Chair of Software Engineering II

Scratch Bug Detection Using the N-gram Language Model

Exposé by

Eva Gründinger

ADVISOR
Prof. Dr. Gordon Fraser

March 10, 2022

Contents

1 Problem

In order to improve code quality and reliability of programs, many rule-based techniques have been researched to detect bugs and code smells. However, these rule-based approaches are relying on highly frequent patterns in the code.

In this bachelor thesis a different technique that uses N-GRAM LANGUAGE MODELs to automatically detect bugs in SCRATCH programs is proposed. This way of approaching the detection of defective code is already successfully applied by BUGRAM [Wan+16] on JAVA code. Token sequences of programs are assessed by their probability in the learned model while low probability ones are marked as bugs. The assumption is that low probability token sequences are unusual, which may indicate bugs, bad practices or special uses of code.

For bug detection the four main components that need to be configured for the model are: *Gram Size, Sequence Length, Reporting Size and Minimum Token Occurrence.* After adjusting these settings, the n-gram Markov model is able to obtain the probabilities of all token sequences. Token sequences in SCRATCH consist of blocks that can be arranged by the user with drag-and-drop. The probability of each block in a sequence is only determined by its previous n-1 tokens. Using a 3-gram model the probability of the sequence s is: $P(s) = P(b_1)P(b_2 \mid b_1)P(b_3 \mid b_1b_2)P(b_4 \mid b_2b_3)$. Then the language model ranks the outcome based on the probabilities in descending order and reports the sequences with the lowest probabilities as potential bugs.

2 State of the art

2.1 Analyzing Scratch programs

The tools Dr. Scratch [MRR15] as well as Hairball [Boe+13] analyze Scratch programs to find bugs and code smells. These are then reported to the user in order to improve the computational thinking and coding skills of novice programmers. Bad programming habits were assessed in a preliminary study by Moreno et al. [MR14] and code smells that are very common in Scratch were analyzed by Vargas-Alba et al. [Var+19]. Stahlbauer et al. [SKF19] introduced Whisker which is a formal testing framework for Scratch. LitterBox, a tool created by Frädrich et al. [Frä+20] that creates an AST of Scratch programs, is used for finding code smells as well as bug patterns. In this bachelor thesis, the goal is to enhance LitterBox by finding bugs with a n-gram language model instead of bug patterns and rules like all tools mentioned above are already able to do.

2.2 N-gram language models

Hindle et al. [Hin+12] first introduced the n-gram language model to show that software source code is highly repetitive and the n-gram language model can be used in code suggestion and completion. This work is the basis for using language models to model source code and demonstrated how they could be used in software tools. A very accurate algorithm by using a Hidden Markov Model for code completion was proposed by Han et al. [HWM09]. SLAMC by Nguyen et al. [Ngu+13], which incorporated semantic information into a n-gram model, presented a method to code suggestion. It demonstrated how tokens can be seen more semantically instead of just syntactically. Raychev et al. [RVY14] investigated the effectiveness of various language models for code completion, i.e., n-gram and recurrent neural networks. By combining program analysis and the n-gram model, they proposed SLANG, which had the goal to predict the sequence of method calls in a software system. This work leverages the n-gram language model on a new domain - software defect detection in Scratch.

2.3 Automatic bug detection

Wang et al. [Wan+16] demonstrated with their tool Bugram that n-gram language models can be used to find defective code in Java programs. Although there are other studies that covered the usage of n-grams for detecting clone bugs [HCN14], localizing faults [Nes+08] and code search [KMA13], these did not leverage n-gram models. In contrast to n-grams that are only token sequences and were primarily used in the papers mentioned above, n-gram models are Markov models built on n-grams. The bachelor thesis will try to adapt this mechanism for Scratch projects.

3 Research questions

The thesis aims to answer the following research questions:

RQ 1 What is the optimal gram size of the language model?

RQ 2 How long should the sequences be?

RQ 3 How many bugs are found in comparison to LitterBox?

RQ 4 What kind of violations are found?

4 Evaluation

Implement a visitor for n-gram language models in LitterBox. Use a big set of Scratch3 projects to study the impact of different parameters on the performance.

RQ 1 What is the optimal gram size of the language model? N-gram models for each project are built with the gram size ranging from two to ten. Calculate the probabilities of all token sequences and rank them based on their probabilities in descending order. Then examine the bottom 10 sequences of each n-gram model and verify whether a token sequence contains a bug or not. Choose gram size with most found bugs.

RQ 2 How long should the sequences be? Use sequence lengths ranging from two to ten. For each built a n-gram model with the optimal gram size from [RQ 1]. Calculate probabilities from all sequences and rank them. Examine the bottom 10 sequences with low probabilities to check how many true bugs are detected. Choose sequence length with most found bugs.

Take another set of Scratch3 projects from our database that can be manually verified.

RQ 3 How many bugs are found in comparison to LitterBox? Use LitterBox to find bugs in the set and count how many of these bugs the n-gram language model can detect as well.

RQ 4 What kind of violations are found? Analyze if n-gram language model found bugs that are not detected by LitterBox. Classify the defective code into these categories: *True Bugs, Refactoring Opportunities or False Positives.*

5 Schedule

Table 5.1: Schedule for bachelor thesis

Weeks	Time slot	Phase	Tasks
1	23.03. - 29.03.	Preparation	Setup LATEX outline, LITTERBOX branch, Search papers
3	30.03. - 19.04.	Programming	Build N-GRAM LANGUAGE MODEL tool: Tokenization, Calculate probabilities for sequences, Rank results and report bugs
3	20.04. - 10.05.	Evaluation	Find optimal parameters for model, Analyze reported bugs with the help of LITTERBOX
3	11.05. - 31.05.	Writing	Structure, Research Questions, Main body, Introduction, Conclusion
1.5	01.06. - 10.06.	Correction	Bibliography, Content, Grammar
0.5	11.06. - 14.06.	Submission	Print and bind bachelor thesis

Bibliography

[Boe+13] Bryce Boe et al. “Hairball: Lint-Inspired Static Analysis of Scratch Projects.” In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education.* SIGCSE 13. Denver, Colorado, USA: Association for Computing Machinery, 2013, pp. 215–220 (cit. on p. 2).

[Frä+20] Christoph Frädrich et al. “Common Bugs in Scratch Programs.” In: *ITiCSE20: 25th annual conference on Innovation and Technology in Computer Science Education.* To appear. Association for Computing Machinery, 2020 (cit. on p. 2).

[HWM09] Sangmok Han, David R. Wallace, and Robert C. Miller. “Code Completion from Abbreviated Input.” In: *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering.* ASE 09. USA: IEEE Computer Society, 2009, pp. 332–343 (cit. on p. 2).

[Hin+12] Abram Hindle et al. “On the Naturalness of Software.” In: *Proceedings of the 34th International Conference on Software Engineering.* ICSE 12. Zurich, Switzerland: IEEE Press, 2012, pp. 837–847 (cit. on p. 2).

[HCN14] Chun-Hung Hsiao, Michael Cafarella, and Satish Narayanasamy. “Using Web Corpus Statistics for Program Analysis.” In: *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications.* OOPSLA 14. Portland, Oregon, USA: Association for Computing Machinery, 2014, pp. 49–65 (cit. on p. 3).

[KMA13] Wei Ming Khoo, Alan Mycroft, and Ross Anderson. “Rendezvous: A Search Engine for Binary Code.” In: *Proceedings of the 10th Working Conference on Mining Software Repositories.* MSR 13. San Francisco, CA, USA: IEEE Press, 2013, pp. 329–338 (cit. on p. 3).

[MR14] J. Moreno and G. Robles. “Automatic detection of bad programming habits in scratch: A preliminary study.” In: *IEEE Frontiers in Education Conference (FIE) Proceedings*. IEEE, Madrid, 2014, pp. 1–4 (cit. on p. 2).

[MRR15] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. “Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking.” In: *RED. Revista de Educación a Distancia* 46 (2015), pp. 1–23 (cit. on p. 2).

[Nes+08] Syeda Nessa et al. “Software Fault Localization Using N-gram Analysis.” In: *Wireless Algorithms, Systems, and Applications*. Ed. by Yingshu Li et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 548–559 (cit. on p. 3).

[Ngu+13] Tung Thanh Nguyen et al. “A Statistical Semantic Language Model for Source Code.” In: *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2013. Saint Petersburg, Russia: Association for Computing Machinery, 2013, pp. 532–542 (cit. on p. 2).

[RVY14] Veselin Raychev, Martin Vechev, and Eran Yahav. “Code Completion with Statistical Language Models.” In: *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 14. Edinburgh, United Kingdom: Association for Computing Machinery, 2014, pp. 419–428 (cit. on p. 2).

[SKF19] Andreas Stahlbauer, Marvin Kreis, and Gordon Fraser. “Testing Scratch Programs Automatically.” In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2019. Tallinn, Estonia: Association for Computing Machinery, 2019, pp. 165–175 (cit. on p. 2).

[Var+19] Angela Vargas-Alba et al. “Bad Smells in Scratch Projects: A Preliminary Analysis.” In: *Proceedings of the 2nd Systems of Assessments for Computational Thinking Learning Workshop*. 2019 (cit. on p. 2).

[Wan+16] Song Wang et al. “Bugram: Bug Detection with n-Gram Language Models.” In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ASE 2016. Singapore, Singapore: Association for Computing Machinery, 2016, pp. 708–719 (cit. on pp. 1, 3).