Towards the Mechanical Verification of Textbook Proofs

CLAUS ZINN, *Division of Informatics, University of Edinburgh, 2 Buccleuch Place, Edinburgh, EH8 9LW, Scotland, UK.*
E-mail: zinn@cogsci.ed.ac.uk

Abstract

Our goal is to implement a program for the machine verification of textbook proofs. We study the task from both the linguistics and automated reasoning perspective and give an in-depth analysis for a sample textbook proof. We propose a framework for natural language proof understanding that extends and integrates state-of-the-art technologies from Natural Language Processing (Discourse Representation Theory) and Automated Reasoning (Proof Planning) in a novel and promising way, having the potential to initiate progress in both of these disciplines.

Keywords: text understanding, proof verification, discourse representation theory, proof planning

1 Motivation

In [18], John McCarthy notes that *"Checking mathematical proofs is potentially one of the most interesting and useful applications of automatic computers"*. In the first half of the 1960s, one of his students, Paul Abrahams, implemented a Lisp program for the machine verification of mathematical proofs [1]. The program, named `Proofchecker`, *"was primarily directed towards the verification of textbook proofs, i.e., proofs resembling those that normally appear in mathematical textbooks and journals"*. Abrahams soon revised his goal. If *"a computer were to check a textbook proof verbatim, it would require far more intelligence than is possible with the current state of the programming art"*. Therefore, *"the user must create a rigorous, i.e., completely formalised, proof that he believes represents the intent of the author of the textbook proof, and use the computer to check this rigorous proof"*. Abrahams points further out that *"it is a trivial task to program a computer to check a rigorous proof; however, it it not a trivial task to create such a proof from a textbook proof"*. Abrahams was right. In his implementation and in all later projects, proofs had to be written in a *formal language* using a *restricted set of proof construction commands* in order to verify them. A human user is required to perform the formalisation task.

Basically, two main approaches towards the formalisation and verification of proofs were taken. In the first, the `Automath/Mizar` approach [20, 24], the user is required to give a *full* and *explicit* construction of a formal proof. The proof component then checks the proof for correctness (the compiler approach). A well-known example of a large formalisation is van Benthem-Jutting's translation of Landau's 'Grundlagen der Analysis' into `aut-qe`, one of the formalisms of the `Automath`-language family [26]. The `Mizar` system offers, besides the rich formal language and the proof checker component, a large library of formalised mathematics which allows one to formalise proofs without excessive preparatory work.

In the second approach, the user constructs the proof interactively with the proof system (the interpreter approach). The user has a set of proof construction commands at hand and asks the system to apply them. The system keeps track of the proof obligations and guarantees that the constructed proof is correct. A well-known example of a proof system of this kind is `Nuprl` [8].
For both approaches, proof planning technology could be used to allow the generation of high-level proofs [4]. During proof construction, instead of carrying out elementary steps at the inference level, *tactics* are applied. A tactic is a program that describes the application of a sequence of inference rules. Tactics are formally described by *methods* which specify the preconditions that must hold for a tactic to be carried out. The `Oyster/Clam` system [5] implements an AI plan formation algorithm which seeks to construct a proof plan by searching a tree of methods that combine to prove a given conjecture. These proof plans lead to formal proofs if each method can successfully execute its associated tactic.

Our goal is to mechanise the machine verification of textbook proofs. In the process, we must answer two questions: What formal representation can be obtained from textbook proofs? And what is needed to formally verify these representations? Building a program that checks informal proofs allows us to study natural language understanding in conjunction with automated mathematical reasoning.
From the linguistics point of view, verifying textbook proofs is a text understanding task. The text genre, however, has a number of characteristics that makes this task feasible. The art of writing good mathematical texts focuses on clearness and conciseness and not on an embellished style of expression [27]. The expert language used by mathematicians is stylised, that is, characterised by the use of standard phrases and keywords [25]. The use of terms and formulae introduce formal parts into mathematical discourse. In addition, textbook proofs are, in general, a highly structured form of discourse. A crucial prerequisite to understanding the course of the argumentation within a proof is to identify the discourse relations (e.g., logical consequence) between sentences of that discourse. Deriving the discourse relations of a given textbook proof means reconstructing the intentional structure (describing how sentences within a discourse segment contribute to a common discourse purpose) and the informational structure (describing how sentences within a segment are related to each other by some relation) of the proof [19].
From the automated reasoning point of view, verifying textbook proofs is a deductive task where one must identify the logical structure of the proof: identifying assumptions and conclusions, the scope and quantification of variables, and substructures which themselves form subproofs. Identifying the proof plan of a textbook proof is a prerequisite for verification. It encapsulates informal mathematical reasoning, and therefore, allows us to follow, as close as possible, the proof that was expressed *informally* in the natural language argument. Often, the form of the theorem presupposes possible proof plans and the concepts it contains often hint to definitions being used in the proof. A proof understander must offer a high-level proof analysis as well as a technical low-level access to details of the proof. Gaps and flaws must be recognised.

2 Example

Fig. 1 depicts LeVeque's existence proof of the fundamental theorem of arithmetic [16]. We analyse this proof from both the linguistic and mathematical point of view.

THEOREM 2-2. Every integer $a > 1$ can be represented as a product of one or more primes.

Proof: The theorem is true for $a = 2$. Assume it to be true for $2, 3, 4, \dots, a-1$. If a is prime, we are through. Otherwise a has a divisor different from 1 and a, and we have $a = bc$, with $1 < b < a$, $1 < c < a$. The induction hypothesis then implies that

$$b = \prod_{i=1}^{s} p'_i, \quad c = \prod_{i=1}^{t} p''_i,$$

with p'_i, p''_i primes and hence $a = p'_1 p'_2 \dots p'_s p''_1 \dots p''_t$.

FIG. 1. A textbook proof taken from [16]

2.1 Linguistic analysis

All kinds of linguistic phenomena which can occur in other text genres also occur in textbook proofs [29]. We discuss a selection of problems for the proof at hand.
A prerequisite for parsing textbook proofs is to be able to parse formulae that occur in these proofs. Parsing formulae in isolation is trivial. Problems arise if the textual context has to be taken into account, and when references from the text to formulae and their parts need to be resolved. This is because constants and variables have a domain and scope which extends across text and formulae. A simple interaction between text and formulae is 'every integer $a > 1$'. This construction makes 'every integer' and 'a' denote the same object.
Assigning names to entities plays an important role in mathematical writing. Having names available for variables enables us to show explicitly and concisely which part of an argument depends on other parts of the argument, and which part does not. This allows for disentanglement and decomposition of the argument. However, the improper use of names can also be a source of confusion. An example is LeVeque's proof. The theorem contains the named variable a. However, for the theorem only, it is not necessary to name this variable — it is used only once. Unfortunately, in LeVeque's proof, the name a is used again – suggesting that both occurrences of a refer to the same mathematical entity. Of course, this is not true. The a in the theorem is a universally quantified variable (its denotation is supposed to vary), the a in the proof is a free variable (it denotes a fixed, unknown object). In informal mathematics, frequently, the difference between bound and free variables is blurred.
Other examples of text–formula interactions include (i) 'a divisor' and the introduction of 'b' and 'c' in the proof; and (ii) the recognition that 'with p'_i, p''_i primes' is not about two individual primes but, taking the context introduced by the $\prod$ formulae into account, reads as two universally quantified statements.
The discourse in Fig. 1 contains three obvious referential expressions: 'the theorem', 'it' and 'the induction hypothesis', all of which are of propositional type. The propositional type is either induced by the verb phrase 'is true for' – only propositions can be true; or by 'implies' – only propositions can imply other propositions.
Mathematical discourse is full of conditional claims, and the discourse in Fig. 1 is no exception. The proof contains a number of clue words (e.g., 'if', 'otherwise', 'assume',

'implies', 'hence'), and cue phrases (e.g., 'is true for', 'we are through') that indicate its logical structure. However, to mechanically reconstruct the logical structure of an informal proof, the use of these cues is not sufficient. Consider LeVeque's proof. The third proof sentence, starting with 'if', could be read as follows: introduce the assumption 'a is prime' and conclude 'we are through' (signalling proof termination). However, this sentence has to be embedded into a larger context. In fact, 'if' introduces a proof per cases construct, where 'a is prime' defines the first case, and where the cue phrase 'we are through' terminates this case segment — discharging the assumption. The cue phrase 'otherwise' is an elliptical construction. It initiates the second case and implicitly introduces the assumption 'a is not prime'. The whole remainder of the proof depends on this assumption. The last clause of the discourse 'hence $a = p'_1 p'_2 \dots p'_s p''_1 \dots p''_t$' contains the discourse marker 'hence'. Unfortunately, the use of 'hence' does not allow us, linguistically, to identify all the premises (and the inference rules) necessary to conclude '$a = p'_1 p'_2 \dots p'_s p''_1 \dots p''_t$'.

2.2 Mathematical analysis

The theorem of Fig. 1 is of the form *For every integer $n \geq n_0$, $P(n)$ holds.* To prove theorems of this logical form, a proof per *Nœtherian induction* (also called *Generalised induction*) can be performed: (i) assume $P(i)$ for $n_0 \leq i < n$; (ii) show that $P(n)$ holds. Knowing this proof method, it is easy to understand LeVeque's proof. The necessity for the first sentence will be explained later. The second sentence states the induction hypothesis. The remainder of the proof constitutes the induction step which itself is structured as a proof per cases: either a is prime or a is not prime. The third proof sentence initiates and terminates the first case. In the second case, forward reasoning from the assumption *a is not prime* takes place. That a has a divisor different from 1 and a follows from definitional expansion of the concept of prime and a proper treatment of negation. Continuing forward reasoning, the statements $a = bc$, $1 < b < a$, and $1 < c < a$ are obtained by definitional expansion of the concept of divisor. The representation of a and b as a product of primes is obtained by applying the induction hypothesis twice (for b using $1 < b < a$, for c using $1 < c < a$). Rewriting $a = bc$ by the product of prime representation of b and c terminates the proof. — The proof makes use of the following definitions and lemmatas:

$$\begin{aligned}
\forall n \in \mathbb{N} : prime(n) &\iff n > 1 \wedge \forall d \in \mathbb{N} : d|n \rightarrow d = 1 \vee d = n \\
\forall a \in \mathbb{N} : \forall b \in \mathbb{N} : a|b &\iff \exists c \in \mathbb{N} : b = ac \wedge a \leq b \wedge c \leq b \\
\forall n \in \mathbb{N} : prod_primes(n) &\iff prime(n) \vee \exists p_1, p_2 \in \mathbb{N} : n = p_1 p_2 \wedge p_1 < n \wedge p_2 < n \\
&\qquad \wedge prod_primes(p_1) \wedge prod_primes(p_2)
\end{aligned}$$

$$\forall a \in \mathbb{N} : \forall b \in \mathbb{N} : a \leq b \wedge \neg(a = b) \rightarrow a < b$$
$$\forall a \in \mathbb{N} : \forall b \in \mathbb{N} : \forall c \in \mathbb{N} : a = bc \wedge \neg(b = 1) \rightarrow c < a$$
$$\forall a \in \mathbb{N} : \forall b \in \mathbb{N} : \forall c \in \mathbb{N} : a = bc \wedge \neg(a = b) \rightarrow c > 1$$

In a more rigorous analysis, we used λ-Clam, a higher-order logic version of the Clam proof planner [5] to construct a corresponding proof plan. This deeper analysis reveiled a number of steps that have no equivalent in the textbook proof.

3 Architecture

A number of issues influenced the design of our architecture: (i) the semantics of an informal mathematical discourse is its corresponding formal proof; (ii) textbook proofs are a highly structured form of discourse; (iii) the recognition of discourse structure is essential from both the linguistic and mathematical perspective; (iv) proof plans allow us to capture the high-level reasoning of informal proofs and impose proof structure; (v) semantics construction must be supported by a pragmatic component, namely the proof planner which has mathematical and meta-mathematical knowledge.

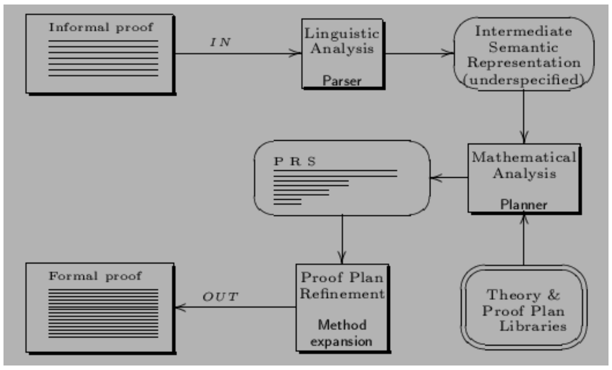

FIG. 2. Architecture of VIP

Fig. 2 depicts the architecture of our system for verifying informal proofs, VIP. VIP gets as input an informal proof. For each sentence of this proof, the **parser** performs a linguistic analysis which results in a set of intermediate semantic representations. These representations are *underspecified* since the grammar underdetermines the content of certain expressions. For example, they contain referential expressions that have to be resolved by subsequent processing since most of the expressions that occur in this intermediate semantic representation are anaphoric. The **proof planner** is the *pragmatics* component; its role is to replace underspecified conditions generated by the grammar with more complete information. The planner incorporates domain knowledge (a library of formalised mathematics) and knows about the uses of argument in mathematical discourse (a library of formalised mathematical reasoning). For each linguistic analysis produced by the parser, the planner determines its full semantic content dynamically by incorporating it in the proof context that has been established by having processed the former sentences. The proof context is represented as a proof representation structure (PRS). It is a hierarchical data structure that consists of discourse entities, discourse statements and discourse markers. Because the planner might be able to incorporate a linguistic analysis in several ways, it has to maintain a set of possible proof continuations. Some of them will be ruled out if they cannot be continued in subsequent processing. If the complete textbook proof has been processed in this manner, the resulting PRS is given to the **proof plan refiner** which expands all proof methods into inference-level steps, such that a formal proof will result. In general, a set of formal proofs will be returned. Each of these formal proofs constitutes one possible semantics for the given informal proof.

3.1 The Parser Module

We have written a definite clause grammar that covers the first proofs of [11], and the proof we discussed above. For semantics construction, our starting point was Kamp's *Discourse Representation Theory (DRT)* [13]. To deal with particular syntactic constructions that occur in mathematical proofs, we modified and added a number of DRT construction rules. For example, one characteristic of mathematical discourse is that mathematicians themselves frequently and explicitly introduce discourse referents. In syntactic constructions like 'let a be an integer > 1', 'a' functions as a discourse referent for the object being introduced by 'an integer'. We therefore defined a DRT construction rule, *naming*, that combines the semantics of 'an integer' (introducing an anonymous discourse referent) with the semantics of the name 'a' (introducing a named discourse referent that anaphorically refers to the object introduced by 'an integer'). This is a simple instance of the problem that variables have a domain and scope which extends across text and formulae.

Another example is to handle the subtle but necessary difference between bound and free variables in mathematical discourse. To compose the semantics of LeVeque's third sentence 'If a is prime, we are through'DRT would apply its construction rule for conditionals. This would result in a universally quantified logical form. However, in the proof context, a must be free. Using the syntactic structure of the sentence alone, it is not possible to determine the domain and scope of variables — one of many requirements to understand mathematical discourse.

Conditionals play a central role in mathematical discourse. In addition to 'if ... then ...', many other forms exist to express the logical relations between arguments. The large number of propositional anaphora makes it necessary to regard mathematical statements as first class citizens of semantics construction. Mathematical statements like 'hence $a = p'_1p'_2 \ldots p'_s p''_1 \ldots p''_t$' express a logical relation between the assertion object introduced by '$a = p'_1p'_2 \ldots p'_s p''_1 \ldots p''_t$' and assertion objects introduced by the former discourse. In this sense, 'therefore A' is anaphoric, i.e., it refers to all premises that are necessary to logically conclude A. Mathematically, not all previous statements of a discourse should be accessible. Proofs are a highly structured form of discourse — proofs consist of sub-proofs, and any theory of semantics must reflect this structure.

For all these reasons, we were forced to considerably modify and extend the DRT framework. First, instead of discourse representation structures, we now speak of *proof representation structures*. Second, the simple algorithm for discourse update in standard DRT (basically, a new sentence is incorporated in the discourse context by the union operator) becomes more complex. The proof planner adds a pragmatic component to the discourse update procedure.

3.2 Proof representation structures

Proof representation structures play a central role in our framework[1]. An example PRS for LeVeque's proof is given in Fig. 3 (please ignore the stars for the moment). PRSs contain discourse referents, discourse conditions and discourse markers. Discourse referents are introduced by a LET construct. Conditions are numbered and can be either assumptions or conclusions, the former are prefixed by ASSUME. Each conclusion can be accompanied by a justification list which is a list of numbers, each of which refers to the condition it prefixes. Instead of boxes, in PRS, accessibility is imposed by a numbering scheme. For example, take the condition labelled 2.2.4. It contains the terms b and a. Their quantification is determined by the closest accessible

[1] PRSs are similar in many respects to Lamport-style proof presentations [14].

LET constructs that introduce these names.

Note that the free variable a shadows the universally bound variable a. For justifying 2.2.4 the conditions 2.2.1-2.2.3 and 1. are accessible, the conditions 2.1.1 and 2.1.2 are not accessible. Discourse markers are used to provide non-number referents to either assumptions (e.g., *the induction hypothesis*) or sub-structures (e.g., *the second case*).

For ease of argumentation, the PRS of Fig. 3 is still semantically underspecified. Variables marked with '?' have scope, but no decision has yet been made about their quantification. Also, no conclusion is justified.

LET $a \in \mathbb{N}$ be universally quantified	$*^1$
THEOREM $a > 1 \rightarrow prod\_primes(a)$	$*^1$
Proof per Nœtherian induction	$*^2$
LET $a \in \mathbb{N}$ be arbitrary	$*^2$
LET $k \in \mathbb{N}$ be universally quantified	$*^2$
1. *induction_hypothesis*	$*^2$
ASSUME $k < a \rightarrow (k > 1 \rightarrow prod\_primes(k))$	$*^2$
PROVE $a > 1 \rightarrow prod\_primes(a)$	$*^2$
2. *Proof per cases*	
2.1. *first_case*	
2.1.1. ASSUME $prime(a)$	
2.1.2. $prod\_primes(a)$ BY ? (QED)	
2.2. *second_case*	
2.2.1. ASSUME $\neg prime(a)$	
LET X be ?	
2.2.2. $divisor(X, a)$ BY ?	
2.2.3. $X \neq 1$ BY ?	
2.2.4. $X \neq a$ BY ?	
LET b, c be ?	
2.2.5. $a = bc$ BY ?	
2.2.6. $1 < b < a$ BY ?	
2.2.7. $1 < c < a$ BY ?	
LET $p'_1, p'_2 \ldots, p''_1, p''_2 \ldots \in$ PRIMES be ?	
LET s, t, i be ?	
2.2.8. $b = \prod_{i=1}^{s} p'_i$ BY ?	
2.2.9. $c = \prod_{i=1}^{t} p''_i$ BY ?	
2.2.10. $a = p'_1 p'_2 \ldots p'_i p''_1 \ldots p''_t$ BY ? (QED)	

FIG. 3. A proof representation structure

3.3 The planner module

The proof planner module gets an underspecified semantic representation of an informal proof sentence and tries to incorporate it into the PRS. The proof planner consults two libraries: a library of mathematical knowledge and a library of meta-mathematical knowledge. The theory library contains domain knowledge, basically, a set of definitions and theorems. Our experience showed that it is useful to have multiple definitions for the same concept. The proof plan library contains a collection of proof plans (more precisely: proof plan schemas).[2] Quite often, the form of the theorem presupposes more than one applicable proof plan. In other cases, the theorem can often be transformed by expanding one of its concepts (definitional rewriting).

LET $n \in \mathbb{N}$ be universally quantified
PROVE $P(n)$
Proof per Nœtherian induction
LET $n \in \mathbb{N}$ be arbitrary
LET $k \in \mathbb{N}$ be universally quantified
induction hypothesis
ASSUME $k < n \rightarrow P(k)$
PROVE $P(n)$

FIG. 4. Nœtherian induction

Fig. 4 depicts one of the applicable proof plans for theorems of the form $\forall n \in \mathbb{N} : P(n)$. It introduces two variables (a free and a bound one), an assumption, a discourse marker, and a remaining proof obligation. For illustrating the PRS construction algorithm, assume an empty PRS, and that the parser returns the following representation for the first sentence of Fig. 1: $[[a, 1|a > 1] \rightarrow [\_|prod\_primes(a)]]$. Since the PRS is empty, the planner assumes it's reading the semantic representation for a theorem (i.e., it assumes a being universally quantified). It adds the enriched semantic representation into the PRS. Now, we have a PRS that consists of the top two-lines of the PRS as depicted in Fig. 3. Instead of interpreting the underspecified representation of LeVeque's first proof sentence, the proof planner tries to figure out future proof continuations. Consulting the proof

[2]However, we cannot hope that any proof plan library is complete. There is a potentially infinite number of different induction schemes, and creative mathematicians often combine existing methods in a novel way. Learning new plans from the content of discourse is a difficult task — also for the human reader!

plan library, applicable proof plans are identified, amongst them the proof plan shown in Fig. 4. Instantiating this proof plan for the theorem at hand results in the lines marked '$*^2$' of the PRS in Fig. 3. These starred lines define a number of expectations, and therefore allow us to view remaining informal statements as anaphoric entities that refer to their place in the PRS.

Unfortunately, the underspecified semantic representation of the first proof sentence of Fig. 1 cannot be matched to the proof plan. For $a = 2$, we have that $\forall k < a : [k > 1 \rightarrow product\_primes(k)]$ is trivially true since there is no k such that $k < a$ and $k > 1$. Therefore, the case $a = 2$ is treated separately.

The second sentence can be matched to one of our expectations if we give to the elliptic construct '$2, 3, 4, \ldots, a - 1$' a universal reading: $\forall n \in \mathbb{N} : n < a$. Note that the a that occurs in this expression refers to the free PRS variable a, and not to the universally quantified PRS variable a that was introduced by processing the theorem.

The third sentence contains the discourse marker 'if' indicating an assumption. Since we cannot match this assumption with the PRS expectations, the proof planner consults the plan library. Unexpected assumptions are often of the form $A \vee \neg A$ (here, A is 'a is prime'), and a proof per cases is likely to take place. The proof planner adds this to the PRS as one possible proof continuation. This allows us to resolve 'if a is prime'. The constituent 'we are through' marks the end of this first case leaving the proof planner with the obligation to resolve the logical relation between $prod\_primes$ (2.1.2) and all statements of the proof context that are accessible from 2.1.2. Note that, due to a wrong proof continuation, we might not be able to identify this logical relation. The occurrence of *otherwise* in the fourth sentence seems to confirm a proof per cases. The remainder of the textbook proof has to fit into the second case yielding to **2.2.2** to **2.2.10**.

The PRS is in Fig. 3 is still semantically underspecified: most of the variables do not have quantifications yet and all conclusions are still unjustified. The proof planner must invest much inference to validate its structure and to compute the necessary information to fill in the semantic gaps.

For step **1.** of the PRS the remaining proof obligation, or *goal*, is of the form $A \rightarrow B$. To prove statements of this form, the *implication method* can be applied: one assumes A and then uses A to show B. This inference is not explicitly part of the textbook proof. It must be derived to facilitate subsequent discourse processing. The proof planner extend the PRS by the lines 'ASSUME $a > 1$' and 'PROVE $prod\_primes(a)$'.

Justifying **Step 2.1.2.** involves a couple of inference steps. *Definitional rewriting* on $prod\_primes$ reduces **2.1.2** to

$$prime(a) \vee \exists p_1 \in \mathbb{N} : \exists p_2 \in \mathbb{N} : a = p_1 p_2 \wedge p_1 < a \wedge p_2 < a \wedge prod\_primes(p_1) \wedge prod\_primes(p_2).$$

Having now a disjunction in the goal, a *proof per elimination* can be performed. There are two possible proof plan schemes (Fig. 5). Applying *proof per elimination-II*, we obtain a new hypothesis (in addition to the induction hypothesis, $a > 1$ and **2.1.1**):

PROVE $A \rightarrow B1 \vee B2$	PROVE $A \rightarrow B1 \vee B2$
Proof per elimination-I	*Proof per elimination-II*
ASSUME A	ASSUME A
ASSUME $\neg B1$	ASSUME $\neg B2$
PROVE $B2$	PROVE $B1$

FIG. 5. Proof per elimination

$$\neg(\exists p_1 \in \mathbb{N} : \exists p_2 \in \mathbb{N} : a = p_1 p_2 \wedge p_1 < a \wedge p_2 < a \wedge product\_primes(p_1) \wedge product\_primes(p_2))$$

and a new goal: $prime(n)$. Since the goal is contained in the hypotheses, we are through. — Due to space restrictions, we omit a detailed analysis of the second case.

3.4 Proof plan refinement.

The step from proof plans to formal proofs is a simple problem. It involves no search because to each large proof step, we can attach its expansion into inference-level steps. Take the proof method of Nœtherian induction (Fig. 4). It is formally defined as (3.1).

$$\forall n \in \mathbb{N} : [(\forall k \in \mathbb{N} : k < n \rightarrow P(k)) \rightarrow P(n)] \rightarrow \forall n \in \mathbb{N} : P(n). \tag{3.1}$$

To prove a formula of the form $\forall n \in \mathbb{N} : P(n)$, we prove

$$\forall n \in \mathbb{N} : [(\forall k \in \mathbb{N} : k < n \rightarrow P(k)) \rightarrow P(n)] \tag{3.2}$$

and apply Modus Ponens using (3.1) and (3.2). To prove (3.2), we can assume n being an arbitrary number and assume $\forall k \in \mathbb{N} : k < n \rightarrow P(k)$. These steps are on the inference level of a Gentzen-style calculus [9].

4 Related and Future Work

The automated verification of textbook proofs has long been ignored. The only citable reference is Simon's Phd thesis [22]. However, Simon fails to seriously address both linguistic and mathematical issues. It remains unclear how Simon handles linguistic phenomena – he does not propose an adequate theory for doing so. It remains also unclear how Simon is able to obtain adequate discourse structure. We regard this as an essential requirement for following the proof author's argumentation line and for subsequent formalisation.

The inverse of our task, translating formal proofs into natural language proofs, has first been described in [7]. More recent work is [12]. Facing the problem that proofs generated by conventional theorem provers are unstructured, tediously long and therefore unreadable, a readable, structured and short proof (omitting all the low-level details) has to be generated. Finding appropriate abstractions from low-level proofs is a hard problem.

Proof planning is an enabling technology for constructing and representing high-level proofs [4]. Proof plans allow us to capture informal mathematical reasoning, and enable us to fill in the details that mathematicians find trivial to write down. Our task to mechanically verify mathematical discourse forces us to closer examine existing mathematical proofs and methods of proof. We share Wang's opinion that *"the existing body of mathematics contains a great wealth of material and constitutes the major source of our understanding of mathematical reasoning. The reasonable course would be to distill from this great reservoir whatever is mechanizable"* [28]. This is a promising approach to make proof machines more usable.

Our work also relates to extensions of DRT, in particular Asher's SDRT [2]. SDRT discourse update mechanism allows to effectively interface semantics with pragmatics. The discourse update procedure consists of *relating* a sentence to its prior discourse (not just adding it). To compute such rhetorical relations (e.g., background, narration, elaboration), Asher and Lascarides make use of domain knowledge and their DICE inference engine [15, 3].

Given the enormous complexity of the entire problem, much implementation work is to be done to complete VIP. However, our definite clause grammar covers already a considerable amount of mathematical text on the sentence level. For this level, DRT has been extended to account for four different kinds of discourse referents:

constants, free variables, quantified variables, and functional terms. A number of DRT construction rules have been implemented that capture typical syntactic constructions in informal mathematical text.

For the discourse level, we extended discourse representation structures to proof representation structures. A novel algorithm for discourse update has been developed. Its core is the proof planner which takes the intermediate semantic representations returned by the parser module, and interprets them in the current proof context. In the future, the discourse update algorithm has to be fully implemented and tested on a substantial number of examples. We would also like to interface to the λ-CLAM proof planning system that we have already used to manually generate formal proofs from informal proofs. This would give us access to a large number of already formalised proof plans and λ-CLAM's proof planning engine.

The view of proofs as highly structured discourse is compelling. Linguistically, we would like to study how our way of handling discourse structure compares with existing discourse models like [10]. Another related line of research is to view the problem of discourse understanding as a plan recognition problem [17],[6], [21]. The proof planner we described certainly generates expectations and tries to match them later. But proof recognition goes hand in hand with constructive planning. The relation between proof construction and plan recognition has to be worked out in greater detail.

5 Conclusion

We proposed an architecture for the automatic verification of informal proofs (mathematically); constructing the semantics and pragmatics of informal mathematical discourse (linguistically). We argued that the recognition of structure plays a decisive role. It enables us to follow the argumentation line of the proof author, and to give us access to a high-level strategic proof understanding. Linguistically, discourse structure allows us to properly handle linguistic phenomena. We argued that proof plans not only provide for such structure; they also generate expectations about possible proof continuations.

The design of VIP extends and integrates state-of-the-art technologies from Natural Language Processing (Discourse Representation Theory) and Automated Reasoning (Proof Planning) in a novel and promising way, For representing mathematical discourse, we introduced proof representation structures as the central data structure. PRSs are a considerable extension to discourse representation structures accommodating our need for representing discourse structure, and handling substructures and mathematical sentences as first-class citizen. For constructing PRSs, we proposed a discourse update algorithm that is powered by pragmatics. A proof planner incorporates an underspecified semantic representation into the proof context by making use of mathematical and meta-mathematical knowledge.

The automatic checking of mathematical textbook proofs is a challenging application scenario. For a long time, human intervention will be necessary to direct theorem provers through the enormous search space of mathematical tautologies. Being able to process textbook proofs allows us to close the gap between the language and the reasoning of mathematicians, and the language and reasoning of proof systems. It is appealing to think of a *natural* language interaction between mathematician and machine. It will enable us to build systems that really assist mathematicians.

References

[1] P. W. Abrahams. *Machine verification of mathematical proofs*. PhD thesis, MIT, 1963.

[2] N. Asher. *Reference to abstract objects in discourse*. Kluwer Academic Publishers, 1993.

[3] N. Asher and A. Lascarides. The semantics and pragmatics of presupposition. *Journal of Semantics*, 15(3):239–300, 1998.

[4] A. Bundy. The use of explicit proof plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Ninth Conference on Automated Deduction*, pages 111 – 120. Springer, 1988.

[5] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam system. In M. E. Stickel, ed., *Proceedings of the 10th International Conference on Automated Deduction*. Springer, 1990.

[6] S. Carberry. Modeling the User's Plans and Goals. *Comp. Linguistics*, 14(3):23–37, 1988.

[7] D. Chester. The Translation of Formal Proofs into English. *Artificial Intelligence*, 7:261–278, 1976.

[8] R. L. Constable. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice Hall, 1986.

[9] G. Gentzen. Untersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 1935.

[10] B.J. Grosz and C.L. Sidner. Attention, Intention, and the Structure of Discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[11] G. Hardy and E. Wright. *An introduction to the theory of numbers*. Oxford at the Clarendon Press, 4th. edition, 1971.

[12] X. Huang and A. Fiedler. Proof verbalization as an application of NLG. In M. E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 965–970. Morgan Kaufmann, 1997.

[13] H. Kamp and U. Reyle. *From Discourse to Logic*. Kluwer Academic Publishers, 1993.

[14] L. Lamport. How to write proofs. In *Global Analysis in Modern Mathematics*, pages 311–321. Publish or Perish, Houston, Texas, U.S.A., February 1993.

[15] A. Lascarides and N. Asher. Discourse relations and defeasible knowledge. In *Proceedings of the 29th Annual Meeting of the Assoc. for Computational Linguistics*, pages 55–63. ACL, 1991.

[16] W. J. LeVeque. *Elementary theory of numbers*. Addison-Wesley, 1965.

[17] D.J. Litman and J.F. Allen. Discourse processing and commonsense plans. In P.R. Cohen, J. Morgan, and M.E. Pollack, eds., *Intentions in communication*, p. 365–388. MIT Press, 1990.

[18] J. McCarthy. Computer programs for checking mathematical proofs. In *Recursive Function Theory, Proceedings of Symposia in Pure Mathematics*, volume 5. AMS, 1962.

[19] J.D. Moore and M.E. Pollack. A Problem for RST: The Need for Multi-Level Discourse Analysis. *Computational Linguistics*, 18(4), 1992.

[20] 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*. North-Holland, 1994.

[21] C.F. Schmidt, N.S. Sridharan, and J.L. Goodson. The Plan Recognition Problem: An Intersection of Psychology and Artificial Intelligence. *Artificial Intelligence*, 11:45–83, 1978.

[22] D. L. Simon. *Checking Number Theory Proofs in Natural Language*. PhD thesis, Austin, 1990.

[23] D. Solow. *How to read and do proofs*. John Wiley & Sons, 1990.

[24] Andrzej Trybulec. The MIZAR-QC/6000 logic information language. *ALLC*, 6:136–140, 1978.

[25] J. Trzeciak. Writing mathematical papers in english. Gdańsk Teacher's Press, Institute of Mathematics, Polish Academy of Science, 1993.

[26] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the Automath system*. PhD thesis, TH Eindhoven, 1977.

[27] A.J.M. van Gasteren. *On the shape of mathematical arguments*, volume 445 of *Lecture Notes in Computer Science*. Springer, 1990.

[28] H. Wang. Towards Mechanical Mathematics. *IBM Journal of research and development*, 4(1), 1960.

[29] C. Zinn. Understanding Mathematical Discourse. In *Proceedings Amstelogue'99, Workshop on the Semantics and Pragmatics of Dialogue*. Amsterdam University, 7-9 May 1999.