Intro to Turn-Lang's formal library - Turn-Formal

A path to formalize critical subjects (in Rust)

Turner, Creator of turn-lang.com

2025-04-19

Presentation Overview

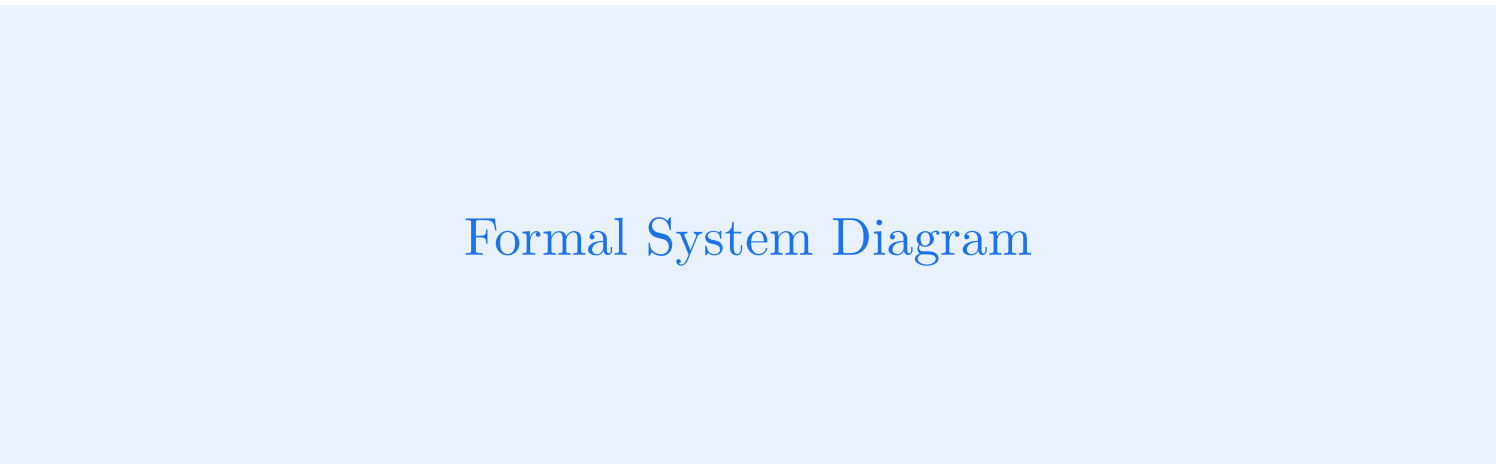
Today we'll discuss:

1. What is formal mathematics and why we care?
2. Why "high-level" formal mathematics?
3. How Turn-Formal is better than Lean4 & others
4. Building a developer-friendly formal system
5. Future roadmap

What is Formal Mathematics?

Formal systems provide:

- Mathematical foundation with machine-checkable proofs
- Precise, unambiguous language and syntax
- Well-defined rules of inference
- Mechanically verifiable correctness



Why We Care About Formalization

Why formalization matters:

- Eliminates ambiguity in mathematical proofs
- Enables machine verification of correctness
- Facilitates the development of verified software
- Bridges the gap between theory and applications

Applications span:

- Cryptographic protocols
- Safety-critical systems
- Verified compilers
- Advanced mathematics
- AI reasoning systems

Why "High-Level" Formal Mathematics?

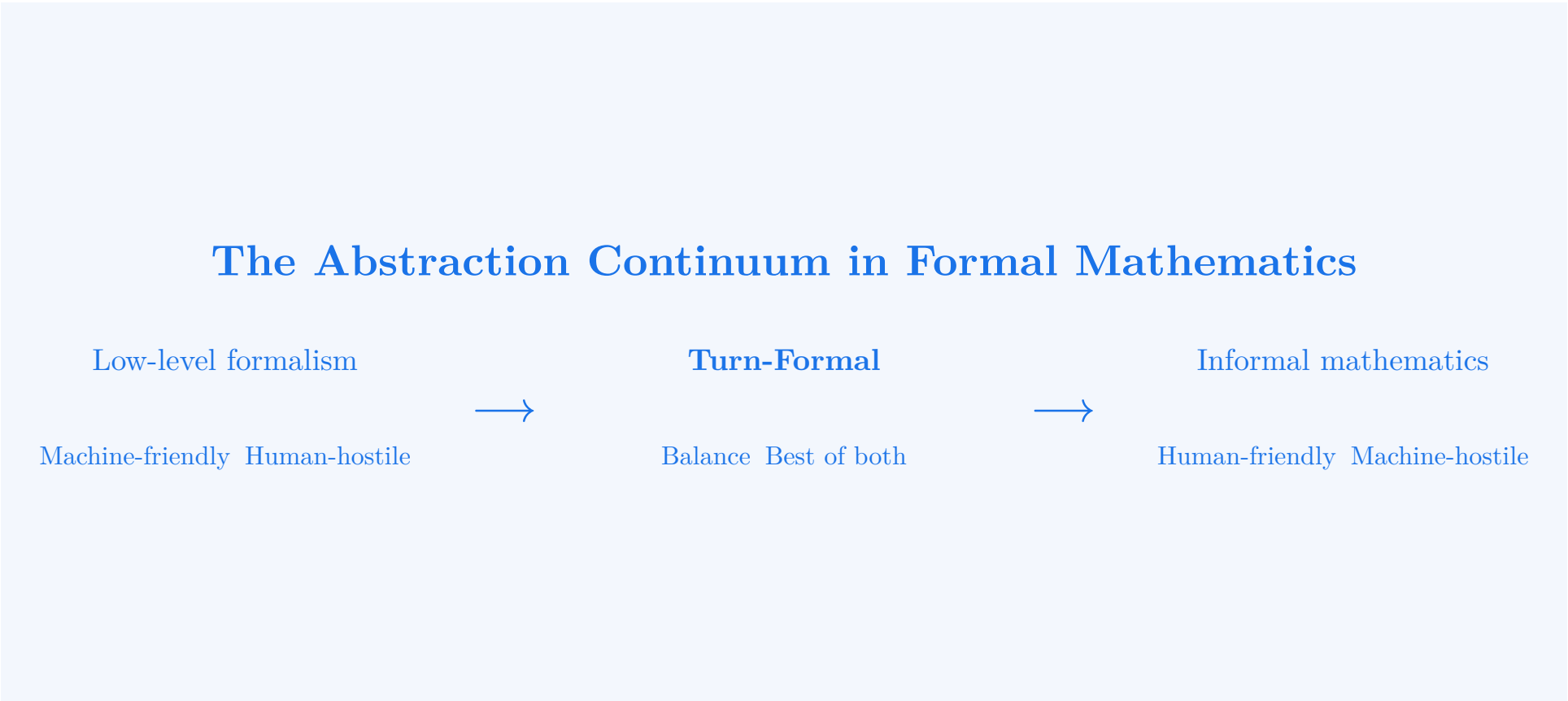
Traditional formal systems operate at a low level of abstraction:

- Human mathematicians don't think in terms of basic inference rules
- Proofs become overly verbose and hard to understand
- The gap between informal and formal proofs is too wide

High-level formal mathematics aims to:

- Match the intuition and workflow of human mathematicians
- Abstract away mechanical details while maintaining rigor
- Provide a natural language-like experience

The Abstraction Gap



How Turn-Formal is Better than Lean4 & Others

Key advantages of Turn-Formal:

- **Performance**: Rust's speed & memory safety
- **Accessibility**: Lower barrier to entry
- **Integration**: Seamless Rust ecosystem interop
- **Modularity**: Flexible architecture
- **Expressiveness**: Rich syntax for intuitive proofs
- **Developer-centric**: Built for engineers

Feature	Turn-Formal	Lean4	Coq
Implementation	Rust	Lean	OCaml
Memory safety	Native	Runtime	Runtime
Learning curve	Moderate	Steep	Steep

Building a Developer-Friendly Formal System

Turn-Formal makes verification accessible through:

- Familiar Rust syntax and semantics
- Strong type system
- Flexible tactics system
- Chainable, fluent API

Core components:

- **ProofState**
- **Tactics**
- **TheoremBuilder**
- **ProofBranch**
- **MathRelation**

Code Example

Example: Creating a proof branch

```
let state = ProofState::new();
let branch1 = state
    .tactics_intro_expr("a", MathExpression::Var(Identifier::E(1)), 0)
    .tactics_intro_expr("b", MathExpression::Var(Identifier::E(2)), 1);

// Add some proof steps
let p1 = branch1.tactics_intro_expr("a", create_var("a"), 1);
let p2 = p1.tactics_intro_expr("b", create_var("b"), 2);

// Mark as complete
let p3 = p2.should_complete();
```

Future Roadmap

Development timeline:

Short-term (2023-2024)

- Core library
- Foundation math
- Basic tactics
- Developer tools

Medium-term (2024-2026)

- Domain libraries
- Proof search
- Interoperability
- Community ecosystem

Long-term (2026+)

- AI integration
- Formal verification for all
- Cross-system translation
- Industrial applications

Our vision: Make formal verification a standard part of software development

Thank You!

Get involved with Turn-Formal:

Website: turn-lang.com GitHub: github.com/turn-lang/turn-formal Documentation: docs.turn-lang.com/turn-formal