Exploring Aspects of Formal Specification Engineering

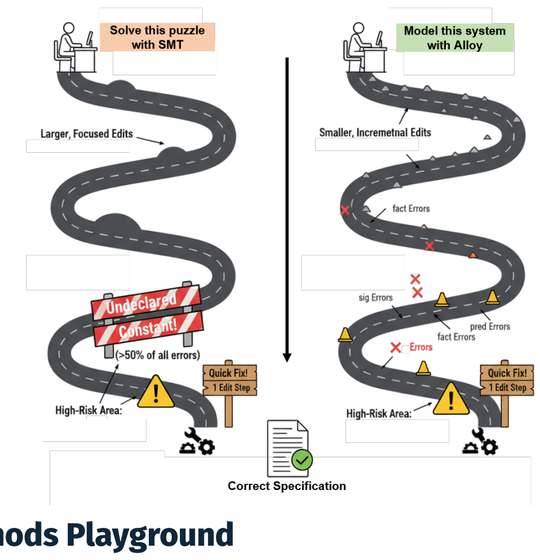
Soaibuzzaman

Bauhaus-University Weimar, Germany

Introduction & Motivation

Formal methods are powerful, but their notoriously steep learning curve hinders adoption. To build better tools and teaching strategies, we must first answer a fundamental question: **How do novices actually write formal specifications?**

- **The Problem:** The actual step-by-step process of how beginners learn, their mistakes, revisions, and strategies, is largely an unobserved 'black box.'
- **Our Approach:** We captured thousands of fine-grained 'edit paths' from students using our Formal Methods Playground, a web-based learning platform.
- **The Contribution:** This provides a unique, side-by-side view into the real-world struggles of learning two distinct formalisms: SMT-LIB (for problem-solving) and Alloy (for modeling).



The Formal Methods Playground

- **Formal Methods Playground** is a web application for writing and analyzing specifications in various modeling and specification languages.
- Offers a user-friendly interface for creating, editing, and evaluating formal specifications **without needing local installation**.
- Provides features like **sharing specifications** through permalinks, storing specifications, syntax highlighting, and more.
- Equips **language support** for various formal specification languages, including SMT-LIB and Spectra.
- **Open source** and **extensible**, allowing users to contribute new languages and features.

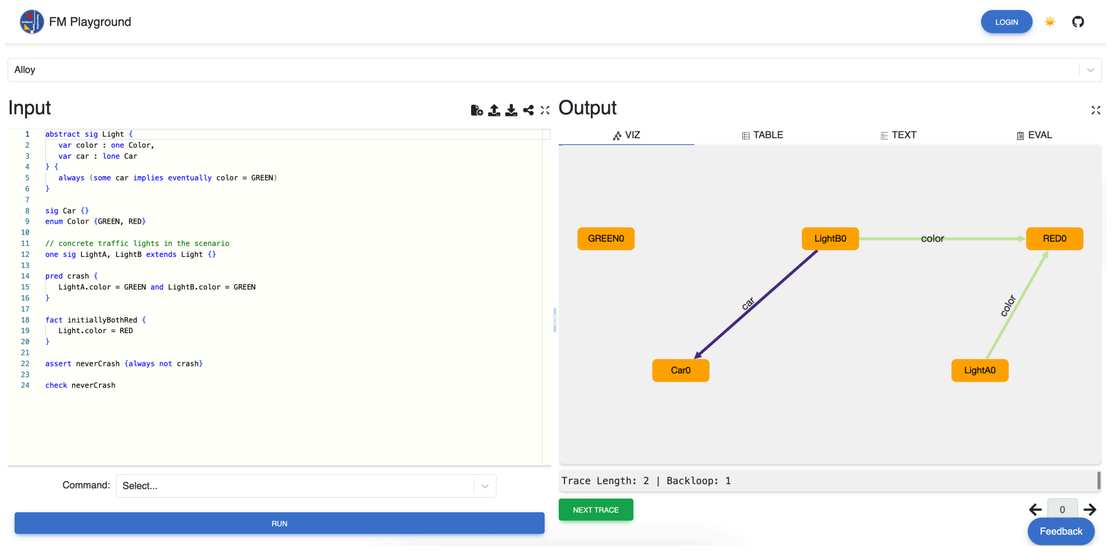


Figure 1. The Formal Methods Playground, showing an example Alloy specification.

Datasets

Our analysis is built on two novel datasets, FMP_{smt} and FMP_{als}, capturing the complete edit history of novices writing formal specifications on the FM Playground.

Reconstructing the Edit Path

Every time a user runs the specification, we capture a complete snapshot of their document and link it to the previous version, along with the timestamp. This creates an **edit path**–a fine-grained, step-by-step record of the specification's evolution.
This structure allows us to precisely analyze:

- User workflow and session length.
- The introduction and correction of errors.
- The distance between edits.

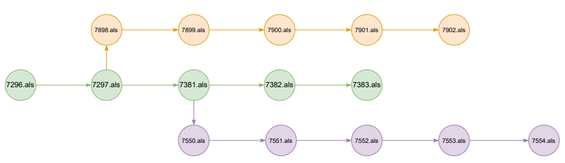


Figure 2. An Example Edit Path from the FMP_{als} Dataset.

The key statistics of both datasets are summarized in Tab. 1.

Feature	SMT-LIB (FMP_{smt})	Alloy (FMP_{als})
Scale and Scope		
Total Specifications	**18,133**	**8,219** models
Unique Edit Paths	2,415	747
User Engagement		
Median Edit Path Length	6 edits	8 edits
Maximum Edit Path Length	321 edits	211 edits
Code Quality and Errors		
Syntactically Unique Specs	9,150 (50.5%)	3,513 (42.7%)
Syntactically Correct (of Unique)	5,971 (65.3%)	1,880 (53.5%)
Edit Paths Containing Errors	59.1%	54.1%

Table 1. Comparative Statistics of the FMP_{smt} and FMP_{als} Datasets.

Research Questions

- **How do specifications evolve over time?**
- **What are the most common syntactic errors users make when writing specifications?**
- **How quickly do users identify and fix errors in their specifications?**
- **How do consecutive edits relate to each other?**

Analysis and Key Findings

Finding 1: Novices struggle with language fundamentals, but in different places.

- **In SMT-LIB**, errors are highly concentrated. The single most common mistake is referencing an **undeclared constant (>50% of errors)**, a fundamental scope issue.
- **In Alloy**, errors are distributed across the entire model structure. Novices find writing **facts (31.6%)** and **predicates (25.7%)** just as challenging as defining **signatures (15.5%)**.

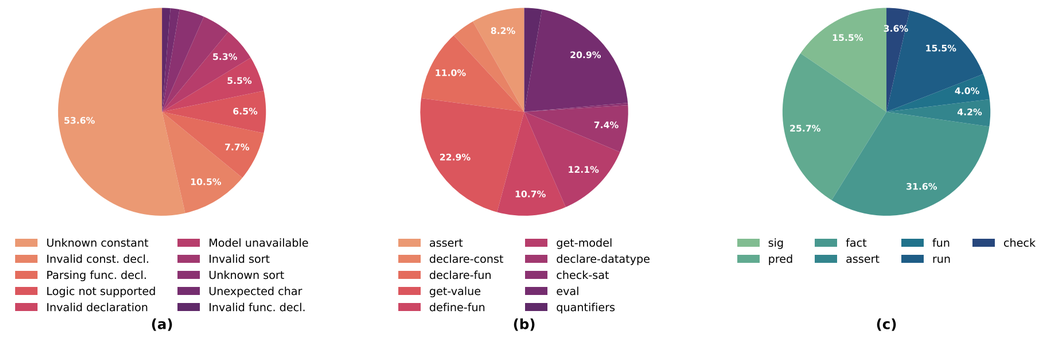


Figure 3. Common Syntactic Errors (a) and Error-Prone Constructs in SMT-LIB (b) and Alloy (c).

Finding 2: The workflow is rapid and incremental, but edit sizes differ.

- Users employ a "trial-and-error" approach. SMT-LIB edits tend to be larger rewrites (**median 51 chars**), while Alloy edits are smaller, more frequent tweaks (**median 25 chars**).
- Recovery is fast. In **both** datasets, syntax errors are typically corrected in a **single edit step**, indicating users quickly identify and fix simple mistakes.

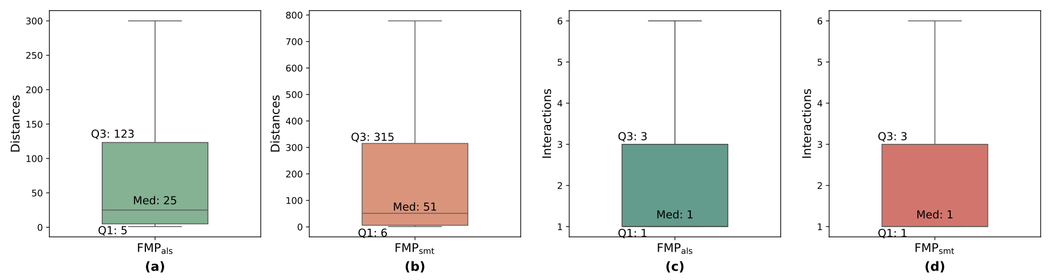


Figure 4. Edit Distance Between Revisions (a, b) and Steps to Fix Errors (c, d).

Finding 3 (SMT-LIB): Edits represent a logical "trial-and-error" process.

- Semantic analysis shows that consecutive scripts are often logically related.
- 23.9% of edits are **semantically equivalent**, suggesting users re-run analyses.
- Another **22%** represents **refinement**, where users incrementally strengthen or weaken their logic, confirming an iterative development process.

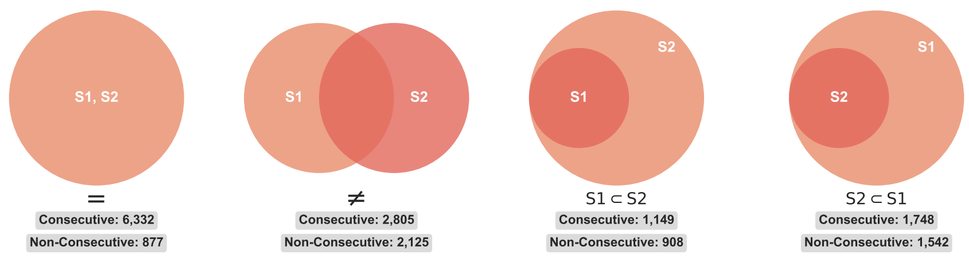


Figure 5. Semantic Relationship Between Consecutive and Non-Consecutive SMT-LIB Scripts.

Implications & Key Takeaways

- **Error-Prone:** Writing specifications is error-prone for novices. Tools should provide better, more targeted support for common mistakes.
- **Incremental Development:** Edits are mostly small and incremental– suitable for interactive feedback and live analysis.
- **Tool Support:** IDEs should provide targeted, context-aware assistance. Providing real-time scope & reference checking and better error messages.

Conclusion & Future Work

- Our analyses reveal common challenges in formal specification engineering:
  - Specifications **evolve incrementally** with frequent small edits.
  - Syntax and typing errors dominate, yet are usually resolved within a few steps.
- These findings highlight the **importance of tool support**, feedback, and teaching strategies that account for the trial-and-error specification writing approach.

Future Work:

- Identifying **code smells** in Alloy models and SMT-LIB scripts to study recurring bad practices.
- Extending analyses to other specification languages (e.g., nuXmv, Spectra).
- Leveraging **robust semantic comparisons** and automated feedback to better support users.

NFDIxCS Bauhaus-Universität Weimar Software Engineering

Scan Me