Designing a Theorem Prover for Reinforcement Learning and Neural Guidance

Jonathan Laurent[1] and André Platzer[1]

Carnegie Mellon University, Pittsburgh, United States
{jonathan.laurent,aplatzer}@cs.cmu.edu

Abstract

We discuss the design of Looprl, an experimental interactive theorem prover for loop invariant synthesis that has been optimized from the ground-up for a clean integration of theorem proving with reinforcement learning and neural guidance.

Context and motivation Augmenting tactic-based interactive theorem provers with neural guidance has been the focus of increased attention in recent years [7, 21, 2, 10, 6]. The dominant approach consists in using imitation learning on large corpora of formalized mathematics. However, despite recent efforts involving self-supervised pre-training [6] or data-augmentation [19], this approach is still limited by the amount of available human-produced training data.

Thus, a promising direction is to use reinforcement learning to train learning agents through sheer interactions with the theorem prover and without resorting to human proofs [8, 3]. Curriculum learning can be used to generate tasks of suitable difficulty for the learner [22]. Unfortunately, RL-based approaches tend to be computationally intensive and sample-inefficient, which raises scalability challenges.

We introduce Looprl, an interactive theorem prover for loop invariant synthesis for programs that is designed from the ground-up for effective neural guidance and which exposes an action space that allows efficient exploration by a reinforcement learning agent.

Loop invariant synthesis Given an imperative program with a loop and a final assertion that is to be proved, a loop invariant is a predicate that i) holds before the loop executes, ii) is preserved by the loop body and iii) implies the subsequent assertion or postcondition when assuming the negation of the loop guard. For example, in the program of Figure 1, a possible invariant that would enable us to prove the final assertion (Line 8) is $y \geq 0 \land x \geq 1 \land x \geq y$.

Loop invariant synthesis is an interesting benchmark for us to consider because it raises many of the same challenges as general theorem-proving (e.g. formal reasoning, need for conjecturing...) while being contained enough to allow for meaningful experiments on limited computing resources and interpretable failure modes. Moreover, it is a largely open problem of great relevance to the verification community, with many natural extensions (e.g. program repair, program synthesis...). Related work exists that uses deep reinforcement learning for invariant synthesis [15, 16]. In the aforementioned work, the agent is trained from scratch on every new problem, which typically takes hours. In comparison, our aim is to train an agent that generalizes across tasks and can therefore solve new problems quickly once it is trained.

The Looprl prover Here are some key features of Looprl:

- **Inducing search spaces with *strategies*:** At the outermost level, the neural network does not interact with a set of deterministic tactics. Rather, building on an idea from Selsam [14, 13], the user can define nondeterministic *strategies* that induce search spaces to be explored by the neural network (using a `Search` monad). This provides a flexible way

to leverage domain knowledge into defining reduced action spaces that are more amenable to the kind of semi-random exploration that is typical in reinforcement learning.

- **Tight integration of proof and synthesis via abductive reasoning:** For example, a typical *strategy* (in the sense defined above) for discovering loop invariants [4, 5] is to start considering the postcondition as an invariant and then try and prove it inductive. If the attempt fails, one can look for a missing assumption that would make it hold and suggest it as a new invariant candidate. (In reality, the default strategy of Looprl is more general and also integrates refinements of function symbols along with some form of forward reasoning.) This form of *abductive* reasoning is a key aspect of how humans find proofs [20] and it has built-in support in Looprl.
- **Tag-based proof guidance:** At its core, Looprl provides an `abduction` tactic that takes a formula as an input and returns either `valid` (if a proof is found) or otherwise a weighted list of suggestions for missing assumptions (or symbol refinements). This tactic is implemented using a rule-based rewriting system. Rewriting is guided by a cost function that is implicitly defined by tags on parts of the input formula. These tags are probabilistic and indicate how favorable it is to use a subformula in the proof, how they should be used (e.g. as a contradictory assumption, for eliminating variable x...) and with what level of certainty that prediction is made. Several factors make this architecture especially well-suited for neural guidance: i) the costly operation of evaluating the neural network only has to be performed once before search starts, ii) tagging the input formula can be done efficiently in a single pass using a Transformer encoder [18] or a Graph Neural Network and iii) the `abduction` tactic naturally leverages the uncertainty estimates given by the neural network.

Note that all these ideas are general and thus potentially applicable beyond the problem of invariant synthesis, which we are only considering here as an initial benchmark.

Learned agent Our agent is reminiscent of the AlphaZero algorithm [17], where a neural network (here, a choice of a Transformer [18, 12] or of a Graph Neural Network [11]) is used as a heuristic for Monte-Carlo Tree Search, which is iteratively improved as more experience becomes available. Training tasks are generated using a simple curriculum-learning scheme where random programs of increasing complexity are sampled. For each program, a random assertion is sampled too for which no easy counterexample can be found and whose validity the current network is uncertain about.

Project status and plans

- **The Looprl theorem prover:** We finished implementing the Looprl theorem prover and wrote a simple user interface (Figure 1) for testing purposes. Using this interface and providing manual guidance, we solved a large sample of problems from the invariant-synthesis track of the SyGUS 2017 competition [1].
- **AlphaZero.jl:** In the context of this project, we have released a novel open-source implementation [9] of Deepmind's AlphaZero algorithm [17] written in the Julia language. This implementation is consistently one to two orders of magnitude faster than competing Python implementations, while being equally simple and flexible.
- **By AITP 2021:** We plan to implement the AlphaZero-like agent mentioned earlier and evaluate it on the SyGUS 2017 benchmark also used in [15, 16].

References

[1] Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. Sygus-comp 2017: Results and analysis. *arXiv preprint arXiv:1711.11438*, 2017.

[2] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, pages 454–463. PMLR, 2019.

[3] Kshitij Bansal, Christian Szegedy, Markus N Rabe, Sarah M Loos, and Viktor Toman. Learning to reason in large theories without imitation. *arXiv preprint arXiv:1905.10501*, 2019.

[4] Isil Dillig, Thomas Dillig, Boyang Li, and Ken McMillan. Inductive invariant generation via abductive inference. *Acm Sigplan Notices*, 48(10):443–456, 2013.

[5] Mnacho Echenim, Nicolas Peltier, and Yanis Sellami. Ilinva: Using abduction to generate loop invariants. In *International Symposium on Frontiers of Combining Systems*, pages 77–93. Springer, 2019.

[6] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

[7] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *arXiv preprint arXiv:1806.00608*, 2018.

[8] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. *arXiv preprint arXiv:1805.07563*, 2018.

[9] Jonathan Laurent. Alphazero.jl: A generic, simple and fast AlphaZero implementation. https://github.com/jonathan-laurent/AlphaZero.jl, 2021.

[10] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence Paulson. Isarstep: a benchmark for high-level mathematical reasoning. 2021.

[11] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974, 2020.

[12] Markus N Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. *arXiv preprint arXiv:2006.04757*, 2020.

[13] Daniel Selsam. The IMO grand challenge. http://aitp-conference.org/2020/slides/DS.pdf, 2020. Talk at AITP 2020.

[14] Daniel Selsam. The IMO grand challenge: A battle of ideas. https://dselsam.github.io/IMO-GC-battle-of-ideas/, 2020.

[15] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Neural Information Processing Systems*, 2018.

[16] Xujie Si, Aaditya Naik, Hanjun Dai, Mayur Naik, and Le Song. Code2inv: a deep learning framework for program verification. In *International Conference on Computer Aided Verification*, pages 151–164. Springer, 2020.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[19] Mingzhe Wang and Jia Deng. Learning to prove theorems by learning to generate theorems. *arXiv preprint arXiv:2002.07019*, 2020.

[20] Yuhuai Wu, Markus Rabe, Wenda Li, Jimmy Ba, Roger Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. *arXiv preprint arXiv:2101.06223*,

2021.

[21] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, pages 6984–6994. PMLR, 2019.

[22] Zsolt Zombori, Adrián Csiszárik, Henryk Michalewski, Cezary Kaliszyk, and Josef Urban. Towards finding longer proofs. *arXiv preprint arXiv:1905.13100*, 2019.

A Looprl Screenshot

Figure 1: A screenshot of the Looprl user interface. Here, the user is manually adding a tag that indicates that the loop guard is likely irrelevant in proving the inductivity of the current invariant candidate.