structures can be captured in this approach.

The major drawback of these approaches is that they do not sufficiently succeed in combining two diverging requirements, namely automatic processibility and readability. Automatic processing requires exact formalization, which in turn requires many details that humans consider obvious or trivial. Whereas detailed steps can be abstracted from in formal proofs by using lemmas, all steps must be included in the formal proof, even if they are easily inferable by the human user. Conversely, because of the omission of easily inferable steps, human-authored derivations often turn out to have gaps when scrutinized formally. The mentioned systems sacrifice the readability in favor of the processibility.

[1] `http://www.mizar.org`

Therefore, we suggest a formal representation language for human-authored proofs where the gaps are filled in by underlying proof assistance systems (without committing ourselves to a specific prover or proof procedure). This formal language mediates between the semantically-annotated natural language representation of the mathematical document in the scientific text editor, where the user enters his input, and the logic representation required by proof assistance systems that check the mathematical content of the documents and fill in gaps. The core idea is to define a formal language that allows for underspecified parts and two systems of rules that check if a proof is correct and at an acceptable level of granularity. These checks are done by decomposing the proof into basic steps that are passed on to proof assistance systems for formal verification.

Clearly, which level of granularity is acceptable depends on many factors, the most prominent ones being the knowledge and skills of the intended audience, the mathematical theory the proven theorem belongs to and the personal style of the author. We do not cope with these factors in this paper, but define one specific level of granularity based on Hilbert's ideas [5] to demonstrate how a specific notion of granularity can be captured by constraining the proof checking rules.

This paper is organized as follows: We start with an overview of our approach. Then, we introduce our formal language for human-oriented proofs, and define a proof checking system for the proofs in that language. Finally, we experiment with means to formally capture notions of granularity. More specifically, we investigate how granularity can be defined as a restriction of the proof checking system.

2 Our Approach

Since many details of a proof, although logically required for a correct derivation, are considered obvious or trivial by human beings, they should be omitted from the proof. For our purposes, we capture the *level of granularity* using the following two distinct aspects:

First, the *level of concept*, at which a proof is done, can be described in terms of the definitions and theorems that can be used in the proof. More precisely, we can identify a mathematical *theory*, to which the theorem that is being proved belongs, as a logical collection of axioms, assumptions, definitions, lemmas and theories (collectively called *assertions*) as well as proofs. We consider a whole hierarchy of theories maintained in a mathematical database, where one theory draws on underlying theories. Now, if the theorem we want to prove belongs to some theory T, then we define the level of concept of the proof as the collection of assertions in theories underlying T plus the assertions of T that logically come before the theorem to be proved.

Second, the *amount of detail* in a proof refers to all facts and inference rules that are explicitly mentioned in the proof. Human-authored proofs are often imprecise in several respects, namely the used inference rule is not mentioned, some of the premises needed for a step in the derivation are not mentioned, and

some steps of the derivation are completely omitted. That natural language texts and utterances are inherently imprecise (i.e., several readings of a sentence are possible) is a well-known phenomenon in linguistics, called *underspecification*. Thus, automated processing of the content of human-authored texts requires the resolution of underspecification by singling out one possible reading.

Our work has been inspired by the work of Abel and colleagues [1], who worked on a tutorial system for intuitionistic logic. In their approach, they defined a linear syntax to represent first-order natural deduction proofs at the assertion level in intuitionistic logic and combined it with a deductive system of proof checking rules for that logic. Thus, the student can write proofs directly in this language and the proofs are automatically checked for correctness. The representation language, however, allows for one possible reading only.

In our approach, we adopt this idea of separating the representation language from the set of checking rules. However, we extend the approach in two dimensions:

First, we suggest a formal representation language for mathematical content detached from any particular logic or calculus. This allows us to represent arbitrary content regardless of the underlying logic. Moreover, the language allows us to represent both different levels of concept and underspecification and is thus particularly well-suited to represent proofs that are authored in a natural way by humans.

Second, we add two deductive systems, namely one for checking the correctness of proofs and one for checking the level of concept. The former decomposes the proof into basic steps, which either can be verified directly by one of the rules of the system or is passed on to an external proof assistance system that checks its correctness, and, if it is successful, provides a correctness proof. As a side effect, underspecified parts of such a basic proof step are resolved. The second deductive system similarly decomposes the proof into basic steps, but now checks if the steps are justified using acceptable inference rules.

We envision that our approach can serve as a first step towards an integration of a scientific text editor with mathematical proof assistance systems. In particular, the deductive systems show how mathematical proof assistance systems can be employed. To achieve the overall goal, however, many additional problems must be tackled, most notably a natural language analysis component that transforms the human-authored proofs into proofs in our representation language. Thus, we require for the time being that the author enter semantically annotated text by using LaTeX-style macros. These macros can then be expanded into a formal representation (cf. [2]), such as our representation language.

3 A Formal Representation Language

In this section we present the formal representation language for proofs (cf. Fig. 1). The language accommodates the mostly linear structure of textual proofs by representing complete proofs as a "*;*"-separated sequence of proof steps. In order to account for the internal structure of the proofs, the language allows

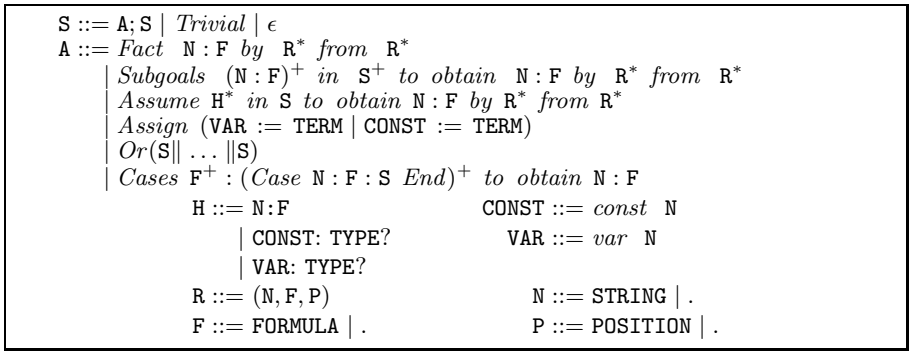

Fig. 1. The grammar of the formal proof language

for complex structures such as the introduction of subgoals or hypotheses, case analysis and induction. In each proof step, which either introduces subgoals or derives a fact, we distinguish in the syntax between the used concepts to justify that proof step (denoted by the keyword *by* in the language) and to which premises or goals the concepts have been applied (denoted by *from*). Finally, in order to support the linguistic analysis of mathematical documents, which is not always able to uniquely categorize a given text fragment, we introduce a nondeterministic branching over possible proofs (`Or`) to represent the different alternative interpretations.

For the definition of the language we assume languages for formulas, terms, and types referred to by the nonterminal grammar symbols `FORMULA`, `TERM`, and `TYPE`, respectively. In the proof language, ϵ denotes an empty (sub)proof while *Trivial* indicates that the (sub)proof should be completed now, for instance, if there is a formula that occurs both as a goal and a hypothesis. The step "*Fact* `N : F` *by* `R*` *from* `R*`" indicates that a fact `F` has been derived from the objects referenced in the *from* slot using the objects referenced in the *by* slot and has been assigned the name `N`. A reference consists of three parts: the *name* of a formula, a *formula* and a *position* denoting a sub-object of that formula or the one referenced by the name; each component of the reference can be left open, which is made explicit by a period ("."). Thus, the sub-language for references explicitly allows for underspecification. A proof step "*Subgoals* `(N : F)+` *in* `S+` *to obtain* `N : F` *by* $\mathtt{R}_1^*$ *from* $\mathtt{R}_2^*$" represents the fact that we introduced a list of subgoals `(N : F)+` for some previous goals $\mathtt{R}_2^*$ and the proofs in `S+` are the subproofs for these subgoals. Note that the facts used to perform that goal reduction may be given in $\mathtt{R}_1^*$. A proof step "*Assume* `H*` *in* `S` *to obtain* `N : F` *by* $\mathtt{R}_1^*$ *from* $\mathtt{R}_2^*$" is used to decompose goals $\mathtt{R}_2^*$ into the new hypotheses `H*` and the new goal `F` of name `N`. The hypotheses can be either named formulas `N : F`, or new constants and variables, possibly with some type. A proof step "*Assign var* $x := t$" allows us to assign a value t to some variable x and "*Assign const* $c := t$" encodes the introduction of an abbreviation c for some expression t. The expression "$Or(S_1 \parallel \ldots \parallel S_n)$" describes a situation, where the linguistic analysis identifies several possible interpretations

If A and B are sets such that $x \in A$ implies that $x \in B$ (that is, every element of A is also an element of B), then we shall say that A is **contained** in B, or that B contains A, or that A is a **subset** of B, and we shall write $A \subseteq B$ or $B \subseteq A$.
[. . .]
1.1.1 Definition Two sets A and B are **equal** if they contain the same elements. If the sets A and B are equal, we write $A = B$.
[. . .]
1.1.4 Theorem *Let A, B, C, be any sets, then*
[. . .]
(d) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$, [. . .]
[. . .]

In order to give a sample proof, we shall prove the first equation in (d). Let x be an element of $A \cap (B \cup C)$, then $x \in A$ and $x \in B \cup C$. This means that $x \in A$, and either $x \in B$ or $x \in C$. Hence we either have (i) $x \in A$ and $x \in B$, or we have (ii) $x \in A$ and $x \in C$. Therefore, either $x \in A \cap B$ or $x \in A \cap C$, so $x \in (A \cap B) \cup (A \cap C)$. This shows that $A \cap (B \cup C)$ is a subset of $(A \cap B) \cup (A \cap C)$.

Conversely, let y be an element of $(A \cap B) \cup (A \cap C)$. Then, either (iii) $y \in A \cap B$, or (iv) $y \in A \cap C$. It follows that $y \in A$, and either $y \in B$ or $y \in C$. Therefore, $y \in A$ and $y \in B \cup C$ so that $y \in A \cap (B \cup C)$. Hence $(A \cap B) \cup (A \cap C)$ is a subset of $A \cap (B \cup C)$.

In view of Definition 1.1.1, we conclude that the sets $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ are equal.

Fig. 2. A textbook example.

1. *Assume* . : $x \in A \cap (B \cup C)$ *in*
 1.1 *Fact* . : $x \in A \wedge x \in B \cup C$ *by* . *from* .;
 1.2 *Fact* . : $x \in A \wedge (x \in B \vee x \in C)$ *by* . *from* .;
 1.3 *Fact* . : $(x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)$ *by* . *from* .;
 1.4 *Fact* . : $(x \in A \cap B) \vee (x \in A \cap C)$ *by* . *from* .;
 1.5 *Fact* . : $x \in (A \cap B) \cup (A \cap C)$ *by* . *from* .;
 to obtain . : $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$ *by* . *from* .;
2. *Assume* . : $y \in (A \cap B) \cup (A \cap C)$ *in*
 2.1 *Fact* . : $y \in A \cap B \vee y \in A \cap C$ *by* . *from* .;
 2.2 *Fact* . : $y \in A \wedge (y \in B \vee y \in C)$ *by* . *from* .;
 2.3 *Fact* . : $y \in A \wedge (y \in B \cup C)$ *by* . *from* .;
 2.4 *Fact* . : $y \in A \cap (B \cup C)$ *by* . *from* .;
 to obtain . : $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$ *by* . *from* .;
3. *Fact* . : $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ *by* Def1.1.1 *from* .;
4. *Trivial*

Fig. 3. An example representation.

resulting in different possible proofs. Finally, case distinctions can be introduced by the *Cases* construct, where for each formula φ in $\mathtt{F}^+$ there is exactly one case $n : \varphi$. However, we consider case analysis as a derived construct that can be encoded by *Subgoals* and *Assume* proof steps. Analogously, induction proof steps can be defined.

To examine an example, let us consider an excerpt from Chapter 1 of the undergraduate analysis textbook *Introduction to Real Analysis* [3], which is shown in Fig. 2. For the purposes of this paper, we neglect the representation of the notation, the definition and the theorem, and focus only on the given proof, which starts with "Let x be an element..." The proof can then be represented in our language as depicted in Fig. 3. Note that the labels of the proof steps are only added for convenience and are not part of the representation language.

4 Proof Checking

Now, having a means of representing proofs, we also want to check the correctness of the represented proofs. To this end, we propose a deductive system consisting of eight rules that allow us to check the encoded proofs by recursively checking each individual proof step starting from the first. For each individual proof step we need to know all declared types and constants, collected in the *signature*, all declared variables, collected in the *context*, and all visible hypotheses and previous goals, which both are lists of named formulas. The result of a successfully checked proof step S is a set of facts derived by the subproof with S as its root.

The deductive system does not directly encode any specific calculus, but collects proof obligations, called *lemmas*, for proof steps. These lemmas need to be verified in order to establish the validity of the corresponding steps. For example, a *Trivial* proof step gives rise to a lemma $\Gamma \Longrightarrow_{\mathsf{Triv}} \Delta$, which states that from the hypotheses Γ some goal in Δ follows "*trivially*". Thus, we also have to provide a specific proof strategy that decides if a proof step is trivial or not. An example would be a simple check if some goal in Δ also occurs in Γ. In general, we allow for specific *strategies* strat to establish the validity of a lemma $\Gamma \Longrightarrow_{\mathsf{strat}} \Delta$. For the purposes of this paper, however, we will not go into the details of the strategies, but consider them as given (e.g., by a call of an automated theorem prover such as the proof planner ΩMEGA [13]).

Formally, a signature *Sig* consists of a list of type declarations $const\ \tau : \mathtt{type}$ and constant declarations $const\ c : \tau$:

$$Sig ::= \epsilon \mid const\ \tau : \mathtt{type}, Sig \mid const\ c : \tau, Sig$$

A context *Ctx* consists of a list of variable declarations:

$$Ctx ::= \epsilon \mid var\ x : \tau, Ctx$$

Now let *Sig* be a signature, *Ctx* a context, and S a proof. Furthermore, Γ and Δ denote sequences consisting of named formulas $N : F$, abbreviations $c \equiv t$ and substitutions $x \leftarrow t$. The judgments are:

– $Sig; Ctx; \Gamma \langle S \rangle \Delta \hookrightarrow \Gamma'$
  Given the signature Sig, the context Ctx, the hypotheses in Γ and the open goals in Δ, the (partial) proof S derives the facts Γ'.
– $P(\Gamma'; \Delta') : \Gamma \Longrightarrow_{\mathsf{strat}} \Delta$
  The proof strategy strat proves the lemma $\Gamma \Longrightarrow_{\mathsf{strat}} \Delta$ and returns the proof object $P(\Gamma'; \Delta')$. This notation for the proof object indicates that the proof requires the subsequences Γ' and Δ' of Γ and Δ, respectively.

The deductive system for proof checking is given in Fig. 4, where an expression $\overline{e}$ means a sequence of expressions $e_1, \ldots, e_n$ and $\uplus$ stands for the disjunctive union. Note that we explicitly refrain from fixing a specific logic in these rules, as we envision the use of our language in different domains. Therefore, we parameterize the proof checking system over the calculus for the specific logic. The connection to the calculus is established via the strategies for the lemmas arising during proof checking.

The individual kinds of lemmas are Triv to establish proof steps considered as trivial by the author, Fact to ensure the validity of derived facts, $\mathsf{Subgoal}$ to prove valid goal reductions, Ass to show that the stated fact can be inferred from some assumptions, Type to verify type correctness, and Subst to ensure the admissibility of a substitution.

To illustrate some of the lemmas arising during proof checking, let Sig_G denote the general signature, which contains, among others, the symbols $const\ A : \mathsf{set}$, $const\ B : \mathsf{set}$, $const\ C : \mathsf{set}$, $const\ \cup : \mathsf{set} \times \mathsf{set} \to \mathsf{set}$, $const\ \cap : \mathsf{set} \times \mathsf{set} \to \mathsf{set}$, and $const\ \subseteq : \mathsf{set} \times \mathsf{set} \to o$. The initial context Ctx is empty, Γ_G denotes the logical context, in which the proof takes place, and contains, among others, the Definition 1.1.1, the definitions of $\cup$ and $\cap$ ($\mathsf{Def}(\cup)$, $\mathsf{Def}(\cap)$), and the abbreviating notation $\subseteq$, which we denote by $\mathsf{Abbrv}(\subseteq)$[2]. Furthermore, let Δ_G be the conclusion $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ of Theorem 1.1.4.

In order to check Step 1 of the proof, the rule Assume requires to first check the subproof 1.1–1.5 where the context is augmented by the local variable x of type elem and the local set of assertions contains the hypothesis $. : x \in A \cap (B \cup C)$. The proof checking of Step 1.1 is then as follows: Assume S' are the proof steps 1.2-1.5, then the Fact proof checking rule is invoked as

$$Sig_G; const\ x : \mathsf{elem}; \Gamma_G, . : x \in A \cap (B \cup C) \quad \langle Fact\ \ . : x \in A \wedge x \in (B \cup C)\ \ by\ \ .\ \ from\ \ .; S' \rangle\ \Delta_G$$

Checking that proof step requires to check the following judgments (cf. the rule Fact from Fig. 4):

1. We first have to check whether the given fact formula is actually derivable from the current assertions ($\Gamma_G, . : x \in A \cap (B \cup C)$) and the current goal ($\Delta_G$). We pass the corresponding lemma to the strategy (indicated by $\Longrightarrow_{\mathsf{Fact}}$) which we use to establish that lemma. The lemma is:

$$\Gamma_G, . : x \in A \cap (B \cup C) \Longrightarrow_{\mathsf{Fact}} . : x \in A \wedge x \in (B \cup C), \Delta_G$$

[2] Assuming, for instance, a higher-order logic language for formulas and terms, the abbreviation $\subseteq$ could be written as $\subseteq \equiv \lambda A, B : \mathsf{set}\ .\ \forall x : \mathsf{elem}\ .\ x \in A \supset x \in B$.

$$\frac{}{Sig;\, Ctx;\, \Gamma\ \langle \epsilon \rangle\ \Delta \hookrightarrow .}\ \textsf{Empty}$$

$$\frac{P(.; N : F) : \Gamma \Longrightarrow_{\textsf{Triv}} \Delta}{Sig;\, Ctx;\, \Gamma\ \langle Trivial \rangle\ \Delta \hookrightarrow N : F}\ \textsf{Trivial}$$

$$\frac{Sig;\, Ctx;\, \Gamma \left\langle S_i, S' \right\rangle \Delta \hookrightarrow \Gamma_i'}{Sig;\, Ctx;\, \Gamma \left\langle Or(S_1 \parallel \ldots \parallel S_n); S' \right\rangle \Delta \hookrightarrow \Gamma_i'}\ \textsf{Or}$$

$$\frac{\begin{array}{l} var\ x : \tau \in Ctx \\ Sig;\, Ctx \Longrightarrow_{\textsf{Type}} t : \tau \\ P : \Gamma \Longrightarrow_{\textsf{Subst}} x \leftarrow t, \Delta \\ Sig;\, Ctx;\, \Gamma, . : x \leftarrow t\ \langle S \rangle\ \Delta \hookrightarrow \Gamma' \end{array}}{Sig;\, Ctx;\, \Gamma\ \langle Assign\ \ var\ x := t; S \rangle\ \Delta \hookrightarrow x \leftarrow t, \Gamma'}\ \textsf{Assign}$$

$$\frac{\begin{array}{l} P(\Gamma_R, \tilde{\Gamma}; N : F, \Delta_R, \tilde{\Delta}) : \Gamma \Longrightarrow_{\textsf{Fact}} N : F, \Delta \\ Sig;\, Ctx;\, \Gamma, N : F\ \langle S \rangle\ \Delta \hookrightarrow \Gamma' \end{array}}{Sig;\, Ctx;\, \Gamma \left\langle Fact\ \ N : F\ \ by\ \ \overline{R}\ \ from\ \ \overline{R'}; S \right\rangle \Delta \hookrightarrow N : F, \Gamma'}\ \textsf{Fact}$$

where $\Gamma_R \uplus \tilde{\Gamma} \subseteq \Gamma$, $\Delta_R \uplus \tilde{\Delta} \subseteq \Delta$ and Γ_R and Δ_R are respectively the subsets of Γ and Δ denoted by $\overline{R}$ and $\overline{R'}$.

$$\frac{\begin{array}{l} Sig;\, Ctx \Longrightarrow_{\textsf{Type}} t : \tau \\ const\ c : \tau \notin Sig \\ Sig;\, Ctx;\, \Gamma, c \equiv t\ \langle S \rangle\ \Delta \hookrightarrow \Gamma' \end{array}}{Sig;\, Ctx;\, \Gamma\ \langle Assign\ const\ c := t; S \rangle\ \Delta \hookrightarrow \Gamma'}\ \textsf{Abbrv}$$

$$\frac{\begin{array}{l} P(\Gamma_R, \tilde{\Gamma}, . : (F_1 \wedge \ldots \wedge F_k); N : F, \Delta_R, \tilde{\Delta}) \\ \quad : \Gamma, . : (F_1 \wedge \ldots \wedge F_k) \Longrightarrow_{\textsf{Subgoal}} N : F, \Delta \\ Sig;\, Ctx;\, \Gamma\ \langle S_1 \rangle\ N_1 : F_1, \Delta \hookrightarrow \Gamma_1 \\ \ldots \\ Sig;\, Ctx;\, \Gamma\ \langle S_k \rangle\ N_k : F_k, \Delta \hookrightarrow \Gamma_k \\ Sig;\, Ctx;\, \Gamma, N : F \left\langle S' \right\rangle \Delta \hookrightarrow \Gamma' \end{array}}{\begin{array}{l} Sig;\, Ctx;\, \Gamma \left\langle \begin{array}{l} Subgoals\ \ N_1 : F_1 \mid \ldots \mid N_k : F_k \\ \quad in\ \ S_1 \mid \ldots \mid S_k \\ \quad to\ obtain\ \ N : F\ \ by\ \ \overline{R}\ \ from\ \ \overline{R}'; S' \end{array} \right\rangle \Delta \\ \hookrightarrow N : F, \Gamma' \end{array}}\ \textsf{Subgoals}$$

where $\Gamma_R \uplus \tilde{\Gamma} \subseteq \Gamma$, $\Delta_R \uplus \tilde{\Delta} \subseteq \Delta$ and Γ_R and Δ_R are respectively the subsets of Γ and Δ denoted by $\overline{R}$ and $\overline{R'}$.

$$\frac{\begin{array}{l} \overline{x_k} \notin Ctx \\ \overline{c_l} \notin Sig \\ S, \overline{const\ c_l : \tau_l}; C, \overline{var\ x_k : \tau_k}; \Gamma, \overline{N_m : F_m}\ \langle S \rangle\ \Delta \hookrightarrow \Gamma_s \\ P(\Gamma_R, \Gamma', . : (\bigwedge_{i=1}^{m} F_i) \supset (\bigwedge_{F \in \Gamma_s} F); N : F, \Delta_R, \Delta') : \Gamma, . : (\bigwedge_{i=1}^{m} F_i) \supset (\bigwedge_{F \in \Gamma_s} F) \Longrightarrow_{\textsf{Ass}} N : F, \Delta \\ Sig;\, Ctx;\, \Gamma, . : (\bigwedge_{F \in \overline{F_m}^{\overline{var\ x_k}}} F) \supset (\bigwedge_{F \in \Gamma_s^{\overline{var\ x_k}}} F), N : F \left\langle S' \right\rangle \Delta \hookrightarrow \Gamma' \end{array}}{\begin{array}{l} Sig;\, Ctx;\, \Gamma \left\langle \begin{array}{l} Assume\ \overline{var\ x_k : \tau_k}, \overline{const\ c_l : \tau_l}, \overline{N_m : F_m}\ \ in\ \ S \\ \quad to\ obtain\ N : F\ \ by\ \overline{R}\ \ from\ \overline{R}; S' \end{array} \right\rangle \Delta \hookrightarrow . : (\bigwedge_{F \in \overline{F_m}^{\overline{var\ x_k}}} F) \supset (\bigwedge_{F \in \Gamma_s^{\overline{var\ x_k}}} F), N : F, \Gamma' \end{array}}\ \textsf{Assume}$$

where Γ_R (resp. Δ_R) is the subset of Γ (resp. Δ) denoted by $\overline{R}$ and $\overline{R}'$ and $\Gamma_s^{\overline{var\ x_k}}$ (resp. $\overline{F_m}^{\overline{var\ x_k}}$) is the subset of formulae from Γ_s (resp. $\overline{F_m}$), which do not contain free variables from $\overline{var\ x_k}$ (i.e., *Assume* serves as a binder for the variables $\overline{var\ x_k}$).

Fig. 4. Proof Checking Rules

From the strategy (for instance a prover or a tactic) we require not only to prove that lemma, but also to return a proof object P. From that proof object we require that it must "rely" on the named formulas provided by the references in the "by" and "from" (denoted as Γ_R and Δ_R) slots as well as on the given fact ($. : x \in A \wedge x \in (B \cup C)$). The fact that they must occur in the proof object is indicated by making them part of the arguments of P, i.e.

$$P(\Gamma_R, \tilde{\Gamma}; , \Delta_R, \tilde{\Delta})$$

In the present case there are no references, and hence Γ_R and Δ_R are empty. The list $\tilde{\Gamma}$ (resp. $\tilde{\Delta}$) denotes all further assertions from Γ_G (resp. goals from Δ_G) on which the proof object P relies. Those lists provide us with the missing references. In our case the proof object is $P(\mathsf{Def}(\cap), . : x \in A \cap (B \cup C); . : x \in A \wedge x \in B \cup C)$ which provides the non-specified reference to the definition of $\cap$.

2. After checking the lemma, the proof checking recurs over the proof steps S' by adding the new fact to the list of usable assertions, which is expressed by $Sig_G; Ctx_G, const\ x : \mathsf{elem}; \Gamma_G, . : x \in A \wedge x \in (B \cup C) \langle S' \rangle \Delta_G$. This returns a list of facts Γ' derived in that subproof S', which is denoted by $\hookrightarrow \Gamma'$. In our case Γ' is $. : x \in (A \cap B) \cup (A \cap C), . : (x \in A \cap B) \vee (x \in A \cap C), . : (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C), . : x \in A \wedge (x \in B \vee x \in C)$ (in that order).

From that last list the result of the proof checking of the fact Step 1.1 are all the named formulas derived in the subproof S' plus $. : x \in A \wedge x \in B \cup C$. This is expressed by $\hookrightarrow N : F, \Gamma'$, which in the present case is $. : x \in A \wedge x \in B \cup C, . : x \in (A \cap B) \cup (A \cap C), . : (x \in A \cap B) \vee (x \in A \cap C), . : (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C), . : x \in A \wedge (x \in B \vee x \in C)$. This list is the result of the subproof inside the *Assume* part and we denote this list by Γ_s. Validating the *Assume* step then requires to prove the lemma

$$\Gamma_G, . : x \in A \cap (B \cup C) \supset \begin{pmatrix} . : x \in A \wedge x \in B \cup C \\ \wedge . : x \in (A \cap B) \cup (A \cap C) \\ \wedge (. : (x \in A \cap B) \vee (x \in A \cap C)) \\ \wedge (. : (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)) \\ \wedge . : x \in A \wedge (x \in B \vee x \in C \end{pmatrix}$$
$$\Longrightarrow_{\mathsf{Ass}} . : A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C), \Delta_G.$$

The proof object for that lemma is $P(. : x \in A \cap (B \cup C) \supset (\bigwedge_{F \in \Gamma_s} F), \mathsf{Abbrv}(\subseteq); . : A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C))$, and provides the missing references to the abbreviation of $\subseteq$ and the used premise $. : x \in A \cap (B \cup C) \supset (\bigwedge_{F \in \Gamma_s} F)$. The result of proof checking this *Assume*-proof step consists of (1) the result Γ' obtained from checking the remaining proof, (2) the obtained (named) formula $. : A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$, and (3) all formulas derived in the subproof of *Assume*, which are not dependent on any local variables. The latter is expressed by the (schematic) formula $(\bigwedge_{F \in \overline{F_m}^{\overline{var\ \ x_k}}} F) \supset (\bigwedge_{F \in \Gamma_s^{\overline{var\ \ x_k}}} F)$, where $\Gamma_s^{\overline{var\ \ x_k}}$ expresses the filtering.[3]

[3] Note that this way any substitution—expressed by an equation $x = t$—for a local variable x inside the subproof of *Assume* is also removed.

Proof checking of the next *Assume*-proof step 2—inclusive its subproof—also succeeds using the rules in Fig. 4. The set of derived facts up to before Step 3 is

- From *Assume*-proof step 1 we obtain $\Gamma_1 := . : A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C), \Gamma_G$.
- From *Assume*-proof step 2 we obtain $\Gamma_2 := . : (A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C), \Gamma_{1.1}$.

Checking the *Fact*-proof step 3 requires to establish the lemma

$$\Gamma_2 \Longrightarrow_{\mathsf{Fact}} . : A \cap (B \cup C) = (A \cap B) \cup (A \cap C), \Delta_G$$

Furthermore, we require from the proof of this lemma which is returned by the strategy, that (i) it uses the goal formula $. : A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ and (ii) it uses the indicated Definition 1.1.1. This is expressed by

$$P(\text{Def1.1.1}, \tilde{\Gamma}; . : A \cap (B \cup C) = (A \cap B) \cup (A \cap C), \Delta_R, \tilde{\Delta})$$

In this case, the used facts from Γ_2 are those obtained in the *Assume*-proof steps, i.e. $\tilde{\Gamma}$ is $. : A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C), . : (A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$ (note that $\tilde{\Delta}$ is empty). Checking the final *Trivial*-proof step reduces to establish the lemma

$$\Gamma_2, . : A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \Longrightarrow_{\mathsf{Triv}} \Delta_G$$

The lemma is trivially provable, since $\Delta_G := \text{Thm}(1.1.4) : A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$, which completes the proof checking of our example proof.

5 Granularity

In this section we investigate the problem how we can check that a proof is at a specific level of granularity. Our focus here is how at all notions of granularity could be captured formally, in order to design proof procedures that not only check the correctness of a (partial) proof, but also if it is at a given level of granularity. Knowing how granularity can be defined is a necessary prerequisite before we can move on to analyze which level of granularity is appropriate in which context and how it could be made user-adaptive.

In general, proof sketches can be at some specific, appropriate level of granularity, without being correct proofs. Conversely, a proof sketch can be correct, but not at a desired level of granularity. In other words, the notion of granularity and correctness do only overlap, but there is not necessarily a subsumption relation in whichever direction. Comparing both notions with respect to the existing means for their formalization, there is a long tradition and a large class of formalisms to represent correct proofs — one of them has been presented in the previous section. On the contrary, there are to our knowledge neither formalizations to check whether a proof is at some specific level of granularity (aside

from, for instance, being a calculus-level proof), nor any other means to reduce the inspection of the granularity to computation.

As we pointed out earlier, there is an overlap between correctness and granularity. In this section we consider this overlap, since there we can hope to exploit the available formalisms for correctness and adapt them to accommodate some notion of granularity. However, it is not obvious a priori, whether any notion of granularity can be captured simply by refining the notion of correctness. With the work presented in the following we explore some aspects of that problem by defining a notion of granularity through restricting the notion of correctness.

The set of proofs which are accepted by the proof checking rules strongly depends on the strength of the strategies used to discharge the arising lemmas *and* on the knowledge, that is, facts and subgoals, the strategies can use during the proof attempt. However, the strength of the strategies can not be controlled at the level of our proof representation language. Moreover, finding and implementing the *right* strategies is difficult as it strongly depends on the individual authors. The only information we can control is which knowledge is actually passed to the strategies, which in turn can be influenced via two criteria: (1) The selection of *locally* available knowledge for the strategies and (2) the *global* flow of knowledge between different parts of the proof. So far in the proof checking rules (1) all locally visible assumptions and subgoals are used, and (2) all possible derived facts from earlier proof steps are passed to subsequent proof steps.

We now show how a specific, intuitive notion of granularity can be formalized by imposing restrictions for (1) and (2). The notion of granularity we define here is inspired by the assertion level proofs by Huang [6] and by a description of what a proof is by Hilbert [5]. This level of granularity can intuitively be described by *What-You-Need-Is-What-You-Stated Granularity*, that is, all necessary facts, assertions and rules are stated explicitly in the proof and the proof is performed at the assertion level. For instance, when we use some fact, we should have stated it before explicitly and not assume it is inferable from the context.

To formally define granularity, we introduce the judgment

$$\mathit{Sig};\ \mathit{Ctx};\ \Gamma\ \langle S\rangle\ \Delta \triangleright \Gamma'$$

which intuitively means that given the signature *Sig*, the context *Ctx*, the hypotheses in Γ and the open goals in Δ, the proof S derives the facts Γ' at the given level of granularity.

The deductive system to check the granularity (cf. Fig. 5) is derived from the proof checking system, by imposing restrictions with respect to criterion (1) for Trivial, Fact, and Subgoals proof steps, and restrictions with respect to both criteria (1) and (2) for Assume proof steps.

For the Fact and Subgoals proof steps, we restrict the rules by selecting from Γ and Δ only those formulas that are explicitly referenced in the proof step description, as well as all substitutions $x \leftarrow t$ and abbreviations $c \equiv t$. Furthermore, we require that the proof object P returned by the strategy strat *rely* on

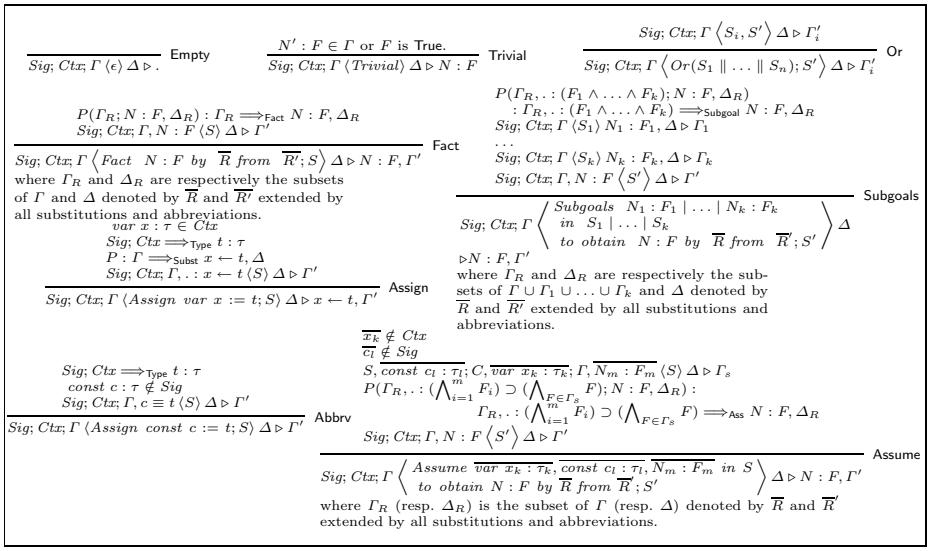

Fig. 5. *What-You-Need-Is-What-You-Stated* Granularity Checking Rules

all the referenced formulas, that is, the deletion of any of these formulas renders the lemma unprovable.

For instance, the Fact and Subgoals rules are strengthened by requiring that the references $\overline{R}$ and $\overline{R'}$ denote *exactly* the set of assumptions and conclusions necessary to derive the new fact. Note that this requires all references to be defined.

The Assume rule is strengthened similarly, but in addition we restrict the information flow to subsequent proof parts by omitting the formula $. : (\bigwedge_{i=1}^{m} F_i) \supset \Gamma_s^{\overline{var\ x_k}}$, as it is only implicitly known and not explicitly stated, and thus violates the intuitive *What-You-State-Is-What-You-Need* condition.

The Trivial rule is restricted by removing the call to the Triv strategy. Instead we require that either there is a trivially valid formula $N : F$ in the accumulated goals or one of the goal formulas occurs as an assumption. This makes this rule analogous to an Axiom rule in a sequent-style calculus.

Let us consider again our example proof to illustrate the checking of the granularity. If we check the granularity of the proof object given in Fig. 3, the check fails. For instance, in Step 1.1, a reference to the definition of $\cap$ is missing, and in Step 3 the references to the two derived subgoals are missing. Note that in Step 1 and 2 no information about $\subseteq$ is needed, since Bartle and Sherbert [3] introduced $\subseteq$ as an abbreviating *notation* and not as a defined concept, and abbreviations are globally visible to prove the lemmas arising during granularity check.

Patching the proof object from Fig. 3 by including the missing references as suggested by the granularity checker consists of: (1) the inclusion of the references to the definitions of $\cup$ and $\cap$ for all but the last *Fact* proof steps, and (2) for the last *Fact* proof step the inclusion of the references to the used premises.

In order to give a sample proof, we shall prove the first equation in (d). Let x be an element of $A \cap (B \cup C)$, then $x \in A$ and $x \in B \cup C$ ***by the definition of*** $\cap$. This means that $x \in A$, and either $x \in B$ or $x \in C$ ***by the definition of*** $\cup$. Hence we either have (i) $x \in A$ and $x \in B$, or we have (ii) $x \in A$ and $x \in C$ ***by the distributivity of "and" over "or"***. Therefore, either $x \in A \cap B$ or $x \in A \cap C$ ***by the definition of*** $\cap$, so $x \in (A \cap B) \cup (A \cap C)$ ***by the definition of*** $\cup$. This shows that $A \cap (B \cup C)$ is a subset of $(A \cap B) \cup (A \cap C)$. ***(1)***

Conversely, let y be an element of $(A \cap B) \cup (A \cap C)$. Then, either (iii) $y \in A \cap B$, or (iv) $y \in A \cap C$ ***by the definition of*** $\cup$. It follows that $y \in A$, and either $y \in B$ or $y \in C$ ***by the definition of*** $\cap$ ***and the distributivity of "and" over "or"***. Therefore, $y \in A$ and $y \in B \cup C$ ***by the definition of*** $\cup$ so that $y \in A \cap (B \cup C)$ ***by the definition of*** $\cap$. Hence $(A \cap B) \cup (A \cap C)$ is a subset of $A \cap (B \cup C)$. ***(2)***

In view of Definition 1.1.1, we conclude ***from (1) and (2)*** that the sets $A \cap (B \cup C)$ and $(A \cap B) \cup (A \cap C)$ are equal.

Fig. 6. The patched textbook proof example.

Exploiting the relationship of the proof steps to individual sentences in the text, we can propagate the additional information back into the textual representation. The resulting proof is shown in Fig. 6, where the added text fragments are set in ***boldface italics*** font.

6 Conclusion

In this paper, we presented a calculus-independent formal representation language for human-authored proofs and two deductive systems that allow for checking the correctness and the level of granularity of the proofs. This formal language mediates between the semantically-annotated natural language representation of the mathematical document in a scientific text editor, where the user enters his input, and the logic representation required by proof assistance systems that check the mathematical content of the documents and fill in gaps. Using an example textbook proof, we showed how this proof is represented in our language and checked its correctness and granularity. We based a first notion of granularity on Hilbert's approach that demands that everything that is needed in the proof must be stated explicitly. This, however, resulted in the failure of the granularity check of the example textbook proof, such that a patch was required that added missing references to used definitions and previously derived facts.

Putting the patched proof under scrutiny, we see that it is now easier to follow the line of reasoning in the proof. However, one could argue that the proof now contains too many details. Therefore, we plan to enhance our notion of granularity in order to allow for implicit references as well. To do so, we need a flexible model of granularity that captures when a reference must be explicit and when it can be implicit, which can only be obtained via empirical studies.

In addition to granularity, a notion of conciseness would be desirable, which could be used to check if a proof is more involved than necessary. It is unclear, though, how such a notion could be captured formally.

Yet, we envision the approach presented in this paper to be a first step towards an integration of a scientific text editor with mathematical proof assistance systems. In particular, the deductive systems show how mathematical proof assistance systems can be employed via the strategies to prove raised lemmas. To achieve the overall goal, however, many additional problems must be tackled, among them the connection to a scientific text editor, and, most notably, a natural language analysis component that transforms the human-authored proofs into proofs in our representation language.

References

1. Andreas Abel, Bor-Yuh Evan Chang, and Frank Pfenning. Human-readable machine-verifiable proofs for teaching constructive logic. In Uwe Egly, Armin Fiedler, Helmut Horacek, and Stephan Schmitt, editors, *Proc. of the Workshop on Proof Transformation, Proof Presentations and Complexity of Proofs (PTP-01)*, pages 37–50. Universitá degli studi di Siena, 2001.
2. Serge Autexier, Christoph Benzmüller, Armin Fiedler, and Henri Lesourd. Integrating proof assistants as reasoning and verification tools into a scientific WYSIWYG editor. In David Aspinall and Christoph Lüth, editors, *User Interfaces for Theorem Provers (UITP 2005)*, pages 16–39, 2005.
3. Robert G. Bartle and Donald Sherbert. *Introduction to Real Analysis*. Wiley, 2 edition, 1982.
4. N. G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In Nederpelt et al. [9], pages 865 – 935.
5. David Hilbert. Die Grundlegung der elementaren Zahlenlehre. *Mathematische Annalen*, 104:485–494, December 1930.
6. Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. Number 112 in DISKI. Infix, Sankt Augustin, Germany, 1996.
7. Fairouz Kamareddine, Manuel Maarek, and Joe Wells. MathLang: An experience driven language of mathematics. *Electronic Notes in Theoretical Computer Science*, (93C):138–160, 2004.
8. Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. *Logic, Language and Information*, 13(3):287–340, 2004.
9. R. P Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 1994.
10. Rob Nederpelt. Weak Type Theory: A formal language for mathematics. Computing Science Report 02-05, Eindhoven University of Technology, Department of Math. and Comp. Sc., May 2002.
11. Tobias Nipkow. Structured Proofs in Isar/HOL. In Herman Geuvers and Freek Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, volume 2646 of *LNCS*, pages 259–278. Springer, 2003.
12. Aarne Ranta. Grammatical framework — a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189, 2004.
13. Jörg Siekmann, Christoph Benzmüller, Vladimir Brezhnev, Lassaad Cheikhrouhou, Armin Fiedler, Andreas Franke, Helmut Horacek, Michael Kohlhase, Andreas Meier, Erica Melis, Markus Moschner, Immanuel Normann, Martin Pollet, Volker Sorge, Carsten Ullrich, Claus-Peter Wirth, and Jürgen Zimmer. Proof development

with ΩMEGA. In Andrei Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE-18)*, number 2392 in LNAI, pages 144–149, Copenhagen, Denmark, 2002. Springer.
14. M. Wenzel and F. Wiedijk. A comparison of the mathematical proof languages Mizar and Isar. *Journal of Automated Reasoning*, (29):389–411, 2002.