# On Writing Alloy Models: Metrics and a new Dataset

Soaibuzzaman[0000−0002−8971−5904], Salar Kalantari, and Jan Oliver Ringert[0000−0002−3610−3920]

Bauhaus-University Weimar, Germany

**Abstract.** Alloy is a modeling language that combines relational first-order logic and temporal logic while providing powerful automated analyses via the Alloy Analyzer. Recent efforts in tool development and teaching of Alloy have contributed the Alloy4Fun dataset enabling many analyses of fine-grained model editing histories.
We present a smaller, but complementary dataset $\mathsf{FMP}_{\mathsf{als}}$ of similar editing granularity. While the Alloy4Fun dataset captures users filling in predefined predicates, our dataset is more diverse and users develop all parts of Alloy models including signatures, fields, facts, and commands. We illustrate the differences between the datasets, define a Halstead metric to measure the difficulty of models, and evaluate model edit paths from both datasets on various metrics.

**Keywords:** Alloy · evolution · metrics · dataset

## 1 Introduction

Alloy [13] is a modeling language that combines relational first-order logic and temporal logic while providing powerful automated analyses via the Alloy Analyzer. Alloy has been applied to modeling software designs [23,40], code testing, debugging, and repair [9,28,38], and to analyze security properties [1,37].

Recent efforts in tool development and teaching of Alloy have contributed the Alloy4Fun platform [18] and dataset [17]. Alloy4Fun provides a web-based editor with selected Alloy models and tasks inside these models. The platform generates feedback in terms of Alloy instances for each user attempt. The Alloy4Fun dataset [18,17] has attracted research interest [41,2,14] as it provides fine-grained model editing histories previously not available.

Alloy4Fun [18] focuses on writing expressions inside predefined predicates that are then semantically evaluated against an instructor's solution. This limits the insight one could obtain about how novice users use Alloy, as the Alloy language also provides elements like signatures, fields, and commands (all briefly introduced in Sect. 2.1), which instructors provide and are not expected to be written or modified by users in the Alloy4Fun dataset. We present a smaller but complementary dataset with similar editing granularity. Our $\mathsf{FMP}_{\mathsf{als}}$ dataset is more diverse, and users develop all parts of Alloy models, including signatures, fields, facts, and commands.

We illustrate the differences between the datasets, define a Halstead metric to measure the difficulty of models, and evaluate model edit paths from both datasets on various metrics.

The remainder of this work is structured as follows. Section 2 briefly presents the foundations of our work. Section 3 lists our research questions, Sect. 4 presents our dataset and data processing. Section 5 presents the evaluation of our research questions on the datasets. We discuss related work in Sect. 6 and conclude in Sect. 7.

## 2 Preliminaries

We now give a brief overview of the Alloy language, the Alloy4Fun platform, the Formal Methods Playground, and Halstead metrics.

### 2.1 Alloy

Alloy [12,13] is a textual modeling language based on relational first-order logic. An example Alloy model is shown in Lst. 1.1 consisting of signature declarations (ll. 1-3) with fields, e.g., field `link` in signature `File` (l. 1). Intuitively, the semantics of an Alloy model are instances consisting of atoms and relations over atoms where each signature is a set (unary relation) of atoms, and each field is an $n$-ary relation, e.g., the field `link` defines a binary relation that relates each `File`-atom with an arbitrary number of `File`-atoms (multiplicity `set` in l. 1). Facts, predicates, and assertions may contain expressions in relational as well as temporal logic, e.g., the expression `no Trash` in predicate `inv1` (l. 5) states that the set `Trash` is empty in all instances satisfying predicate `inv1`. Alloy models can be automatically analyzed [13] by the Alloy Analyzer in a bounded scope (bounding number of atoms) via a reduction to SAT [36].

```
1 sig File { link : set File }
2 sig Trash in File {}
3 sig Protected in File {}
4
5 pred inv1 { /* The trash is empty. */ /* solution: */ no Trash }
6 pred inv2 { /* All files are deleted. */ }
7 pred inv3 { /* Some file is deleted. */ }
```

Listing 1.1: An example Alloy model from [17] with three out of 10 predicates for the user to complete, e.g., as attempted in predicate `inv1`.

### 2.2 Alloy4Fun

Alloy4Fun [18] is a web application for writing and analyzing Alloy models intended for teaching Alloy. Alloy4Fun offers automated assessment and feedback by requiring users to fill in predefined predicates (see the predicates in Lst. 1.1, ll. 5-7). Each predicate is independent of the others to avoid "distracting problems corresponding to failures of other properties" [18].

Interactions with Alloy4Fun are captured and published in the Alloy4Fun dataset [17] collected mainly from master students' submissions at the University of Minho and the University of Porto between Fall 2019 to Spring 2023.

### 2.3 Formal Methods Playground

The Formal Methods Playground[1] is a web application for writing and analyzing models in various modeling and specification languages. We have developed this application mainly for teaching, e.g., slides in our Formal Methods for Software Engineering lecture [33] contain permalinks to example models on the Formal Methods Playground for direct analysis in the browser. Currently, the Formal Methods Playground supports Limboole[2], Z3 [21], Alloy [13], nuXmv [3] and Spectra [20] specifications.

### 2.4 Halstead Metrics (also *Theory of Software Science*)

Halstead [11] introduced various measures for software, e.g., the effort related to the time required to write the program or the difficulty $D$ of understanding a program when reading or writing it. Halstead metrics are computed based on the numbers of unique operators $\eta_1$ and operands $\eta_2$, and the total numbers of occurrences of operators $N_1$ and operands $N_2$. Halstead difficulty is defined as

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}, \text{ i.e., } \frac{\#\text{ unique operators}}{2} \times \frac{\#\text{ occurrences of operands}}{\#\text{ unique operands}} \quad (1)$$

Shen et al. [30] have summarized early critical assessment of Halstead metrics as well as some "tentative support" [30] from empirical studies. Shepperd [31] criticizes three representative metrics (including Halstead's) based on their definition and general (ab-)use. We reflect on this criticism in Sect. 4.4 and Sect. 5.6.

## 3 Research Questions

We aim to understand better the differences between the existing Alloy4Fun dataset and our new dataset and how Alloy models evolve in these.

We define the following research questions:

- **RQ1**: In what characteristics do the two datasets differ?
- **RQ2**: What is the Halstead difficulty for typical Alloy Models?
- **RQ3**: How does Halstead difficulty evolve in Alloy modeling tasks?
- **RQ4**: Is Halstead difficulty related to making and fixing errors?
- **RQ5**: How large are editing steps in Alloy modeling tasks?

[1] See https://play.formal-methods.net and https://www.youtube.com/playlist?list=PLGyeoukah9NYq9ULsIuADG2r2QjX530nf

[2] See https://fmv.jku.at/limboole/

# 4 Data Processing and Metrics Computation

## 4.1 Experimental Data

In this study, we utilize two datasets: the publicly accessible Alloy4Fun (A4F) dataset [17] and our new Formal Methods Playground Alloy ($\mathsf{FMP}_{\mathsf{als}}$) dataset [34].

Our $\mathsf{FMP}_{\mathsf{als}}$ dataset contains Alloy models executed on the Formal Methods Playground from November 2023 to January 2025. Unlike with Alloy4Fun, users usually initiate their work with a blank canvas rather than a starter model, and there are no fixed predicates to encode. Students at Bauhaus-University Weimar use this platform as part of our Formal Methods for Software Engineering [33] module. Based on user activity and the models authored, at least one additional university uses the platform. We capture models with their analyses, timestamps, and historical derivations structured in a parent-child relationship.

The A4F dataset includes a total of 97,755 models. However, 1,358 of these models only serve as starting points for users and do not include user edits or lack an executed command (cmd_i). The remaining A4F dataset consists of 96,397 Alloy models. In contrast, the $\mathsf{FMP}_{\mathsf{als}}$ dataset contains 8,219 Alloy models.

## 4.2 Edit Paths

Both the A4F and $\mathsf{FMP}_{\mathsf{als}}$ datasets maintain records of the previous revision of each Alloy model. Each revision is a user submission [17], i.e., the current model whenever the user executes an analysis. We utilize this information to reconstruct the *edit path*[3], allowing us to capture the sequences of edits/submissions made by users (these edits are typically small, see Sect. 5.5).

The A4F dataset comprises a total of 5,268 unique edit paths, whereas the $\mathsf{FMP}_{\mathsf{als}}$ dataset consists of 747 unique edit paths. In particular, the top 25% of the edit paths have a length greater than 28 for A4F and 22 for $\mathsf{FMP}_{\mathsf{als}}$, with median lengths of 11 and 8, respectively. The edit paths in the A4F dataset are all derived from 19 distinct models[4] that each define multiple tasks. In contrast, the $\mathsf{FMP}_{\mathsf{als}}$ dataset has 392 unique initial models[5] within the edit paths.

## 4.3 Alloy4Fun Edit Paths Partitioning (from A4F to A4FpT)

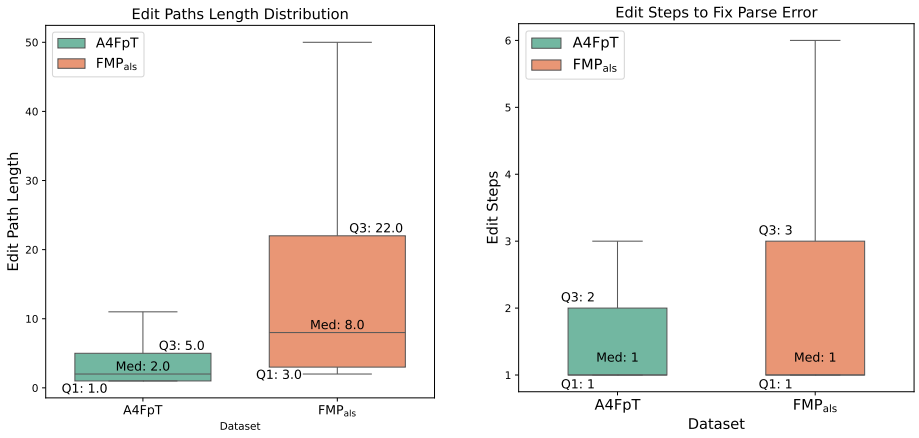
The Alloy4Fun platform offers a variety of starter models, each defined by unique signatures and empty predicates that describe distinct tasks. Users may solve these tasks across multiple edits in any order as the tasks are independent [18] of each other. Thus, analyzing entire edit paths might lead to wrong conclusions when trying to understand how individual tasks are solved. We, therefore,

[3] We adopt the terminology of [14] although *interaction paths* might be more fitting as shown in Table. 3.

[4] "original: the first ancestor with secrets (always the same within an exercise)"[17]

[5] While most edit paths start from scratch, some edit paths share initial models provided by instructors [33].

partitioned the original edit paths of the A4F dataset. In addition to the executed commands, Alloy4Fun provides information about the predicates that users evaluated. We utilized this information to develop new edit paths that capture users' efforts per task and refer to this dataset as A4FpT (Alloy4Fun per Task). Technically, we remove the task predicates for unrelated tasks from these models. They all remain stand-alone Alloy models with the common signatures and facts defined in their 19 starter models. For more information, readers can refer to our replication package [32].




(a) (b)

Fig. 1: (a) Distribution of the edit paths length and (b) Edit steps required to fix errors in edit paths

Fig. 1a illustrates the distribution of edit paths of the A4FpT dataset. The partitioning process resulted in a total of 24,592 edit paths. The box plot reveals that 75% of the edit paths have lengths of $\leq 5$ for A4FpT and $\leq 22$ for $\mathsf{FMP}_{\mathsf{als}}$, with median lengths of 2 and 8, respectively (outliers excluded from box plot).

### 4.4 A Halstead Metric for Alloy

The Halstead metrics, while offering an intuitive static analysis framework, have received various criticism [30,31] on practical challenges associated with the methodology of counting operators and operands. Halstead [11] does not provide explicit definitions for these terms but instead characterizes their meanings as "intuitively obvious". Paige [24] defines operators as "all language elements which must be used to allow the operands to be operated on". As suggested by Salt [27] we document our counting strategy by definitions of operators and operands and provide an implementation in [32].

| Operator | Frequency | Operand | Frequency |
|---|---|---|---|
| sig | 3 | inv1 | 1 |
| no | 1 | inv2 | 1 |
| set | 1 | inv3 | 1 |
| in | 2 | Protected | 1 |
| pred | 3 | File | 4 |
| | | link | 1 |
| | | Trash | 2 |

Table 1: Counting strategy of operators and operands of a model from Lst. 1.1

**Counting Strategy** Our counting strategy roughly follows the Alloy grammar [12, App. B] extended with temporal operators added in Alloy 6 as described in the Alloy Language Reference[6].

1. Only parts of the current model are considered; the contents of imported (operator `open`) models, and comments are ignored.
2. Operators are:
   - Keywords: `abstract`, `extends`, `var`, `enum`, `steps`, `sig`, `fun`, `pred`, `assert`, `check`, `run`, `but`, `else`, `module`, `open`, `disj`, `as`, `let`, `for`, `fact`, `exactly`
   - Multiplicity and quantifiers: `lone`, `some`, `one`, `all`, `sum`, `no`
   - Unary operators: `!`, `not`, `no`, `set`, `#`, `~`, `*`, `^`, `always`, `eventually`, `after`, `before`, `historically`, `once`, `'`
   - Binary operators: $\vee$, `or`, $\wedge$, `and`, $\iff$, `iff`, $\implies$, `implies`, `&`, `+`, `-`, `++`, `<:`, `:>`, `.`, `until`, `releases`, `since`, `triggered`, `;`, `in`, $=$, `<`, `>`, $\leq$, $\geq$, `->`, `[]` (box join)
3. Operands are:
   - literals of `Int` and `String`; and constants: `none`, `univ`, `iden`
   - names of modules, signatures, fields, variables, predicates, functions, and asserts
   - names and types of parameters and types of predicates and functions
4. Overloaded elements (fields, predicates, or functions) are counted once.
5. Parentheses and curly brackets are neither operators nor operands.

Table 1 illustrates an example of counting operators and operands for the Alloy model presented in Lst. 1.1, following our established counting strategy. The unique counts of operators and operands are 5 ($\eta_1$) and 7 ($\eta_2$), respectively, while the total occurrences of operators and operands are 10 ($N_1$) and 11 ($N_2$), respectively. Utilizing Eq. 1, we can calculate the Halstead difficulty as

$$D = \frac{5}{2} \times \frac{11}{6} = 4.58$$

An implementation of this counting strategy is available from [32].

[6] https://alloytools.org/spec.html

## 5 Evaluation

We now present data to answer the research questions defined in Sect. 3.

### 5.1 RQ1: Dataset Characteristics

We evaluate characteristics of the A4F and the $\mathsf{FMP}_{\mathsf{als}}$ datasets along (slightly modified) research questions from [14], a recent, thorough analysis of the A4F dataset. We had to slightly modify the research questions of [14] as detailed below and omit replication of RQs 4, 6, and 7 due to the absence of an oracle/fixed task for models from $\mathsf{FMP}_{\mathsf{als}}$, which would allow for deciding correct, under-, or over-specified attempts, in this more open dataset.

*Errors Users Make* We build on the research questions from [14], which examine the classification of correct and incorrect user submissions ([14], RQ1) and mistakes in writing formulas ([14], RQ5). We modify and extend these to identify top-level language constructs where errors are made.

Approximately two-thirds of the models are syntactically correct, at 70.9% for A4F and 66.3% for the $\mathsf{FMP}_{\mathsf{als}}$ dataset. Conversely, around one-third are syntactically incorrect, with 29.1% for A4F and 33.7% for the $\mathsf{FMP}_{\mathsf{als}}$ dataset.

The syntactically incorrect models include both syntax and type errors. As shown in Table 2, these error types are evenly distributed in the A4F dataset. In contrast, the $\mathsf{FMP}_{\mathsf{als}}$ dataset has a significant prevalence of syntax errors at 77.6%, with type errors making up just 22.4%.

Finally, Table 2 indicates top-level language constructs where users face the greatest challenges. In A4F, nearly all errors are found within predicates, which is expected since users must only complete these. Conversely, the $\mathsf{FMP}_{\mathsf{als}}$ dataset indicates that users similarly struggle with writing predicates and facts (25.7% and 31.6% of errors), but also with signatures (15.5%) and commands (15.5% + 3.6%). This shows the importance of additional datasets like ours as users also make errors in parts not assessed by the A4F dataset.

| Dataset | | Type | Syntax | sig | pred | fact | assert | fun | run | check |
|---|---|---|---|---|---|---|---|---|---|---|
| A4F | # | 13 657 | 13 734 | 72 | 27 202 | 26 | 1 | 52 | 22 | 11 |
| | % | 49.9 | 50.1 | 0.002 | 99.3 | $\approx 0.0$ | $\approx 0.0$ | 0.001 | $\approx 0.0$ | $\approx 0.0$ |
| $\mathsf{FMP}_{\mathsf{als}}$ | # | 566 | 1 962 | 376 | 625 | 769 | 101 | 97 | 378 | 87 |
| | % | 22.4 | 77.6 | 15.5 | 25.7 | 31.6 | 4.2 | 4.0 | 15.5 | 3.6 |

Table 2: Error category and location of the errors for A4F and $\mathsf{FMP}_{\mathsf{als}}$ dataset

*Submission Similarity* RQ2 from [14] examines the prevalence of syntactically and semantically unique submissions. We focus on syntactic similarity as a semantic comparison is easy on the predicate-level (sufficient for A4F), but more complex on the model-level [26] (as it would have been required for $\mathsf{FMP}_{\mathsf{als}}$).

| | A4FpT | | $\mathsf{FMP}_{\mathsf{als}}$ | |
|---|---|---|---|---|
| | # | % | # | % |
| Syntactically Unique Models | 57 777 | 59.9 | 3 513 | 42.7 |
| Syntactically Correct Models (in unique models) | 37 024 | 64.1 | 1 880 | 53.5 |
| Syntax Error (in unique models) | 20 753 | 35.9 | 1 633 | 46.5 |
| Models within single edit paths: | | | | |
| Consecutive Identical Models | 4 664 | 4.64 | 3 174 | 25.58 |
| Non-Consecutive Identical Models | 5 758 | 5.73 | 667 | 5.38 |

Table 3: Syntactically unique models in the A4FpT and $\mathsf{FMP}_{\mathsf{als}}$ datasets

Table 3 demonstrates that many user submissions comprise syntactically unique models. Specifically, the A4F dataset reveals that 59.9% of models maintain syntactic uniqueness, in contrast to 42.7% observed within the $\mathsf{FMP}_{\mathsf{als}}$ dataset. Among these submissions, 64.1% of models in the A4F dataset are syntactically correct, whereas 35.9% are incorrect. Conversely, the $\mathsf{FMP}_{\mathsf{als}}$ dataset indicates a syntactic correctness rate of 53.2% among the unique models, with 46.4% of the models being incorrect. The A4F dataset demonstrates significantly more unique models than the $\mathsf{FMP}_{\mathsf{als}}$ dataset. Further analysis reveals (Table 3, bottom) that for $\mathsf{FMP}_{\mathsf{als}}$ 25.6% of consecutive models in edit paths are identical (only 4.6% in A4FpT). We believe that users repeatedly browse instances and thus analyze the same model again. The Formal Methods Playground, as the official Alloy Analyzer, only allows showing the next instances, whereas Alloy4Fun also allows for navigating previous ones. It might be worthwhile implementing this backward navigation feature in the Formal Methods Playground and the Alloy Analyzer as well. Additional contributors to this difference might be that instances displayed on the Alloy4Fun platform are usually counterexamples that come with semantic classifications. This task-specific information might reduce the amount of instances users choose to inspect.

*Fixing Errors* We adapt RQ3 from [14], which examines invalid submissions and the effectiveness of Alloy's compiler-based error reporting, by focusing specifically on how users fix errors over multiple edit steps.

Approximately one-third of the models are deemed invalid in both the A4FpT and $\mathsf{FMP}_{\mathsf{als}}$ datasets. To gain further insights into how users address these issues, we analyze the presence of errors within the edit paths associated with the models. The results are summarized in Table 4. In the A4FpT dataset, 39.24% of the edit paths contain at least one erroneous model, with 3.80% consisting entirely of erroneous models. In comparison, the $\mathsf{FMP}_{\mathsf{als}}$ dataset reveals that 54.08% of the edit paths include at least one erroneous model, and 6.55% are comprised entirely of erroneous models.

Fig. 1b depicts the edit steps necessary to correct the errors, i.e., the numbers of consecutive, syntactically invalid models until reaching a syntactically correct one. In the A4FpT dataset, users generally require a median of 1 revision to resolve errors, with 75% needing less than or equal to two revisions for complete

| | A4FpT | $\mathsf{FMP}_{\mathsf{als}}$ |
|---|---|---|
| Edit paths (#) | 24 592 | 747 |
| With Invalid Models (%) | 39.24 | 54.08 |
| Without Valid Models (%) | 3.80 | 6.55 |
| Edit Path Length $\geq 5$ (%) | 25.93 | 64.79 |
| Max Edit Path Length | 107 | 211 |

Table 4: Details of syntactically invalid models in edit paths

correction. Likewise, in the $\mathsf{FMP}_{\mathsf{als}}$ dataset, users also show a median of 1 revision; however, 75% of them need less than or equal to 3 revisions to address the errors. This could point to more complex errors experienced by users within the $\mathsf{FMP}_{\mathsf{als}}$ dataset. A distinction between specific error types might be helpful for future analyses.

### 5.2 RQ2: Halstead Metrics for Typical Models

We aim to assess modeling difficulty using Halstead difficulty as defined in Sect. 4.4. As an intuitive baseline, we analyze the sample models provided with the Alloy Analyzer to assess the Halstead difficulty of well-known Alloy models. These ca. 80 models comprise four categories: Algorithm, Book, Case Study, and Temporal. Fig. 2a presents the Halstead difficulty for each category.

The Book category comprises example models from [12]. Most of these models exhibit a difficulty rating ranging from 11.9 to 60.4, with a median difficulty of 27.2. In contrast, the Case Studies category includes Alloy case studies, e.g., analyses of the Firewire [8] and Chord [39] protocols, and exhibits much higher Halstead difficulties between 130.8 and 176.4, with a median value 140.8.

We selected the final submitted Alloy model from each edit path to compare the Halstead difficulty of the A4FpT and $\mathsf{FMP}_{\mathsf{als}}$ datasets with typical Alloy models. Fig. 2b presents a box plot of the Halstead difficulty for these final submissions across both datasets. The results indicate that the A4FpT and $\mathsf{FMP}_{\mathsf{als}}$ datasets exhibit similar Halstead difficulty levels to the Book category of typical Alloy models shown in Fig 2a. This is no surprise, as both platforms are mainly used in teaching contexts. However, the $\mathsf{FMP}_{\mathsf{als}}$ dataset demonstrates greater Halstead difficulty than A4FpT, with median values of 20.3 and 16.3, respectively. Furthermore, the plot suggests that the $\mathsf{FMP}_{\mathsf{als}}$ dataset demonstrates greater variability in Halstead difficulty compared to A4FpT.

### 5.3 RQ3: Evolution of Halstead difficulty

We analyzed both datasets using clustering and standard deviation to examine how Halstead difficulty evolves during Alloy modeling tasks.

We performed KMeans clustering on the Halstead difficulty scores of models in both datasets. The clustering allowed us to group edit paths with similar difficulty levels, revealing how difficulty changes over time across different revisions

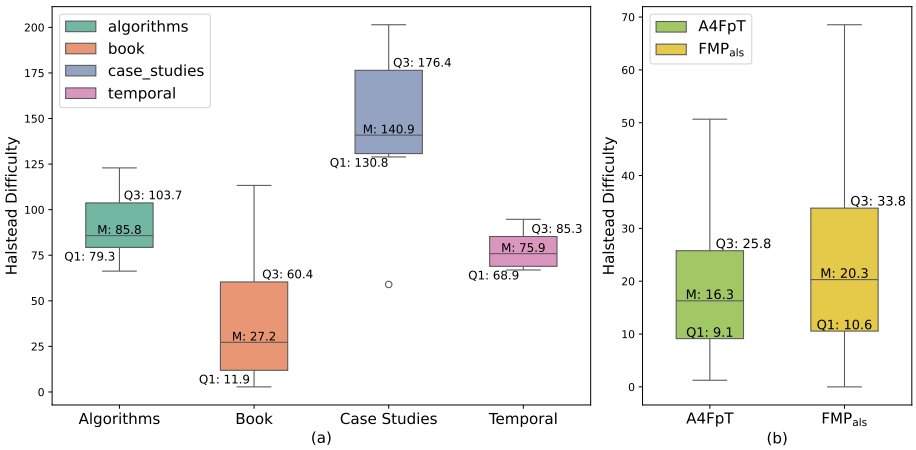



Fig. 2: Halstead difficulty of (a) typical Alloy models from the sample models of Alloy Analyzer and (b) last submitted models of the A4FpT & $\mathsf{FMP}_{\mathsf{als}}$ edit paths

of the models. We identified eight distinct clusters in the A4FpT dataset, while the $\mathsf{FMP}_{\mathsf{als}}$ dataset revealed three clusters. This variation reflects the differing complexity and characteristics of the models within each dataset.

In addition, we incorporated standard deviation into our analysis to assess the consistency of difficulty levels within each cluster. It offers insights into whether the evolution of difficulty is stable or erratic across edit paths within a cluster.

Figure 3 and Fig. 4 demonstrate the evolution of Halstead difficulty for the A4FpT and $\mathsf{FMP}_{\mathsf{als}}$ datasets, respectively. These plots illustrate the mean difficulty scores for each cluster over time, spanning across edit path steps. Shaded regions represent the standard deviation, offering insights into how the difficulty of models within each cluster varies as users refine their Alloy models.

In our analysis of Halstead difficulty within the A4FpT dataset, we found that the standard deviation for seven of the eight clusters was relatively low, ranging from 0 to 20. This indicates that the models within these clusters display consistent difficulty levels throughout their revisions.

Cluster 4 is the largest, comprising 9,675 edit paths, and is characterized by relatively low difficulty levels, with fewer than 20 revision steps required. Other significant clusters with larger model counts include Clusters 1, 3, 5, and 8, each containing thousands of edit paths. These clusters typically involve less than 30 revision steps, suggesting that the tasks in these groups are generally less complex requiring fewer edits.

Cluster 6, which consists of only 24 edit paths, displays significantly higher difficulty levels, ranging from 30 to 50. Further investigation revealed that most of the models in this cluster originate from the Train Station modeling problem from the EM 20/21 dataset. This suggests that this particular task may be inherently more complex than others in the dataset.

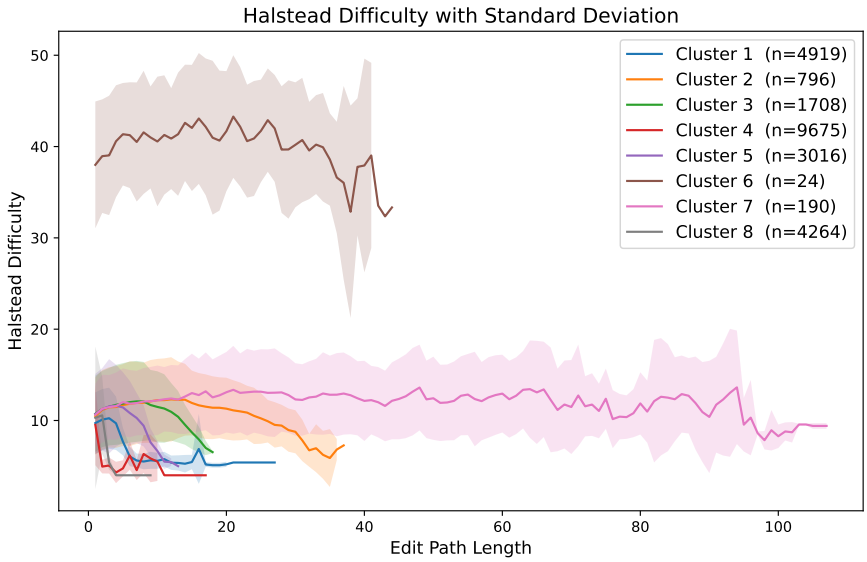



Fig. 3: Clustered Halstead difficulty of A4FpT dataset. The shaded region represents the standard deviation for each cluster

```
always all f:File | (f not in Protected and f not in Trash) implies f in
    ↪ Protected'
always all f:File | always f not in Protected implies f in Protected'
always all f:File | eventually f not in Protected implies f in Protected'
always all f:File | f not in Protected implies after f in Protected
always all f:File | f not in Protected implies f in Protected'
```

Listing 1.2: An example investigation on the decline of difficulty over edit paths

A common observation across nearly all clusters of the A4FpT dataset is a noticeable decrease in difficulty at the end of the editing process, accompanied by a narrow standard deviation. We investigated this trend and discovered that the decline may be linked to users making significant changes, such as removing entire constraints to fix errors in their models. Users often begin with a more complex constraint and gradually simplify it over time. Listing 1.2 provides an example of this revision strategy where each line is a separate edit. Note that Listing 1.2 shows the predicate body only while the Halstead difficulty is always computed for the whole model.

In the $\mathsf{FMP}_{\mathsf{als}}$ dataset, we identified three clusters of Halstead difficulty. The largest cluster, Cluster 1, with 592 edit paths, demonstrates a broad range of Halstead difficulty values, spanning from 0 to 50, which indicates variability in model complexity within this group. Cluster 2 is noteworthy due to its elevated difficulty levels, ranging from 25 to 175. Upon further analysis, we discovered

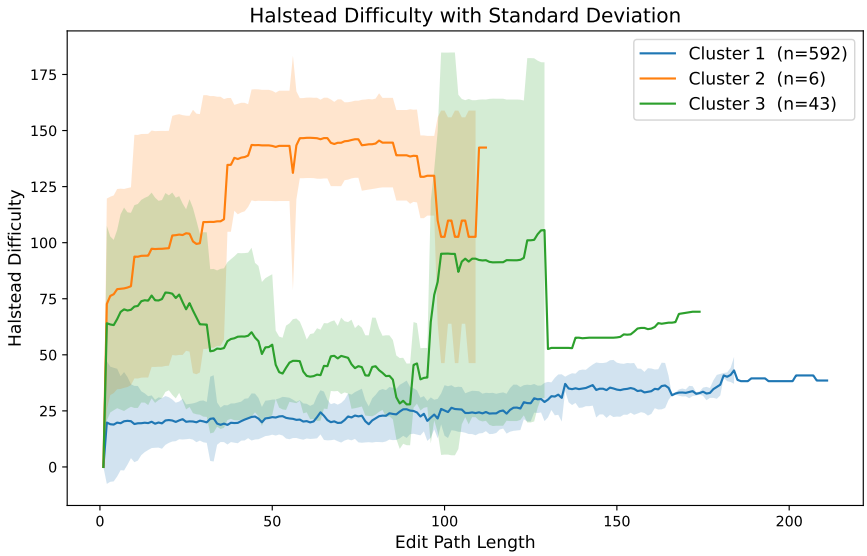



Fig. 4: Clustered Halstead difficulty of $\mathsf{FMP}_{\mathsf{als}}$ dataset. The shaded region represents the standard deviation for each cluster

that most of the models in this cluster were generated automatically[7] rather than being crafted by users, which may account for the higher difficulty levels observed. Cluster 3, consisting of only 43 edit paths, is the most diverse regarding difficulty, reflecting a broad spectrum of complexity. This diversity suggests that the models in this cluster embody a broad array of challenges.

### 5.4 RQ4: Metrics related to Errors/Fixing

*Correlation Between Halstead difficulty and Time to Fix Errors* To investigate whether higher Halstead difficulty is linked to error-fixing times, we calculated the Spearman correlation coefficient between the Halstead difficulty of each first syntactically incorrect model and the total time required to rectify its errors. We focused our analysis on models that underwent at least one revision and excluded cases where the time difference exceeded 600 seconds to concentrate on active revision sessions.

For the A4FpT dataset, the Spearman correlation coefficient was $-0.032$ ($p = 0.006$). This negative correlation indicates a weak association, suggesting that higher Halstead difficulty is correlated with slightly shorter error-fixing times. However, the small effect size and low correlation strength imply that this relationship is likely not practically significant.

Conversely, for the $\mathsf{FMP}_{\mathsf{als}}$ dataset, the Spearman correlation coefficient was 0.236 ($p < 0.0001$), revealing a weak yet statistically significant positive cor-

[7] The models were generated as part of a student project based on [29]

relation between Halstead difficulty and the time taken to correct errors. This suggests that in the $\mathsf{FMP}_{\mathsf{als}}$ dataset, models with higher Halstead difficulty tend to require more time for error correction. Although this correlation is stronger than that observed in the A4FpT dataset, it still remains relatively weak.

These weak correlations indicate that other factors—such as model structure, user expertise, or the nature of the errors—may play a more significant role in determining error resolution time.

*Correlation Between Halstead difficulty and Error Occurrence* To explore the impact of Halstead's Difficulty on the likelihood of errors, we performed a logistic regression analysis on both datasets. In this analysis, the dependent variable indicated whether a model contained an error, with Halstead difficulty as the independent variable.

For the A4FpT dataset, the results revealed a weak negative correlation between Halstead difficulty and error occurrence, with a coefficient of $-0.0167$ ($z = -10.84, p < 0.001$). Although this finding is statistically significant, the effect size is minimal, as demonstrated by the low $pseudo - R^2$ value of 0.0010 and the Point-Biserial correlation of $-0.0339$.

This suggests that models with higher Halstead difficulty are slightly less likely to contain errors. However, this difference is negligible, suggesting that factors beyond difficulty have a more pronounced influence on error occurrence.

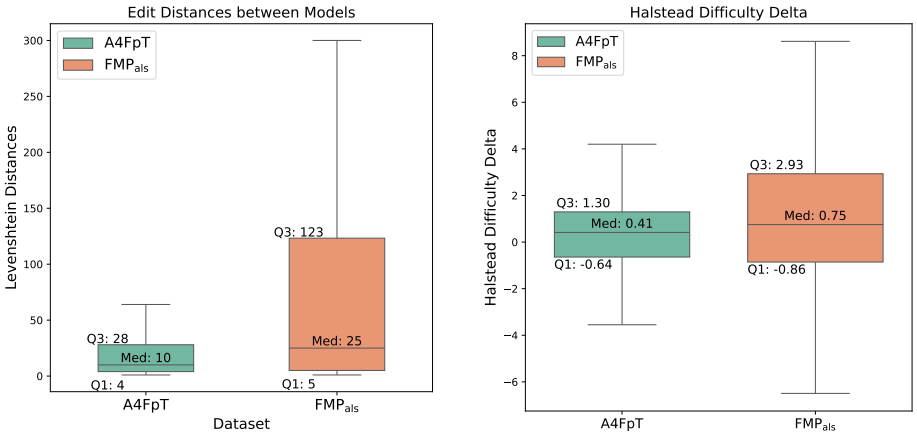
In contrast, the $\mathsf{FMP}_{\mathsf{als}}$ dataset indicates a weak positive correlation between Halstead difficulty and error occurrence, with a coefficient of 0.0034 ($z = 6.28, p < 0.001$). While this relationship is statistically significant, it remains small, as indicated by the $pseudo - R^2$ value of 0.0027 and a Point-Biserial correlation of 0.0587. This suggests that models with higher Halstead difficulty are slightly more likely to experience errors in the $\mathsf{FMP}_{\mathsf{als}}$ dataset.

In summary, while the statistical analysis suggests a modest association between Halstead difficulty and error occurrence, the effect sizes in both datasets are petite.

### 5.5 RQ5: Edit Distance and Difficulty Delta

To better understand how users evolve Alloy models, we have computed Levenshtein distances [15] between consecutive models, i.e., the minimal number of characters modified to transform one into the other. We show box plots of Levenshtein distances between consecutive, non-identical models for both datasets in Fig. 5a. The median and $75^{th}$ percentile of the edit distance in the A4FpT dataset are relatively small, with 10 and 28. They are significantly larger in the $\mathsf{FMP}_{\mathsf{als}}$ dataset with 25 and 123. This is expected for the A4FpT dataset, as most edits are confined to a single line in a predicate. It also shows that users typically analyze their models frequently, not only when solving predefined tasks.

In addition to Levenshtein distances, we have also computed and aggregated changes in Halstead difficulty in Fig. 5b. Note that Fig. 5b aggregates the difference not in absolute terms and thus also shows decreases in Halstead difficulty,




(a) (b)

Fig. 5: (a) Distribution of the edit distance (Levenshtein distance) and (b) Halstead difficulty differences in edit chain

e.g., for both datasets more than 25% of the edit steps decrease the model's Halstead difficulty. This is no surprise as we have seen negative trends in particular for the A4FpT dataset already in Fig. 3 and Fig. 4. Again, the median values and $75^{th}$ percentile show that larger change sizes in the $\mathsf{FMP}_{\mathsf{als}}$ are also reflected in larger differences of Halstead difficulty between edited models.

### 5.6 Threats to Validity

We now identify and discuss various threats to the validity of our analyses.

First, we have processed the A4F dataset from [17] to obtain edits per task as described in Sect. 4.3. To ensure correctness we have performed manual inspection of the resulting edit paths as the excerpt shown in Lst. 1.2. In addition, for transparency and reproducibility we provide the implementation of our processing in [32].

Second, we assume that the use of the Formal Methods Playground across the dataset is similar to that of our students, i.e., small models are created and analyzed. However, we have no control over how other users use the publicly available platform, e.g., it could be used for teaching with very narrow task definitions. To assess this uncertainty, we have assessed unique initial nodes of edit paths in Sect. 4.2, giving some confidence in the models' variability (392 in $\mathsf{FMP}_{\mathsf{als}}$ vs. 19 in A4FpT). In addition, we have manually inspected the edit paths' clusters as described in Sect. 5.3, identifying a small number of (partially) generated models inside the dataset.

Third, existing resources on Halstead metrics do not contain formal definitions of operators and operands and we are not aware of previous applications of the metrics to Alloy. Following best practices suggested by Salt [27] we have

clearly defined our counting strategy in Sect. 4.4 and provide an implementation for reproducibility and inspection at [32]. We cannot rule out that different counting strategies would change the data presented across Sect. 5.

Finally, the use of metrics like the Halstead difficulty has been criticized for being employed as indicators when they show low or no evidence of correlation with various phenomena [30,31]. Our use of the metrics is rather descriptive, and we have carefully analyzed possible correlations with errors and times required to fix these in Sect. 5.4.

## 6 Related Work

In recent years, an increasing amount of work has been invested to understand how novice users use formal methods and, in particular, the Alloy language. Mansoor et al. [19] performed an exploratory study where users fix Alloy models and also create them from scratch. They report that users (both novices and non-novices) find it hard to start Alloy models from scratch [19]. Our new $\mathsf{FMP}_{\mathsf{als}}$ dataset [34] exhibits these challenges.

Many works have analyzed the popular Alloy4Fun dataset [18,17]. Zheng et al. [41] and Cerqueira et al. [4] use it to evaluate their model repair approaches. Our analysis in Sect. 5.1 shows that the $\mathsf{FMP}_{\mathsf{als}}$ dataset is complementary in locations and kinds of issues exposed and might be beneficial for works on model repair. Barros et al. [2] used the Alloy4Fun dataset to generate suggestions for the next edits based on similar models written by other users. This approach relies on fixed tasks and known goals of edits, which are not given for our dataset.

Cunha et al. [6] have assessed what kind of hints best support novice users in fixing faulty Alloy models. Datasets like ours and platforms like Alloy4Fun and the Formal Methods Playground would allow large-scale comparative analyses of the strategies suggested above on diverse models and modeling tasks.

Another line of work has focused on improving instance generation for Alloy models [22,16,35,25] and understanding how different strategies support users in understanding Alloy models [10,5]. These and the ambitious early user study by Danas et al. [7] highlight the difficulty of setting up controlled studies and evaluating data from multiple sources, e.g., interviews and tool instrumentation. While our dataset is much more diverse, we are only able to make general observations due to absent task descriptions and characteristics of individual users.

Closest to our current work is the analysis of the Alloy4Fun dataset by Jovanovic and Sullivan [14]. They extensively analyze user edits syntactically as well as semantically, i.e., whether predicates are over- or under- constrained. We adapt their research questions as described in Sect. 5.1 to compare our new dataset to the one of Alloy4Fun. In addition, our analysis introduces and investigates a Halstead difficulty measure for Alloy models and strongly focuses on edits performed on edit paths.

## 7 Conclusion

We have presented the Formal Methods Playground Alloy ($\mathsf{FMP}_{\mathsf{als}}$) dataset and compared it to the well-known Alloy4Fun (A4F) dataset. Our comparison shows that additional datasets are worthwhile as the A4F dataset is limited to user edits in predicates while our $\mathsf{FMP}_{\mathsf{als}}$ dataset shows challenges in writing other language constructs as well.

Our analysis focused on the evolution of model complexity as characterized by the Halstead difficulty metric we adapted for Alloy models. The $\mathsf{FMP}_{\mathsf{als}}$ dataset exhibits more stable growth of the Halstead difficulty of models in edit paths, while the A4FpT shows interesting dips as edit paths terminate. These trends indicate iteratively growing models in the $\mathsf{FMP}_{\mathsf{als}}$ dataset and debugging behavior in the A4FpT dataset. Interestingly, the Halstead difficulty shows only a (very) weak correlation with error prevalence and fixing times in both datasets, i.e., it does not seem to be suitable indicator for the difficulty of fixing errors in Alloy models.

Finally, an observation on repeated analyses of identical models in the $\mathsf{FMP}_{\mathsf{als}}$ dataset suggests for tool improvements and for further analyses of how users interact with generated instances.

## 8 Data Availability

We have made the Formal Methods Playground Alloy ($\mathsf{FMP}_{\mathsf{als}}$) dataset publicly available on Zenodo as [34]. As the use of the Formal Methods Playground increases, we plan to follow the example of [17] and provide updates to this dataset.

In addition, to support reproducibility of our metrics computations and the preprocessing of the A4FpT dataset, we have made the implementation used for our analyses available in a GitHub repository as [32].

## References

1. Bagheri, H., Kang, E., Malek, S., Jackson, D.: A formal approach for detection of security flaws in the android permission system. Formal Aspects Comput. **30**(5), 525–544 (2018). https://doi.org/10.1007/S00165-017-0445-Z
2. Barros, A., Neto, H., Cunha, A., Macedo, N., Paiva, A.C.R.: Alloy repair hint generation based on historical data. In: FM 2024. LNCS, vol. 14934, pp. 104–121. Springer (2024). https://doi.org/10.1007/978-3-031-71177-0_8
3. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuxmv symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV. LNCS, vol. 8559, pp. 334–342. Springer (2014)
4. Cerqueira, J., Cunha, A., Macedo, N.: Timely specification repair for alloy 6. In: SEFM 2022. LNCS, vol. 13550, pp. 288–303. Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_18

5. Cornejo, C., Novaira, M.M., Permigiani, S., Aguirre, N., Frias, M.F., Brida, S.G., Regis, G.: An analysis of the impact of field-value instance navigation in alloy's model finding. In: ABZ 2024. LNCS, vol. 14759, pp. 141–159. Springer (2024). https://doi.org/10.1007/978-3-031-63790-2_9
6. Cunha, A., Macedo, N., Campos, J.C., Margolis, I., Sousa, E.: Assessing the impact of hints in learning formal specification. In: SEET@ICSE 2024. pp. 151–161. ACM (2024). https://doi.org/10.1145/3639474.3640050
7. Danas, N., Nelson, T., Harrison, L., Krishnamurthi, S., Dougherty, D.J.: User studies of principled model finder output. In: SEFM 2017. LNCS, vol. 10469, pp. 168–184. Springer (2017). https://doi.org/10.1007/978-3-319-66197-1_11
8. Devillers, M., Griffioen, W.O.D., Romijn, J., Vaandrager, F.W.: Verification of a leader election protocol: Formal methods applied to IEEE 1394. Formal Methods Syst. Des. **16**(3), 307–320 (2000). https://doi.org/10.1023/A:1008764923992
9. Dini, N., Yelen, C., Alrmaih, Z., Kulkarni, A., Khurshid, S.: Korat-api: a framework to enhance korat to better support testing and reliability techniques. In: SAC 2018. pp. 1934–1943. ACM (2018). https://doi.org/10.1145/3167132.3167339
10. Dyer, T., Nelson, T., Fisler, K., Krishnamurthi, S.: Applying cognitive principles to model-finding output: the positive value of negative information. Proc. ACM Program. Lang. **6**(OOPSLA1), 1–29 (2022). https://doi.org/10.1145/3527323
11. Halstead, M.H.: Elements of Software Science. Operating and Programming Systems, Elsevier Science Inc. (1977)
12. Jackson, D.: Software Abstractions - Logic, Language, and Analysis. MIT Press (2006), http://mitpress.mit.edu/catalog/item/default.asp?ttype=2&tid=10928
13. Jackson, D.: Alloy: a language and tool for exploring software designs. Commun. ACM **62**(9), 66–76 (2019). https://doi.org/10.1145/3338843
14. Jovanovic, A., Sullivan, A.: Right or wrong - understanding how users write software models in alloy. In: SEFM 2024. LNCS, vol. 15280, pp. 309–327. Springer (2024). https://doi.org/10.1007/978-3-031-77382-2_18
15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Proceedings of the Soviet physics doklady (1966)
16. Macedo, N., Cunha, A., Guimarães, T.: Exploring scenario exploration. In: FASE 2015. LNCS, vol. 9033, pp. 301–315. Springer (2015). https://doi.org/10.1007/978-3-662-46675-9_20
17. Macedo, N., Cunha, A., Paiva, A.C.R.: Alloy4fun dataset for 2022/23 (Jul 2023). https://doi.org/10.5281/zenodo.8123547
18. Macedo, N., Cunha, A., Pereira, J., Carvalho, R., Silva, R., Paiva, A.C.R., Ramalho, M.S., Silva, D.C.: Experiences on teaching alloy with an automated assessment platform. Sci. Comput. Program. **211**, 102690 (2021). https://doi.org/10.1016/J.SCICO.2021.102690
19. Mansoor, N., Bagheri, H., Kang, E., Sharif, B.: An empirical study assessing software modeling in alloy. In: FormaliSE 2023. pp. 44–54. IEEE (2023). https://doi.org/10.1109/FORMALISE58978.2023.00013
20. Maoz, S., Ringert, J.O.: Spectra: a specification language for reactive systems. Softw. Syst. Model. **20**(5), 1553–1586 (2021). https://doi.org/10.1007/S10270-021-00868-Z
21. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24
22. Nelson, T., Saghafi, S., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: Aluminum: principled scenario exploration through minimality. In: ICSE 2013. pp. 232–241. IEEE Computer Society (2013). https://doi.org/10.1109/ICSE.2013.6606569

23. Nelson, T., Barratt, C., Dougherty, D.J., Fisler, K., Krishnamurthi, S.: The margrave tool for firewall analysis. In: LISA 2010. USENIX Association (2010), https://www.usenix.org/conference/lisa10/margrave-tool-firewall-analysis
24. Paige, M.: A metric for software test planning. In: Conference Proceedings of COMPSAC. vol. 80, pp. 499–504 (1980)
25. Ringert, J.O., Sullivan, A.: Abstract alloy instances. In: FM 2023. LNCS, vol. 14000, pp. 364–382. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_21
26. Ringert, J.O., Wali, S.W.: Semantic comparisons of alloy models. In: MoDELS 2020. pp. 165–174. ACM (2020). https://doi.org/10.1145/3365438.3410955
27. Salt, N.F.: Defining software science counting strategies. ACM SIGPLAN Notices **17**(3), 58–67 (1982). https://doi.org/10.1145/947912.947916
28. Samimi, H., Aung, E.D., Millstein, T.D.: Falling back on executable specifications. In: ECOOP 2010. LNCS, vol. 6183, pp. 552–576. Springer (2010). https://doi.org/10.1007/978-3-642-14107-2_26
29. Schnabel, T., Weckesser, M., Kluge, R., Lochau, M., Schürr, A.: Cardygan: Tool support for cardinality-based feature models. In: VaMoS 2016. pp. 33–40. ACM (2016). https://doi.org/10.1145/2866614.2866619
30. Shen, V.Y., Conte, S.D., Dunsmore, H.E.: Software science revisited: A critical analysis of the theory and its empirical support. IEEE Trans. Software Eng. **9**(2), 155–165 (1983). https://doi.org/10.1109/TSE.1983.236460
31. Shepperd, M.J., Ince, D.C.: A critique of three metrics. J. Syst. Softw. **26**(3), 197–210 (1994). https://doi.org/10.1016/0164-1212(94)90011-6
32. Soaibuzzaman, Kalantari, S., Ringert, J.O.: Alloy metrics replication package (2025), available from https://github.com/se-buw/alloy-metrics
33. Soaibuzzaman, Ringert, J.O.: Introducing github classroom into a formal methods module. In: FMTea 2024. LNCS, vol. 14939, pp. 25–42. Springer (2024). https://doi.org/10.1007/978-3-031-71379-8_2
34. Soaibuzzaman, Ringert, J.O.: Formal methods playground alloy dataset (Feb 2025). https://doi.org/10.5281/zenodo.14865553
35. Sullivan, A.: Hawkeye: User-guided enumeration of scenarios. In: ISSRE 2021. pp. 569–578. IEEE (2021). https://doi.org/10.1109/ISSRE52982.2021.00064
36. Torlak, E., Jackson, D.: Kodkod: A relational model finder. In: TACAS 2007. LNCS, vol. 4424, pp. 632–647. Springer (2007). https://doi.org/10.1007/978-3-540-71209-1_49
37. Trippel, C., Lustig, D., Martonosi, M.: Security verification via automatic hardware-aware exploit synthesis: The checkmate approach. IEEE Micro **39**(3), 84–93 (2019). https://doi.org/10.1109/MM.2019.2910010
38. Zaeem, R.N., Khurshid, S.: Contract-based data structure repair using alloy. In: ECOOP 2010. LNCS, vol. 6183, pp. 577–598. Springer (2010). https://doi.org/10.1007/978-3-642-14107-2_27
39. Zave, P.: Using lightweight modeling to understand chord. Comput. Commun. Rev. **42**(2), 49–57 (2012). https://doi.org/10.1145/2185376.2185383, https://doi.org/10.1145/2185376.2185383
40. Zave, P.: Reasoning about identifier spaces: How to make chord correct. IEEE Trans. Software Eng. **43**(12), 1144–1156 (2017). https://doi.org/10.1109/TSE.2017.2655056
41. Zheng, G., Nguyen, T., Brida, S.G., Regis, G., Aguirre, N., Frias, M.F., Bagheri, H.: ATR: template-based repair for alloy specifications. In: ISSTA 2022. pp. 666–677. ACM (2022). https://doi.org/10.1145/3533767.3534369