Kernel-level expression generator

Job Petrovčič, Sebastian Mežnar, and Ljupčo Todorovski

University of Ljubljana, Jadranska 19, 1000 Ljubljana, Slovenia
jp10210@student.uni-lj.si
Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Abstract. We introduce a neural generator that produces valid Lean 4 expressions directly at the kernel level, bypassing Lean's elaboration process. To train the generator, we implement Lean's core type checking in Python and integrate it into a reinforcement learning environment that assigns rewards for well-typed subexpressions. We evaluate the model's ability to learn and adapt within this environment.

Keywords: Proof assistant · Lean · Type checker · Abstract syntax tree · Reinforcement learning · Recurrent neural network

1 Introduction

Machine learning (ML) has demonstrated its potential for formalized math, as highlighted by recent advancements in neural proof search [5,9,12] and recommendation systems [2,6,7]. Reinforcement learning (RL) is a popular choice to train these models [12,4] for tasks like premise or tactic selection. Some authors combine RL with supervised training [3,8]. In RL frameworks, the agent interacts with a proof assistant (PA) in various ways. A popular approach is for the agent to communicate an action in the form of a tactic. The PA runs the tactic to generate kernel-level expressions, which are then evaluated by the type checker.

Here, we introduce an RL agent that engages directly with the type checker. Our method enables the agent to construct an expression tree while an online type checker validates the tree, and the environment assigns rewards based on the tree's correctness. We further show that a neural network can be trained within this environment to generate valid Lean 4 expressions.

2 Methods

The RL framework consists of the environment, which is based on a type checker, and a neural model that acts as the agent. The agent incrementally creates an expression by adding nodes to the corresponding abstract syntax tree (AST) in a structural manner. After each iteration, the partially created expression is passed to the environment's type checker to assign rewards based on compliance with Lean 4's type theory. We briefly describe each component of the framework.

Expressions and type checker The expressions are represented as ASTs that correspond to Lean's formal grammar. The grammar consists of 13 possible

primitive constructors, some of which require names or indices. A special symbol is reserved for holes, representing missing parts of an expression. Following [1], we adapt a simplified version of Lean 4's kernel, which processes the agent's output. We implement the kernel in Python to fully integrate it into the framework for greater flexibility and efficiency while also leveraging Python's robust ML support.

RL framework The agent's action is to replace the first hole (in the preorder traversal of the current AST) by inserting a node using one of the constructors. If the constructor requires children, sub-holes are added in place. The initial state of the agent is an AST with only one node, a hole. The environment rewards or penalizes the agent based on completed subtree validity, evaluated by the type checker. Invalid subtrees terminate the process with a penalty, valid ones earn rewards, and unfinished subtrees allow continuation.

Neural agent model We use a recurrent neural network (RNN) [11] to implement the agent. The input comprises the hidden embedding of the parent and the relative position of the hole among the parent's children. The model outputs a probability distribution over the constructors used to fill the hole, from which we sample an action. Additionally, it may compute a distribution for selecting bound variables or names when necessary.

Training We train the model using policy gradient [10] and shape the reward exponentially, with the adjustment inversely proportional to the root node's depth of the subtree, promoting globally correct trees over merely correct subtrees. We dampen rewards for smaller trees to encourage longer expressions and reward diverse use of constructors and constants to prevent trivial outputs.

Initially, the model generates expressions, of which less than 22% are valid. Among the valid expressions, the average number of nodes is 2. After 100 epochs of training with a batch size 8192, we find that the model learns to generate valid Lean 4 expressions 96% of the time with expressions averaging around 15 nodes. It learns to use basic constants from a fixed set of 17, with half having at least a 5% chance of being used in the expression.

3 Conclusion

Our RL framework can generate valid Lean expressions with high accuracy. We currently examine how rewards in this environment can be better utilized to navigate the model towards more correct expressions. Some constants are still neglected by the model, and we will explore how to shape the rewards further for more diverse expressions.

In further work, we aim to explore how models can generate representative embeddings of ASTs of Lean 4 expressions. We plan to consider two approaches. The first approach is to use the RNN to embed the ASTs. The second is to use the model as a generator of expressions that can be used to train a secondary model for embedding. Finally, we plan to use the embeddings for downstream tasks, particularly for premise selection.

Acknowledgements

The authors acknowledge the financial support of the Slovenian Research Agency via the Gravity project *AI for Science*, GC-0001, and by the ARRS Grant for young researchers (second author). The authors especially appreciate the helpful comments and suggestions by Andrej Bauer.

References

1. Bailey, C.: Type inference in Lean 4, https://ammkrn.github.io/type_checking_in_lean4/, last accessed: 2025-01-22
2. Bauer, A., Petković, M., Todorovski, L.: MLFMF: Data sets for machine learning for mathematical formalization. In: Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., Levine, S. (eds.) Proceedings of the 37th International Conference on Neural Information Processing Systems (Datasets and Benchmarks). pp. 50730–50741 (2023), https://dl.acm.org/doi/abs/10.5555/3666122.3668329
3. Crouse, M., Abdelaziz, I., Makni, B., Whitehead, S., Cornelio, C., Kapanipathi, P., Srinivas, K., Thost, V., Witbrock, M., Fokoue, A.: A deep reinforcement learning approach to first-order logic theorem proving. Proceedings of the AAAI Conference on Artificial Intelligence **35**(7), 6279–6287 (2021). https://doi.org/10.1609/aaai.v35i7.16780
4. Huang, D., Dhariwal, P., Song, D., Sutskever, I.: Gamepad: A learning environment for theorem proving. In: International Conference on Learning Representations (2019). https://doi.org/10.48550/arXiv.1806.00608
5. Jiang, A.Q., Li, W., Tworkowski, S., Czechowski, K., Odrzygóźdź, T., Miłoś, P., Wu, Y., Jamnik, M.: Thor: Wielding hammers to integrate language models and automated theorem provers. In: Oh, A.H., Agarwal, A., Belgrave, D., Cho, K. (eds.) Proceedings of the 36th International Conference on Neural Information Processing Systems. pp. 8360–8373 (2022), https://dl.acm.org/doi/10.5555/3600270.3600878
6. Kogkalidis, K., Melkonian, O., Bernardy, J.P.: Learning structure-aware representations of dependent types. In: The Thirty-eighth Annual Conference on Neural Information Processing Systems. vol. 38 (2024). https://doi.org/10.48550/arXiv.2402.02104
7. Mikuła, M., Tworkowski, S., Antoniak, S., Piotrowski, B., Jiang, A.Q., Zhou, J.P., Szegedy, C., Kuciński, Ł., Miłoś, P., Wu, Y.: Magnushammer: A transformer-based approach to premise selection. In: The Twelfth International Conference on Learning Representations (2024). https://doi.org/10.48550/arXiv.2303.04488
8. Polu, S., Han, J.M., Zheng, K., Baksys, M., Babuschkin, I., Sutskever, I.: Formal mathematics statement curriculum learning. In: The Eleventh International Conference on Learning Representations (2023). https://doi.org/10.48550/arXiv.2202.01344
9. Polu, S., Sutskever, I.: Generative language modeling for automated theorem proving (2020), https://arxiv.org/abs/2009.03393
10. Prince, S.J.: Understanding Deep Learning. The MIT Press, United Kingdom (2023), http://udlbook.com
11. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation, p. 318–362. MIT Press, Cambridge, MA, USA (1986)

12. Wu, M., Norrish, M., Walder, C., Dezfouli, A.: Tacticzero: learning to prove theorems from scratch with deep reinforcement learning. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. pp. 9330–9342. Curran Associates Inc., Red Hook, NY, USA (2021), https://dl.acm.org/doi/10.5555/3540261.3540975