Learning Conjecturing from Scratch

Thibault Gauthier, Josef Urban

Czech Technical University in Prague, Czech Republic

Abstract

We develop a self-learning approach for conjecturing induction predicates on a dataset of 14005 problems derived from the OEIS. These problems are hard for today's SMT and ATP systems because they require a combination of inductive and arithmetical reasoning. Starting from scratch, our approach consists of a feedback loop that iterates between (i) training a neural translator to learn the correspondence between the problems solved so far and the induction predicates useful for them, (ii) using the trained neural system to generate many new induction predicates for the problems, (iii) fast runs of the Z3 prover attempting to prove the problems using the generated predicates, (iv) using heuristics such as predicate size and solution speed on the proved problems to choose the best predicates for the next iteration of training. The algorithm discovers on its own many interesting induction predicates, ultimately solving 3590 problems, compared to 835 problems solved by CVC5, Vampire or Z3 in 60 seconds.

Introduction Proof by induction is a fundamental tool in mathematics, essential for reasoning about inductive structures such as trees, lists or integers. Efficient automation of induction is a key challenge in automated theorem proving (ATP), with direct implications for software verification and mathematical research. Reasoning about integer sequences is a major source of induction problems. The Online Encyclopedia of Integer Sequences (OEIS) [7] catalogs over 350,000 sequences, often with multiple proposed explanations. Verifying the equivalence of these explanations ranges from easy to extremely difficult, with some cases still open conjectures. Our program synthesis AI [5] has generated millions of such explanations, covering over 130,000 OEIS sequences. New algorithms discovered by the system are typically tested on finite prefixes, but ideally, we would like to prove their correctness and also their equivalence to known algorithms. Automated theorem provers (ATPs) capable of this would significantly improve the reliability of such AI-generated algorithms, allowing also their verified optimization.[1] To evaluate the ATP performance on such tasks, the OEIS ATP benchmark [4][2] was introduced. It consists of SMT [1] problems requiring proofs of equivalence between a *small* and a *fast* program for the same OEIS sequence. Baseline results show that current ATPs and SMTs struggle: out of 14,005 problems, the best system (CVC5) solved only 601.

Approach We develop a self-learning approach for synthesizing instances of second-order induction which are useful for solving the OEIS benchmark problems with the `z3` [3] SMT solver. Our general approach is to adapt our AI-based approach for synthesizing OEIS programs to create conjectures to be proven by induction. Such conjectures are referred to as *induction predicates*.

Induction Predicate Language To this end, we first implement a translation from the language of the OEIS programs to the SMT format. We define the language of induction predicates and particularly recursive functions that can be derived from loop constructions in our original language. This way, we translate a program equivalence into an equivalence between recursive functions and conjecture induction predicates (properties) on these functions.

[1] See https://t.ly/qd626 for a discussion of an AI-invented algorithm for $\sqrt[3]{2}$.

[2] https://github.com/ai4reason/oeis-atp-benchmark

Initial Brute-Force Generation In our language for the induction predicates, we develop an *initial brute-force method* for generating a set of sufficiently diverse predicates, suitable for starting the self-learning loop. Here, we introduce several *pruning methods* useful for exploring the large space of our induction predicates. This includes semantic evaluation, selection of only the true predicates, fingerprinting to avoid generation of equivalent predicates, and restricting the language to the most relevant functions.

Evaluation of Predicates We introduce methods for *evaluation, selection and minimization* of the invented predicates using z3. This is an important part of the overall system, which evaluates the quality of our conjectures. A conjecture (or set of conjectures) is *selected* if it is either smaller or makes the proof faster than previous conjectures solving the same problem. Selected conjectures are added to our dataset for training our machine learning model.

Predicate Generation via Self-Learning We then develop our *self-learning loop* which iterates between (i) training a neural machine translation (NMT) model [6] on previously discovered problem/solution pairs, (ii) generating new predicates for all problems, and (iii) evaluating them with z3. This process continuously improves the generation and selection of predicates, which are in turn used for training, after applying data-augmentation methods. We experiment with several long self-learning runs which ultimately prove 3590 of the 14005 problems. This also results in a strong trained NMT+ATP system that proves 3501 problems in at most 48 seconds, and a large dataset of useful induction predicates. For comparison, we develop a strong baseline methods which use manual heuristics for induction. The strongest one proves 2497 problems in 10 seconds, and the union of these baseline methods prove 3285 problems. We show that the union of the manual and self-learning approaches solves 4350 problems, i.e., the feedback loop adds 1065 problems to those solved by manual heuristics.

Example One of the 1065 problems solved only by our AI-based conjecture generation involves the OEIS sequence A26532, defined by $a(0) = 1$ and $a(n+1) =$ if n is even then $2 \times a(n)$ else $3 \times a(n)$. Our system automatically conjectured a small and a fast program for A26532:

$$\overbrace{\prod_{k=1}^{x}(2 + k \bmod 2)}^{small} = \overbrace{6^{\lfloor \frac{x}{2} \rfloor} \times \begin{cases} 1 & \text{if } x \bmod 2 = 0 \\ 3 & \text{otherwise} \end{cases}}^{fast}$$

It then proved the equality between two programs by synthesizing induction predicates. While this can be proved manually using induction, synthesizing and verifying such identities automatically remains a core challenge in automated reasoning.

The code for our project is available at https://github.com/barakeel/oeis-synthesis.

References

[1] Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, Scotland)*, volume 13, page 14, 2010.

[2] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 2005.

[3] Leonardo Mendonça de Moura and Nikolaj S. Bjørner. Z3: an efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[4] Thibault Gauthier, Chad E. Brown, Mikolas Janota, and Josef Urban. A mathematical benchmark for inductive theorem provers. In Ruzica Piskac and Andrei Voronkov, editors, *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023*, volume 94 of *EPiC Series in Computing*, pages 224–237. EasyChair, 2023.

[5] Thibault Gauthier, Miroslav Olsák, and Josef Urban. Alien coding. *Int. J. Approx. Reason.*, 162:109009, 2023.

[6] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[7] Neil J. A. Sloane. The On-Line Encyclopedia of Integer Sequences. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings*, volume 4573 of *Lecture Notes in Computer Science*, page 130. Springer, 2007.

A Examples

We present a selection of example problems found during the self-learning runs but not found when manually adding induction predicates. In more detail, given an induction predicate Q, we always produce a first-order axiom (over $\mathbb{Z}$) by instantiating the following second-order induction axiom with Q:

$$\forall P.\ ((\forall y.\ P(0,y)) \wedge (\forall xy.\ P(x,y) \Rightarrow P(x+1,y))) \Rightarrow (\forall xy.\ 0 \leq x \Rightarrow P(x,y)) \qquad \text{(Ind)}$$

Example 1 (A2411: Pentagonal pyramidal numbers)
This problem is a variation of the famous Gaussian sum.

Problem: $\underbrace{loop(X+Y,X,0)}_{v} \times X = (((X \times X) + X)\ div\ 2) \times X$

Math: $x \times \sum_{k=1}^{x} k = \frac{x \times x + x}{2} \times x$ Solution (predicate): $0 \leq x \wedge x \times x + x = 2 \times v(x)$

The proof requires one predicate on the *loop function* v defining $loop(X+Y,X,0)$. We can also observe that our machine learning algorithm restricted the induction step to nonnegative integers by adding the conjunct $0 \leq x$.

Example 2 (A59826)
This problem is another summation problem that additionally requires a generalization step as a direct induction on the problem equality would fail.

$$\text{Problem: } \mathit{loop}(X+Y+Y, X\times X, 1) = 1+(1+X\times X)\times(X\times X)$$
$$\text{Math: } 1+\sum_{k=1}^{x\times x} 2k = x^4+x^2+1 \quad \text{Solution (predicate): } u(x,1)-1 = x\times x+x \tag{P2}$$

The function u is the helper function for the $\mathit{loop}(X+Y+Y, X\times X, 1)$. Thus, $u(x,y) = y+\sum_{k=1}^{x} 2k$ and $u(x,1) = 1+\sum_{k=1}^{x} 2k = 1+(x\times x+x)$. Our system conjectured the general lemma P2 and indeed proved it by induction. P2 can then be instantiated by $x\times x$ to prove the problem. There is no easy way how to prove the problem directly by induction over n previous steps where n is fixed.

Example 3 (A1026: powers of 17) This problem presents two ways of computing the powers of 17. The fast program w saves time by storing the value of $17 = 1+(2\times(2\times(2+2)))$ into the second component s of *loop2* whereas the small program v recomputes 17 at every iteration of *loop*. The value 17 is computed as $1+(2\times(2\times(2+2)))$ in the fast program whereas in the small program the update function f defining $\mathit{loop}(X\times X, 2, 2)\times X+X$ can be simplified to $16\times X+X = 17\times X$.

$$\text{Problem: } \underbrace{\mathit{loop}(17\times X, X, 1)}_{v} = \underbrace{\mathit{loop2}(X\times Y, Y, X, 1, 17)}_{w,s}$$
$$\text{Math: } v(x) \text{ and } w(x) \text{ are equivalent, where } v(0)=1, v(x+1)=17\times v(x)$$
$$\text{and } (w(0),s(0))=(1,17), (w(x+1),s(x+1))=(w(x)\times s(x), s(x))$$
$$\text{Solution: } v(x)=w(x)\wedge s(x)=v(1)$$

In the solution, the function v corresponds to the outermost *loop* of the left-hand side and the functions w and s correspond to the *loop2* program on the right-hand side. The second conjunct states s is a constant function. This fact is helpful to inductively prove the first conjunct expressing the equality between the two loop functions v and w.

Example 4 (A205646: empty faces in Freij's family of Hansen polytopes)
We first simplify the original problem by evaluating constant programs and reducing polynomials. The problem can then be restated as:

$$\underbrace{\mathit{loop}(3\times X, X, 1)}_{v}+16 = \underbrace{\mathit{loop2}(X\times Y, Y, X \ \mathit{div}\ 2, \mathit{loop}(3, x \ \mathit{mod}\ 2, 1), 9))}_{w,s}+16$$

Mathematically, the equation is $3^x+16 = 3^{(x \bmod 2)}\times 9^{(x \operatorname{div} 2)}+16$. The fast program is saving time by computing 3^x as $3^{(x \bmod 2)}\times 9^{(x \operatorname{div} 2)}$ (this is the basis of the *fast exponentiation* algorithm [2]). Similarly to the previous example, it also stores the value of $9 = 1+2\times(2+2)$ in the second component of the loop y to avoid recomputing it at each iteration of the loop. The solution consists of 6 predicates:

$$w(1+x)=v(1+x)\wedge w(x)=v(x), v(x)=w(x)\wedge w(1+x)=v(1+x)$$
$$x\le w(2), x\le w(x), s(x)=s(1), s(x+1)=s(x)$$

The first two predicates (which are equivalent) allow z3 to prove the statement by doing an induction over the two previous steps, which is necessary and thus z3 can perform case splitting on whether the number of iteration is even or odd. The last two predicates state that s is constant. We are not sure of the purpose of the predicates $x \leq w(2)$ and $x \leq w(x)$. Yet, we know that z3 is not able to find a proof when these predicates are removed because the minimization step would have removed them already.