Symbolic World Models in Lean 4 for Reinforcement Learning

Matěj Kripner

Keywords: Model-based RL, symbolic regression, Lean, evolutionary algorithm

Summary

We propose a novel approach to model-based reinforcement learning by synthesizing symbolic world models in the Lean 4 proof assistant. Leveraging Lean's formal language for mathematics, we encode environment dynamics as interpretable, verifiable rules. Our system integrates a planning agent, an evolutionary algorithm inspired by AlphaEvolve, and a Lean server that predicts the dynamics of an environment using a set of already synthesized rules. We evaluate our approach on a custom cellular automaton environment called `FireHelicopter`. This environment simulates the dynamics of a forest fire and requires the agent to maximize forest preservation. We explore two training objectives: a pragmatic one focused on maximizing agent's return, and a descriptive one prioritizing accurate world prediction. To our knowledge, this is the first use of a general formal mathematics language for model-based RL. We hypothesize that this is a promising avenue for sample-efficient, safe, and interpretable reinforcement learning in real-world scenarios.

Contribution(s)

1. We propose a novel connection of model-based RL and a general formal mathematics language.
   Context: Other approaches have experimented with synthesizing symbolic world models using custom languages, but none used a general-purpose mathematics language.
2. We implement an evolutionary algorithm for symbolic classification where the mutation function is guided by a large language model, similarly to AlphaEvolve (Novikov et al., 2025).
   Context: None
3. We evaluate our approach on two distinct world modeling objectives – reward maximization and truthfulness.
   Context: None
4. We release a cellular automaton-based environment suitable for evaluating model-based RL approaches.
   Context: A similar environment already exists but is unsuitable in our case (see Section 2).

Symbolic World Models in Lean 4 for Reinforcement Learning

Matěj Kripner[1]
kripner@ufal.mff.cuni.cz

[1]**Charles University, Faculty of Mathematics and Physics**

Abstract

We propose a novel approach to model-based reinforcement learning by synthesizing symbolic world models in the Lean 4 proof assistant. Leveraging Lean's formal language for mathematics, we encode environment dynamics as interpretable, verifiable rules. Our system integrates a planning agent, an evolutionary algorithm inspired by AlphaEvolve, and a Lean server that predicts the dynamics of an environment using a set of already synthesized rules. We evaluate our approach on a custom cellular automaton environment called `FireHelicopter`. This environment simulates the dynamics of a forest fire and requires the agent to maximize forest preservation. We explore two training objectives: a pragmatic one focused on maximizing agent's return, and a descriptive one prioritizing accurate world prediction. To our knowledge, this is the first use of a general formal mathematics language for model-based RL. We hypothesize that this is a promising avenue for sample-efficient, safe, and interpretable reinforcement learning in real-world scenarios.

1 Introduction

Modern reinforcement learning (RL) approaches drive progress in complex real-world environments with unknown dynamics such as robotics or autonomous driving. To this end, model-based RL (Sutton, 1991; Moerland et al., 2023) is a traditional technique of addressing some of the challenges that arise when an environment simulator is not readily available and when actions of the agent can have real-world consequences. Specifically, a learned world model can 1) improve sample efficiency, reducing the number of necessary interactions with the environment, 2) improve generalization by enabling transfer to unforeseen tasks, serving as a foundational model for the environment, 3) allow test-time planning, improving performance in complex environments necessitating backtracking where some actions are not reversible, and 4) lead to safer exploration where the agent can evaluate risky strategies before executing them.

Additionally, learned world models can seek to be interpretable by humans. This offers the possibility of understanding previously unknown dynamics of a complex system and obtaining some guarantees on the behavior of the learned policy.

One avenue for obtaining interpretable world models is to construct them using a symbolic language which can be directly read by humans. For example, the field of genetic programming strives to automatically synthesize programs expressed as trees using a domain-specific set of operators and constants. More generally, symbolic regression is the problem of finding a mathematical formula or program which models an unknown function given a dataset of its inputs and outputs, optimizing for accuracy and simplicity. Symbolic regression has also been applied to RL, using a model found by a genetic algorithm to solve classic control RL environments with excellent sample efficiency (Kamienny et al., 2022; Gorodetskiy et al., 2024).

Building on top of these contributions, we recognize that the real world together with many artificial RL environments can be modeled by the laws of physics which in turn can be described using mathematics. Therefore, we hypothesize that it is beneficial to construct world models of such environments using formal languages designed for mathematics, the most prominent of which is Lean (Moura & Ullrich, 2021).

Lean has been used successfully in the formalization of mathematics (van Doorn et al., 2023), physics (Tooby-Smith, 2024), and chemistry (Bobbin et al., 2024), along with applications in software verification (Avigad et al., 2025) and cryptography (Doussot, 2024). The standard library of Lean currently offers more than 100k[1] formal definitions and 200k formal statements together with their automatically verifiable proofs, which can be utilized in the formalization of new theories. Instead of being limited to a specific domain, Lean is as expressive as mathematics itself.

In this paper, we present our preliminary results of applying an evolutionary algorithm as a symbolic classification engine to model the dynamics of a simple cellular automaton-based RL environment. The model is learned online, gathering experience using a simple planning agent. Our contributions are as follows:

- We present an end-to-end learning system consisting of a planning agent which collects experience in an unknown environment, an evolutionary algorithm inspired by AlphaEvolve (Novikov et al., 2025), and a Lean server that predicts the dynamics of the environment using a set of already synthesized rules written in Lean. This serves as a proof of concept for using Lean to model RL environments. To the best of our knowledge, this is the first such attempt.
- We evaluate our approach on two distinct world modeling objectives.
- We present a Gymnasium-compatible (Towers et al., 2024) environment `FireHelicopter` suitable for evaluating model-based RL approaches. We release all our code as open source[2] and the `gym-cellular` environment as a user-friendly Python package[3].

2 Related Work

Symbolic Regression in Reinforcement Learning Symbolic regression has been applied to RL to construct interpretable policies that generalize well from limited data. Kamienny et al. (2022) used a genetic algorithm on a predefined number of operators to synthesize a symbolic world model for the Cartpole environment (Barto et al., 1983), planning using the cross-entropy method (Rubinstein, 1999) and achieving state-of-the-art at the time in terms of sample efficiency. Similarly, genetic programming was used by Hein et al. (2018) to directly learn symbolic policies on classical control tasks and by Gorodetskiy et al. (2024) to generate training data for a model-free algorithm.

Evolutionary and Genetic Algorithms Evolutionary algorithms provide a general framework for optimizing black-box fitness functions, drawing inspiration from the evolutionary process in nature. Classical methods such as genetic programming (Koza, 1994) and NEAT (Stanley & Miikkulainen, 2002) demonstrated the potential of this line of research, but struggled to find practical applications. More recently, evolutionary strategies were revived as scalable RL optimizers, illustrating that evolutionary methods can match deep RL performance in some cases due to being better suited for massively parallel computation. Subsequently, AutoML-Zero (Real et al., 2020) used evolutionary search to automatically synthesize simple machine learning algorithms. Recently, AlphaEvolve (Novikov et al., 2025) successfully scaled evolutionary program synthesis, guiding the mutation function with a large language model and discovering state-of-the-art solutions to a range of problems in mathematics, computer science, hardware design, and more.

Cellular Automata Environments In designing the `FireHelicopter` environment, we take heavy inspiration from `gym-cellular-automata` – an existing cellular automaton-based en-

[1] Retrieved in Jun, 2025.
[2] https://github.com/Kripner/lean-worlds
[3] https://pypi.org/project/gym-cellular

vironment modeling a forest fire.[4] Instead of utilizing this existing work, we ultimately decided to implement our own environment for three reasons.

First, we find the cellular automaton implemented in `gym-cellular-automata` to be too simple for the purpose of evaluating automatic synthesis of a world model. Second, we encountered several technical difficulties when interacting with `gym-cellular-automata`. For example, its interactive rendering logic does not work out-of-the-box. Most importantly, `gym-cellular-automata` is a stochastic environment which prevents the application of our approach. We discuss this limitation in the Conclusion.

3 Cellular Automaton Environment

We present a reinforcement learning environment based on Gymnasium (Towers et al., 2024) driven by a cellular automaton, called `FireHelicopter`. The environment is a simplified simulator of a forest fire where the goal of the agent is to guide a firefighting helicopter and keep as many trees alive as possible. Below, we detail the mechanics of this environment. For comparison with existing environments, refer to Section 2.

In `FireHelicopter`, the agent is positioned on a 10×10 grid where each cell takes on one of the following values: Empty (**E**), Tree (**T**), Igniting Fire ($\mathbf{F}_1$), Developing Fire ($\mathbf{F}_2$), Developed Fire ($\mathbf{F}_3$), and Rock (**R**). The cell values evolve as a cellular automaton, where the change of a cell value is determined only by the values of its four neighbors, and all values are updated simultaneously.

The precise local rule of the cellular automaton is shown in Figure 1 and a progression of an example episode is shown in Figure 2. In the initial state, Trees span the whole grid except for one square with Igniting Fire. Additionally, approximately 10% of the squares are occupied by Rocks which remain inert throughout the episode.

Once an Igniting Fire progresses through Developing Fire into a Developed Fire, it spreads to any Trees in its 4-neighborhood, leaving the burned square Empty. The fire is extinguished at any stage if the agent moves to the given square. Finally, a Tree spreads to any Empty neighbor, regenerating the forest.

Formulated as a Markov Decision Process (MDP), the agent's *observation* is its position together with the current value for each cell in the grid. The set of *actions* are the four directions in which the agent can move. Finally, the reward at any given timestep is the number of existing Trees divided by 100. An episode is terminated after 100 steps have been reached.

We hypothesize that the described environment is suitable for evaluating planning agents with a learned world model, since predicting the location of fire squares at any point in the future is critical for deciding where to move next.

4 World Model Synthesis

In general, we identify two distinct objectives against which a world model can be optimized and evaluated. First, a *pragmatic* objective seeks to find a world model that enables a planning agent to achieve the highest possible return. To this end, a pragmatic objective only forces learning the environment dynamics needed to choose high-reward actions, without necessitating other aspects of the environment to be modeled.

In contrast, a *descriptive* objective requires a world model to correctly capture all aspects of an environment, independent of any single downstream task. As such, a world model optimized using a descriptive objective is more general in the sense that it can be deployed in a different context or facilitate the interpretability of the environment dynamics by humans.

[4] https://github.com/elbecerrasoto/gym-cellular-automata

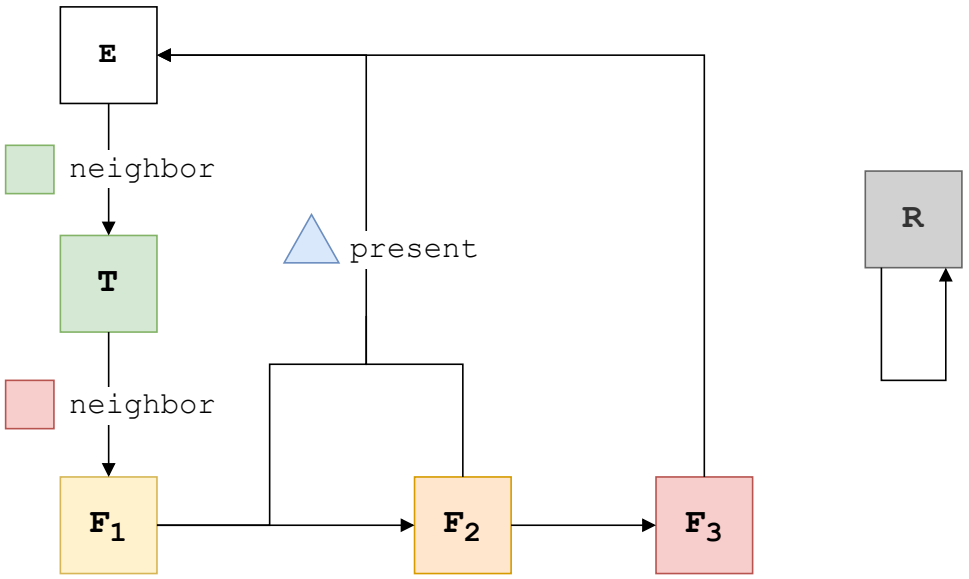
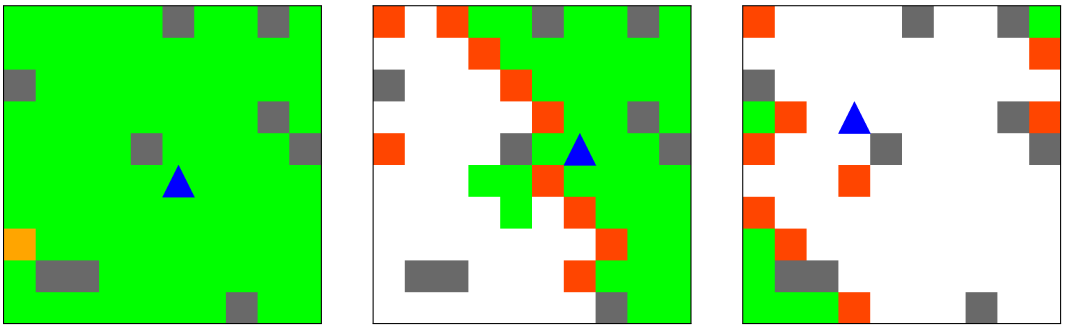


Figure 1: The cellular automaton rule guiding one step of our environment. A tree (**T**) can grow to a neighboring empty square (**E**) or catch fire from a neighboring developed fire ($\mathbf{F}_3$), yielding an igniting fire ($\mathbf{F}_1$). Igniting fire transitions through a developing fire ($\mathbf{F}_2$) before becoming fully developed. The fire can be extinguished at any point if the agent (blue triangle) moves to the given square. After a fire is extinguished or burns through all the stages, it transitions to an empty square. Additionally, rocks (**R**) stay unchanged throughout the episode. If no transition is applicable, the given cell does not change.



(a) Initial position. (b) Fighting the fire. (c) The forest is lost.

Figure 2: Various stages of the episode. Initially, only one tree is burning and the agent is placed in the center of the grid. When the fire is not contained and all trees are burned down, the forest will not regenerate and the reward will be zero until episode end.

In light of this distinction, we optimize our learning system individually for each of the two objectives. Both approaches differ only in the fitness function provided to the evolutionary algorithm, which we detail in Section 4.2. First, Section 4.1 presents a world model is formalized in Lean.

4.1 Lean Formalization

In our case, a world model attempts to predict the next value for each grid cell given the current state of the grid and the position of the agent. We use Lean 4 to represent this dynamics.

Specifically, we factorize the world model into a set of *rules* that attempt to describe the local rule for a given cell in the cellular automaton. Each rule consists of a *body* and a *target*. The body of a rule is a predicate written in Lean which evaluates to either `true` or `false` given the current state of the grid, the position of the agent and the coordinates of the given cell. If the rule activates for a given cell, evaluating to `true`, the value of the cell is updated to match the rule's target.

Conflicts arising from the activation of multiple rules are resolved by assigning a precedence to cell types, which is equal to the order in which they are presented in Section 3. When no rule activates for a given cell, the cell's value stays unchanged.

As an example, Figure 3 lists a set of hand-written rules that model the dynamics of the `FireHelicopter` environment.

$$
\begin{aligned}
\texttt{E} &\leftarrow \texttt{grid}\ p = \texttt{F}_3 \\
\texttt{E} &\leftarrow \texttt{grid}\ p = \texttt{F}_2 \wedge p = a \\
\texttt{E} &\leftarrow \texttt{grid}\ p = \texttt{F}_1 \wedge p = a \\
\texttt{T} &\leftarrow \texttt{grid}\ p = \texttt{E} \wedge \texttt{neighbor}\ p\ (\lambda p \mapsto \texttt{grid}\ p = \texttt{T}) \\
\texttt{F}_1 &\leftarrow \texttt{grid}\ p = \texttt{T} \wedge \texttt{neighbor}\ p\ (\lambda p \mapsto \texttt{grid}\ p = \texttt{F}_3) \\
\texttt{F}_2 &\leftarrow \texttt{grid}\ p = \texttt{F}_1 \wedge p \neq a \\
\texttt{F}_3 &\leftarrow \texttt{grid}\ p = \texttt{F}_2 \wedge p \neq a
\end{aligned}
$$

Figure 3: Hand-written Lean rules corresponding to the true behavior of the `FireHelicopter` automaton.

We note that the presented world model factorization is relatively natural in Lean due to the functional nature of Lean, as compared to imperative languages.

4.2 Evolutionary Symbolic Classification

We employ an evolutionary algorithm to synthesize a symbolic world model. Although several evolutionary algorithm libraries are readily available (notably the LEAP Library (Coletti et al., 2020)), we found their interface to be constrained for our specific needs. Specifically, our approach benefits from evaluating both the fitness function and the mutation function in batches rather than individually. To achieve this, we opted to implement an evolutionary algorithm manually. The algorithm, which we detail below, follows the familiar scheme of fitness evaluation, selection, and mutation.

Fitness evaluation. Following the distinction of two different world model objectives described at the beginning of this section, we define two different fitness functions. In the pragmatic setting, the fitness function of a world model is simply the return achieved by a planning agent guided by the world model, averaged over k rollouts.

On the other hand, in the descriptive setting, we set the fitness function to be the F-score of the world model's predictions when viewed as a binary classification. Specifically, for each timestep, we define true positives `tp` as the number of cells that changed and the world model predicted them correctly, Analogously, false negatives `fn` correspond to the cells for which no rule was activated by their value did change, and false positives `fp` correspond to cells which did not change although the world model predicted that they would.

We find that the F-score, calculated as $2 \cdot \texttt{tp} \,/\, (2 \cdot \texttt{tp} + \texttt{fp} + \texttt{fn})$, better represents the performance of a world model since, in most timesteps, the majority of cells do not change, and thus predicting them is trivial.

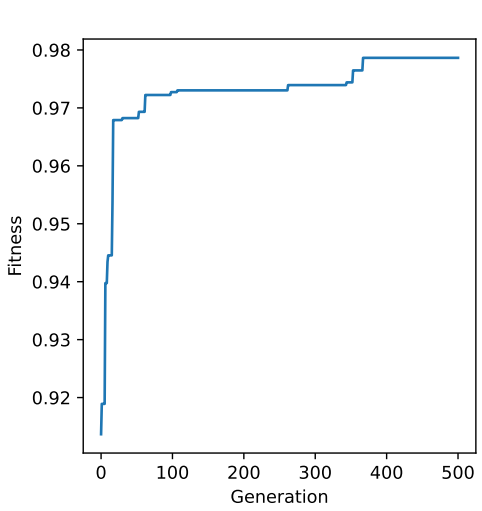
Selection. To select parents for crossover, we use fitness-proportionate selection implemented using Stochastic Universal Sampling (SUS) (Brindle, 1980) with temperature t. In this approach, the expected number of times an individual with fitness f is selected is proportional to f^t.

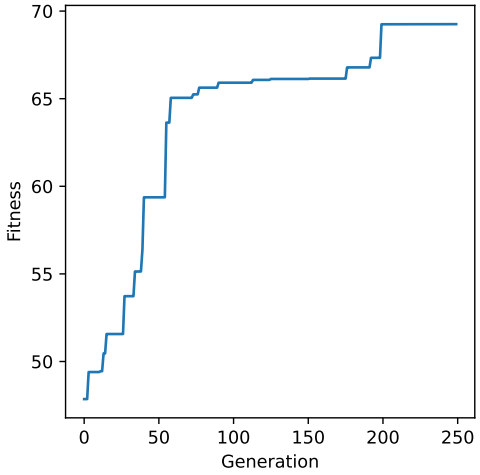
Mutation. Similarly to AlphaEvolve (Novikov et al., 2025), we guide mutations using a large language model (LLM) rather than hand-crafted rules. Specifically, we use Qwen3-4B (Yang et al., 2025) prompted with a mutation prompt listed in Appendix A.

Importantly, the premises presented to the LLM are obfuscated by variable renaming to mitigate the possibility that the model infers any connection with a cellular automaton or the spread of fire. The goal of this is to make our approach as general as possible.

5 Experiments

We apply our world model synthesis approach presented in Section 4 to the cellular automaton-based environment described in Section 3 using both a descriptive and a pragmatic objective. For experience collection and evaluation with the learned world model, we use a simple planning agent that considers all future trajectories up to depth $d = 4$ and selects the one maximizing its return.





$$
\begin{aligned}
\texttt{E} &\leftarrow \texttt{grid}\ \ p = \texttt{F}_3 \\
\texttt{E} &\leftarrow \texttt{grid}\ \ p = \texttt{E} \\
\texttt{F}_3 &\leftarrow \texttt{grid}\ \ p = \texttt{F}_2 \\
\texttt{F}_1 &\leftarrow \texttt{neighbor}\ \ p\ \ (\lambda x \mapsto \texttt{grid}\ \ x = \texttt{F}_3) \\
\texttt{F}_1 &\leftarrow (\texttt{grid}\ \ a = \texttt{R}) \wedge (\texttt{grid}\ \ p = 6) \\
\texttt{E} &\leftarrow \texttt{grid}\ \ p = \texttt{F}_3 \\
\texttt{F}_2 &\leftarrow \texttt{grid}\ \ p = \texttt{F}_1
\end{aligned}
$$

(a) Descriptive objective.

$$
\begin{aligned}
\texttt{T} &\leftarrow \texttt{neighbor}\ \ p\ \ (\lambda x \mapsto \texttt{grid}\ \ x = \texttt{F}_3\ \ \wedge \\
&\qquad \texttt{grid}\ \ a = \texttt{F}_3 \wedge x = a) \\
\texttt{F}_1 &\leftarrow \texttt{grid}\ \ p = \texttt{F}_2 \\
\texttt{T} &\leftarrow \texttt{R} = \texttt{F}_2 \\
\texttt{E} &\leftarrow \texttt{grid}\ \ p = \texttt{F}_1 \\
\texttt{T} &\leftarrow \texttt{grid}\ \ p = \texttt{F}_1 \\
\texttt{T} &\leftarrow \texttt{grid}\ \ a = \texttt{F}_1 \wedge \texttt{true} \\
\texttt{F}_1 &\leftarrow \texttt{neighbor}\ \ p\ \ (\lambda x \mapsto \neg(\texttt{grid}\ \ x \neq \texttt{F}_1))
\end{aligned}
$$

(b) Pragmatic objective.

Figure 4: Training plots and resulting world models for both objectives.

Figure 4 shows the training plots for both objectives together with the best individual in the last generation. The results indicate that the evolutionary algorithm is capable of optimizing both objectives individually. In the descriptive case, the resulting world model qualitatively resembles the true environment dynamics, but still contains inaccuracies, redundancies, and omissions. For example, one of its rules states that a cell transitions to an Igniting Fire if it is currently occupied by a Developed Fire, which is not true.

In the pragmatic case, the world model is qualitatively bad. Still, it enables the planning agent to significantly improve its performance compared to a random model. Crucially, the synthesized

model contains a rule stating that a Tree grows in a cell if it is currently occupied by a Developed Fire under the condition that the agent (a firefighting helicopter) is present. Although the rule is not accurate, as such a cell transitions to an Empty state before a Tree can grow, it allows the agent to understand that when a fire is extinguished, a Tree will later be able to grow.

Conclusion

We presented a novel connection of symbolic model-based RL and the formal verification language Lean, evaluating an end-to-end learning system individually to achieve high reward in an environment and to model the environment accurately.

Limitations and future work. Although we demonstrated the technical viability of such an approach, future work remains before the system is useful in a practical application. Firstly, we used a standard off-the-shelf evolutionary algorithm without tuning it for our specific use case. Our plan is to integrate some of the techniques used by FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025), such as island-based population and diversity score. Additionally, while the advantage of our mutation function is its generality, it might instead be beneficial to provide the LLM guiding the mutation with some context about the problem statement or the current best solution.

Second, we did not address stochastic environments which are commonplace in the real world. Although Lean is well suited for working with probability distributions,[5] additional work is required before stochastic world models can be synthesized.

Third, our approach does not utilize the potential of Lean as a proof assistant. Aside from being well suited for formalizing definitions and statements, the main capability of Lean lies in the formalization and verification of mathematical proofs. Future work should explore the possibility of synthesizing formal proofs of various aspects of the world model, such as its internal consistency.

Finally, more robust experimental results are needed to evaluate the potential of symbolic formal world models in RL. For example, while we evaluated the optimization of both the descriptive and the pragmatic objectives individually, it would be beneficial to explore how the two objectives interact and how they can be optimized in parallel. In addition, our experiments should be replicated to provide error estimates.

Acknowledgments

This project was supported by the grant no. 25-18031S of the Czech Science Foundation (GAČR). This research was partially supported by SVV project number 260 821. We acknowledge VSB – Technical University of Ostrava, IT4Innovations National Supercomputing Center, Czech Republic, for awarding this project access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CSC (Finland) and the LUMI consortium through the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (grant ID: 90254). The author thanks Milan Straka and Matej Straka for helpful suggestions and corrections.

[5] The standard library of Lean offers more than 14k formal declarations in Probability and Measure Theory.

References

Jeremy Avigad, Lior Goldberg, David Levit, Yoav Seginer, and Alon Titelman. A proof-producing compiler for blockchain applications. *J. Autom. Reason.*, 69(2), April 2025. ISSN 0168-7433. DOI: 10.1007/s10817-025-09723-y. URL https://doi.org/10.1007/s10817-025-09723-y.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. DOI: 10.1109/TSMC.1983.6313077.

Maxwell P Bobbin, Samiha Sharlin, Parivash Feyzishendi, An Hong Dang, Catherine M Wraback, and Tyler R Josephson. Formalizing chemical physics using the lean theorem prover. *Digital Discovery*, 3(2):264–280, 2024.

Anne Brindle. *Genetic algorithms for function optimization*. PhD thesis, University of Alberta, 1980. URL https://doi.org/10.7939/R3FB4WS2W. Doctoral thesis, University of Alberta. Available under non-commercial use only.

Mark A. Coletti, Eric O. Scott, and Jeffrey K. Bassett. Library for evolutionary algorithms in python (leap). In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, GECCO '20, pp. 1571–1579, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371278. DOI: 10.1145/3377929.3398147. URL https://doi.org/10.1145/3377929.3398147.

Gérald Doussot. Cryptography experiments in lean 4: SHA-3 implementation. Cryptology ePrint Archive, Paper 2024/1880, 2024. URL https://eprint.iacr.org/2024/1880.

Andrey Gorodetskiy, Konstantin Mironov, and Aleksandr Panov. Model-based policy optimization using symbolic world model. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 664–669. IEEE, 2024.

Daniel Hein, Steffen Udluft, and Thomas A Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018.

Pierre-Alexandre Kamienny et al. Symbolic-model-based reinforcement learning. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4:87–112, 1994.

Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, pp. 625–635, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-79875-8. DOI: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

Alexander Novikov, Ngân Vu, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. Technical report, Technical report, Google DeepMind, 05 2025. URL https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/alphaevolve-a-gemini-powered-coding-agent-for-designing-advanced-algorithms/AlphaEvolve.pdf, 2025.

Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International conference on machine learning*, pp. 8007–8019. PMLR, 2020.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. URL http://nn.cs.utexas.edu/?stanley:ec02.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bull.*, 2(4):160–163, July 1991. ISSN 0163-5719. DOI: 10.1145/122344.122377. URL https://doi.org/10.1145/122344.122377.

Joseph Tooby-Smith. Formalization of physics index notation in lean 4. *ArXiv*, abs/2411.07667, 2024. URL https://api.semanticscholar.org/CorpusID:273970152.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Floris van Doorn, Patrick Massot, and Oliver Nash. Formalising the h-principle and sphere eversion. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, pp. 121–134, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700262. DOI: 10.1145/3573105.3575688. URL https://doi.org/10.1145/3573105.3575688.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

A Mutation Prompt

````
You will get a Lean 4 expression.
Suggest 3 (three) different mutations of it.
Enclose each one in ```lean4 ... ``` tags.
Be creative, try out different things!

There are three types:
- bool (boolean)
- int (integer)
- Foo (an opaque type)

Allowed binary infix operators:
= : int -> int -> bool
= : Foo -> Foo -> bool
∧ : bool -> bool -> bool

Allowed unary prefix operators:
g : Foo -> int
¬: bool -> bool

Allowed constants:
0 : int
...
5 : int
a : Foo
c : Foo

No other operators or constants are allowed!
Make sure the expression is type-correct.
You can use parenthesis to signify precedence.

Additionally, you can use the exists_foo quantifier over Foo
like this: `exists_foo (fun x => ...)`

There can be at most one exists_foo operator in each formula.
Inside the exists_foo clause, you can additionally use the x
variable, which has type Foo.

Generate a diverse set of mutations - do not repeat the same idea
twice.
The three mutations should accomplish the following:
- One modifies the formula.
- One extends the formula.
- One simplifies the formula.

Now, this is your input formula to mutate:

```lean4
{}
```

<think>
... OK, done. Now I will write the 3 different mutations.
</think>

```lean4
````
````