DOI: https://doi.org/10.26565/2304-6201-2026-69-03

UDC 004.85:510.6:004.421.2

Yevdokymov Oleksandr *PhD Student, Department of Mathematical Modeling and Artificial Intelligence, National Aerospace University "Kharkiv Aviation Institute", Vadym Manko St., 17, Kharkiv, Ukraine, 61070*
e-mail: o.yevdokymov@khai.edu
https://orcid.org/0009-0008-9687-6344

Luchsheva Oksana *Senior Lecturer, Department of Software Engineering National Aerospace University "Kharkiv Aviation Institute", Vadym Manko St., 17, Kharkiv, Ukraine, 61070*
e-mail: o.luchsheva@khai.edu ;
https://orcid.org/0000-0003-3855-2815

A Mathematical Model of Automatic Verification of Formalized Proofs and a Conservative Presentation Interface over Lean

Relevance. Interactive theorem provers such as Lean have fundamentally transformed the verification of mathematical statements by transferring the final control of correctness from the human mathematician to a small trusted kernel. Nevertheless, their widespread adoption in research and higher education is still limited by a significant gap between the strict formal language of the kernel, based on dependent type theory, and the usual intuitive mathematical notation, symbols, and natural-language reasoning used by mathematicians and students. This discrepancy creates serious obstacles for teaching proof writing, formalizing new results, and developing effective intelligent educational systems.
Objective. The purpose of this paper is to construct a rigorous mathematical model of automatic verification of formalized proofs in Lean and to provide a formal justification for building a conservative human-oriented presentation interface together with an untrusted generative pedagogical component over the trusted kernel without any loss of mathematical rigor.
Methods. The study relies on dependent type theory, metatheoretic properties of the calculus of constructions (soundness, strong normalization, decidability of type checking), models of elaboration and backward projection between the presentation language and the kernel, and formal predicates of term-typing correctness. Interface conservativity is expressed as the inclusion of the set of derivable statements of the presentation layer in the set of kernel theorems. The safety of the generative AI-component is defined by the condition that every executable artefact must be projected back to the kernel and re-verified by the trusted type-checker.
Results. The developed model defines the formalizable fragment of mathematical discourse $\mathcal{ML}$, the kernel-verification predicate VerL, the course formalizability functional $\mathbb{F}(\mathcal{S})$, a complete conservative interface model, and a safety condition for the generative extension. It is proved that syntactic sugar, macros, notation abbreviations, and AI-generated educational artefacts do not enlarge the set of derivable statements as long as the trusted Lean kernel remains unchanged and all executable fragments are re-checked by the kernel. An illustrative example with four theorems from algebra and group theory (uniqueness of the identity element, intersection of subgroups, kernel of a homomorphism is normal, and binomial identity) demonstrates that human-oriented notation is fully elaborated into kernel terms while preserving mathematical meaning and verifiability. The interface fidelity reaches the maximum value of 1.
Conclusions. The obtained results demonstrate that the formal language of Lean can be equipped with a convenient human-oriented interface and intelligent educational services suitable for scientific and instructional use without compromising rigor. The only criterion of truth remains repeated verification by the small trusted kernel. The proposed separation between verified kernel knowledge, conservative presentation layer, and untrusted generative services creates a reliable foundation for building intelligent educational systems in higher mathematics, discrete mathematics, algebra, and mathematical logic, where formal correctness is guaranteed independently of the quality of generated explanations and exercises.
Keywords: *Lean, dependent types, automatic proof verification, formalized mathematics, conservative interface, generative pedagogical module.*

How to quote: O. Yevdokymov, and O. Luchsheva, "A Mathematical Model of Automatic Verification of Formalized Proofs and a Conservative Presentation Interface over Lean", *Bulletin of V. N. Karazin Kharkiv National University, series Mathematical modelling. Information technology. Automated control systems*, vol. 69, pp. 33–40, 2026. https://doi.org/10.26565/2304-6201-2026-69-03

Як цитувати: Yevdokymov O., and Luchsheva O. A Mathematical Model of Automatic Verification of Formalized Proofs and a Conservative Presentation Interface over Lean. *Вісник Харківського національного університету імені В. Н. Каразіна, серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління».* 2026. вип. 69. С. 33–40. https://doi.org/10.26565/2304-6201-2026-69-03

1. Introduction

Modern interactive theorem provers have transformed the machine verification of mathematical statements from a purely logical task into a practical instrument for the formalization of substantial fragments of contemporary mathematics. Lean is characterized by a small trusted kernel, dependent types as the basis of its internal language, and an advanced mechanism for syntax extension, elaboration, tactics, and inference support, while the mathlib library accumulates a large corpus of already verified definitions, lemmas, and theorems [1–3]. The theoretical foundations of such systems are provided by intuitionistic type theory, the calculus of constructions, and the correspondence "propositions as types, proofs as terms" [4–6].

For research and education this means that the control of mathematical correctness can be transferred from the human agent to the system kernel, whereas the human remains responsible for formulating ideas, choosing a strategy, selecting notation, and constructing an explanatory layer. Experience with Lean in proof education, as well as work on controlled natural language and human-oriented interfaces such as Verbose Lean and Waterproof, shows that the principal obstacle is not verification itself but rather the mode of interaction between the mathematician and the formal language [7–12].

At the same time, practical intelligent educational systems impose two nontrivial requirements. First, the mathematically trusted layer must be strictly separated from untrusted service extensions. Second, it must be formally justified why ordinary mathematical notation, macros, syntactic sugar, or AI-generated explanations do not alter the set of derivable theorems whenever final verification is performed by the kernel. Existing studies have already taken steps toward the operationalization of the formalizability of mathematical tasks for intelligent tutoring systems, as well as toward the representation of mathematical expressions by structural trees [13–15]; however, the mathematical model of a conservative human-oriented interface over Lean still requires a separate justification.

The aim of the paper is to construct a mathematical model of the automatic verification of formalized proofs in Lean and to rigorously justify the conservativity of a human-oriented presentation interface and an untrusted generative component with respect to the trusted kernel of the system.

2. Formal problem statement

Let L be a Lean-like formal system built over dependent type theory. At the minimal level of abstraction it is convenient to represent it by the tuple

$$L = (\Sigma, Ctx, Term, Type, \vdash, \equiv, K) \tag{2.1}$$

where Σ is the signature of constants and inductive objects, *Ctx* is the set of contexts, *Term* is the set of terms, *Type* is the set of types, $\vdash$ is the typing relation, $\equiv$ is definitional equality, and *K* is the trusted verification kernel. The basic judgment has the form

$$\Gamma \vdash t : T \tag{2.2}$$

where Γ is the context of local hypotheses and instances, *t* is a term, and *T* is a type. By the Curry-Howard correspondence, a proof of a statement T is represented by a term p of that type [4]–[6].

$$\mathrm{Th}(L) = \{ T \in \mathrm{Type} \mid \exists p \in \mathrm{Term}: \varnothing \vdash p : T \} \tag{2.3}$$

Let $\mathcal{M}$ denote the set of statements formulated in ordinary mathematical language, and let $\mathcal{M}_L \subset \mathcal{M}$ be the fragment for which an adequate encoding into L exists. Then

$$\mathcal{M}_L = \{\sigma \in \mathcal{M} \mid \exists \varphi(\sigma) \in \mathrm{Type},\ \exists p \in \mathrm{Term}: \varnothing \vdash p : \varphi(\sigma)\} \tag{2.4}$$

Here φ is the mapping of a mathematical statement into a formal type. For a finite set of statements $\mathcal{S} = \{\sigma 1, \ldots, \sigma N\}$, we introduce the formalizability coefficient

$$\mathfrak{F}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{\sigma \in \mathcal{S}} \mathbf{1}_{\sigma \in \mathcal{M}_L} \tag{2.5}$$

The quantity $\mathfrak{F}(\mathcal{S})$ generalizes the intuitive claim about what proportion of a course, topic, or problem bank admits full formalization and automatic verification [13]. In order to avoid an excessively strong metaphysical thesis about the formalization of "any" proof without qualifications, universality of automatic verification will henceforth mean universality on the set $\mathcal{M}_L$ only.

A formalization will be considered adequate if it preserves the mathematical meaning of the statement:

$$[\![\sigma]\!]_{\mathcal{M}} = [\![\varphi(\sigma)]\!]_L \tag{2.6}$$

where $[\![\cdot]\!]_{\mathcal{M}}$ and $[\![\cdot]\!]_L$ denote the semantic interpretations at the levels of informal and formal mathematics, respectively. Equality (1.6) is the condition of faithfulness of translation from the language of the mathematician into the language of the kernel.

3. Model of automatic proof verification

At the surface level, the user may build a proof by a tactic script, a term expression, or a combination of both. After elaboration, the surface text is reduced to a kernel term. Therefore, the model of automatic verification can be defined through the Boolean predicate

$$\mathrm{Ver}_L(T, p) = 1 \Leftrightarrow K \text{ accepts } p{:}\, T \tag{3.1}$$

In the opposite case, we assume that *VerL(T, p) = 0*. From the mathematical point of view, any formal proof may be represented as a derivation tree

$$D = (V_D, E_D, r, \lambda), \qquad \lambda(v) = \Gamma_v \vdash t_v{:}\, T_v \tag{3.2}$$

where the vertices of the tree correspond to local judgments and the edges correspond to applications of inference rules. If the leaves of *D* are axioms or directly checkable constructor steps, and the root coincides with $\vdash p : T$, then *D* encodes a completed proof of the theorem *T*.

Proposition 1. For any closed $T \in Type$ and $p \in Term$, the predicate *VerL(T, p)* is computable.

Proof. Within the kernel, correctness checking reduces to solving the typing problem $p : T$ while taking definitional equality into account. For the calculus of constructions and its practical restrictions used in Lean, the type-checking procedure is algorithmic; therefore, there exists an algorithm that terminates on every closed input with value 0 or 1 [1], [2], [4], [5].

Theorem 1. Let $\sigma \in \mathcal{M}L$ and suppose the adequacy condition (1.6) holds. If $VerL(\varphi(\sigma), p) = 1$, then the statement σ is *true* in the intended mathematical interpretation.

Proof. From $VerL(\varphi(\sigma), p) = 1$ we obtain $\vdash p : \varphi(\sigma)$. By the metatheoretic soundness of the kernel, this implies the semantic truth of the formal statement $\varphi(\sigma)$ in every model of the system *L* compatible with the context. By the faithfulness of translation (2.6), the truth of $\varphi(\sigma)$ is equivalent to the truth of σ in the mathematical interpretation.

Thus, any mathematical proof belonging to the formalizable fragment $\mathcal{M}L$ and equipped with a constructed proof object *p* is automatically checkable by the kernel. This is the mathematical meaning of the thesis of automatic proof verification in Lean.

4. Conservative model of a human-oriented presentation interface

To make work with a formal system acceptable for a researcher, teacher, or student, a presentation layer *P* is introduced over the kernel language: ordinary mathematical notation, abbreviated forms of quantifiers, symbols such as $x \in H$, $\ker f$, $H \trianglelefteq G$, templates of natural-language steps, and macros. The properties of Lean 4 as an extensible environment with a programmable parser, elaborator, and pretty-printer make such an extension natural [2], while educational systems such as Verbose Lean and Waterproof demonstrate the pedagogical value of human-oriented syntactic layers [11], [12].

Let the interface be specified by the tuple

$$I = (P, C, E, R, \Pi, K) \tag{4.1}$$

where *P* is the presentation language, *C* is the Lean kernel language, $E : P \rightharpoonup C$ is elaboration, $R : C \to P$ is rendering into a readable form, $\Pi : P \rightharpoonup C$ is the projection of executable fragments into the kernel, and *K* is the trusted kernel. The set of statements accessible to the user in language *P* is defined as

$$\mathrm{Th}(P) = \{\, s \in P \mid \exists p \in \mathrm{Term}{:}\, \mathrm{Ver}_L(E(s), p) = 1 \,\} \tag{4.2}$$

Then the formal expression of interface conservativity is the inclusion

$$E\big(\mathrm{Th}(P)\big) = \mathrm{Th}(C) \cap \mathrm{Im}(E) \tag{4.3}$$

Theorem 2. Suppose every extension of the syntax of *P* is eliminated by the function *E* into terms of the language *C*, while the rules of the kernel *K* remain unchanged. Then *P* is a conservative extension over *C*.

Proof. Let $s \in P$ be derivable. By definition (4.2), there exists p such that $VerL(E(s), p) = 1$, that is, *E(s)* belongs to *Th(C)*. Hence no statement becomes derivable solely due to the syntactic extension.

Conversely, any term $c \in C$ may be regarded as a trivial element of the presentation language or may be rendered into it via *R*. Therefore, *P* does not enlarge the set of theorems but only changes their mode of expression.

For practice, the stability of the cycle "kernel → readable form → kernel" is also useful:

$$\mathrm{NF}\left(E\left(R(c)\right)\right) = \mathrm{NF}(c), \qquad c \in C \tag{4.4}$$

where *NF* denotes normalization up to *α*-, *β*-, *δ*-, *ι*-, and *ζ*-conversions. If equality (3.4) holds on the relevant class of terms, the interface preserves not only the set of derivable statements but also syntactic identity after normalization. For a finite test set $X = \{c1, \ldots, cm\}$, interface fidelity may be measured by the indicator

$$Q_I(X) = \frac{1}{|X|}\sum_{c\in X} \mathbf{1}_{\mathrm{NF}\left(E\left(R(c)\right)\right)=\mathrm{NF}(c)} \tag{4.5}$$

In the case $QI(X) = 1$, the interface is fully faithful on the chosen set of formal objects. Several typical examples of translating ordinary mathematical notation into kernel constructions are given in Table 1.

Табл. 1. Відображення звичайної математичної нотації в конструкції Lean-ядра
Table 1. Mapping ordinary mathematical notation to Lean kernel constructions

Mathematician's language	Kernel form / elaboration	Purpose
∀ x ∈ A, P(x)	∀ x, x ∈ A → P x	Bounded quantifier abbreviation
x ∈ ker f	f x = 1	Expansion of the definition of the kernel of a homomorphism
H ⊴ G	Normal H	Abbreviated notation for a normal subgroup
a + b = b + a	Eq (a + b) (b + a)	Equality as a type object
"Let x be arbitrary …"	fun x => ...	Natural-language introduction of a quantifier

The general architecture of the proposed conservative interface is shown in Fig. 1

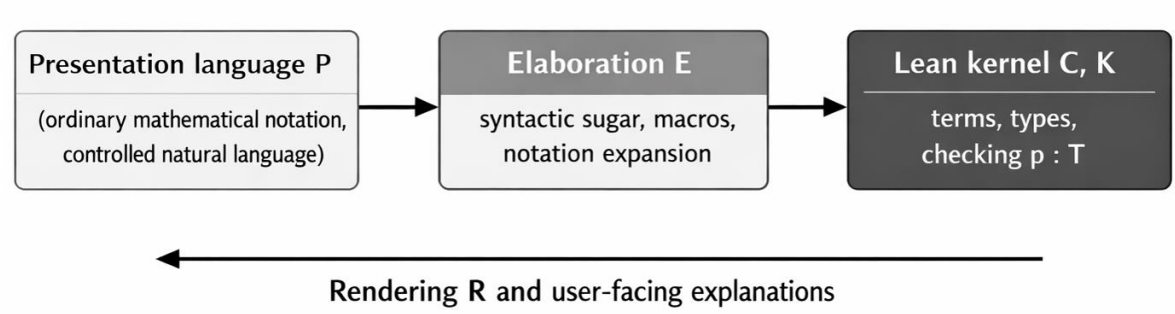

Рис. 1. Консервативна модель інтерфейсу над довіреним ядром
Fig. 1. Conservative interface model over the trusted kernel

5. Generative component as an untrusted pedagogical extension

The construction of a convenient mathematical interface is compatible with the use of AI provided that AI does not enter the trusted kernel. Let *A* be a nondeterministic operator that receives a kernel-level formal object as input and returns a set of pedagogical artefacts: explanations, examples, micro-lectures, generated exercises, and occasionally fragments of executable Lean code. Formally, let

$$A: C \rightsquigarrow Y, \qquad Y = Y_{\mathrm{text}} \cup Y_{\mathrm{exec}} \tag{5.1}$$

The elements of the set Y_{text} are purely textual and therefore do not directly affect mathematical correctness. By contrast, the elements of Y_{exec} may contain Lean fragments or other executable objects; therefore, they must be returned to the kernel via the projection function *Π*. We call the extension *A* safe if the condition

$$\mathrm{Safe}(A) \Leftrightarrow \forall c{:}T\ \forall y \in A(c){:}\ y \in Y_{\mathrm{exec}} \Rightarrow \exists c' = \Pi(y)\ \wedge\ \mathrm{Ver}_L(T, c') = 1 \tag{5.2}$$

Theorem 3. If the AI-extension *A* is safe in the sense of (5.2), then adding it to the interface does not change the set of derivable theorems and does not weaken the soundness of the system.

Proof. Artefacts from Y_{text} are non-executable and therefore logically inert: they do not generate new objects of type theorem/proof. For every $y \in Y_{exec}$, condition (5.2) requires the existence of $c' = \Pi(y)$, which is accepted by the kernel. Hence all executable AI outputs pass through the same verification channel as ordinary user proofs. It follows that the set of accepted statements coincides with $Th(C) \cap Im(E)$, whereas soundness is determined solely by the kernel *K*.

The safe interaction between the generative component and the kernel is illustrated in Fig. 2.

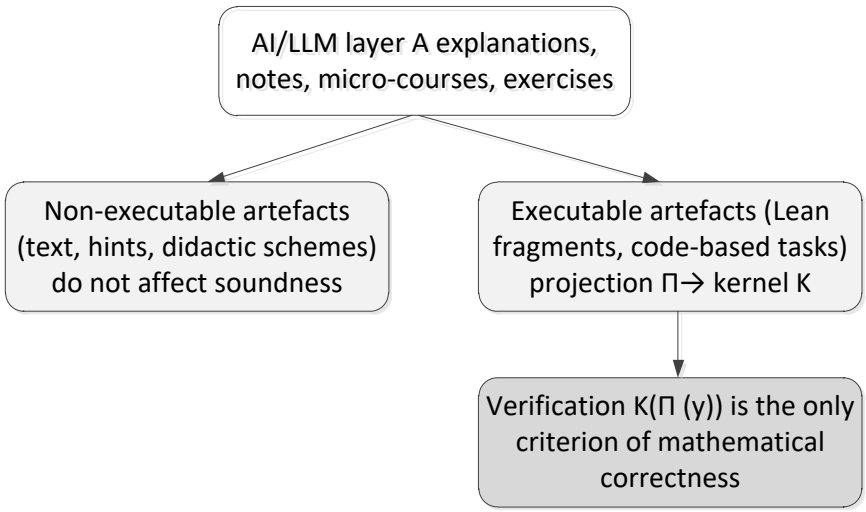

Рис. 2. Безпечне генеративне розширення: лише виконувані артефакти, що проектуються назад у ядро, можуть впливати на коректність
Fig. 2. Safe generative extension: only executable artefacts projected back into the kernel may affect correctness

It is precisely this property that provides a mathematical justification for an AI-generated learning module built over formally verified theorems. The generative component may produce explanations, short notes, exercises, and examples; however, every executable fragment must be returned to the kernel via *Π* and verified by the predicate $K(\Pi(y))$. Therefore, mathematical rigor is determined not by the stylistic quality of generation but by the architecture of interaction with the trusted kernel.

$$\mathrm{Course}_A(\Theta) = \bigcup_{T\in\Theta} \{A(T),\, R(T),\, \mathrm{examples}(T),\, \mathrm{tasks}(T)\} \tag{5.3}$$

In other words, the rigor of the course is determined not by the textual quality of the AI output but by the fact that all executable artefacts from the sets *tasks(T)* or *examples(T)* are returned to the kernel through *Π* and checked by it. In this architecture, an LLM is not a proven "mathematician" in the trusted sense; it is merely a generator of human-oriented presentation, whereas the final judge of correctness is the small trusted kernel [1], [2].

6. Illustrative example and didactic implications of formalization

To illustrate the model, consider a small set of theorems $\Theta = \{T1, T2, T3, T4\}$, covering both elementary algebra and basic group theory. All statements were selected so that they admit natural representation both in ordinary mathematical language and in compact Lean-like notation. Let us introduce the proportion of automatically verified statements in the set Θ:

$$\eta(\Theta) = \frac{1}{|\Theta|}\sum_{T\in\Theta} \mathbf{1}_{\exists p:\mathrm{Ver}_L(T,p)=1} \tag{6.1}$$

If formal proof objects have been constructed for all theorems in the set, then $\eta(\Theta) = 1$. In this illustrative case, precisely such a situation is considered: the purpose is not empirical measurement but a demonstration of how a human-oriented interface and kernel-level verification interact on specific mathematical examples.

For the same set one may evaluate interface fidelity using functional (4.5). If presentation macros for the symbols $\in$, *ker*, $\trianglelefteq$, as well as abbreviations of the form "$\forall x \in A$", are translated by the elaborator into standard Lean terms without change of meaning, then for the selected set $X = \{T1, T2, T3, T4\}$ we have

QI(X) = 1. Hence the user operates with a familiar language, while the kernel operates with the canonical form.

The four selected theorems together with their formal representations and the role of the interface are shown in Table 2.

Табл. 2. Ілюстративний набір формалізованих тверджень
Table 2. Illustrative set of formalized statements

Label	Mathematical statement	Formal image type	VerL	Role of the interface
T1	In a group, the identity element is unique	∀ e_1 e_2 : G, Id e_1 → Id e_2 → e_1 = e_2	1	Abbreviated notation Id, G
T2	The intersection of two subgroups is a subgroup	Sub(H) → Sub(K) → Sub(H ∩ K)	1	Notation ∩ and x ∈ H
T3	The kernel of a homomorphism is a normal subgroup	Hom(f : G → G′) → Normal(ker f)	1	Abbreviations ker, ⊴
T4	Binomial identity	∀ x y : R, (x+y)^2 = x^2 + 2xy + y^2	1	Standard algebraic notation

It is characteristic that the same principle extends to the construction of a learning module. For example, for *T3* the generative component may produce a short explanation such as “to prove that the kernel is normal, it is sufficient to show closure under conjugation”, as well as auxiliary exercises on computing *ker f* and working with the definition of *Normal*. However, the correctness of such materials is not established by the explanatory text itself; it is guaranteed only when all executable fragments are returned through *Π* to the kernel and re-checked by the predicate *K(Π(y))*. This is why even an AI-supported instructional module may remain mathematically rigorous.

7. Educational opportunities and limits of applicability

The proposed model has a direct didactic consequence: to every formally verified statement one may assign a learning object $\mathcal{U}(T, p)$, consisting of an explanation, a system of verified examples, a set of exercises, and hints. Only those components of such an object that are projected into executable kernel artefacts are mathematically significant; informal texts may improve intelligibility but do not participate in establishing the truth of the statement.

Therefore a natural requirement for a learning module is the inclusion $Exec(\mathcal{U}(T, p)) \subseteq \mathcal{P}ver$, where $\mathcal{P}ver$ is the set of all fragments of the kernel language that successfully pass type checking and thus belong to verified formal content. Under this condition, any example, proof template, or exercise with a code fragment preserves the same mathematical correctness as the original proof object *p*.

This also yields a methodological restriction: the generative component cannot be an independent arbiter of correctness; it can only serve as a means of explanation, reformulation, and structuring of already verified content. Accordingly, in the educational process it is expedient to distinguish three levels: verified kernel knowledge, the presentation interface, and generative support services.

It is precisely such a separation that makes it possible to construct courses for researchers, teachers, and students in which formal rigor is not lost in the transition to ordinary mathematical language. At the level of instructional design, this means the possibility of building verifiable examples, parameterized exercises, and systems of contextual hints for which the only criterion of correctness remains repeated verification of executable fragments by the Lean kernel.

8. Conclusions

The paper has developed a mathematical model of automatic verification of formalized proofs in Lean and a model of a conservative human-oriented presentation interface over its trusted kernel. The formalizable fragment of mathematical discourse, the kernel-verification predicate, a course formalizability functional, the elaboration and rendering model, and the safety condition for a generative extension have been defined.

It has been proved that, given adequate encoding, any statement from the set $\mathscr{ML}$ for which a proof object p has been constructed can be verified automatically. It has also been shown that ordinary mathematical notation, abbreviations, macros, and natural-language explanations do not enlarge the set of derivable statements whenever they are eliminated into kernel terms without changing the rules *K*.

A separate result is the formal safety condition for the generative component. It shows that generative technologies may be used to construct learning modules, lecture notes, explanations, examples, and assessment materials, but should not be regarded as a source of mathematical truth. The practical significance of the results lies in the possibility of building intelligent educational systems for courses in higher mathematics, discrete mathematics, algebra, and mathematical logic, in which the rigor of formalization is combined with an interface suitable for educational use.

REFERENCES

1. L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer, “The Lean Theorem Prover (System Description),” in *Automated Deduction – CADE-25*, vol. 9195, 2015, pp. 378–388, https://doi.org/doi: 10.1007/978-3-319-21401-6_26.
2. L. de Moura and S. Ullrich, “The Lean 4 Theorem Prover and Programming Language,” in *Automated Deduction – CADE 28*, vol. 12699, 2021, pp. 625–635, doi: 10.1007/978-3-030-79876-5_37.
3. The mathlib Community, “The Lean Mathematical Library,” in *Proc. CPP 2020*, 2020, pp. 367–381, https://doi.org/doi: 10.1145/3372885.3373824.
4. T. Coquand and G. Huet, “The calculus of constructions,” *Inf. Comput.*, vol. 76, no. 2–3, pp. 95–120, 1988, doi: 10.1016/0890-5401(88)90005-3.
5. P. Martin-Löf, *Intuitionistic Type Theory*. Napoli, Italy: Bibliopolis, 1984.
6. P. Wadler, “Propositions as Types,” *Commun. ACM*, vol. 58, no. 12, pp. 75–84, 2015, https://doi.org/10.1145/2699407.
7. J. Avigad and P. Massot, *Mathematics in Lean*. Lean community tutorial, 2025. [Online]. Available: https://leanprover-community.github.io/mathematics_in_lean/
8. A. Thoma and P. Iannone, “Learning about Proof with the Theorem Prover LEAN: the Abundant Numbers Task,” *Int. J. Res. Undergrad. Math. Educ.*, vol. 8, pp. 64–93, 2022, doi: 10.1007/s40753-021-00140-1.
9. P. Iannone and A. Thoma, “Interactive theorem provers for university mathematics: an exploratory study of students’ perceptions,” *Int. J. Math. Educ. Sci. Technol*., 2024, https://doi.org/doi: 10.1080/0020739X.2023.2178981.
10. X. K. Yan and G. Hanna, “Using the Lean interactive theorem prover in undergraduate mathematics,” *Int. J. Math. Educ. Sci. Technol*., 2023, https://doi.org/doi: 10.1080/0020739X.2023.2227191.
11. P. Massot, “Teaching Mathematics Using Lean and Controlled Natural Language,” in *Leibniz Int. Proc. Inform. (LIPIcs)*, vol. 309, 2024, Art. no. 27, https://doi.org/doi: 10.4230/LIPIcs.ITP.2024.27.
12. J. Wemmenhove et al., “Waterproof: Educational Software for Learning How to Write Mathematical Proofs,” arXiv:2211.13513, 2022, https://doi.org/doi: 10.48550/arXiv.2211.13513.
13. O. Yevdokymov, A. Chukhray, and T. Stoliarenko, “Operationalizing the Formalizability of Mathematics Problems for Intelligent Tutoring Systems: Taxonomy, Measurement Protocol, and Educational Impact,” *CEUR Workshop Proc*., vol. 4164, paper 25, 2026.
14. A. Chukhray, D. Dvinskykh, V. Narozhnyy, and T. Stoliarenko, “Using an expression tree for adaptive learning,” in 2023 13th Int. Conf. Dependable Syst., Services Technol. (DESSERT), Athens, Greece, 2023, pp. 1–5, https://doi.org/10.1109/DESSERT61349.2023.10416497.
15. A. Chukhray, T. Stoliarenko, O. Yevdokymov, and V. Demyanenko, “Possibilities of using Intelligent Tutoring Systems (ITS) in Higher Mathematics Courses,” *Open Inf. Comput. Integr. Technol.*, no. 102, pp. 92–119, 2025, https://doi.org/10.32620/oikit.2024.102.07.

Євдокимов Олександр *аспірант, кафедра математичного моделювання та штучного інтелекту, Національний аерокосмічний університет «Харківський авіаційний інститут», вул. Вадима Манька, 17, Харків, Україна, 61070*
e-mail: o.yevdokymov@khai.edu
https://orcid.org/0009-0008-9687-6344

Лучшева Оксана *старший викладач, кафедра інженерії програмного забезпечення, Національний аерокосмічний університет «Харківський авіаційний інститут», вул. Вадима Манька, 17, Харків, Україна, 61070*
e-mail: o.luchsheva@khai.edu ;
https://orcid.org/0000-0003-3855-2815

Математична модель автоматичної верифікації формалізованих доказів і консервативний інтерфейс представлення в Lean

Актуальність. Інтерактивні системи доведення теорем, такі як Lean, дозволяють перенести остаточний контроль математичної правильності від людини до малого довіреного ядра. Однак їх широке застосування в дослідженнях і вищій освіті все ще обмежене значним розривом між строгим формальним мовою ядра, заснованою на залежних типах, та звичною інтуїтивною математичною нотацією, символами й міркуваннями природною мовою, якими користуються математики та студенти. Цей розрив створює серйозні перешкоди для навчання доведенню, формалізації нових результатів і розробки ефективних інтелектуальних освітніх систем.
Мета. Метою роботи є побудова строгої математичної моделі автоматичної верифікації формалізованих доказів у Lean та надання формального обґрунтування можливості створення консервативного інтерфейсу, орієнтованого на користувача, разом з ненадійним генеративним педагогічним компонентом над довіреним ядром без втрати математичної строгості.
Методи. Дослідження ґрунтується на теорії залежних типів, метатеоретичних властивостях обчислення конструкцій (коректність, сильна нормалізація, розв'язність перевірки типів), моделях розробки та зворотної проекції між мовою представлення і мовою ядра, а також формальних предикатах правильності типізації термінів. Безпека генеративного AI-компонента визначається умовою, що кожен виконуваний артефакт має бути спроектований назад у ядро і повторно перевірений довіреним механізмом перевірки типів.
Результати. У розробленій моделі визначено формалізований фрагмент математичного дискурсу $\mathscr{M}$L, предикат верифікації ядра VerL, функціонал формалізованості курсу $\mathbb{F}(\mathcal{S})$, повну модель консервативного інтерфейсу та умову безпеки генеративного розширення. Доведено, що синтаксичний цукор, макроси, скорочення нотацій та AI-генеровані освітні артефакти не розширюють множину виводних тверджень, якщо довірене ядро Lean залишається незмінним і всі виконувані фрагменти повторно перевіряються ядром. Ілюстративний приклад з чотирма теоремами з алгебри та теорії груп демонструє, що нотація, орієнтована на користувача, повністю розгортається в терміни ядра зі збереженням математичного змісту та перевірюваності. Показник вірності інтерфейсу на тестовому наборі досягає максимального значення 1.
Висновки. Отримані результати демонструють, що формальну мову Lean можна оснастити зручним інтерфейсом, орієнтованим на користувача, та інтелектуальними освітніми сервісами, придатними для наукового та навчального використання, без втрати математичної строгості. Єдиним критерієм істинності залишається повторна верифікація малим довіреним ядром. Запропоноване розділення між верифікованими знаннями ядра, консервативним шаром представлення та ненадійними генеративними сервісами створює надійну основу для побудови інтелектуальних освітніх систем з вищої математики, дискретної математики, алгебри та математичної логіки, у яких формальна правильність гарантується незалежно від якості згенерованих пояснень і вправ.

Ключові слова: *Lean, залежні типи, автоматична верифікація доказів, формалізована математика, консервативний інтерфейс, генеративний педагогічний модуль.*

Надійшла у першій редакції 20.01.2026; рекомендована до друку 04.03.2026; опублікована 03.04.2026.
The first version has been received on 20.01.2026; was recommended for printing on 04.03.2026; published 03.04.2026.