Integration of controlled natural language in formal mathematics systems

David M. Cerna Adrian De Lon Peter Koepke
Shashank Pathak Aarne Ranta

October 1, 2025

Abstract

This is a brief and informal overview of the rôle of controlled natural language in formal mathematics. We list existing systems, summarize open challenges and opportunities, and present some ongoing projects. This report also serves as the start of a bibliography of the field.

1 Motivation and background

Mathematical writing is sufficiently precise for human readers, but too ambiguous and informal for machines. Formal proof languages used in proof assistants are rigorous but can be difficult for humans to read and write. *Controlled natural languages* (CNLs) [21] offer a middle ground, providing a readable yet formally defined way to express mathematics.

For the purposes of this report in the context of formal mathematics, a *controlled natural language* is a restricted sublanguage of natural language, such as English, with a fully formal specification of its syntax and semantics. We do not address the full problem of interpreting arbitrary mathematical writing, but focus instead on deliberately designed sublanguages with well-defined rules.

Controlled languages can function as an intuitive interface between the user and the machine, making formal mathematics more accessible while ensuring that every statement has a precise, predictable, and traceable interpretation.

2 Overview of existing systems

2.1 Mizar

The Mizar Project [20] initiated by Andrzej Trybulec in the 1970s, which was the first to combine interactive and automated theorem proving with a long term plan for readable output in a natural or quasinatural language. The Mizar system uses a quasinatural formalization language, which is not a sublanguage of English, but still models many of the essential features of mathematical English

[27]. This project has lead to the development of the large Mizar Mathematical Library (MML). Articles of the MML are published in the Journal of Formalized Mathematics and a natural language summary paper is automatically derived as part of the publishing process. Mizar has been influential on the design of subsequent proof assistants and proof vernaculars.

2.2 Informalization tools for Rocq

Various informalizations tools have been developed for the informalization of Rocq proofs. Examples include work by Coscoy et al. [30], Bedford [18], as well as Poulsen et al. [8].

2.3 SAD

The System for Automated Deduction (SAD) is a proof assistant developed by Andrei Paskevich as part of his doctoral work [26]. It represents the culmination the long-running Evidence Algorithm (EA) project [15] initiated by V. M. Glushkov in the 1970s, which aimed to build systems that support mathematical reasoning in a way that is natural for humans. The key feature of SAD is its controlled natural language ForTheL (Formal Theory Language) [29], which allows mathematical texts to be written in a style close to ordinary mathematical English. A ForTheL document is parsed into a first-order logical representation and then processed by a verification engine, which can work together with external first-order automated theorem provers to verify proof steps. In line with the EA vision, SAD can track *evidences* during the proof checking process, approximating the way a human reader incrementally understands a mathematical text. SAD was the first proof assistant to make the direct formalization of mathematics in controlled natural language practical. This iteration of SAD was in some sense a continuation of an earlier version of SAD, which featured a Russian controlled natural language. SAD was roughly contemporaneous with the first iteration of Isabelle's Sledgehammer, which was released in 2007 (the same year Paskevich defended his thesis), and likewise relied on external automated theorem provers to provide powerful proof automation.

2.4 Naproche (2003–2013)

The Naproche project started as an interdisciplinary collaboration, studying the logical and linguistic aspects of mathematical language, aiming to model it through a controlled natural language. The first phase of this project culminated in Marcos Cramer's implementation of the Naproche system [22]. This system used a novel foundation called Ackermann-like Function Theory and employed various linguistic techniques, such as discourse representation structures, to interpret texts written in TEX-style markup. Compared to SAD, the Naproche system took a more linguistic approach, but it was weaker in terms of proof automation, making formalization difficult.

2.5 Naproche (2015–now)

Naproche-SAD [11] is a direct continuation of SAD. It adds integration with Isabelle/PIDE as an interactive editing environment with incremental proof checking, a LaTeX dialect of ForTheL that allows for *literate formalization* (mixing commentary and formalization within the same document), performance optimizations, and makes it harder for users to introduce accidental logical inconsistencies. Naproche-SAD uses Kelley–Morse class theory as foundation and comes with an expanded included library. Various Bachelor and Master student completed formalization projects in Naproche-SAD, covering subjects such as representation theory, elementary number theory, set theory, and category theory [14, 10, 9]. Naproche-SAD is available as part of the Isabelle distribution.

2.6 Naproche-ZF

Naproche-ZF [4] is an experimental reimplementation of key ideas of Naproche and SAD, motivated by trying to scale the Naproche approach beyond chapter-sized formalization and offering a test bed for new features. In contrast to Naproche, which uses parser combinators, it uses a declarative grammar-based approach to parsing, which makes it easier to extend its controlled natural language when adding new features or extending language coverage. Naproche-ZF also has more efficient proof automation, making better use of multicore machines and doing less potentially now unnecessary preprocessing on proof tasks exported to external automated theorem provers.

2.7 Elfe

Elfe [19] is an interactive theorem prover designed to help undergraduate students learn and practice formal proofs. It allows users to write proofs in a quasinatural mathematical language that resembles textbook notation. Elfe translates these texts into first-order logic and generates proof obligations in TPTP format, which are then checked using automated theorem provers such as E, SPASS, Z3, and Beagle. If a proof step is incorrect, Elfe provides counterexamples to help students understand their mistakes. Elfe includes a basic library on sets, relations, and functions, and offers both a command line and a web interface. The focus lies on accessibility and ease of use, lowering the barrier for students to engage with formal proof systems and preparing them to transition to advanced theorem provers.

2.8 Diproche

Diproche (Didactic Proof Checking) [7] is a proof assistant aimed at supporting students in the early stages of university-level mathematics by helping them develop basic proving skills. The system allows students to write proofs in a controlled sublanguage of German that is tailored to beginner-level exercises, The CNL is translated into an internal formal representation, which is used to generate proof tasks for a purpose-built ATP. Unlike Naproche, which uses the strongest ATPs available, this ATP is designed and configured to only accept elementary proof steps, as deemed appropriate for a given topic and educational level. Diproche provides immediate feedback on logical correctness, type correctness, failed steps, and proof goals.

2.9 GFLean

GFLean [6] is a software tool for autoformalization of mathematics into the Lean theorem prover [13], built on the Grammatical Framework (GF) [24]. It provides a pipeline that translates mathematical statements, expressed in a controlled natural language, into Lean code. The input language is a simplified version of ForTheL [29], a formalised fragment of natural mathematical English. Statements written in this language are parsed into abstract syntax trees (ASTs), simplified and translated into Lean-compatible ASTs, and finally linearised into Lean expressions. The generated Lean code expresses the intended semantics of the statements, with placeholder proofs standing in for reasoning steps.

When applied to a representative set of 62 undergraduate-level statements [25], it successfully formalises 42 statements with minor rephrasing, with the remaining 20 statements failing primarily due to limitations in linguistic coverage. The system already supports quantifiers, predicates, modifiers, and a range of mathematical constructions, demonstrating that grammar-based autoformalization is feasible for nontrivial mathematical input.

The framework does, however, highlight several challenges in scaling autoformalization. The lexicon must be substantially expanded to cover the breadth of mathematical terminology. The reliance on a simplified ForTheL grammar excludes common linguistic phenomena, such as conjunctions of terms and multiple adjectives, which appear frequently in mathematical texts. Moreover, the GF grammars used for GFLean are static, making it difficult to incorporate user-defined notions dynamically, an essential feature for real-world mathematical practice. At present, GFLean also restricts itself to the formalisation of statements, leaving proof automation as a future direction.

Looking forward, GFLean is designed to serve as a foundation for more advanced systems. Extensions to full ForTheL, integration with Lean's environment for dynamic vocabulary and definitions, and the combination of grammar-based precision with the broader coverage of neural translation methods represent natural next steps. By bridging controlled natural language with type-theoretic representations, GFLean demonstrates a viable path toward scalable and practical autoformalization of mathematics.

2.10 Verbose Lean

Verbose Lean 4 [2] is a collection of tactics and commands that enables users to write Lean 4 proofs in a small CNL. Originally developed in Lean 3 for instructional purposes, it has been ported to Lean 4 with support for both French and

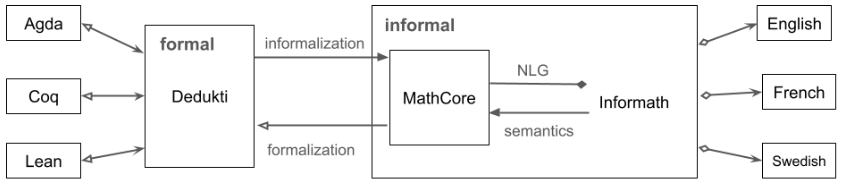
English, built on a shared backend. Its primary aim is pedagogical, intending to bridge the gap between formal verification and traditional mathematical exposition, improving readability and accessibility in teaching contexts. The project is actively used in higher-education settings and provides pedagogical resources, including examples, documentation, and configuration tools, to support classroom deployment.

3 Open challenges

Several challenges remain for controlled natural languages (CNLs) in formal mathematics. Existing production systems are typically limited to one or two languages, such as English, French, German, or Russian. Current CNL approaches have not been scaled to libraries of the size of Lean's mathlib, Isabelle's Archive of Formal Proofs, or Mizar's MML, and it is unclear how to maintain readability and consistency at that scale. There is also a need for methods to reformulate, compress, or expand proofs while preserving verifiability, so that the same argument can be presented at different levels of detail. In the context of autoformalization and proof automation, we would also want a reliable way to turn proofs produced by automated theorem provers into text that humans can read and check without losing essential information. Finally, there is potential for CNLs to serve as a structured interface between formal systems and large language models.

4 Ongoing projects

4.1 Informath



Informalization — conversion from formal to informal language — is a radically different task from formalization: it can be made into a total function, whereas conversions from informal to formal are bound to be only partial (unless quality is sacrificed). The reason is that natural, informal languages have no well-defined bounds, whereas formal languages have. From this difference, it follows that informalization is a more feasible task than formalization. It can be implemented with symbolic, formally analysable rules, which can convert all expressions of a formal language into a language that looks natural and is readable without knowing the formalism. It can thereby form a trusted component in a system that combines formal and informal mathematics.

The most obvious task of informalization is to make formal mathematics available in natural language. This was the starting point of Coscoy et al [30], who built a translator from Rocq (then Coq) proofs to English texts. The rationale was that the users of Rocq would not need to rely on the machine-built proof, which was in a formal code too complex for a human to understand, but they could read a more human-friendly text and understand themselves. This purpose is more relevant than ever in federated systems such as Google's AlphaProof, where a machine-checked formal proof (in Lean in this case) is to be made readable by humans. Symbolic informalization makes it possible to produce natural language from formal proofs in an automated and reliable way.

Informalization is also relevant for the opposite direction, formalization. If built in a reversible grammar formalism (where generation rules are automatically convertible to parsing rules), formalization is an immediate consequence of informalization. However, this kind of formalization is partial in the same way as parsers of CNLs in general are. A more wide-coverage usage of informalization is to let it produce synthetic data for training neural autoformalization [17, 5]. If formal code is available, formal-informal pairs can be mechanically produced in huge quantities. The resulting autoformalization system is still unreliable, but informalization can help assess its quality by producing feedback texts from suggested formalizations.

The main problem Coscoy et al. [30] addressed was the quality of the mechanically produced text. When formal proofs grow large, texts informalized from them easily become hopeless to read, and, in any case, boring and mechanical. Jiang et al. [5] noted the same problem, as one of three fundamental problems with symbolic informalization:

- generated text lacks natural variation and fluency;
- symbolic systems are laborious to build;
- such systems only address one single formalism at a time.

The project Informath [3] takes these three problems as important challenges to meet.

The third challenge is met by using Dedukti as an interlingua between formalisms: due to existing conversions from Agda, Lean, and Rocq (and some others) to Dedukti, it is enough to use Dedukti as the starting point to informalize all these systems.

The second challenge is addressed by using the grammar engineering tools of Grammatical Framework (GF; [24, 16]). Due to the high level of coding and the availability of resource grammar libraries (RGL), a single line of GF can replace hundreds of context-free rules. GF also supports the extraction of grammar rules from data-driven systems such as Universal Dependencies (UD, [12]). With these assets, comprehensive informalization systems can be built as a matter of weeks or months.

The first challenge is the most open-ended one. The goal of Informath is to approach informal mathematical language in all its richness described by Ganesalingam [23]. To achieve this, Informath builds on classical techniques of Natural Language Generation (NLG, [28]) and develops them further. By combining different techniques, one can easily produce hundreds of alternative ways to express a given formula. Due to the reversibility of GF, such extensions also widen the parsing capabilities of the grammar. Extensions are developed in a controlled way, so that each new way of expressing a formula comes with a semantic rule that proves that it is a conservative extension.

Informath is not meant to be yet another stand-alone system, but rather a component and a library enabling informalization and helping autoformalization of other systems. For formal proof systems, it can provide automated documentation and an ultimate form of syntactic sugar for user interaction. For a CNL, it can provide an extension of its capabilities: Informath can be extended to cover the CNL if it does not already do so (which will also profit Informath itself). As a result, the CNL gets extended parsing and generation capabilities, and it can be translated to numerous formal systems via Dedukti. Via the multilinguality of GF, it can also be ported to multiple natural languages.

4.2 Mizaproche

The Mizaproche [1] projects aims to combine the Mizar proof system with a Naproche-like controlled natural language. Its goal is to translate the Mizar Mathematical Library into a form of mathematical English that is both human-readable and mechanically checkable. The translation is designed to be reversible, so that nothing is lost and the original Mizar proofs can be recovered. This will produce a large parallel corpus of Mizar proofs and their natural-language equivalents. The project also aims to develop tools for *proof summarization*, reducing the many small steps in Mizar proofs to shorter arguments that are verifiable by automated theorem provers. Finally, it will explore *autoformalization*, the automatic creation of formal proofs from informal mathematical text, targeting controlled natural language as an intermediate format both that is both readable to humans and easily processed by the machine.

References

[1] Mario Carneiro, Adrian De Lon, Atle Hahn, Peter Koepke, and Josef Urban. "Le Miz s'approche: Informalization and Autoformalization with Mizar and Naproche". In: *AITP 2025.* 2025.

[2] Patrick Massot. *Verbose Lean 4.* 2025. URL: `https://github.com/PatrickMassot/verbose-lean4`.

[3] Aarne Ranta. *Informath.* 2025. URL: `https://github.com/GrammaticalFramework/informath`.

[4] Adrian De Lon. "The Naproche-ZF Theorem Prover (Short Paper)". In: *Automated Reasoning.* Ed. by Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt. Cham: Springer Nature Switzerland, 2024, pp. 105–114. ISBN: 978-3-031-63498-7.

[5] Albert Q. Jiang, Wenda Li, and Mateja Jamnik. “Multi-language Diversity Benefits Autoformalization”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang. Vol. 37. Curran Associates, Inc., 2024, pp. 83600–83626. URL: `https://proceedings.neurips.cc/paper_files/paper/2024/file/984de836e696ba653bbfbbbfce31d3bc-Paper-Conference.pdf`.

[6] Shashank Pathak. *GFLean: An Autoformalisation Framework for Lean via GF*. 2024. arXiv: `2404.01234 [cs.CL]`. URL: `https://arxiv.org/abs/2404.01234`.

[7] Merlin Carl, Hinrich Lorenzen, and Michael Schmitz. “Natural Language Proof Checking in Introduction to Proof Classes – First Experiences with Diproche”. In: *Electronic Proceedings in Theoretical Computer Science* 354 (Feb. 2022), pp. 59–70. URL: `http://dx.doi.org/10.4204/EPTCS.354.5`.

[8] Seth Poulsen, Matthew West, and Talia Ringer. “Autogenerating Natural Language Proofs for Proof Education”. In: *Coq Workshop 2022*. 2022.

[9] Adrian De Lon, Peter Koepke, and Anton Lorenzen. “A Natural Formalization of the Mutilated Checkerboard Problem in Naproche”. In: *12th International Conference on Interactive Theorem Proving (ITP 2021)*. Ed. by Liron Cohen and Cezary Kaliszyk. Vol. 193. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 16:1–16:11. ISBN: 978-3-95977-188-7. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2021.16`.

[10] Adrian De Lon, Peter Koepke, Anton Lorenzen, Adrian Marti, Marcel Schütz, and Erik Sturzenhecker. “Beautiful Formalizations in Isabelle/Naproche”. In: *Intelligent Computer Mathematics*. Ed. by Fairouz Kamareddine and Claudio Sacerdoti Coen. Cham: Springer International Publishing, 2021, pp. 19–31. ISBN: 978-3-030-81097-9.

[11] Adrian De Lon, Peter Koepke, Anton Lorenzen, Adrian Marti, Marcel Schütz, and Makarius Wenzel. “The Isabelle/Naproche Natural Language Proof Assistant”. In: *Automated Deduction – CADE 28*. Ed. by André Platzer and Geoff Sutcliffe. Cham: Springer International Publishing, 2021, pp. 614–624. ISBN: 978-3-030-79876-5. URL: `https://doi.org/10.1007/978-3-030-79876-5_36`.

[12] Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, and Daniel Zeman. “Universal Dependencies”. In: *Computational Linguistics* 47.2 (July 2021), pp. 255–308.

[13] Leonardo de Moura and Sebastian Ullrich. “The Lean 4 Theorem Prover and Programming Language”. In: *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 625–635. ISBN: 978-3-030-79875-8. URL: `https://doi.org/10.1007/978-3-030-79876-5_37`.

[14] Peter Koepke, Jan Penquitt, Marcel Schütz, and Erik Sturzenhecker. “Formalizing Foundational Notions in Naproche-SAD”. In: *Workshop on Natural Formal Mathematics at CICM 2020*. 2020. URL: `https://cicm-conference.org/2020/cicm.php?event=NFM`.

[15] Alexander Lyaletski and Alexandre Lyaletsky. “Evidence Algorithm Approach to Automated Theorem Proving and SAD Systems”. In: *IT&I-2020 Information Technology and Interactions*. 2020.

[16] Aarne Ranta, Krasimir Angelov, Normunds Gruzitis, and Prasanth Kolachina. "Abstract Syntax as Interlingua: Scaling Up the Grammatical Framework from Controlled Languages to Robust Pipelines". In: *Computational Linguistics* 46.2 (June 2020), pp. 425–486. URL: `https://doi.org/10.1162/coli%5C_a%5C_00378`.

[17] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. *First Experiments with Neural Translation of Informal to Formal Mathematics.* 2018. arXiv: `1805.06502 [cs.CL]`. URL: `https://arxiv.org/abs/1805.06502`.

[18] Andrew Bedford. "Coqatoo: Generating Natural Language Versions of Coq Proofs". In: *International Workshop on Coq for Programming Languages (CoqPL 2018)*. 2017.

[19] Maximilian Doré. "Elfe: An interactive theorem prover for undergraduate students". Bachelor's Thesis. 2017.

[20] Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. "Four Decades of Mizar". In: *Journal of Automated Reasoning* 55 (Oct. 2015), p. 2015.

[21] Tobias Kuhn. "A Survey and Classification of Controlled Natural Languages". In: *Computational Linguistics* 40 (Mar. 2014), pp. 121–170.

[22] Marcos Cramer. "Proof-checking mathematical texts in controlled natural language". PhD thesis. University of Bonn, 2013.

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

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

[25] G. Chartrand, A.D. Polimeni, and P. Zhang. *Mathematical Proofs: A Transition to Advanced Mathematics.* Pearson/Addison Wesley, 2008. ISBN: 9780321390530. URL: `https://books.google.co.uk/books?id=dCogAQAAIAAJ`.

[26] Andriy Paskevych. "Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques". PhD thesis. Université Paris 12, 2007.

[27] Josef Urban and Grzegorz Bancerek. "Presenting and Explaining Mizar". In: *Electronic Notes in Theoretical Computer Science* 174 (May 2007), pp. 63–74.

[28] Ehud Reiter and Robert Dale. *Building Natural Language Generation Systems.* Cambridge University Press, 2000.

[29] Konstantin Vershinin and Andrey Paskevich. "ForTheL—the language of formal theories". In: *International Journal of Information Theories and Applications* 7.3 (2000), pp. 120–126.

[30] Y. Coscoy, G. Kahn, and L. Thery. "Extracting text from proofs". In: *Proc. Second Int. Conf. on Typed Lambda Calculi and Applications.* Ed. by M. Dezani-Ciancaglini and G. Plotkin. Vol. 902. LNCS. 1995, pp. 109–123.