Arithmetical Mini-Games*

Thibault Gauthier and Josef Urban

Czech Technical University in Prague
email@thibaultgauthier.fr josef.urban@gmail.com

Mini-games are playing an important role in the field of artificial intelligence for video games [8, 5]. The principle is that a particularly complex problem can be split it into smaller tasks presenting gentle learning curves. Lessons learned from these easier challenges should prove valuable for solving the original problem.

This divide and conquer strategy could also benefit the automated theorem proving community. That is why we propose three mini-games, each of them being a simplified challenge representative of a larger class of theorem proving problems.

Arithmetic Let $T =_{rec} \{(0, S\, t_1, t_1 + t_2, t_1 \times t_2 \mid (t_1, t_2) \in T^2\}$ and $E =_{def} \{t_1 = t_2 \mid (t_1, t_2) \in T^2\}$. Let x, y be variables and $A =_{def} \{x + 0 = x,\ x \times 0 = 0,\ x + S\ y = S\ (x + y),\ x \times S\ y = x \times y + x\}$. We define R to be rewrite rules for both orientations of equations in A with the exception of $0 \to x \times 0$. For any $e_1, e_2 \in E$ and $t \in T$ the following rules hold:

$$\frac{}{t = t}\ \text{reflexivity} \qquad \frac{e_2}{e_1}\ \text{if } e_1 \to e_2$$

Predictors We compare the performance of two predictors P_{Near} and P_{Tree}. The nearest neighbor algorithm P_{Near} relies on subterm features and is one of the best performing predictor for premise selection tasks in hammers [2]. The tree neural network P_{Tree} mimics the structure of the input formula. Each operator $0, S, +, \times, =$ is replaced by a feed-forward neural network [11] with one hidden layer. The embedding space has dimension four.

Reinforcement Learning The mini-games 2 and 3 are solved by reinforcement learning. We rely on a Monte Carlo tree search [4] to explore different configurations according to their reward potential and progressively improve our predictions form this feedback. Such methodology is described in detail in [7, 10]. Since P_{Tree} is faster than P_{Near}, we use 1600 simulation per move for P_{Tree} and 100 for P_{Near}.

Mini-Game 1: Validity of a Formula

The goal of this mini-game is to decide whether a formula is true or false. By solving it, a predictor will gain the ability to guess the truth value of a formula. Such knowledge could help automated theorem provers avoid spending time searching for a proof of a formula if it is likely to be false. We run a specific instance of this mini-game with a training set of 3200 equations. On a test set of 400 true equations and 400 false equations, we get a percentage of correct guesses of:

$$53.0\% \text{ for } P_{Near} \text{ and } 100.0\% \text{ for } P_{Tree}$$

*Supported by the ERC Consolidator grant no. 649043 AI4REASON and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

The estimation of the truth of a formula given by P_{Near} to estimate is very close to a random guess. For example, the equations $t = u$ and $t \times 0 = u$ may not have the same truth value although they are syntactically close. Moreover, by comparing P_{Tree} to P_{Near}, we can observe that P_{Tree} is not just memorizing the truth values. Otherwise, it would have gotten the same score or a worse score than P_{Near}.

Mini-Game 2: Proof by Rewriting

The goal of this mini-game is to prove a formula through MCTS-guided reinforcement learning using the backward reasoning steps defined by the calculus. The predictor decides at which position in the current term a rewrite step should be applied and the rewrite rule to apply. The choice of a position is made through a series of finite branching choices. Starting from the root position of the equation, the predictor chooses whether to take the left or the right branch and whether to stop or descend further. Therefore, many sub-steps are required to perform a single proof step.

The dataset consists of 100 true equations with shortest proof length ranging from 2 to 6 (20 of each). The number of proven equations after each generation is:

$$0 \times (0 + 0) = 0 \times (0 + S\ 0)$$
$$0 \times (0 + 0) + 0 = 0 \times (0 + S\ 0)$$
$$0 \times (0 + 0) + 0 = 0 \times S\ (0 + 0)$$
$$0 \times (0 + 0) + 0 = 0 \times (0 + 0) + 0$$

P_{Near}	19	32	53	75	81	88	98	99	100	100
P_{Tree}	22	24	32	52	68	88	93	97	93	100

Both P_{Near} and P_{Tree} gradually prove all targets. This shows that this mini-game can be mastered from a dataset of self-generated examples and the syntactical distance given by P_{Near}. Scaling the problem to larger terms and longer proofs may require better generalizations.

Mini-Game 3: Term Synthesis

The goal of this mini-game is to replicate a target term. It provides a simple example of a term building process. Finding a witness (or counter-example) [6, 3] is an natural extension of this mini-game. In order to apply reinforcement learning, we re-frame term synthesis as a search problem. The copy is built starting from the root. At each node, the predictor chooses the operator it wants to imitate. When branching occurs, we first construct the left part of the tree. The replication fails if the copy reaches a size bigger than the target.

The dataset consists of 100 target terms with term size ranging form 6 to 10 (20 of each). The number of replicated term after each generation is:

$$\square,\ S\ \square,\ S\ (\square \times \square),\ S\ (0 \times \square),$$
$$S\ (0 \times (\square + \square)),\ S\ (0 \times (0 + \square)),$$
$$S\ (0 \times (0 + \square)),\ S\ (0 \times (0 + S\ 0))$$

P_{Near}	5	5	8	8	8	9	9	9	9	9
P_{Tree}	11	11	11	11	11	11	11	12	11	11

The discrepancy observed between P_{Near} and P_{Tree} is simply due to the different number of simulations. Since there is exactly one path to the target, any deviation leads to a state where the goal is not reachable anymore. This issue could be solved by adding the possibility to mutate the partially constructed term, making the underlying term rewriting system confluent.

Road Map To conclude, we propose extensions to these mini-games that would move us closer to our final goal of creating stronger automated theorem provers. Our plan is to gradually increase the difficulty of these three problems, test other kind of predictors and change the underlying theory/calculus. At the same time, we would like to design more mini-games, providing a way to evaluate fairly and eventually master other proving [9] and para-proving abilities [1, 12].

References

[1] David Aspinall and Cezary Kaliszyk. What's in a theorem name? (rough diamond). In Jasmin C. Blanchette and Stephan Merz, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 9807 of *LNCS*, pages 459–465. Springer, 2016.

[2] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016.

[3] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[5] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[6] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of Haskell programs. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000.*, pages 268–279. ACM, 2000.

[7] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olsák. Reinforcement learning of theorem proving. *CoRR*, abs/1805.07563, 2018.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Bartosz Piotrowski and Josef Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018.

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, 2017.

[11] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[12] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2018.