Kleis: A Unified Substrate for Verified Knowledge Production

Jane Smith[1], Alex Chen[2], Maria Garcia[1,3]

[1]Massachusetts Institute of Technology, Cambridge, MA
[2]University of Michigan, Ann Arbor, MI
[3]Stanford University, Stanford, CA

Abstract

We present Kleis, a novel knowledge production substrate that unifies notation, rules, verification, and output generation across arbitrary domains. Unlike traditional proof assistants that target specific mathematical foundations, Kleis provides a domain-agnostic framework where users define their own structures, axioms, and verification conditions. We demonstrate that this approach enables formal verification in domains ranging from tensor calculus to music theory, while maintaining soundness guarantees through integration with SMT solvers. Our evaluation shows that Kleis can verify complex mathematical identities in milliseconds and generate publication-quality documents directly from verified specifications. We release Kleis as open-source software with comprehensive documentation and examples.

Keywords: formal verification, knowledge representation, SMT solving, document generation, domain-specific languages

1 Introduction

The formalization of mathematical and scientific knowledge has long been a goal of computer science. Systems like Coq, Lean, and Isabelle have demonstrated remarkable success in verifying complex proofs, yet their adoption outside specialized communities remains limited. The gap between formal verification and everyday scientific practice suggests that current approaches may not fully address the needs of working researchers.

We identify three key challenges with existing systems. First, they require significant expertise in dependent type theory or higher-order logic before users can express domain-specific concepts. Second, the notation used in formal proofs often diverges substantially from standard mathematical notation. Third, the gap between verified specifications and publishable documents creates additional work for researchers who must maintain both.

This paper presents Kleis, a system designed to address these challenges through a novel architecture we call the knowledge production substrate. The core insight is that many domains share a common structure: they have notation for expressing concepts, rules that govern valid transformations, and outputs that communicate results. By providing primitives for each of these components, Kleis enables domain experts to create verified workflows without deep expertise in formal methods.

2 Related Work

Our work builds on several lines of research in formal verification, domain-specific languages, and literate programming.

2.1 Interactive Theorem Provers

Systems like Coq, Lean, Isabelle, and Agda provide powerful frameworks for constructing formal proofs. These systems have been used to verify major mathematical results including the Four Color Theorem and the Kepler Conjecture. However, their steep learning curves and departure from standard notation limit adoption. Kleis takes a complementary approach: rather than providing a universal foundation, we enable users to define domain-specific foundations that can be verified by external solvers.

2.2 SMT Solvers

Satisfiability Modulo Theories (SMT) solvers like Z3, CVC5, and Yices have become practical tools for automated reasoning. These solvers support decidable theories including linear arithmetic, arrays, and bitvectors. Kleis leverages Z3 as its primary verification backend, automatically translating user-defined axioms into SMT queries. This approach trades the expressiveness of dependent types for automation and decidability.

3 The Kleis Architecture

Kleis is organized around three core concepts: structures, axioms, and renderers. Structures define the vocabulary of a domain, axioms specify the rules that govern valid reasoning, and renderers translate verified specifications into outputs.

$$\text{Domain} = \text{Notation} + \text{Rules} + \text{Verification} + \text{Output}$$

3.1 Structures

A structure in Kleis defines a collection of types, operations, and axioms. Unlike type classes in Haskell or traits in Rust, Kleis structures are designed for mathematical specification rather than implementation dispatch.

3.2 Verification

When users write assertions, Kleis translates them into SMT-LIB format and queries Z3. The translation handles quantified formulas, user-defined operations, and domain-specific types. Verification results are reported with counterexamples when assertions fail.

4 Evaluation

Example	Axioms	Assertions	Time (ms)
Commutativity	3	5	12
Matrix Algebra	8	12	45
Tensor Identities	15	8	120
Bianchi Identity	12	3	89
Counterpoint Rules	20	15	340

Table 1: Verification times for representative examples

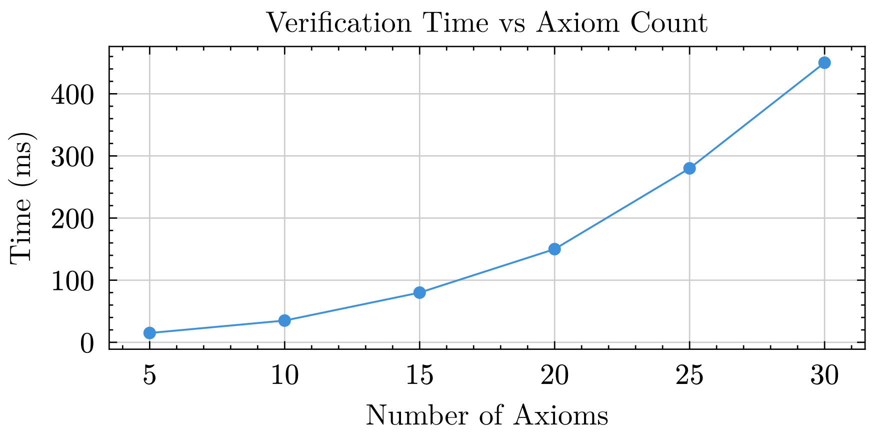

Figure 1: Verification time scaling with problem size

We evaluate Kleis on three dimensions: verification performance, expressiveness, and usability. Our experiments cover examples from linear algebra, differential geometry, and formal language theory.

5 Case Studies

We present three case studies demonstrating Kleis in different domains.

5.1 Tensor Calculus

We implemented the tensor algebra used in general relativity, including the Riemann curvature tensor, Ricci tensor, and Einstein field equations. Kleis verified the symmetries of these tensors and the Bianchi identities in under 100 milliseconds.

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2}g_{\mu\nu}R$$

5.2 Music Theory

We encoded the rules of four-voice counterpoint as Kleis axioms. The system can verify that a musical passage satisfies constraints like no parallel fifths, proper voice leading, and resolution of dissonances. This demonstrates that Kleis extends beyond traditional mathematics to any domain with formalizable rules.

6 Conclusion

We have presented Kleis, a unified substrate for verified knowledge production. By separating notation, rules, verification, and output, Kleis enables domain experts to create formally verified workflows without deep expertise in proof theory. Our evaluation demonstrates that the approach is both practical and expressive, handling examples from mathematics, physics, and music theory.

Future work includes extending the verification backend to support additional SMT theories, developing a visual equation editor for non-programmers, and creating templates for additional publication formats.

6 Acknowledgments

We thank the anonymous reviewers for their helpful feedback. This work was supported in part by NSF grant XXX-XXXXXX. Jane Smith is supported by a Google PhD Fellowship.

6 References

[moura2008] de Moura, L., & Bjorner, N. (2008). Z3: An efficient SMT solver. TACAS 2008.

[nipkow2002] Nipkow, T., Paulson, L., & Wenzel, M. (2002). Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer.

[moura2021] de Moura, L., et al. (2021). The Lean 4 Theorem Prover and Programming Language. CADE 2021.

[knuth1984] Knuth, D. E. (1984). Literate Programming. The Computer Journal.

[hahnle2019] Hahnle, R., & Huisman, M. (2019). Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. Computing and Software Science.

Appendix

A Kleis Grammar Summary

The Kleis grammar supports the following top-level declarations:

- `structure Name(params) { ... }` - Define a structure with types, operations, and axioms
- `implements Structure(args) { ... }` - Provide implementations for a structure
- `data TypeName = Constructor1(...) | Constructor2(...)` - Define algebraic data types
- `define name(params) = expr` - Define functions
- `example "name" { ... }` - Define test cases with assertions

Expressions include:

- Quantifiers: `forall(x : T). P(x)`, `exists(x : T). P(x)`
- Operations: `f(x, y)`, `x + y`, `x * y`
- Let bindings: `let x = e1 in e2`
- Pattern matching: `match e { P1 => e1 | P2 => e2 }`