Z3-alpha: A Reinforcement Learning Guided SMT Solver

SMT-COMP 2023

Zhengyang Lu*, Stefan Siemer†, Piyush Jha*, Florin Manea†, Joel Day‡, Vijay Ganesh*
*University of Waterloo, Canada
†University of Göttingen, Germany
‡Loughborough University, UK

***Abstract*—In this note, we introduce a Reinforcement Learning guided SMT solver, dubbed Z3-alpha, that supports *QF_NonLinearIntArith*, *QF_NonLinearRealArith* and *QF_Strings*. Z3-alpha uses a framework that combines Reinforcement Learning with logical reasoning in a corrective feedback loop, with the goal of dynamically constructing a strategy for any given input SMT formula.**

I. System Overview

Z3-alpha is a Reinforcement Learning (RL) Guided SMT Solver, which dynamically constructs, in expectation, the *best strategy* for a given input query by minimizing some objective function (typically, solver run time). The term *strategy* refers to a solving algorithm that is constructed by sequencing individual *tactics* as steps in the said strategy. A tactic is a well-defined and implemented reasoning step in an SMT solver, such as *simplify* (apply simplificaton rules), *nla2bv* (convert a nonlinear arithmetic problem into a bit-vector problem), *smt* (the core DPLL(T) SMT solver) [1]. Z3-alpha uses an RL-Logic framework that combines deep Monte-Carlo Tree Search (MCTS) and logical reasoning in a corrective feedback loop. The RL component guides the search for the optimal strategy and the logical part provides corrective feedback. The combined framework enables the solver to learn how to adaptively select the best tactic for a given class of formulas.

Z3-alpha is a derived solver based on Z3 [2], i.e., it selects tactics from Z3's built-in ones. For the *QF_Strings* division, Z3-alpha also additionally chooses from three tactics from our string constraint solver Z3str4 [3]: *Z3str3*, *LAS* (the length abstraction solver), and *string-rewrite-extension*. *Z3str3* is an arrangement based string solver, which is integrated into Z3. *LAS* is a CEGAR-style algorithm that can quickly solve certain string formulas based on abstractions and refinements of integer constraints implied by string equations. The *string-rewrite-extension* tactic implements a set of rewrite rules in the string theory.

Although Z3-alpha can dynamically construct strategies for any given input, the version we submitted to SMT-COMP 2023 is a preliminary one that uses a pre-trained "best" static strategies that we could find using our RL-Logic framework during experimentation over the SMT-COMP benchmarks. The full version of Z3-alpha is available at https://github.com/JohnLyu2/AlphaSMT, which requires a list of Python packages with specific versions. Some of these packages are not available on StarExec, the SMT-COMP running platform, and we currently do not have the bandwidth to support the compatibility of the full version on StarExec.

II. The Combined Reinforcement Learning and Logic Architecture

The deep MCTS algorithm was first introduced in AlphaZero [4], a Go playing program. We have made important changes to adapt the AlphaZero algorithm to the SMT tactic selection problem, in terms of how MCTS is used, the neural network architecture, as well as corrective feedback via logical reasoning.

In our method, a deep neural network, f_θ, serves as both the value function and the policy, evaluating state-action pairs and making *tactic decisions* (the choice function over tactics). The goal of the RL algorithm is to train a neural network f_{θ^*} that approximates the optimal policy and value function. The training happens by iterations. In each iteration, multiple sampling episodes are executed to collect neural network training samples. Each episode works on one formula φ picked from the training benchmark set, and sequentially applies tactics until the formula is solved, or the episode time exceeds a threshold timeout. The tactic applied at each step is selected by a lookahead search algorithm, MCTS. The selected tactic is executed by the base logical solver Z3. This logical reasoning step provides feedback to the RL agent in two ways: (1) state transitions caused by formula rewrites; (2) a reward R collected at the end of the episode, awarding effective and efficient tactic sequences. This reward is considered the sample return for the episode and is used to update the value function, i.e., the neural network f_θ.

Specifically, at the end of an episode, for every step t in this episode, one neural network sample $d_t = (s_t, \pi_t, R)$ is generated, where s_t is the solving state representation, and π_t is the tactic probability distribution output by the MCTS, and as stated above R is the reward. When all sampling episodes of an iteration end, the neural network f_θ is trained upon the training samples from all episode steps.

The rationale behind this framework is as follows: MCTS, which is a lookahead technique, can potentially make better selections over previous ones, i.e., it is a policy improvement step; the sampling episodes, which enable the neural network to make better value predictions based on logical feedback, are doing the policy evaluation. Thus, each training iteration is considered as one policy iteration loop. After an appropriate number iterations, the neural network is expected to be closer to the optimal policy and value function for a given class of input formulas.

The use of RL in solvers is not entirely new [5]. It was perhaps first used by Lagoudakis and Littman [6] and then achieved superior performance in MapleSAT [7], both for branching heuristics. Following works either use RL to improve one aspect of solver heuristics [8], or to select among different solver algorithms [9], [10]. Our proposed framework continues this line of research, and tackles the problem which is more comprehensive than just improving one specific heuristic and more granular than algorithm selection.

III. MCTS Search Algorithm

MCTS is a decision-time planning algorithm, which helps make a better decision at a given environment state by simulating multiple paths from this state [11]. The selection of the simulated paths are guided by both a prior policy and the rewards collected from previous simulations, in order to keep a balance between exploiting expected more rewarding trajectories and exploring less visited ones.

As stated earlier, MCTS serves as a lookahead search at each step t of the sampling episodes, in order to improve the tactic selection at t. MCTS takes two inputs, namely, the current state s_t and the prior policy f_θ. By simulating paths starting at s_t, MCTS keeps updating its estimation of how rewarding it is to take each action from s_t. After many simulation experiences, MCTS outputs a probability distribution recommending tactics to apply at s_t, $\pi(\cdot|s_t) = \alpha_\theta(s_t)$, based on the estimations collected from the simulations. This recommended distribution is assumed to be a better policy than prior policy given by f_θ.

IV. Deep Neural Network

The deep neural network f_θ with parameters θ takes as input a state embedding x, and outputs a tactic probability distribution $\boldsymbol{p}$ and a value estimation v, i.e., $\boldsymbol{p}, v = f_\theta(x)$. The neural network uses transformers [12] to encode the time series relationship between tactic applications. The neural network also applies techniques such as batch normalisation [13], dropout, and ReLU activation. The training aims to adjust the neural network parameters θ, so that the differences between the neural network predicted distribution $\boldsymbol{p}$ and the MCTS output π, and between the predicted v and the recorded R, are minimized.

V. Logical Corrective Feedback

Each tactic application is a logical reasoning step, which is executed by the base solver Z3. Tactics rewrite formulas, thus resulting in the state transition in the RL problem. Since the RL agent makes decisions based on the state it is in now, tactic applications which change the state, provide feedback to the agent in terms of how it should proceed. Moreover, the application of a strategy, i.e., a sequence of tactics, on a formula generates the reward (whether the formula is solved in an efficient manner) in our context. Therefore, logical reasoning plays the role of the environment in the RL modelling, directing and evaluating the agent actions.

Acknowledgements

The submitted solver's source code (especially the string constraint solver part) is largely contributed by the team's previous members: Murphy Berzish, Yunhui Zheng, Mitja Kulczynski, Federico Mora, and Dirk Nowotka.

References

[1] L. De Moura and G. O. Passmore, "The strategy challenge in smt solving," *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, pp. 15–44, 2013.

[2] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[3] F. Mora, M. Berzish, M. Kulczynski, D. Nowotka, and V. Ganesh, "Z3str4: a multi-armed string solver," in *International Symposium on Formal Methods*. Springer, 2021, pp. 389–406.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[5] S. B. Holden *et al.*, "Machine learning for automated theorem proving: Learning to solve sat and qsat," *Foundations and Trends® in Machine Learning*, vol. 14, no. 6, pp. 807–989, 2021.

[6] M. G. Lagoudakis and M. L. Littman, "Learning to select branching rules in the dpll procedure for satisfiability," *Electronic Notes in Discrete Mathematics*, vol. 9, pp. 344–359, 2001.

[7] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for sat solvers," in *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*. Springer, 2016, pp. 123–140.

[8] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro, "Improving sat solver heuristics with graph networks and reinforcement learning," *arXiv preprint arXiv: 1909.11830*, 2019.

[9] M. Balunovic, P. Bielik, and M. Vechev, "Learning to solve smt formulas," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 10 337–10 348. [Online]. Available: http://papers.nips.cc/paper/8233-learning-to-solve-smt-formulas.pdf

[10] N. Pimpalkhare, F. Mora, E. Polgreen, and S. A. Seshia, "Medleysolver: online smt algorithm selection," in *Theory and Applications of Satisfiability Testing–SAT 2021: 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings 24*. Springer, 2021, pp. 453–470.

[11] 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*, vol. 4, no. 1, pp. 1–43, 2012.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456.