From Proof Objects to Linear Proof Texts

Aarne Ranta[1] and Jan von Plato[2]

[1] Department of Computer Science and Engineering,
Chalmers University of Technology and University of Gothenburg
aarne.ranta@cse.gu.se
[2] Department of Philosophy, University of Helsinki
jan.vonplato@helsinki.fi

1 Introduction

Gentzen's natural deduction trees are an explicit notation for the relations between conclusions, premisses, and hypotheses in logical proofs [6]. Via the Curry-Howard isomorphism, constructive type theory generalizes them to proof objects where the tree structure is expressed by function applications and the dependence on hypotheses by lambda abstractions [7]. This is exploited in systems such as Agda and Lean, where proof checking becomes type checking.

Type-theoretical terms are an elegant and efficient representation of proofs. But they can be hard for humans to read. Visualization as natural deduction trees works for small proof objects in logic teaching. But for proofs in general, linear texts (with indentation adding partial tree structure) is the format of choice. The need for this was stated in [2], particularly when proofs are produced automatically. The need is even larger in recent systems where proofs are produced by large language models (LLMs) [11]. A proof system can guarantee their correctness, but it can be hard even to see if the proven theorem is the one intended. Letting an LLM "explain" the proof breaks the chain of reliable tools, because it may produce text that does not correspond to the formal proof that has been checked.

The work reported here builds on the ideas of [2]. It belongs to the project Informath [10], where informalization has been implemented for arbitrary expressions of Dedukti [1] and thereby, in principle, for other systems that can be converted to Dedukti, including Agda, Lean, and Rocq. Since Informath can convert arbitrary Dedukti expressions, it can deal with proofs as well. However, the current compositional conversion of abstractions and applications leads to texts that are horrible to read. We will here break it up to a two-step approach:

- from proof objects to sequences of proof lines, following [2] and [12],
- from proof lines to natural language, using GF [9].

Our focus is on the first step, where each line is still expressed as a formula. The second step leads to ordinary mathematical language, a mixture of words and symbols [5]; this is the main novelty in comparison to [12]. In comparison to [2] and our earlier work [8], the novelty is the division of one step into two.

2 The algorithm

Our algorithm is a generalization from natural deduction as presented in [12], to proof objects in Dedukti. The proof steps in [12] are lines of Jaśkowski-Fitch-Prawitz style natural deduction. Each line has a number (ranging from 1 without gaps) and can be represented as a sextuple of the form:

$$<\text{context}, \text{line-number}, \text{formula}, \text{rule-label}, \text{premisses}, \text{discharged}>$$

where context, premisses, and discharged are lists of zero or more line numbers. An example is the following proof of natural deduction in both tree and linear form:

$$\dfrac{\dfrac{\dfrac{1.}{A\&B}}{B}\,\&E2 \qquad \dfrac{\dfrac{1.}{A\&B}}{A}\,\&E1}{\dfrac{B\&A}{A\&B \supset B\&A}\,\supset I,1}\,\&I$$

1	1.	$A\&B$	*hypo*		
1	2.	B	$\&E2$	1	
1	3.	A	$\&E1$	1	
1	4.	$B\&A$	$\&I$	2, 3	
	5.	$A\&B \supset B\&A$	$\supset I$	4	[1]

On lines 1 to 4, the context is shown as the open hypothesis 1. Line 5 is free from this hypothesis, which is indicated by the discharge list [1]. Contexts are usually marked not with the hypothesis numbers but with nested vertical lines; this notation can be generated by replacing the numbers by such lines.

A proof object representation for the above tree is

$$\supset I(A\&B, B\&A, (\lambda h.\&I(B, A, \&E2(A, B, h : A\&B) : B, \&E1(A, B, h : A\&B) : A) : B\&A))$$

It is built from expression of three kinds:

- abstraction: $\lambda x.p$
- application: $c(p_1, \ldots, p_n) : A$
- variable: $x : A$

Applications and variables are annotated by types that represent the formulas to be shown on each step. The type annotation is a preparatory step of the linearization, like in [2]. Applications are assumed to be total applications of constants c, which are functions that encode rule labels. For example, the implication introduction rule is the function

$$\supset I : (A : \text{Prop}) \to (B : \text{Prop}) \to (\text{Proof}(A) \to \text{Proof}(B)) \to \text{Proof}(A \supset B)$$

The conversion from proof objects to sequences of lines is a composition of two functions:

- recursion on proof expressions producing lists of lines (as sextuples),
- removal of repeated lines and renumbering of lines and the pointers to them.

Furthermore, the linearization of a function application typically does not show all of the arguments: in $\supset I$, it must drop the first two arguments to reach the original natural deduction. The algorithm therefore reads a symbol table that indicates what arguments are omitted. They are usually the same as the implicit arguments in systems like Agda and Lean.

3 Further processing

The linearization of a proof term results in a sequence of lines whose length is proportionate to the number of non-hidden subterms. Each line moreover shows information that the user normally does not want to see, because it can be inferred from the context. Of the sextuple data structure of a proof line, only the formula is typically shown. The rule label can be shown if it corresponds to a named theorem, and the premisses might be indicated by referring to numbered assumptions. Deciding what information to omit is an important further step in the linearization of proof objects to natural-looking texts.

An even more substantial improvement is to omit intermediate steps. Type-theoretically, this means the replacement of primitive proof functions (such as those of natural deduction) by larger ones. The formalization can help this by defining and using such functions, but they could also be introduced automatically by probing what steps are "obvious" enough to be shown as single steps [3]. The ultimate goal is to perform conversions that invert the de Bruijn factor [4]: if formalization produces a proof that is 10-20 times larger than the informal text, informalization should shrink the size by at least 90%.

Acknowledgements

This work is supported by the European Research Council through the grants *NextReason* (Grant ID 101200949), *MALINCA* (Grant ID 101167526), and *HERITAGE* (Grant ID 101198605).

References

[1] Ali Assaf, Guillaume Burel, Raphaël Cauderlier, David Delahaye, Gilles Dowek, Catherine Dubois, Frédéric Gilbert, Pierre Halmagrand, Olivier Hermant, and Ronan Saillard. Dedukti: a logical framework based on the $\lambda\pi$-calculus modulo theory, 2023.

[2] Yann Coscoy, Gilles Kahn, and Laurent Thery. Extracting text from proofs. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Proc. Second Int. Conf. on Typed Lambda Calculi and Applications*, volume 902 of *LNCS*, pages 109–123, 1995.

[3] Martin Davis. Obvious logical inferences. In Patrick J. Hayes, editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence, IJCAI '81, Vancouver, BC, Canada, August 24-28, 1981*, pages 530–531. William Kaufmann, 1981.

[4] N. G. de Bruijn. A survey of the project Automath. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–606. Academic Press, New York, 1980.

[5] Mohan Ganesalingam. *The Language of Mathematics: A Linguistic and Philosophical Investigation*. Springer, 2013.

[6] Gerhard Gentzen. Untersuchungen ueber das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210 and 405–431, 1934.

[7] Per Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, Napoli, 1984.

[8] Aarne Ranta. Context-relative syntactic categories and the formalization of mathematical text. In S. Berardi and M. Coppo, editors, *Selected papers from TYPES'95: Int. Workshop on Types for Proofs and Programs, Trento, Italy*, volume 1158 of *LNCS*, pages 231–248. Springer-Verlag, 1996.

[9] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011.

[10] Aarne Ranta. Informath: Informalization and Autoformalization of Formal Mathematics. https://github.com/GrammaticalFramework/informath, 2026.

[11] Josef Urban. 130k lines of formal topology in two weeks: Simple and cheap autoformalization for everyone? arXiv 2601.03298, https://arxiv.org/abs/2601.03298, 2026.

[12] Jan von Plato. From Gentzen to Jaskowski and Back: Algorithmic Translation of Derivations Between the Two Main Systems of Natural Deduction. *Bulletin of the Section of Logic*, 46:65–73, 2017.