Contents

Kleis: A Structural Mathematics Editor and Verification Engine

Version: 0.1.0
Date: December 2025
Author: Kleis Development Team

Executive Summary

Kleis is a structural mathematics editor that treats mathematical expressions as typed, verifiable data structures rather than strings of text. Unlike LaTeX, which renders symbols without understanding their meaning, Kleis maintains the semantic structure of mathematics, enabling type-checking, verification, and programmatic manipulation.

The ultimate vision is for Kleis to become a **universal verification layer for structured knowledge**, applicable not just to mathematics but to any domain requiring formal reasoning: physics, business rules, legal contracts, and AI-generated content.

---

1. The Problem with Current Mathematical Tools

1.1 LaTeX: Beautiful but Meaningless

LaTeX produces beautiful mathematical typography, but treats formulas as strings:

```
\frac{d}{dx} \int_0^x f(t) \, dt = f(x)
```

To LaTeX, this is just layout instructions. It doesn't know that: - The left side is a derivative of an integral - `f` is a function, `t` is a bound variable, `x` is free - This is the Fundamental Theorem of Calculus - The equation is only valid if `f` is continuous

1.2 The Consequences

1. **No error checking:** You can write `dx = x²` and LaTeX will render it beautifully
2. **No verification:** Cannot check if equations are dimensionally consistent
3. **No reuse:** Can't import definitions from one paper to another
4. **AI vulnerability:** LLMs generate plausible-looking but incorrect formulas

1.3 The Need

We need a mathematical representation that is: - **Structured:** Expressions are trees, not strings - **Typed:** Every symbol has a declared type - **Verifiable:** Consistency can be checked automatically - **Renderable:** Still produces beautiful output

---

2. The Kleis Solution

2.1 Core Concept: Expressions as ASTs

In Kleis, the expression `f/ x` is not a string but an Abstract Syntax Tree:

```
Operation {
  name: "partial",
  args: [
    Object("f"),
    Object("x")
  ]
}
```

This structure preserves meaning: - We know this is a partial derivative - We know `f` is being differentiated with respect to `x` - We can transform, simplify, or verify this expression

2.2 Multi-Target Rendering

The same AST renders to multiple formats:

Target	Output
Unicode	`f/ x`
LaTeX	`\frac{\partial f}{\partial x}`
HTML/MathML	`<mfrac>...</mfrac>`
Typst	`(diff f)/(diff x)`

2.3 The Template System

Kleis includes 54+ mathematical templates covering: - Calculus (derivatives, integrals, limits) - Linear Algebra (matrices, determinants, traces) - Quantum Mechanics (bra-ket notation, operators) - Set Theory (membership, quantifiers) - Physics (tensors, field equations) - POT Operations (projections, modal integrals)

---

3. The Type System

3.1 Why Types Matter

Consider Einstein's field equation:

```
G_  + Λg_  = T_
```

Without types, these are just symbols. With types:

```
G : Tensor(M, [lower, lower])      // Einstein tensor
g : Tensor(M, [lower, lower])      // Metric tensor
T : Tensor(M, [lower, lower])      // Stress-energy tensor
Λ : Scalar                          // Cosmological constant
  : Scalar                         // Coupling constant
```

Now Kleis can verify: - Both sides are (0,2) tensors □ - Index positions match □ - Dimensions are consistent □

3.2 Algebraic Hierarchy

Kleis implements a formal algebraic hierarchy:

```
Magma
     Semigroup (associativity)
           Monoid (identity)
                 Group (inverse)
                       AbelianGroup (commutativity)
                             Ring (two operations)
                                   Field (multiplicative inverse)
                                         VectorSpace
                                               HilbertSpace
```

Each structure inherits axioms from its parents.

3.3 Scope Hierarchy

Kleis implements a 5-level scope model for academic papers:

1. **Package Scope:** Imported libraries (`std.quantum`, `std.pot`)
2. **Paper Scope:** Document-wide notation
3. **Section Scope:** Local overrides (with explicit `shadow` keyword)
4. **Block Scope:** Theorem/definition locality
5. **Cell Scope:** Ephemeral scratch work

This matches how mathematicians actually write papers.

---

4. Projected Ontology Theory (POT) Integration

4.1 What is POT?

Projected Ontology Theory is a framework where: - **Ontology resides in an abstract modal space** (Hilbert Ontology, □ $_o$□ □) - **Spacetime (□ ^{4}) is a projection** of this deeper structure - **Physical "constants" are projection residues**, not fundamental

4.2 The Projection Kernel

The central object is the projection operator:

```
Π[ ](x) =  _  K(x,m)  (m) dm
```

Where: - is a modal state - `K(x,m)` is the projection kernel - `x  M` is a spacetime point

4.3 Variable Constants

In POT, the speed of light `c(x)` is a field, not a constant:

```
define c : M →
axiom local_constancy:  c(x)   0  // approximately constant locally
axiom projection_residue: c(x) = Res[K, "lightcone"](x)
```

This eliminates the need for cosmological inflation and resolves tensions in current physics.

4.4 Kleis Operations for POT

Kleis includes specialized templates:

Operation	Symbol	Purpose
`projection`	`Π[ ](x)`	Modal-to-spacetime projection
`modal_integral`	`_  ... dm`	Integration over modal space
`projection_kernel`	`K(x,m)`	The kernel function
`projection_residue`	`Res[K,s]`	Extract projection residue
`causal_bound`	`_c(x,r)`	Variable-c causal boundary
`hont`		Hilbert Ontology space

---

5. Use Cases

5.1 Mathematical Research

- Build formulas structurally, not as strings
- Type-check equations before submission
- Verify dimensional consistency
- Export to LaTeX, Typst, or HTML

5.2 AI-Assisted Mathematics

The workflow: 1. LLM generates formula 2. Kleis type-checks it 3. Errors flagged automatically 4. Human reviews verified result

This catches AI hallucinations like: - Incompatible Hilbert spaces - Dimension mismatches - Invalid operations

5.3 Physics and Engineering

- Model field theories with proper tensor types
- Verify gauge invariance
- Check conservation laws
- Type system prevents nonsense (e.g., adding scalar to vector)

5.4 Education

- Students build formulas visually
- Immediate feedback on type errors
- Understand mathematical structure
- Learn why certain operations are invalid

5.5 Future: Business and Legal

The same type system can verify: - Purchase orders (totals match line items) - Contracts (clauses are consistent) - Compliance rules (axioms satisfied)

---

6. Technical Architecture

6.1 Core Components

```
                    Web UI (HTML/JS)
                  Structural Editor

                    Axum Server
              REST API + WebSocket

                   Kleis Core (Rust)

   AST         Renderer      Type Checker
 (Expression)     (4 targets)     (in progress)


                  Typst Compiler
              (SVG generation)
```

6.2 Expression AST

```
enum Expression {
    Const(String),           // "42", " "
    Object(String),          // "x", " "
    Placeholder { id, hint }, //   (unfilled slot)
    Operation { name, args }, // "frac", "integral", etc.
}
```

6.3 Template Registry

54+ templates with 4 rendering targets each = 200+ rendering rules.

Each template specifies: - Operation name - Argument structure - Rendering patterns for Unicode, LaTeX, HTML, Typst

6.4 Semantic Positioning

Kleis tracks bounding boxes for each sub-expression, enabling: - Click-to-edit - Cursor navigation - Precise overlay positioning

---

7. Vision and Roadmap

Phase 1: Mathematical Foundation (Current)

- □ Structural editor
- □ Multi-target rendering
- □ Template system (54+ operations)
- □ POT operations
- □ Type inference
- □ Scope management

Phase 2: Verification Engine

- Type checking for expressions
- Axiom validation
- Dimensional analysis
- Proof obligations

Phase 3: AI Integration

- `/api/verify` endpoint
- LLM □ Kleis □ Human workflow
- Type error feedback
- Suggested fixes

Phase 4: Visual Authoring

- Define custom operations
- Design new glyphs
- Create type signatures
- Package distribution

Phase 5: Executable Mathematics

- Notebook interface
- Context management
- Proof assistant integration
- Collaborative reasoning

Ultimate Vision

Kleis becomes the universal verification layer for structured knowledge.

In the AI era: LLMs generate content quickly; Kleis verifies it rigorously. The combination makes AI-assisted formal reasoning actually reliable.

---

8. The arXiv Vision

8.1 Current State

Academic papers are PDFs—static, unverifiable, un-importable.

8.2 The Kleis Future

Papers become `.kleis` documents: - Type definitions are importable - Equations are verifiable - Citations become imports - Peer review includes type checking

```
import "arxiv:2301.12345" as EFE
use EFE.einstein_tensor as G
```

8.3 Quality vs Quantity

arXiv has 2+ million papers. Quality is uneven. With Kleis: - Verified papers are marked as such - Type errors are visible - Trust is earned, not assumed

---

9. Philosophy: Simplification as Cognition

9.1 The Insight

Simplification isn't about computation—it's about human understanding.

`x² - 1` and `(x+1)(x-1)` are mathematically identical but cognitively different.

9.2 Kleis Approach

- Evaluation is minimal and meaning-preserving
- Simplification is optional and human-directed
- Context matters (what's "simpler" depends on purpose)

9.3 The Larger Point

Mathematics is a human activity. Tools should serve cognition, not just computation.

---

10. Conclusion

Kleis represents a paradigm shift in mathematical software:

Traditional Tools	Kleis
Strings	Structures
Rendering	Verification
Documents	Programs
Trust	Proof

By treating mathematics as typed data, Kleis enables: - Error detection - Semantic manipulation - Cross-document imports - AI verification

The goal is not just better math rendering, but **trustworthy mathematical communication** in an age of AI-generated content.

References

- ADR-001 through ADR-013: Architecture Decision Records
- docs/KLEIS_TYPE_UX.md: Type System UX Design
- docs/kleis_vision_executable_math.md: Long-term Vision
- docs/POT_FORMALISM.md: Projected Ontology Theory
- README.md: Technical Documentation

Appendix A: Template Gallery (Selected)

Calculus

- `frac(a, b)` □ a/b
- `derivative(f, x)` □ df/dx
- `integral(f, a, b, x)` □ $\int_a^b$ f dx
- `limit(f, x, a)` □ lim_{x□ a} f

Linear Algebra

- `matrix2x2(a,b,c,d)` □ 2×2 matrix
- `determinant(A)` □ det(A)
- `trace(A)` □ tr(A)
- `transpose(A)` □ A□

Quantum Mechanics

- `ket( )` □ |ψ□
- `bra( )` □ □ ψ|
- `inner(u, v)` □ □ u, v□
- `outer( , )` □ |ψ□ □ ϕ|
- `commutator(A, B)` □ [A, B]

POT Operations

- `projection( , x)` □ Πψ
- `modal_integral(f, , m)` □ ∫_□ f dm
- `projection_kernel(x, m)` □ K(x, m)
- `projection_residue(K, s)` □ Res[K, s]
- `hont` □ □$_o$□ □

Appendix B: Glossary

AST: Abstract Syntax Tree - tree representation of expression structure

Hilbert Ontology (□$_o$□ □): The abstract modal space in POT where ontology resides

Modal Space: The fundamental space from which spacetime is projected

POT: Projected Ontology Theory - framework where spacetime is a projection

Projection Kernel: Function K(x,m) mapping modal states to spacetime points

Projection Residue: Physical “constants” as stable features of the projection

Structural Editor: Editor that manipulates AST directly, not text

Template: Definition of a mathematical operation with rendering rules

Type System: Formal system assigning types to expressions and checking consistency

VSL: Variable Speed of Light - the idea that c can vary as a field

---

Kleis: Where formal structure meets executable mathematics.