Self-Learned Formula Synthesis in Set Theory*

Chad E. Brown and Thibault Gauthier

Czech Technical University, Prague

One of the most difficult tasks in higher-order theorem proving is the instantiation of set variables [3, 4]. An important class of theorem proving problems requiring instantiation of a set variable are those requiring induction [6]. Instantiating a set variable often requires synthesizing a formula satisfying some properties. In our work we apply machine learning to the task of synthesizing formulas satisfying a collection of semantic properties. Previous work applying machine learning to induction theorem proving can be found in [10].

Hereditarily finite sets In [1] Ackermann proved consistency of Zermelo's axioms of set theory without an axiom of infinity by interpeting natural numbers $0, 1, 2, \ldots$ as sets. Membership $m \in n$ is taken to hold if bit m is 1 in the binary representation of n, e.g., $0 \in 1$, $1 \in 2$ and $0 \notin 2$. This is known as the *Ackermann encoding* of hereditarily finite sets. We will always consider terms and formulas to be interpreted via the model given by this encoding.

As terms s, t we take variables $x, y, z, \ldots$ as well as $\wp(t)$ (power set of t), $\{t\}$, and $s \cup t$. As atomic formulas we take $s \in t$, $s \notin t$, $s \subseteq t$, $s \not\subseteq t$, $s = t$ and $s \neq t$. Formulas φ, ψ are either atomic formulas or of the form $\varphi \Rightarrow \psi$, $\varphi \wedge \psi$, $\forall x \in s.\varphi$, $\exists x \in s.\varphi$, $\forall x \subseteq s.\varphi$ or $\exists x \subseteq s.\varphi$. Note that all our quantifiers are bounded. As a consequence, for every assignment of free variables to natural numbers we can always (in principle) calculate the truth value for a formula under the assignment. In practice if certain bounds are exceeded evaluation fails.

Formula Generation All formulas up to size 15 with at most one free variable x were generated. For each of these formulas we attempted to evaluate the formula with x assigned to values between 0 and 63. We call this list of truth values the *graph* of the formula. We omitted each formula that failed to evaluate on any of these values. For the remaining formulas, we kept one representative formula (of minimal size) for each subset of $\{0, \ldots, 63\}$ that resulted from an evaluation. This resulted in a set $\mathbb{F}$ of 6750 formulas varying in size from 3 to 15 distributed as indicated in Table 1.

The Formula Synthesis Problem The goal of the synthesis task is to create a formula with one free variable for a given graph. To ensure that the task can be achieved, we choose the graphs of the generated formulas as inputs to our problems. For each formula $\varphi \in \mathbb{F}$ the associated problem is to find a formula ψ that has the same graph as φ by only observing the graph of φ. We restrict ourselves to solutions that construct ψ from left to right if represented in prefix notation.

Size	3	4	5	6	7	8	9	10	11	12	13	14	15
No. of formulas	6	8	22	60	88	260	472	960	638	992	1582	1056	606

Table 1: Number of generated formulas of each size

*Supported by the ERC Consolidator grant no. 649043 AI4REASON

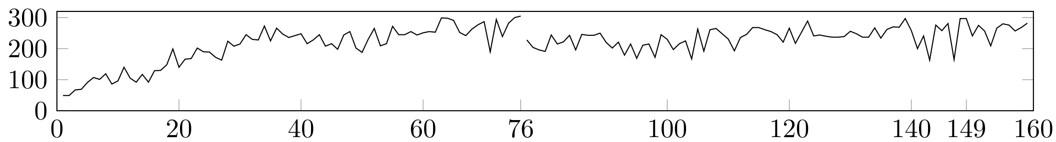

Figure 1: Number of successful formula synthesis (y) at generation (x)

A Solution by Reinforcement Learning Our reinforcement learning framework [7] relies on a curriculum learning approach. It perfects its synthesis abilities on the easiest problems first before moving to harder ones. The difficulty of a problem derived from a formula $\varphi \in \mathbb{F}$ is defined to be the size of φ. Each level consists of 400 graphs with lower levels containing easier problems. Each *generation* consists of an exploration phase and a training phase.

During the exploration phase, the algorithm attempts to find a solution for 400 graphs taken in equal measure from each level lower or equal to the current level. An attempt for a graph g consists of a series of big steps. The number of big steps is limited to twice the size of φ. One big step consists of one call to Monte Carlo tree search algorithm with a partial formula ψ as the root of the search tree. The number of search steps for one MCTS call is set at 50000. Then, the step from the root with the highest number of visits is chosen. This adds one operator to ψ. The updated formula becomes the root of the search in the next big step. The algorithm moves to the next level when it solves strictly more than 75% of the problems in one phase.

During the training phase, a tree neural network (TNN) that predicts both the value and the policy is trained on the 200000 newest examples. Each of those examples is extracted from the root tree statistics after one big step. Since we perform searches for many different graphs, the information about the targeted graph g is given to our network in addition to the partially constructed formula ψ. They are represented together in the tree structure by $concat(g', \psi)$ where $g' \in \mathbb{R}^{64}$ is the embedding of g and *concat* is an additional helper operator. When guiding the MCTS algorithm, noise is added to the predicted policy to favor exploration.

Results In Figure 1, the success rate at each generation of the reinforcement learning run is shown. Level 1 is passed at generation 76 with 305 formulas synthesized. The run is stopped at generation 159. In Table 2, the TNN from generation 149 is tested without noise (Guided) on problems from level 1, 2 and 3. To produce a baseline, we replace the MCTS algorithm by a breadth-first search algorithm (Breadth-first). We also try to figure how much the input graph influences the search by masking its embedding g' (Hidden-graph).

	Breadth-first	Hidden-graph	Guided
Level 1, 2, 3	68, 0, 0	270, 126, 59	338, 240, 165

Table 2: Number of successful formula synthesis in level 1, 2 and 3 respectively

The formula $\varphi = \exists y \in x.\ x \not\subseteq \wp(y) \in \mathbb{F}$ does not seem to have an obvious meaning. From the graph of φ, the equivalent formula $\psi = \exists y \in x.\ \{y\} \neq x$ is synthesized by our algorithm. This reveals that the formula defines the predicate for x having at least two elements.

Conclusion This work indicates that formula synthesis for an assignment of truth values can be learned progressively using only guided exploration as an improvement mechanism. In the future, we consider improving the techniques developed and integrating them in automated theorem provers [5, 13, 12] and in general automation [9, 2, 11, 8] for proof assistants.

References

[1] Wilhelm Ackermann. Die Widerspruchsfreiheit der allgemeinen Mengenlehre. *Mathematische Annalen*, 114(1):305–315, 1937.

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

[3] W. W. Bledsoe and Guohui Feng. Set-var. *Journal of Automated Reasoning*, 11(3):293–314, 1993.

[4] Chad E. Brown. Solving for set variables in higher-order theorem proving. In Andrei Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, volume 2392 of *LNCS*, pages 408–422. Springer, 2002.

[5] Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

[6] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In Maria Paola Bonacina, editor, *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 392–406. Springer, 2013.

[7] Thibault Gauthier. Deep reinforcement learning in HOL4. *CoRR*, abs/1910.11797, 2019.

[8] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In *Certified Programs and Proofs (CPP'15)*, LNCS. Springer, 2015.

[9] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Learning to prove with tactics. *CoRR*, 2018.

[10] Yaqing Jiang, Petros Papapanagiotou, and Jacques D. Fleuriot. Machine learning for inductive theorem proving. In Jacques D. Fleuriot, Dongming Wang, and Jacques Calmet, editors, *Artificial Intelligence and Symbolic Computation - 13th International Conference, AISC 2018, Suzhou, China, September 16-19, 2018, Proceedings*, volume 11110 of *LNCS*, pages 87–103. Springer, 2018.

[11] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014.

[12] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[13] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.