Data Generation for SyGuS Problems

Julian Parsert[1,2] and Elizabeth Polgreen[2]

[1] University of Oxford, Oxford, UK
julian.parsert@gmail.com
[2] University of Edinburgh, Edinburgh, UK
elizabeth.polgreen@ed.ac.uk

Syntax-Guided Synthesis [2, 4] (SyGuS) is a paradigm where synthesis problems can be created using syntactic as well as semantic constraints. The SyGuS input format (SyGuS-IF) is a standardised language that allows users to formulate synthesis problems restricting the problem syntactically (using a grammar) and semantically (formal specification).

Machine learning-based synthesis methods range from DeepCoder [5] to the latest large-language model based code synthesis tools [6]. Here a learner is trained on large swathes of training data. We hypothesise that the key reason these methods have not been applied to SyGuS problems is the lack of training data. Despite the broad applications of syntax-guided synthesis [17, 1, 12, 15, 10], the number of SyGuS benchmarks publicly available is roughly limited to the number in the SyGuS competition [3], which comprises of a suite of a few hundred hand-written benchmarks. Efforts to integrate machine learning methods into formal synthesis algorithms mostly limit themselves to input-output example synthesis (or "programming by example") [5, 14, 8, 9, 11, 7, 13]. Automatically generating programming by example problems is relatively straightforward. Si et al [16] apply a deep learning framework to the problem of synthesising invariants. They augment a very small training set by mutating the loops in ways that are guaranteed not to affect the invariant to generate training data.

Automatically generating meaningful logical specifications is challenge that is not tackled in the literature, and, we believe, has limited the progress of machine learning methods for SyGuS up until now. In this work we present an algorithm for producing SyGuS problems and corresponding solutions from SMT problems of the same theory.

Algorithm We assume that we have a valid SMT problem P. This can be obtained by either negating an `unsat` problem or using the solution to a `sat` problem to obtain a valid problem. Our goal is to generate a SyGuS-Problem and corresponding solution. The algorithm performs the following steps:

1. generate a set S of sub-terms in P.
2. perform anti-unification on S to obtain least general generalization (LGG) L
3. replace sub-terms S in P with fresh second-order variable
4. use unification to determine order of arguments applied to second-order variable.
5. translate P (with second-order variable) to SyGuS problem P_s.

In the end the pair P_s is a SyGuS problem and L its solution. While steps 2 to 5 are static, the first step allows for a large amount of variability. This step directly influences the quality and quantity of the produced SyGuS problem/solution pairs. By using different techniques to search for "good" sub-terms we can create different training data/ground truths pairs. Further investigation could lead to using metrics such as largest common prefix or even apply some adversarial learning mechanisms to identify sub-terms in P that lead to good problem/solution pairs.

References

[1] A. Abate, I. Bessa, D. Cattaruzza, L. C. Cordeiro, C. David, P. Kesseli, D. Kroening, and E. Polgreen. Automated formal synthesis of digital controllers for state-space physical plants. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2017.

[2] R. Alur, R. Bodík, E. Dallal, D. Fisman, P. Garg, G. Juniwal, H. Kress-Gazit, P. Madhusudan, M. M. K. Martin, M. Raghothaman, S. Saha, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, volume 40 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 1–25. IOS Press, 2015.

[3] R. Alur, D. Fisman, R. Singh, and A. Udupa. Syntax guided synthesis competition. `http://sygus.seas.upenn.edu/SyGuS-COMP2017.html`, 2017.

[4] R. Alur, R. Singh, D. Fisman, and A. Solar-Lezama. Search-based program synthesis. *Commun. ACM*, 61(12):84–93, 2018.

[5] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow. Deepcoder: Learning to write programs. In *ICLR (Poster)*. OpenReview.net, 2017.

[6] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021.

[7] Y. Chen, C. Wang, O. Bastani, I. Dillig, and Y. Feng. Program synthesis using deduction-guided reinforcement learning. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 587–610. Springer, 2020.

[8] J. Devlin, J. Uesato, S. Bhupatiraju, R. Singh, A.-r. Mohamed, and P. Kohli. Robustfill: Neural program learning under noisy i/o. *arXiv preprint arXiv:1703.07469*, 2017.

[9] K. Ellis, C. Wong, M. I. Nye, M. Sablé-Meyer, L. Morales, L. B. Hewitt, L. Cary, A. Solar-Lezama, and J. B. Tenenbaum. Dreamcoder: bootstrapping inductive program synthesis with wake-sleep library learning. In *PLDI*, pages 835–850. ACM, 2021.

[10] G. Fedyukovich, S. Prabhu, K. Madhukar, and A. Gupta. Quantified invariants via syntax-guided synthesis. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 259–277. Springer, 2019.

[11] K. Morton, W. T. Hallahan, E. Shum, R. Piskac, and M. Santolucito. Grammar filtering for syntax-guided synthesis. pages 1611–1618, 2020.

[12] A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli. Syntax-guided quantifier instantiation. In *TACAS (2)*, volume 12652 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2021.

[13] M. I. Nye, L. B. Hewitt, J. B. Tenenbaum, and A. Solar-Lezama. Learning to infer program sketches. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 4861–4870. PMLR, 2019.

[14] E. Parisotto, A.-r. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli. Neuro-symbolic program synthesis. *arXiv preprint arXiv:1611.01855*, 2016.

[15] A. Reynolds, H. Barbosa, D. Larraz, and C. Tinelli. Scalable algorithms for abduction via enumerative syntax-guided synthesis. In *IJCAR (1)*, volume 12166 of *Lecture Notes in Computer Science*, pages 141–160. Springer, 2020.

[16] X. Si, A. Naik, H. Dai, M. Naik, and L. Song. Code2inv: A deep learning framework for program verification. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 151–164.

Springer, 2020.

[17] P. Subramanyan, Y. Vizel, S. Ray, and S. Malik. Template-based synthesis of instruction-level abstractions for soc verification. In *FMCAD*, pages 160–167. IEEE, 2015.