English summaries of mathematical proofs

English Summaries of Mathematical Proofs

Marianthi Alexoudi[1], Claus Zinn, Alan Bundy
Division of Informatics, The University of Edinburgh

Abstract

Automated theorem proving is becoming more important as the volume of applications in industrial and practical research areas increases. Due to the formalism of theorem provers and the massive amount of information included in machine-oriented proofs, formal proofs are difficult to understand without specific training. A verbalisation system, ClamNL, was developed to generate English text from formal representations of inductive proofs, as produced by the Clam proof planner. The aim was to generate natural language proofs that resemble the presentation of proofs found in mathematical textbooks and that contain only the mathematically interesting parts of the proof.

1 Introduction

Automated theorem proving is becoming more important as the volume of its applications increases. It is a powerful tool for hardware design, as well as for the verification of software systems. Additionally, formal methods are increasingly applied in mathematics e.g. for the construction of proofs for conjectures and the composition of formalised theories.

Machine-generated proofs are difficult to understand without specific training and familiarity with the given system's formalism and the calculus being used. They are represented in a specialised and artificial language, using a notation that seems incomprehensible to an inexperienced reader. Furthermore, the massive amount of information presented in a formal proof leads to an over-detailed, and therefore hardly readable proof, even for an experienced reader. One could also claim that a formal proof is mathematically 'unstructured', in the sense that it is hard to understand its overall logical structure and identify the important definitions, lemmas and other logical dependencies. Therefore machine-found proofs seem to be insufficient for an effective communication between theorem provers and their users, particularly in terms of their presentation.

Due to the 'unreadability' of formal proofs, the task of their verbalisation and the development of systems, whose output could be conveniently comprehended by non-experts and effortlessly explained by experts became evident. In fact, one of the most challenging tasks in the area of automated theorem proving is the realisation of an effective translation of machine-found (formal) into human-oriented (informal) mathematical proofs, and vice versa. Such a translation would eliminate the gap between the mathematicians' and the proof systems' language and reasoning.

[1]M.Alexoudi@sms.ed.ac.uk,{zinn,bundy}@inf.ed.ac.uk

This paper presents an implemented proof presentation system that generates Natural Language (NL) proofs at various levels of abstraction from (inductive) proof plans. In section 2, related systems are introduced and their weaknesses are briefly discussed. Section 3 provides an overview of ClamNL and examples of its output are presented. A summary of the experimental results is presented in section 4 and the current state and further work on this project are discussed in section 5. Finally, section 6 concludes.

2 Literature Survey

Several efforts have been made to improve the readability of machine-oriented proofs by generating (English) NL versions of proofs. Numerous systems have been designed and developed to produce informal proofs from formal ones that were produced using various deduction techniques and calculi, such as Natural Deduction (ND), resolution and λ-calculus.

Previous work can be classified into three main categories with respect to the output that the existing systems have generated. The first category involves the first generation of verbalisation systems, such as EXPOUND [5] and χ-proof [9] that generated low-level NL proofs, definitely more readable and coherent from machine-found ones but still obscure. Additionally, both Coq [6] and ILF [7] theorem proving systems have a NL front-end that allows the generation of (pseudo)-NL proofs. Although these systems use different methods to generate natural language proofs, most of them suffer from the same problem. The NL versions of machine-oriented proofs were produced by translating a great number of low-level steps, and thus they contained 'obvious' and unnecessary information, such that even trivial proofs might be confusing. Furthermore, most of the informal proofs produced by the above systems preserved the logical formulae in their original form and text was inserted between them either in the form of introductory phrases or as explanations of the inference rules. Thus, the NL proofs are characterised by a mixture of formal and informal representation of the proof steps that reduces their readability. The second group comprises systems such as PROVERB [12] and the NLG module of the Nuprl theorem prover [11], that used more composite and sophisticated techniques to eliminate the drawbacks of previous ones and generated more abstract and human-like NL proofs. Regardless of the sophisticated methods used by these systems, their output is still restricted in some aspects. These systems produce a unique NL version of the corresponding machine-generated proof at a fixed level of abstraction independent of the reader's knowledge. Their output might be too advanced for novice users and too elementary for experts, since it assumes a certain audience with specific knowledge and it does not allow shifting between multiple abstraction levels. The last category embodies systems such as THEOREMA [2] and P.rex [10] that are capable to output various informal proofs for a single formal one.

An essential feature of proofs that enhance their readability is the resemblance to human-written proofs and especially to those written by mathematicians, in terms both of content and presentation. However, the attempts made mostly focus in resembling the way that mathematicians write their proofs, rather than the way that mathematicians reason during proof construction. In many cases this is an issue that arises from the prover rather than the proof presentation system. The formal language and the

deduction techniques used for the construction of a formal proof, not only limit the degree of similarity between informal and textbook proofs, but also restrict the level of abstraction and the readability of an NL proof. More precisely, resolution calculi based formal proofs are difficult to manipulate in order to produce coherent NL proofs, due to the existence of a single calculus rule. As far as the ND calculus is concerned, although ND proofs have more potential than resolution ones, it is still complex to abstract the important proof steps from low level inference rules. On the other hand, it is more likely to generate comprehensive and easy readable NL versions of formal proofs produced by tactic-based environments, since related inference rules are grouped into tactics, each of which approximates a single human inference step.

Therefore, aiming at the construction of informal proofs similar to those presented in mathematical textbooks, we need to use formal proofs resembling the way that mathematicians analyse and work out proofs. For instance, mathematicians recognise families of proofs containing common structure and they use previously encountered proofs to assist them in discovering new ones. The way that mathematicians work out their proofs can be captured using the proof planning technique for constructing and representing high-level proofs [3]. Proof plans are abstract representations of proofs at a level that is better suited for manipulation beacuse of the absence of low-level derivations.

3 System Overview

ClamNL [1] is a proof presentation system built upon the Clam proof planning environment [4] that generates NL proof at various levels of abstraction, similar to those found in mathematical textbooks. Clam is a first-order predicate logic proof planner that was used for the construction of proof plans for theorems whose proofs require the application of various kind of mathematical induction over different data types.

The use of high-level representations of proofs, known as proof plans, enhances the generation of abstract NL proofs. Furthermore, the process of formal proof conversion is informed by a notion of 'interestingness' of proof steps. A set of heuristics has been employed to remove obvious and trivial parts of the proof and highlight its mathematically interesting points.

ClamNL's architecture is presented in Figure 1. Of the modules illustrated in Figure 1, the proof planner (Clam) and the XSLT-based software (Natural Language Generator) that processes the templates are existing software. Each of the remaining components is described in one of the following sections. ClamNL consists of three main modules, the *Abstraction Controller* that enables the interaction of the system with the user and the proof planner; the *Structure Planner* that handles the structure, the contents and the presentation of the NL proofs to be generated; and the *NL Generator* that translates the extracted parts of a proof plan to English text in a template-based manner.

3.1 Abstraction Controller

The Abstraction Controller (AC) controls the level of detail of a proof plan and in turns the level of abstraction of the resulting NL proof. The AC, given a theorem name, determines the number of proof versions that can be generated for that theorem.

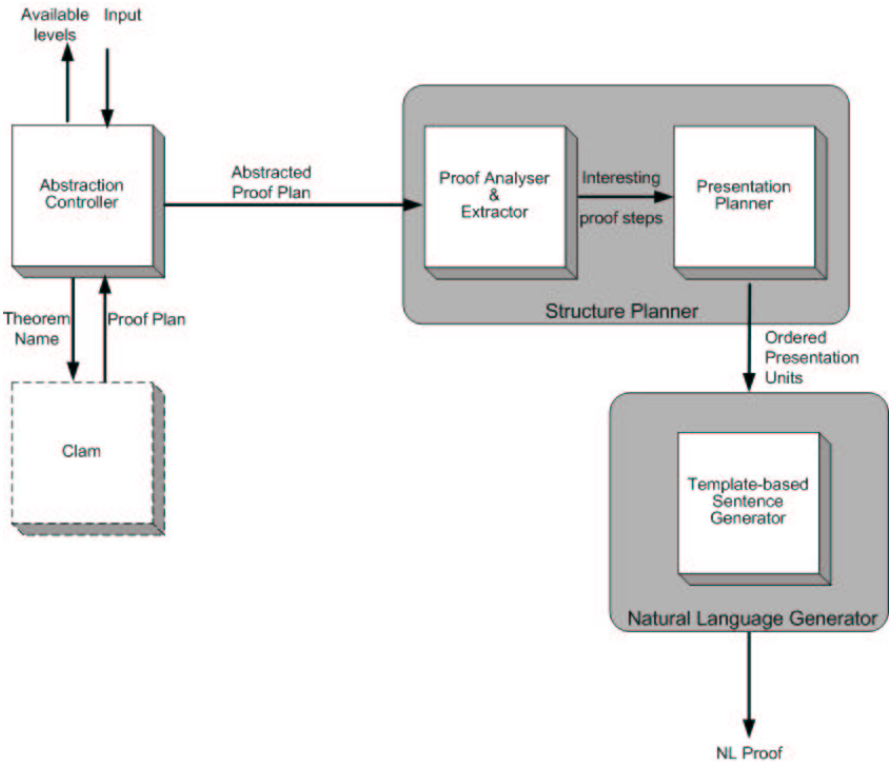

Figure 3.1: Architecture of the ClamNL verbalisation system

Moreover, given a theorem name and the required level of abstraction, it initialises the process of generating the requested NL proof.

The process of abstracting a proof plan depends entirely on the theorem to be proven. If the proof of a theorem involves the proof of another theorem, then the AC discards from the original proof plan the subproof and passes the remaining proof plan to the Proof Analyser and Extractor (PAE). This process can be repeated as many times as the total number of theorems used to prove the original one. Therefore, the number of NL proofs at different levels of abstraction depends on the number of theorems that are used in the proof of the original one. In this case the resultant NL proof presents the proof of the original theorem, in which other theorems are used for its completion. Although none of them is proven, it is assumed that they hold.

3.2 Proof Analyser and Extractor

The Proof Analyser & Extractor PAE determines the nodes of an abstracted proof plan to be included into a NL proof. Given an abstracted proof plan, PAE extracts the mathematically interesting parts of a proof and omits standard and easily deducible ones. During the extraction of interesting proof steps the proof plan tree is linearised and every node (subproof) is handled separately. The output of PAE is a forest of proof steps that is then passed to the Presentation Planner. An example of such transformation is presented in Figure 2.

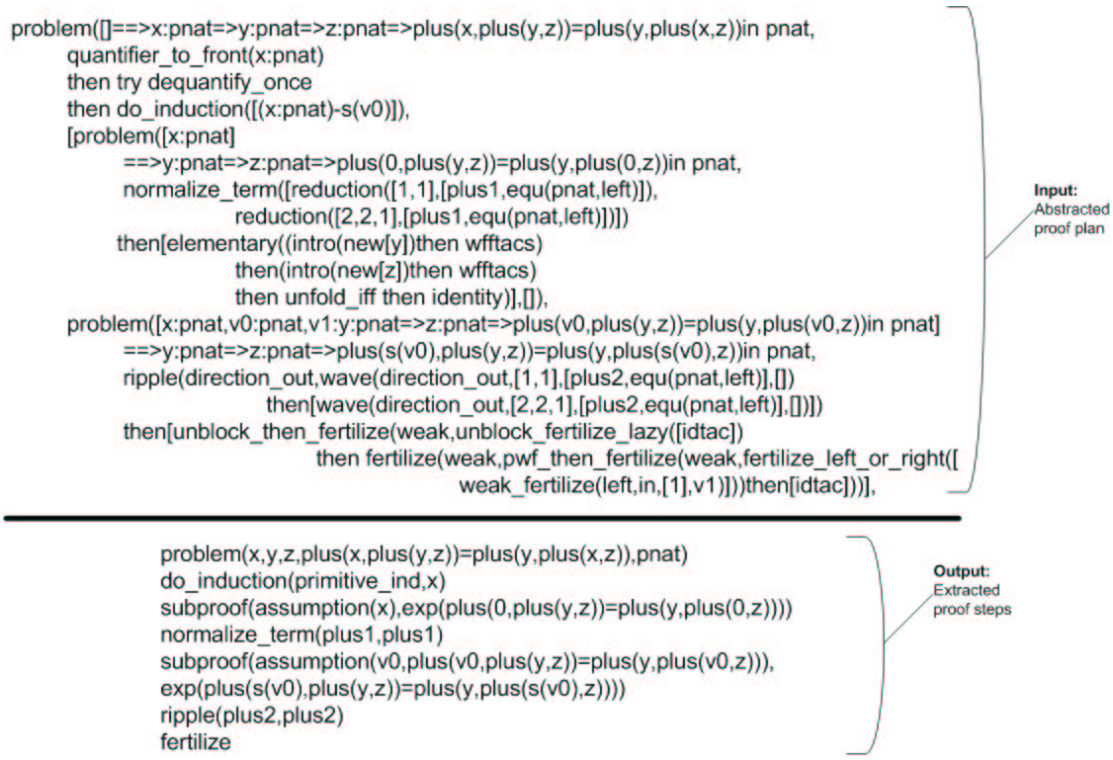

Figure 3.2: Extraction of mathematically interesting proof steps from a proof plan.

The 'interestingness' of proofs steps is defined by a knowledge base according to which a proof step can be interesting, non-interesting, or partially interesting.

In general, apart from the theorem statement, the focus of interest in inductive proofs is on the induction scheme used to prove the theorem and the induction variable to which the induction scheme is applied. Also, the base and the step cases of an inductive proof should be clearly stated. In the base case, the induction variable and the constant to which the induction variable is instantiated, as well as the base case resulting expression should be specified. Similarly, in a step case, the induction hypothesis and the conclusion should also be specified. On the other hand, axioms and low-level methods are classified as mathematically non-interesting proof steps and thus are discarded. Moreover, rewriting on conjectures that produce the same conjecture as the one to which they were applied should be ignored, as well as their resultant conjecture (i.e. tautology). As partially interesting are charecterised proof steps whose some of their contents are useful and some of them are not. An example of this category is the list of hypotheses available every time a new goal is introduced, which might contain a new assumption.

3.3 Presentation Planner

The Presentation Planner (PP)performs three vital tasks. It rearranges the contents of the proof steps, inserts additional elements where appropriate, and transforms the

proof steps to an intermediate representation format consisting of presentation units. The ordered presentation units are then passed to Natural Language Generator module to be verbalised.

The PP features two stages of reordering the contents of proof steps but not the proof steps themselves. The first involves the checking of the proof steps ordering, in case their order was lost during linearisation, and the reordering of the terms of a mathematical formula from infix to prefix. The second involves the mapping of certain compound terms of a conjecture into a more human-oriented representation and the ordering of the new terms in the conjecture.

One feature involves the random selection of justification tokens that will be used for the verbalisation of certain units. The Presentation Planner handles, in a non-sophisticated way, commonly used tokens to avoid the repetition of identical standard phrases in the proof outline. Every justification token corresponds to a single sentence in the template-based sentence generator. In principal, this involves the identification of base and step cases proof steps in order to avoid incorrect verbalisations. Another approach to avoid multiple interesting proof steps in the proof outline involves the merging of similar, adjacent proof steps.

Finally, the presentation of a proof in enhanced by organising the proof steps into paragraphs and indenting them so that the proof structure is clear and thus coherent. Also, an axiom table, consisting of all the axioms' definitions used throughout the proof, is appended to the end of the proof. During the structuring of the proof steps, Presentation Planner (PP) converts the proof steps to XML elements and produces the XML document to be fetched to the NLG module.

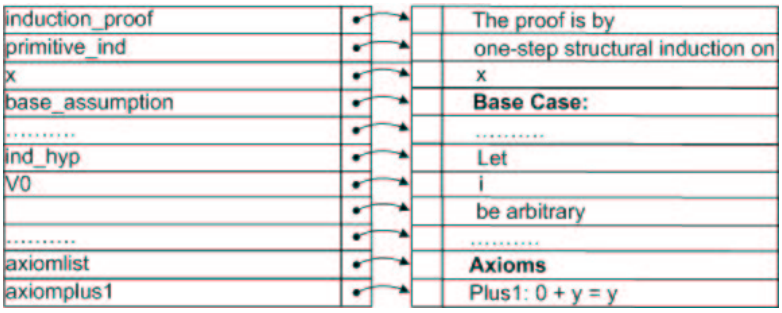

Figure 3.3: An example of presentation units to NL templates mapping.

3.4 NLG

The template-based generator maps presentation units to English statements. Text is generated by mapping individual presentation units (XML elements) into mathematical notions, concepts, variable names, words and sentences (XML templates). Figure 3 illustrates a sample of such a mapping. The Template-based generator, given an input XML document, outputs a NL proof in a HTML or XML file format.

As a tool for the text generation, a major component of the EXEMPLARS framework [14], the *text builder* was used as a wrapper for the XSLT text building transformation.

Also, a XSL stylesheet consisting of more than 400 templates was created for the mapping of the XML elements to English text. The XSLT engine matches the template rules contained in the XSL stylesheet with the XML elements of the input document and the text is generated, based on the XSL vocabulary.

NL proofs such as the one presented in Figure 4 consist of the theorem statement and the outline of the proof. The actual proofs are constructed using a Lamport-style proof presentation [13], consisting of the theorem statement, the proof outline and the proof of the theorem.

3.5 Sample Output

The commutativity of addition is provided, as an example of the system's output for illustration purposes. The proof of the provided theorem is an example of a proof by induction over natural numbers, one of the various induction schema available in Clam. Others include one and two step induction on lists and trees.

Figure 4 presents an example of a NL version of a machine-found proof corresponding to the commutativity of addition theorem, generated by ClamNL. This kind of output is called a proof summary, since it states how a certain theorem can be proven. More precisely, it is more like a proof description about what one should do in order to prove a theorem, rather than a proof itself.

Theorem: (x + (y + z)) = (y + (x + z) for all natural numbers x y z
The proof is by one-step induction on x In order to complete the step case, the conjecture is generalised by introducing a universal variable i.e. k . The generalised conjecture can be proven by one-step induction on y and finally the proof is completed

Figure 3.4: The proof summary of the commutativity of addition theorem.

The actual proof of the commutativity of addition theorem, but more abstract than the complete one is presented in Figure 5. Such kinds of proofs involve the progress of the proof until the stage where another known theorem is used to complete the proof of the actual theorem.

Figure 6 [2] shows a more detailed proof version of the commutativity of addition theorem. Although the complete proof of the theorem is presented, it is not a direct translation of the original, machine-found proof plan, since lots of proof steps and trivial parts of the proof are omitted.

4 Experimental Results

ClamNL was developed to generate NL versions of formal mathematical proofs that would be fluently readable and abstract and would resemble those found in mathematical textbooks. To demonstrate and validate the project's claims, a corpus of inductive

[2] In certain cases the template mapping produces minor grammatical errors, whose elimination is in high priority

Theorem: (x + (y + z)) = (y + (x + z)) for all natural numbers x y z
The proof is by one-step structural induction on x
Base Case: x = 0
(0 + (y + z)) = (y + (0 + z)) by applying 2 times axiom "plus1"
Inductive Hypothesis: Let i be arbitrary
(i + (y + z)) = (y + (i + z))
Induction Step: We need to prove ((i + 1) + (y + z)) = (y + ((i + 1) + z))
by applying 2 times axiom "plus2" and using the induction hypothesis we get:
((y + (i + z)) + 1) = (y + ((i + z) + 1))
Let: k = (i + z)
we are left to prove: ((y + k) + 1) = (y + (k + 1))
Since by plus2right theorem, we have that (x + (y + 1)) = (x + y) +1 is true,
then (x + y) +1 = (x + (y + 1)) is also true. Thus, the proof is completed.

Axioms
plus1: 0 + y = y
plus2: (x + 1) + y = (x + y + 1)

Figure 3.5: A NL proof of the commutativity of addition theorem at an intermediate level of abstraction

theorems was collected from approximately 130 supported theorems. The corpus was selected so as to provide a wide range of theorems covering various degrees of difficulty and complexity, as well as the use of various different induction schema.

Two groups of subjects were used, Clam experts and non-experts with different mathematical background, in order to ensure that the NL proofs were accessible and beneficial to a wide range of audience with various levels of expertise. Although the number of participants was too limited to obtain a statistically representative sample, the results gathered were beneficial and encouraging.

The results are classified into three categories, each of which corresponds to and supports the project's objectives.

Readability: The NL proofs were characterised as easily readable and coherent on two counts. First, in terms of the linguistic nature of the proofs, since it is considerable easier for a human to comprehend a proof in the (natural) language of mathematicians, rather than rules represented in the given proof system's formalism. Second, the use of indentation was quite helpful in keeping track of deeply nested proofs (i.e. subproofs by induction). However, in cases of deeply nested proofs it would be preferable to have a hypertext-based or applet-based approach to hide or unfold parts of the proof presentation on the fly upon user requests.

Abstraction Level: The availability of proofs at various levels of detail was found extremely useful in digesting proofs. As regards the content of the NL proofs at different levels of abstraction, different opinions were expressed, possibly because of the dissimilar mathematical background of the participants. The majority of the subjects claimed that each proof contained the right amount of information on the progress of the proof, given the corresponding detail level. In particular, proofs declared as abstract were indeed seen

Theorem: (x + (y + z)) = (y + (x + z)) for all natural numbers x y z
The proof is by one-step structural induction on x
Base Case: x = 0
(0 + (y + z)) = (y + (0 + z)) by applying 2 times axiom "plus1"
Inductive Hypothesis: Let i be arbitrary
(i + (y + z)) = (y + (i + z))
Induction Step: We need to prove ((i + 1) + (y + z)) = (y + ((i + 1) + z)) by applying 2 times axiom "plus2" and using the induction hypothesis we get:
((y + (i + z)) + 1) = (y + ((i + z) + 1))
Let: k = (i + z) we are left to prove:
((y + k) + 1) = (y + (k + 1))
The proof is by one-step structural induction on y
Base Case: y = 0
((0 + k) + 1) = (0 + (k + 1)) by applying 2 times axiom "plus1"
Inductive Hypothesis: Let o be arbitrary
((o + k) + 1) = (o + (k + 1))
Induction Step: We need to prove (((o + 1) + k) + 1) = ((o + 1) + (k + 1)) by applying 2 times axiom "plus2" and using the induction hypothesis the case is completed.

Axioms
plus1: 0 + y = y
plus2: (x + 1) + y = (x + y + 1)

Figure 3.6: The most detailed proof version of the commutativity of addition theorem.

as abstract and those declared as detailed were indeed seen as more detailed. However, there were cases where participants claimed that the omission of some steps would be desirable, though their presence was not irritating.
Similarity to textbook proofs: The participants, based on their experience in mathematical literature, estimated that the presentation of the NL proofs approximated the presentation of proofs found in mathematical textbooks.

5 Current State and Further Work

ClamNL has been developed as the first author's undergraduate project. It offers many opportunities for improvements and extensions, which we will discuss next.

The next step in the development of NL versions of machine-oriented proofs would be the generation of partial NL proofs of unsuccessfully proven theorems. Proof presentation systems developed so far, require their respective theorem provers to compute complete proofs. Our idea is to verbalise a formal proof until the point that it has been successfully developed and then try to explain in natural language the reasons that lead to a fallible proof, and if possible, suggest patches about how a certain failure can be overcome in natural language.

Currently, we are also investigating to adapt our verbalisation system so that it can handle IsaPlanner[8]-generated proof plans. IsaPlanner is a generic framework for proof

planning build upon the interactive theorem prover Isabelle that facilitates reasoning techniques to conjecture and prove theorems automatically. In constrast to Clam that limits the implementation of partial NL proofs, IsaPlanner supports the generation of both complete and incomplete proof plans. Furthermore, IsaPlanner is available to a wider audience compared with Clam, which is nowdays used by a limites number of people.

As far as the verbalisation of complete proofs is concerned, the current system generates various abstract NL proofs that contain only mathematically 'interesting' parts of the proof, rather than the complete proof in terms of low-level steps. At the moment, the 'interestingness' of proof steps is system-defined. Thus, the next step would be to modify it so that users would be able to obtain customised versions of NL proofs by defining what they consider to be interesting, depending on their knowledge and interest.

The provision of user interaction is an important feature in systems of this nature. The user can interact with the system either by requesting the number of available NL versions of the proof of some theorem or by requesting a certain proof of a theorem. However, the interaction between the user and the system is managed through a unix shell. This is clearly a limitation that we would like to resolve in future by designing a simple and user-friendly interface. Furthermore, although the proofs are expandable, in the sense that some are more detailed than others, hypertext-based versions of proofs would be extremely useful, since users could unfold parts on the fly rather than look for another version of the proof.

Finally, additional features would be the gen eration of multilingual proofs and proofs of different presentation styles targeted at two different groups of users. A *mathematical-style* for users that are interested in mathematical aspects of the proof of a given theorem and a *compositional-style* that will target people interested in the process of constructing proofs.

6 Conclusion

This paper presents and proposes a multi-step approach for the presentation of machine-oriented proofs. The automatic generation of NL versions of formal proofs aims to improve the readability and comprehensiveness, as well as the usefulness of machine-found proofs and extend/enable their availability to a wider audience.

The generation of human-readable proofs at different levels of abstraction can be succeeded using the proof planning technique. Proof plans offer an ideal solution, since they provide high-level presentation and low-level interpretation of proofs. In addition, the process of a formal proof abstraction and the availability of proofs at different levels of details is enhanced by the notion of interestingness. Currently, the interestingness of proof steps is system defined, but in future we aim to a more dynamic and interactive approach to proof presentation and explanation.

Bibliography

[1] M. Alexoudi. English Summaries of Mathematical Proofs, 4th Year Undergraduate Project Report. School of Informatics, University of Edinburgh, 2003.

[2] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuţa, and D. Vāsaru. A Survey on the THEOREMA Project. In *ISSAC '97. Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, pages 384–391. ACM Press, 1997.

[3] A. Bundy. A Science of Reasoning. In *J. L. Lassez & G. Plotkin (Eds.), Computational Logic: Essays in Honor of Alan Robinson.* The MIT Press, 1991.

[4] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The Oyster-Clam System. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*, volume 449 of *LNAI*, pages 647–648, Kaiserslautern, FRG, July 1990. Springer Verlag.

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

[6] Y. Coscoy. A Natural Language Explanation for Formal Proofs. In *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, volume 1328 of *LNAI*, pages 149–167, Berlin, September 23–25 1997. Springer.

[7] B. I. Dahn and A. Wolf. Natural language presentation and combination of automatically generated proofs. In *Frontiers of Combining Systems (FroCos)*, pages 175–192, 1996.

[8] L. Dixon and J. D. Fleuriot. IsaPlanner: A Prototype Proof Planner in Isabelle. In *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 279–283. Springer, 2003.

[9] A. Felty and D. Miller. Proof Explanation and Revision. Technical Report MS-CIS-88-17, University of Pennsylvainia, 1987.

[10] A. Fiedler. *P.rex: An Interactive Proof Explainer.* In Rejeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning — 1st International Joint Conference, IJCAR 2001*, number 2083 in LNAI, pages 416–420, Siena, Italy, 2001. Springer Verlag.

[11] A. M. Holland, R. Barzilay, and R. L. Constable. Verbalization of High Level Formal Proofs. *In American Association for Artificial Intelligence*, 1999.

[12] X. Huang and A. Fiedler. Presenting machine-found proofs. In *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-96)*, volume 1104 of *LNAI*, pages 221–225, Berlin, July 30–August 3 1996. Springer.

[13] L. Lamport. How to Write a Proof. Technical Report TR 94, Digital Systems Research Center, February 1993.

[14] M. White and T. Caldwell. EXEMPLARS: A Practical, Extensible Framework for Dynamic Text Generation. In Eduard Hovy, editor, *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 266–275. Association for Computational Linguistics, New Brunswick, New Jersey, 1998.