AITP 2020

Fifth Conference on
Artificial Intelligence and Theorem Proving

Abstracts of the Talks

September 13–19, 2019, Aussois, France

Preface

This volume contains the abstracts of the talks presented at AITP 2020: Fifth Conference on Artificial Intelligence and Theorem Proving held September 13–19, 2020 in Aussois, France and streamed online.

We are organizing AITP because we believe that large-scale semantic processing and strong computer assistance of mathematics and science is our inevitable future. New combinations of AI and reasoning methods and tools deployed over large mathematical and scientific corpora will be instrumental to this task. We hope that the AITP conference will become the forum for discussing how to get there as soon as possible, and the force driving the progress towards that. AITP 2020 consists of several sessions discussing connections between modern AI, ATP, ITP and (formal) mathematics. The sessions are discussion oriented and based on 30 contributed talks.

We would like to thank the CNRS conference center in Aussois for hosting AITP 2020. Many thanks also to Andrei Voronkov and his EasyChair for their support with paper reviewing and proceedings creation. The conference was partly funded from the European Research Council (ERC) under the EU-H2020 projects SMART (no. 714034) and AI4REASON (no. 649043), and the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15003/0000466 and the European Regional Development Fund. Finally, we are grateful to all the speakers, participants and PC members for their interest in discussing and pushing forward these exciting topics!

Septemer 2020

Thomas C. Hales
Cezary Kaliszyk
Ramana Kumar
Stephan Schulz
Josef Urban

Program Committee

Jasmin Christian Blanchette	INRIA Nancy
Ulrich Furbach	University of Koblenz
Tibault Gauthier	Czech Technical University in Prague
Thomas C. Hales	University of Pittsburgh
Sean Holden	University of Cambridge
Mikoláš Janota	University of Lisbon
Cezary Kaliszyk	University of Innsbruck
Michael Kinyon	University of Denver
Peter Koepke	University of Bonn
Michael Kohlhase	FAU Erlangen-Nürnberg
Konstantin Korovin	The University of Manchester
Ramana Kumar	DeepMind
Sarah Loos	Google Research
Stephan Schulz	DHBW Stuttgart
Geoff Sutcliffe	University of Miami
Josef Urban	Czech Technical University in Prague
Sarah Winkler	University of Innsbruck

Additional Reviewers

Adam Pease	Florian Rabe
Dennis Müller	Alexander Bentkamp
Joshua Chen	Mathias Fleury
Alexander Bentkamp	Martin Suda
Yutaka Nagashima	Henryk Michalewski
Chad Brown	Qingxiang Wang
Chad Brown	Zarathustra Goertzel

Table of Contents

Learning alignment between formal & informal mathematics

Kshitij Bansal[1] and Christian Szegedy[2]

[1] Google Research
kbk@google.com
[2] Google Research
szegedy@google.com

1 Introduction

In this talk, we explore the possibility of training an alignment model between informal and formal mathematical corpora in a semi-supervised manner. Though there is a lot of informal mathematics available in natural language (textbooks, papers), the fully formalized and computer checked mathematical content is limited. Availability of alignment information between the two is even further limited. That said, an alignment model between formal and informal mathematics would be essential for the task of autoformalization [3] and could result in dramatically growing the corpus of formalized mathematics. This could open up the possibility for an open-endedly improving system by training proof-guidance and alignment models in lockstep. We look into the currently available resources for bootstrapping such a system, and share our findings.

2 Learning an alignment model

Unsupervised (and weakly-supervised) neural approaches to machine translation relying on learning semantic representations for languages and an alignment model between them have shown great promise (e.g. [4]). We look into various aspects from the point of view of incorporating such ideas for learning an alignment model between formal and informal mathematics.

One of the key aspects is learning semantic representations from large unstructured corpora in a self-supervised manner. On the natural language side (generally, not specifically for mathematics) this is a well-studied area with a lot of progress over the past few years alone [2, 6, 8]. In general, research has established that training current deep neural network based models on proxy-tasks for natural language modeling can be fine-tuned to several downstream tasks such as machine translation, semantic search, sentiment analysis and question answering. Moreover, these tasks did not need as large amounts of data, yet yielded significant gains. For mathematics, on the informal side, there is also significant semantic information in the formulas, equations, diagrams, etc. which would be crucial to leverage for autoformalization work. The availability of large (unlabeled) corpora of informal mathemetics is not necessarily an issue, even if work is required for collecting such datasets for our puprose.

Perhaps less systematically explored and established, nevertheless, various works on theorem proving using neural approaches have looked into learning semantic representations on the formal side. Examples include tasks such as predicting the relevance of premises for proving a statement [1], predicting latent representations of rewrites [5], and labeling a formula with symbols using its structure alone [7]. One can argue that formal mathematical content is even more amenable to unsupervised pretraining as there is a larger number of conceivable self-supervised tasks than in the case of natural language processing. For that, we can leverage

the well-defined graph structure of formulas and the ability to systematically transform them (using, say, rewrite rules and substitutions).

Given the success of unsupervised pretraining on the natural language side and encouraging initial results of semantic embeddings of formal mathematical content, the main task that remains is to train an alignment model between the two sets of embeddings. One key idea is to use cycle consistency [10]. We are especially inspired by its use for learning machine translation models on non-aligned corpora [4]. We propose a similar approach in conjunction with requiring that the translations should utilize similar notions. We explore models that translate natural language text to formal mathematical content (in HOL Light) and vice versa, with several constraints: after back and forth translation the embedding of the resulting statement should stay close to the input in the embedding space; put a loss on the network to enforce that the set of notions referred to by the two translations contain similar notions; and we maximize the probability of the translated sentence looking natural (or being a valid formal sentence). Using these constraints (that is, a combination of the associated losses) we have trained sequence-to-sequence models based on the transformer network [9] with end-to-end backpropagation.

To summarize, using corpora derived from formalization efforts in HOL Light proof assistant on the formal side, we will dicuss the different aspects of the approach:

- sources of datasets,
- language models for informal mathematics including formulas/equations,
- semantic embedding models and discussion of training tasks for formal mathemetics,
- training translation models with the various (cycle consistency, notion-similarity and naturality) requirements,
- neural network architecture choices and
- qualitative evaluation of our first alignment and translation models.

References

[1] Alex A Alemi, Francois Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. Deepmath-deep sequence models for premise selection. *arXiv preprint arXiv:1606.04442*, 2016.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Automating formalization by statistical and semantic parsing of mathematics. In *International Conference on Interactive Theorem Proving*, pages 12–27. Springer, 2017.

[4] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *arXiv preprint arXiv:1711.00043*, 2017.

[5] Dennis Lee, Christian Szegedy, Markus N Rabe, Sarah M Loos, and Kshitij Bansal. Mathematical reasoning in latent space. *arXiv preprint arXiv:1909.11851*, 2019.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[7] Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *arXiv preprint arXiv:1911.12073*, 2019.

[8] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[10] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

Project Proposal: Machine Learning Good Symbol Precedences*

Filip Bártek and Martin Suda

Czech Technical University in Prague, Czech Republic

1 Project Overview

Modern saturation-based Automated Theorem Provers (ATPs) such as E [13] or Vampire [4] rely on the superposition calculus [7] for their underlying inference system. Superposition is built around the paramodulation inference [12] crucially constrained by a simplification ordering on terms, which comes as a parameter of the calculus. Each of the two classes of simplification orderings used in practice, i.e., the Knuth-Bendix Ordering (KBO) [3] and the Lexicographic Path Ordering (LPO) [2], is mainly determined by a pair of symbol precedences—permutations on the predicate and function symbols, respectively.[1] Note that since a symbol precedence only makes sense in the context of a particular problem signature, it is genuinely problem-specific.

The choice of the precedences and thus of the simplification ordering may have a significant impact on how long it takes to solve a given problem. In a well-known example, prioritizing predicates introduced during the Tseitin transformation of an input formula [16] exposes the corresponding literals to resolution inference during early stages of the proof search, with the effect of essentially undoing the transformation and thus threatening with an exponential blow-up [11]. ATPs typically offer a few heuristics for generating the symbol precedences. For example, the successful `invfreq` heuristic in E orders the symbols by the number of occurrences in the input problem, prioritizing symbols that occur the least often for early inferences. Experiments with random precedences have shown that the existing heuristics often fail to come close to the optimum precedence [10], revealing there is a large potential for further improvements.

The ultimate goal of this project is to implement a system that, when presented with a First-Order Logic (FOL) problem, proposes symbol precedences that will likely lead to solving the problem quickly. We plan to use the techniques of supervised learning and extract such theorem-proving knowledge from successful (and unsuccessful) runs of the Vampire theorem prover when run over a variety of FOL problems equipped by randomly sampled symbol precedences. Our basic assumption is that there are abstract properties of signature symbols[2] which the learned model can utilize to determine their placement in good (from the perspective of the proving process) precedences. Moreover, we assume that by succeeding to solve already solvable problems fast, the learned knowledge will generalize to solving problems previously out of reach. This general assumption is shared with other projects for learning good theorem proving strategies from previous experience, such as the MaLeS system [6].

Note that while the domain of each precedence depends on the problem signature, we still aim to generalize the model across a wide range of FOL problems, learning theorem proving knowledge in a signature agnostic way. To this aim we are planning to use Graph Neural Networks (GNNs) [18] to extract abstract representations (feature embeddings) of the symbols

*Supported by the ERC Consolidator grant AI4REASON no. 649043 under the EU-H2020 programme.

[1] KBO is further parameterized by symbol weights, but our reference implementation in Vampire [4] uses for efficiency reasons only weights equal to 1 [5] and so we do not consider varying this parameter here.

[2] Such as the number the occurrences used by `invfreq` mentioned above.

while learning to distinguish good precedences from bad ones. The viability of using GNNs for learning in related theorem proving tasks has recently been established [17, 14, 9, 8] and it has been shown that a lot of problem structure can be captured in a way that does not hard-code symbol names into the model. This makes GNNs an ideal target architecture also for our work.

2 Problem-wise Predicate Precedence Learning

While keeping finer details of the overall architecture open for now, we start by focusing on learning good precedences for each problem in isolation. This preliminary task allows us to establish how much "signal to learn from" we can expect to be available. Moreover, learning from permutations presents already several interesting challenges to deal with. For one, there is a priori no obvious way how to characterise a permutation by real-valued features to serve as inputs for a learning algorithm, or how to do it in a way which does not presuppose a fixed domain size. Moreover, even with a regression model ready to predict the prover's performance under a particular precedence Π, we still need solve the task of finding the ideally optimal precedence Π^* according to this model.

2.1 Initial Experiment

We based our initial experiment on the assumption that each pair of predicates (p_1, p_2) contributes to the performance a precedence that orders p_1 before p_2 independently of the ordering of the other predicates in the precedence. This is in accord with how a typical human designed heuristics would be justified: by declaring that a certain class of symbols should be larger or smaller than others. Under this assumption we can aggregate the values from all of the up to $n!$ precedence-wise performance measurements (where n is the number of predicates in the problem under consideration) into $n(n-1)$ pairwise predicate *preference* values.

In our first implementation, we choose the preference value of a pair of predicates (p_1, p_2) to be an empirical estimate of the expected number of saturation loop iterations of a successful Vampire run that orders p_1 before p_2. We obtain this value by running Vampire with 1000 uniformly random predicate precedences for each of the training problems, attributing a high constant value to the runs that time out. The number of saturation loop iterations in a successful run is generally a good measure of the quality of a precedence, because if we reduce the number of saturation loop iterations for a wide variety of problems, we are likely to solve previously unsolved problems.

Once we have access to good pairwise preference values $\mathit{pref}(p_1, p_2)$, we proceed to construct a predicate precedence Π^* that approximately optimizes the cumulative preference value using a greedy algorithm proposed by Cohen et al. [1]. Essentially, the algorithm always looks for a predicate p such that $\sum_{p' \in \mathit{Remaining}} \mathit{pref}(p, p')$ is the smallest/largest and moves such p from *Remaining* to the next position in the (from left to right) constructed precedence.

To summarize the initial experiment design, we evaluate the feasibility of precedence learning from pairwise preference values, trying to answer the following question:

> To what extent is it possible to construct a good predicate precedence if we know a good preference value for each pair of predicate symbols?

In the talk, we will present encouraging experimental results showing that when using the averaged pairwise preference values the greedy algorithm from Cohen et al. [1] reaches or surpasses the performance of Vampire's equivalent of E's `invfreq` heuristic on 85 out of 113 problems

from TPTP [15] (we selected problems that exhibit an especially high variation in number of saturation loop iterations), reducing the number of saturation loop iterations by an average (geometric mean) factor of 0.47.

References

[1] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *CoRR*, abs/1105.5464, 2011. URL http://arxiv.org/abs/1105.5464.

[2] Samuel N. Kamin and Jacques Lévy. Two generalizations of the recursive path ordering. 1980.

[3] D. E. Knuth and P. B. Bendix. *Simple Word Problems in Universal Algebras*, pages 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_23. URL https://doi.org/10.1007/978-3-642-81955-1_23.

[4] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 1–35, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39799-8. URL http://www.vprover.org/cav2013.pdf.

[5] Laura Kovács, Georg Moser, and Andrei Voronkov. On transfinite knuth-bendix orders. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 384–399. Springer, 2011. ISBN 978-3-642-22437-9. doi: 10.1007/978-3-642-22438-6_29. URL https://doi.org/10.1007/978-3-642-22438-6_29.

[6] Daniel Kühlwein and Josef Urban. Males: A framework for automatic tuning of automated theorem provers. *J. Autom. Reasoning*, 55(2):91–116, 2015. doi: 10.1007/s10817-015-9329-1. URL https://doi.org/10.1007/s10817-015-9329-1.

[7] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 371–443. 2001.

[8] Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *CoRR*, 2019.

[9] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *CoRR*, abs/1905.10006, 2019. URL http://arxiv.org/abs/1905.10006.

[10] Giles Reger and Martin Suda. Measuring progress to predict success: Can a good proof strategy be evolved? In *AITP 2017*, 2017.

[11] Giles Reger, Martin Suda, and Andrei Voronkov. New techniques in clausal form generation. In Christoph Benzm\"uller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016. doi: 10.29007/dzfz. URL https://easychair.org/publications/paper/XncX.

[12] G. Robinson and L. Wos. Paramodulation and theorem-proving in first-order theories with equality. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 298–313, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_19. URL https://doi.org/10.1007/978-3-642-81955-1_19.

[13] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

[14] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 336–353. Springer, 2019. ISBN 978-3-030-24257-2. doi: 10.1007/978-3-030-24258-9_24. URL https://doi.org/10.1007/978-3-030-24258-9_24.

[15] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[16] G. S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pages 466–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. ISBN 978-3-642-81955-1. doi: 10.1007/978-3-642-81955-1_28. URL https://doi.org/10.1007/978-3-642-81955-1_28.

[17] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2786–2796, 2017. URL http://papers.nips.cc/paper/6871-premise-selection-for-theorem-proving-by-deep-graph-embedding.

[18] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks, 2019.

Project Proposal: Relieving User Effort for the Auto Tactic in Coq with Machine Learning

Lasse Blaauwbroek*

Czech Institute for Informatics, Robotics and Cybernetics, Czech Republic
Radboud University Nijmegen, the Netherlands
lasse@blaauwbroek.eu

We propose to enhance the `auto` tactic in Coq with machine learning, aiming to reduce the effort the user has to put in designing hint databases for `auto`. We seek ideas and advice regarding the specific type of ML that would be appropriate in this context.

Proof Styles in Coq The Coq Proof Assistant supports many methods of proving a theorem. One can either directly write proof terms, or choose one of the tactical languages like Ltac [2] or Mtac [3]. Then there are custom sets of tactics for Ltac like SSReflect [5]. However, even within one of these paradigms there are still different styles of proving available. Some people advocate structured proofs using Coq's built-in bullet points, writing every step of the proof explicitly in hopes of increasing readability. Other people try to write very compact and tailored tactic scripts that prove a lemma in one step. This usually results in shorter and easier to maintain proofs, often at the cost of readability. All of these styles have their place, depending on the mathematical domain one is trying to formalize.

In this proposal we will focus on one specific proving style described and popularized by Adam Chlipala [1]. The concept is to provide as little proof information as possible within the tactic script of a lemma. Usually this means that one critical step of the proof is explicitly stated in the script, while the rest of the proof is pieced together by automation. For example, the critical step can be to use induction on a specific variable of the lemma. The resulting cases of the induction principle then have to be solved by the built-in `auto` tactic. This tactic is a generic prover that uses hints previously provided by the user to guide proof search. These hints usually consist of a recipe on how to use a previously declared sublemma of the proof. However, it is also possible to teach the `auto` tactic how and when to use custom tactics and complete decision procedures.

This approach has two main advantages. (1) It keeps the actual proof scripts short and therefore maintainable. If something in the development changes it should be easy to go through the development and fix the hints and proofs, if necessary at all. (2) By only using the `auto` tactic the user is forced to tease out important information about the proof and refactor this into a lemma, hint or tactical procedure. In this way, all the crucial steps will be explicitly declared and can be easily understood by readers without bogging them down with the straightforward details of the proof. The truth of a lemma would ideally be evident to a reader simply by thinking about previously provided hints for a bit, just like it is to Coq.

The `auto` Tactic Fundamentally, the auto tactic is a simple search procedure. For a proof state it can compile a list of possible actions to take, together with a priority for these actions. The resulting search space is traversed in BFS or DFS fashion until either a full proof is found or a limit is reached. The interesting part is that the list of possible actions is compiled from so-called hint-databases. These databases are meant to contain usage information for lemmas and tactics in the current development. Users can add and remove information from a database on the fly by using variations of the `Hint` vernacular. We give some examples that add hints to a database.

- `Hint Resolve thmx` will tell `auto` to try and unify the current goal with the conclusion of theorem `thmx`. On success, auto will replace the current goal with the assumptions of `thmx`.

*This work was supported by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466)

- Let `thmy` be a theorem that has an equality as its conclusion. `Hint Rewrite -> thmy` will tell auto to rewrite the goal using `thmy` if the goal unifies with the left-hand side of the equality. Any possible assumptions of `thmy` are added to the proof state.
- `Hint Extern tac` can be used to register any tactic `tac` to be run by auto. This can be useful for making `auto` do things like normalize terms or other simple steps that never go wrong. Also, since this vernacular gives us access to the full power of the tactical language, it allows us to encode much more complicated hints, as we will elaborate below.

Hint databases have to be designed with great care. Adding the wrong lemmas to a database can lead to a very large search space and even infinite loops. The larger and more complicated a development is, the more problematic this becomes. To reduce the branching factor in such developments, a hint can have a gate specifying the conditions that need to be met before the hint is used. In its simplest form, this can be a pattern that must be matched to the goal before the hint applies. It is, however, possible to write arbitrarily complicated gates using Ltac as a programming language. This way, the hint can be accepted or rejected based on the full contents of the proof state. Philosophically speaking, the goal of writing a gate is to capture the domain specific knowledge and intuition that the user has on how and when to use a lemma or tactic. A simple example is a gate for the lemma $a < b \to b < c \to a < c$. We want to apply this lemma to a goal $x < z$ only if we can expect to find a suitable y. Therefore, the gate will be a pattern on the proof state: `?x < ?y,...,?y < ?z` $\vdash$ `?x < ?z`. Note that this gate is very strict, and a much more complicated one might be required in practice.

Experience tells us that for a decently sized development, the branching factor of the search performed by `auto` has to be kept well below 1.5 to keep the system usable.[1] The gating required to reach this can be quite laborious. Conversely, it tends to not be very difficult to achieve a branching factor smaller than five. Our proposal is to bridge the gap between these factors using machine learning, bringing together the best of human intuition and the computers ability to do the grunt work.

Machine Learning for `auto` Our proposal to incorporate machine learning into auto consists of gathering information on previous runs of the auto tactic. The idea is that at the beginning of a development, hints and proofs are usually much simpler, allowing auto to find a proof easily. We can then record which hints ended up being fruitful in the context of which proof state. As the development progresses, the system can then start to leverage this information to prioritize the list of available actions to `auto` in a proof state. Actions will be more important if they have been used previously in similar states. The amount of actions the machine learning has to choose from will be quite limited because the gating of the user has already weeded out most inapplicable actions.

One fundamental challenge is that the system will not have a lot of data to learn from. This is because within a development a hint associated with a lemma would normally be used tens or at most hundreds of times. The system needs to learn quickly in terms of data. On the other hand, because there will be very few choices to be made at each point, there will be quite a lot of time to consider each choice. For these reasons, most traditional learning techniques, like neural networks, will not be immediately applicable. The simplest approach is to extract features from proof states, and perform a direct comparison with previous states. However, more symbolic methods such as approximate substring matching between goal and lemma may also be applicable [4]. During AITP we would like to gather feedback about other techniques that may suit this setting.

[1] This is partially due to the fact that Coq users generally do not, can not, and often do not want to replace the auto tactic with the found solution like is common in Isabelle. Therefore, to get a good experience, the search has to be completed within seconds.

References

[1] Adam Chlipala. *Certified Programming with Dependent Types - A Pragmatic Introduction to the Coq Proof Assistant*. MIT Press, 2013.

[2] David Delahaye. A tactic language for the system coq. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000.

[3] Jan-Oliver Kaiser, Beta Ziliani, Robbert Krebbers, Yann Régis-Gianas, and Derek Dreyer. Mtac2: typed tactics for backward reasoning in coq. *PACMPL*, 2(ICFP):78:1–78:31, 2018.

[4] Jiaying Wang, Xiaochun Yang, Bin Wang, and Chengfei Liu. An adaptive approach of approximate substring matching. In *Database Systems for Advanced Applications - 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part I*, pages 501–516, 2016.

[5] Iain Whiteside, David Aspinall, and Gudmund Grov. An essence of ssreflect. In *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*, pages 186–201, 2012.

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

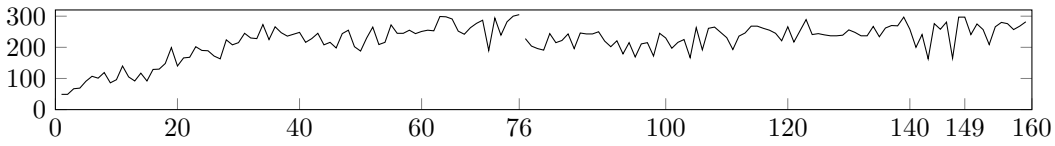


Figure 1: Number of successful formula synthesis (y) at generation (x)

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.

Learning to Advise an Equational Prover*

Chad E. Brown[1], Bartosz Piotrowski[1,2], and Josef Urban[1]

[1] Czech Technical University, Prague,
[2] University of Warsaw, Poland

We describe a simple first-order equational theorem prover, `aimleap`, capable of interacting with an advisor. The prover takes 87 (fixed) quantified unit equations as axioms (all of which are true in AIM loops [4]). The conjecture is then given as an equation and `aimleap` attempts to find a proof by applying paramodulation using one of the axioms to one term occurrence in either the left or right side of the current goal. An external advisor can be used to filter and rank possible paramodulation steps. We report results of using an advisor trained using XGBoost [1] on data described below.

Proof Search We briefly describe the way `aimleap` searches for a proof of $s = t$. Note that s and t may contain constants and variables, where variables are allowed to be instantiated. Let $\mathcal{A}$ be a set of known equations. The search is limited by a bound n (maximum allowed distance) which is now always initialized to 10. We also use an abstract time limit (number of proof search inferences during) a that is set to 100 by default.

1. If s and t are unifiable, then report success.
2. If $n = 0$, then report failure.
3. Compute a finite set of *paramodulants* $s_i = t_i$. These are defined as rewrites of $s = t$ by a single equation from $\mathcal{A}$. Choose only one representative for each equivalence class of paramodulants w.r.t. renaming of variables.
4. Order these paramodulants using an advisor, filtering out those which the advisor deems to require more than $n - 1$ paramodulation steps to complete the proof, and for each one ask if $s_i = t_i$ is provable in $n - 1$ steps. I.e., this is a simple best-first search.

Primary Dataset Veroff has obtained a large number of AIM proofs using Prover9 [5]. Analysing some of these proofs it was possible to obtain 3468 equations provable from the 87 axioms within 2-10 paramodulation steps. This gave us our initial data for training and testing. Roughly half of the examples, 47.3%, was the result of 2 paramodulation steps. Another 25% resulted from 3 paramodulation steps and 10.2% resulted from 4 paramodulation steps. The remaining 18.5% resulted from 5-10 paramodulation steps, with fewer examples as the number of steps increased. We later augmented the training data with roughly 10,000 synthetically generated data, while still restricting testing to the original 3468 equations.

Rote Learner As a sanity test, an "oracle" advisor was used first. This advisor returns the known distance[1] for goals and subgoals that were seen in the training proofs. For other equations it returned a high distance of 50 (essentially forcing these equations to be pruned from the paramodulant options). We call this the "rote learning" (memorizing) advisor. As expected, the prover could reprove all 3468 problems given the full memorized information. In some cases, shorter proofs than the training proofs were found.[2] In 132 cases, new proofs involving only one paramodulation step were found. The rote learner also gave us a way to create negative training data by recording each (redundant) step where the rote learner returned 50. We have also split the 3468 problems into ten parts and used a rote learner that is allowed to look up only the values from the other nine parts. `aimleap` was only able to prove 800 (21.9%) problems in this cross-validation scenario.

*Supported by the ERC Consolidator grant no. 649043 AI4REASON

[1] Note that the *known distance* may be an over-approximation of the *real (minimal) distance*.

[2] Even though we only used the memorized data, they may still lead to alternative proofs.

Constant Distance We also tested an advisor that always returns a constant distance $d \in \{1, ..., 9\}$ whenever the left and right hand sides of the current goal are not equal (and 0 if equal). When $d = 9$, `aimleap` considers all possible 1 step proofs and some 2 step proofs,[3] approximating breadth-first search. Since just over half the problems actually do have a 1 or 2 step proof, this can solve 1739 problems (50.1%). When $d < 9$, the maximum number of problems solved was 138 (4%). Most of the search is then spent in greater depths and the abstract time of 100 runs out before trying many alternatives that may lead to a short proof.

Training the Advisor For providing machine-learned advice we used XGBoost. Training examples from the data set were fed into the model as features of pairs of terms and the corresponding distance between them. We used ENIGMA [2]-style features, i.e., paths of lengths 1-3 from the parse tree, with numbers of their occurrences. The more significant hyperparameters of XGBoost we used were as follows: objective function – mean squared error, number of boosting rounds – 1000, maximal depth of a decision tree – 10, learning rate – 0.1.

Accuracy of the Advisor We trained the model and measured its performance using the cross-validation split. The model on testing parts achieved average root mean square error (RMSE) 1.1 and average accuracy of 0.59. This means that the model with our features is able to estimate the distances between terms reasonably well. This in turn suggests its usability as an advisor for the prover.

Search Results using the Advisor Again, we run evaluation respecting the cross-validation split. The prover was given time limit of 60 s. Using the advice from the trained XGBoost models (one per cross-validation partition) we obtained proofs of 299 problems, which is 9% out of all 3468. This number is optimistic in a sense of being better than the average number of problems solved with the constant advice (but not better when $d = 9$). Because the interaction with the advisor slows the prover significantly, we believe that improvement of the implementation can make the proving performance considerably better. What is worth noting though is the number of problems solved with the advisor which were not solved by the rote learner – 135. There are also 18 problems solved with the advisor which were not solved with any d.

First-Order Automated Provers For further comparison we gave the problems to three automated provers with a timeout of 60 s: Prover9 [5], E [7] and Waldmeister [3]. E proved 2684 problems (77.4%), Waldmeister 2170 problems (62.6%) and Prover9 2037 problems (58.7%). The number of problems which were solved by `aimleap` with the machine-learned advice and not solved by these provers is 113, 92 and 49 for E, Prover9 and Waldmeister, respectively.

Conclusion and Future Work We have described an equational dataset based on the AIM project and an equational prover (`aimleap`) capable of interacting with an advisor. This provides a framework for testing the degree to which an advisor (typically one trained using machine learning techniques) is helpful during proof search. With perfect advice all the problems are easily provable by `aimleap`, however they are not all easy for standard ATPs. The current results for machine-learned advice, although preliminary, seem to be interesting and provide a baseline for further experimentation. One direction we will be investigating is using neural networks for *conjecturing* the intermediate steps. Sequence-to-sequence neural models have recently shown interesting performance in related tasks, such as predicting literals in Tableaux proofs [6], or even constructing terms from symbols. `aimleap` is already prepared for this mode of interaction.

[3] This is because the abstract time is 100 and the number of equations only 87. Deeper proof searches are pruned as soon as $d > n$.

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 785–794. ACM, 2016.

[2] Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2019.

[3] T. Hillenbrand, A. Jaeger, and B. Löchner. Waldmeister - Improvements in Performance and Ease of Use. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction*, number 1632 in Lecture Notes in Artificial Intelligence, pages 232–236. Springer-Verlag, 1999.

[4] Michael K. Kinyon, Robert Veroff, and Petr Vojtechovský. Loops with abelian inner mapping groups: An application of automated deduction. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 151–164. Springer, 2013.

[5] W. McCune. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[6] Bartosz Piotrowski and Josef Urban. Guiding theorem proving by recurrent neural networks. *CoRR*, abs/1905.07961, 2019.

[7] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

From proofs to theorems*

Karel Chvalovský and Josef Urban

Czech Technical University in Prague, Czech republic,
karel@chvalovsky.cz and josef.urban@gmail.com

In automated theorem proving (ATP) the essential task is, not surprisingly, to produce a proof for a given theorem. However, for human mathematicians such a task usually involves also producing various conjectures that are proved, refuted, or more likely modified and which help to clarify the problem in question. Clearly, to mimic such an approach in ATP is a very challenging task. Moreover, the approaches proposed for computer generated conjecturing have produced mostly toy or domain specific conjectures, see e.g. [9, 5, 4, 8, 6].

One of the core problems is to even decide whether a produced conjecture is fruitful. This clearly depends on the particular task for which we want to use the conjectures. An activity usually rich on generating conjectures is reading mathematical texts. For example, the reader may anticipate the flow of the paper by guessing the next theorem based on the previous text. This seems to be an interesting machine learning task that requires a non-trivial understanding of a mathematical text. Another reading of this task is given a proof attempt what is a useful lemma that helps to complete the proof. Hence the task is hard and it is probably better to start with a related and more approachable problem namely the inverse task to what automated theorem provers do—to produce a theorem given a proof. Clearly, even this can be, especially without a proper context of the rest of the paper and for informal proofs, an extremely hard to impossible task. Nevertheless, at least there are data available for learning. Hence we are here interested in the problem of transforming a proof into a corresponding theorem.

It is much easier if we use a formalized mathematical library, in many cases it can be even trivial to produce such a transformation. However, usually it requires some, at least statistical, insight. For example, take the following tokenized[1] proof from Mizar Mathematical Library [2] (MML, contains over 50K theorems)

```
proof let L , M be non empty RelStr such that A1 : L , M are_isomorphic and A2 : L is
reflexive ; let x be Element of M ; M , L are_isomorphic by A1 , WAYBEL_1 : 6 ; then
consider f being Function of M , L such that A3 : f is isomorphic ; reconsider
fx = f . x as Element of L ; fx <= fx by A2 ; hence thesis by A3 , WAYBEL_0 : 66 ; end ;
```

as an input. The corresponding theorem is

```
theorem for L , M being non empty RelStr st L , M are_isomorphic & L is reflexive holds
M is reflexive
```

Our system is able to correctly produce this theorem. Although it is easy to extract the assumptions from the proof, in this particular case, a bit of work is required to statistically infer that we want to know that `M is reflexive` holds, because this fact does not occur explicitly in the proof. Moreover, the system provides the correct output only as its third option with `for x being Element of M holds x is reflexive` being the top candidate.

A preliminary version of our very simple system is based on a popular neural machine translation toolkit OpenNMT-py and basically follows an approach [7] developed for text summarization, because we can loosely speaking understand our task as a summarization task.

*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.

[1] We tokenize all the inputs and outputs to make the task better suited to our tools.

The model we employ is based on the sequence to sequence approach using a variant [1] of the attention mechanism and importantly it is able to copy words directly from the input to the output. Hence it can handle even words that it did not see during the training phase. We also experimented with the Transformer [10] model, but the results have been slightly worse. However, it is well known that this model is sensitive to the right choice of hyperparameters and after tuning them it is likely to outperform the former model.

However, we should emphasize that although natural language processing (NLP) tools have proven to be useful, they still suffer from several problems. Among them is the problem that sentences produced by such systems are in many cases logically inconsistent. Hence it may seem a bit silly to use the exactly same approach to produce mathematical theorems. However, as our task boils down basically to extracting correct sub-sequences from a proof and adding a bit of statistically plausible knowledge, it seems powerful enough for our purposes. Moreover, we do not claim that such a simple approach should provide surprisingly complicated results, but it has been experimentally shown [11] that NMT can be used to produce statistically plausible results for MML. Hence it is not so surprising that for MML we get decent results: on a test set the success rate, which means that we produce an exact match, is 0.28 (0.39 if compared against the ten most probable outputs).

A clearly challenging task is to test a similar approach on LaTeX documents from arXiv.org, where it is in many cases easy to identify theorems and proofs, but the format of proofs vary widely. We have performed a few preliminary experiments using the Stacks project, which provides a curated and coherent playground suitable for our purposes. Not surprisingly, our simple approach produces very poor results in this context. A relatively, for our purposes, small size of the dataset (ca. 12K theorems) may contribute to this, but more likely the main problem is that in such a general setting the task is no longer about selecting the right sub-sequences and guessing a statistically plausible conclusion. It requires at least a superficial understating of the problem in its entirety. Moreover, it would be helpful to take a bit more of context into account, for example, use the previous theorems as a part of the input. In fact, it helps slightly to see the previous theorems, because they provide additional sources of data.[2] Although it is possible to modify our task in many such ways, it is unlikely that such modifications will be sufficient to produce reasonable results on informal texts.

We believe that producing conjectures based on a mathematical text is an important task. And although narrowing it down to producing theorems from proofs looks much less interesting, it is a task that connects both these notions in a non-proof theoretical way, a potentially useful viewpoint on its own. Moreover, the attention mechanism makes it possible to weight the contribution of tokens in the input and use this knowledge elsewhere, for example, for fingerprinting mathematical object, cf. [3].

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred

[2] However, it is important to carefully split our dataset into training, validation, and testing sets. Clearly, the results are much "better" if an output theorem in the test set appears (literally) also among the previous theorems in the training set.

Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015.

[3] Sara C. Billey and Bridget E. Tenner. Fingerprint databases for theorems. *Notices Amer. Math. Soc.*, 60(8):1034–1039, 2013.

[4] Simon Colton. *Automated Theory Formation in Pure Mathematics*. Distinguished Dissertations. Springer London, 2012.

[5] Siemion Fajtlowicz. On conjectures of Graffiti. *Annals of Discrete Mathematics*, 72(1–3):113–118, 1988.

[6] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Initial experiments with statistical conjecturing over large formal corpora. In Andrea Kohlhase, Paul Libbrecht, Bruce R. Miller, Adam Naumowicz, Walther Neuper, Pedro Quaresma, Frank Wm. Tompa, and Martin Suda, editors, *Joint Proceedings of the FM4M, MathUI, and ThEdu Workshops, Doctoral Program, and Work in Progress at the Conference on Intelligent Computer Mathematics 2016 co-located with the 9th Conference on Intelligent Computer Mathematics (CICM 2016), Bialystok, Poland, July 25-29, 2016.*, volume 1785 of *CEUR Workshop Proceedings*, pages 219–228. CEUR-WS.org, 2016.

[7] Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, 2018.

[8] Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory exploration in a proof assistant. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2014.

[9] Douglas Bruce Lenat. *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*. PhD thesis, Stanford, 1976.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[11] 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.

Solving Arithmetic Problems on a Checkered Paper*

Adrián Csiszárik[1], Beatrix Benkő[2], Gergely Stomfai[1], and Milán Vásárhelyi[1]

[1] Alfréd Rényi Institute of Mathematics
[2] Eötvös Loránd University

1 Introduction

> In [3] Clark and Chalmers highlight a conversation taken verbatim from Gleik's (1992) biography of the famous scientist, Richard Feynman and the historian, Charles Weiner. In this conversation, Weiner mentions that Feynman's notes are "a record of the day-to-day work". Feynman retorts: "I actually did the work on paper". Weiner then suggests: "The work was done in your head, but the record of it is still here" to which Feynman counters: "No, it's not a record, not really. It's working. You have to work on paper and this is the paper. Okay?".

Our ongoing project explores how simple arithmetic reasoning tasks can be carried out by an artificial neural network that operates on a checkered paper, without utilizing any external prover. Our general setup can be formulated with the notions of reinforcement learning as follows.

- The environment is a checkered paper represented by an $N \times M$ grid, with at most one symbol in each cell from a fixed finite symbol set S.
- The starting state is a paper with an arithmetic problem written on it with the symbol set S. (E.g., with `"-6.1213 + 2543.073?"` or `"44342.23412 * -534.24?"` written on the first line of the paper.)
- The action the agent can take in each step is to write a symbol in a cell.
- The agent receives a reward when the correct solution is written on the paper (again with the symbol set S) at a prescribed position, followed by a special symbol ■ to "hand in" the paper for evaluation.

Why simple arithmetic problems? Harnessing artificial neural networks to solve arithmetic problems is a long pursued goal for the field at large [2, 4, 5, 6, 7]. The motivation behind this ambition is the idea that solving tasks of elementary school difficulty with artificial neural networks could bring us closer to understanding how one can approach the general goal of human-level intelligent behavior with machines. The subject of elementary school mathematics serves as a particularly suitable testbed in this regard: the necessary theory required to resolve such problems is narrow, one has exact solutions with solid reasoning steps, and perhaps most importantly, one can easily generate synthetic datasets to train the models. Despite the ubiquitousness of such tasks, handling arithmetic problems utilizing neural networks still seem to be challenging. (See, e.g., [7], where utilizing LSTMs or even a powerful Transformer model still fails to solve several seemingly easy tasks of e.g., multiplication. State-of-the-art learning systems utilizing theorem provers struggle with these tasks, [8] provides several experimental results.)

*With an artificial neural network, but without an external prover.

Why on a checkered paper? The starting point of our project is considering the semiotic perspective implicitly appearing in our motto behind the words of Feynman. Indeed, utilizing a (checkered) paper for solving arithmetic problems is a rather natural approach for humans. A human being mechanically executes straightforward steps of algorithms engrained in his mind, which have been learned and memorized over several years in elementary school. In the process, the arrangement of symbols in a spatial structure plays a key role. Our working hypothesis for this project is that more complex mathematical reasoning also relies on the same fundamental approach of pattern matching (of course with more convoluted notions).

2 Preliminary results and project plan

Our preliminary experiments demonstrated that utilizing a supervised training scheme with a simple attention augmented convolutional network [1] performs surprisingly well on the task of predicting the next symbol to be written on the paper by an algorithm that is utilized by humans to solve that problem. (E.g., predicting the next symbol to be written on the paper when utilizing the classic multiplication algorithm taught for children in elementary school.)

Based on this, our goal is to solve various simple mathematical problems by learning to carry out step-by-step reasoning procedures with our proposed approach. In particular, as the next step, we plan to implement step-by-step solutions for the problem set presented in [7].

In our talk, we will present the general idea, the above mentioned preliminary results, and report our progress on how far we can reach with this approach.

References

[1] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention augmented convolutional networks. *CoRR*, abs/1904.09925, 2019.

[2] Kaiyu Chen, Yihan Dong, Xipeng Qiu, and Zitian Chen. Neural arithmetic expression calculator. *ArXiv*, abs/1809.08590, 2018.

[3] Andy Clark and David Chalmers. The extended mind. *Analysis*, 58(1):7–19, 1998.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. In *NIPS*, 2015.

[6] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[7] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.

[8] Zsolt Zombori, Adrián Csiszárik, Henryk Michalewski, Cezary Kaliszyk, and Josef Urban. Towards finding longer proofs. *CoRR*, abs/1905.13100, 2019.

Computer-assisted identification of splittings in subvariety lattices *

Wesley Fussner

Laboratoire J.A. Dieudonné, CNRS, and Université Côte d'Azur
Nice, France
wfussner@unice.fr

Abstract

We describe a heuristic approach to identifying splittings in some lattices of subvarieties using computer-assisted methods, in particular McCune's Prover9/Mace4. This approach shows promise to facilitate the analysis of subvariety lattices for many classes of algebraic structures, in particular in situations where the large size of the algebras involved in the splittings make human-executed proofs infeasible.

Equational reasoning can often be fruitfully analyzed from a semantic perspective by considering *varieties*—i.e., classes of algebraic structures satisfying some set of equations, or equivalently that are closed under taking subalgebras, direct products, and homomorphic images. Indeed, given an equational theory $\mathcal{E}$ and the the variety $\mathcal{V}$ of algebras modeling $\mathcal{E}$, there is a bijective correspondence between equational extensions of $\mathcal{E}$ and varieties contained in $\mathcal{V}$. Ordered under inclusion, the varieties contained in $\mathcal{V}$ form a complete lattice. This subvariety lattice of $\mathcal{V}$ completely encodes equational inference in the presence of $\mathcal{E}$ due to the previously mentioned correspondence, and thus understanding the structure of the subvariety lattice is crucial. Because a complete description of the subvariety lattice of a given variety $\mathcal{V}$ is usually not possible, its structure is often illuminated in terms of its *splittings*. These consists of pairs $(\mathcal{W}_1, \mathcal{W}_2)$ of subvarieties of $\mathcal{V}$ such that for each subvariety $\mathcal{W}$ of $\mathcal{V}$, $\mathcal{W} \subseteq \mathcal{W}_1$ if and only if $\mathcal{W}_2 \not\subseteq \mathcal{W}$. The splittings of a subvariety lattice provide ways of partitioning it, and have been studied extensively both in general (see, e.g., [1]) and for certain important varieties of algebras (see, e.g., [4]).

This work presents a case study in identifying such splittings using a computer-assisted approach. Our case study illustrates a heuristic method of finding splittings in suitably-chosen subvariety lattices, and relies on a human-guided computer search. This approach facilitates the understanding of subvariety lattices in situations where human-executed approaches are rendered infeasible by the size of the structures involved, necessitating the use of computational resources. In particular, we execute our heuristic approach using McCune's Prover9/Mace4 [6] to identify some important splittings in the subvariety lattice of *involutive lattices*. The latter comprise a class of lattice-ordered algebraic structures that contain, *inter alia*, ortholattices and Boolean algebras. We focus on the Kleene identity

$$x \wedge \neg x \leq y \vee \neg y, \tag{K}$$

which plays an extremely important role in the theory of distributive involutive lattices (see, e.g., [5, 7, 2, 3]). We characterize the failure of (K) in arbitrary (not necessarily distributive) involutive lattices by the presence of six forbidden configurations $\mathbf{F}_i$, $i \in \{4, 5, 6, 8, 10, 12\}$.

*This project received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program (grant agreement No. 670624).

These forbidden configurations are found by guided use of Mace4 to construct countermodels witnessing each manner in which (K) may fail. First, Mace4 is asked to produce small countermodels witnessing the failure of (K). After appropriate candidates for forbidden configurations are identified, we scrutinize these for salient features (such as the presence of an involution fixed point, distributivity, or special equations satisfied by generators). This yields a set of conditions $\Sigma_{\mathbf{F}}$, which we conjecture guarantees the presence of the candidate forbidden configuration $\mathbf{F}$ as a subalgebra in any involutive lattice $\mathbf{L}$ refuting (K) and satisfying $\Sigma_{\mathbf{F}}$. We then query Mace4 for countermodels of (K) that refute some conditions in $\Sigma_{\mathbf{F}}$. This process is then iterated.

Although there is no reason *a priori* why this process must terminate in general, for involutive lattices we obtain six involutive lattices $\mathbf{F}_i$, $i \in \{4, 5, 6, 8, 10, 12\}$, together with six jointly-exhaustive sets of conditions Σ_i, $i \in \{4, 5, 6, 8, 10, 12\}$ (where i identifies the cardinality of the involutive lattice $\mathbf{F}_i$). In order to prove that these forbidden configurations suffice to characterize the failure of (K), for each i we examine countermodels $\mathbf{L}$ of (K) containing $\mathbf{F}_i$ in order to understand generators of $\mathbf{F}_i$ in $\mathbf{L}$. Specifically, given an involutive lattice $\mathbf{L}$ containing $\mathbf{F}_i$ and a pair of elements $a, b \in L$ witnessing the failure of (K), we identify term functions in the language of involutive lattices that, when given a, b as inputs, produce generators for $\mathbf{F}_i$ as a subalgebra of $\mathbf{L}$.

Once candidate terms of the above kind are identified, we use Prover9 to derive machine proofs that the subalgebras generated by these terms are isomorphic to the appropriate forbidden configuration $\mathbf{F}_i$. It follows from [5] that the involutive lattice $\mathbf{F}_4$ generates the variety of distributive involutive lattices. We further show that for each $i \in \{5, 6, 8, 10, 12\}$, the variety generated by $\mathbf{F}_i$ contains $\mathbf{F}_4$. This produces the following splitting result.

Theorem 1. *Let $\mathcal{V}$ be a variety of involutive lattices. Then one of the following holds.*

1. *$\mathcal{V}$ is contained in the variety of involutive lattices satisfying (K).*
2. *$\mathcal{V}$ contains all distributive involutive lattices.*

We expect that the heuristic approach outlined here will be successful in the study of many subvariety lattices, in particular for varieties consisting of lattice-ordered algebraic structures. In addition to discussing these paths forward, we discuss some limitations of this heuristic approach, as well as potential avenues for further automation. In particular, we discuss the possibility of using techniques from machine learning to automate the production of the conditions $\Sigma_{\mathbf{F}}$, as well as the term functions that produce generators of forbidden subalgebras.

References

[1] Day, A.: Splitting algebras and a weak notion of projectivity. Algebra Universalis. 5:153–162, 1975.

[2] Davey, B.A. and Werner, H.: Dualities and equivalences for varieties of algebras. In A.P. Huhn and E.T. Schmidt, editors, *Contributions to Lattice Theory*, pages 101–275. North-Holland, Amsterdam, New York, 1983.

[3] Fussner, W. and Galatos, N.: Categories of models of R-mingle. Ann. Pure Appl. Logic 170(10):1188–1242, 2019.

[4] Jipsen, P. and Rose, H.: *Varieties of Lattices*. Springer-Verlag, Berlin, Heidelberg, 1992.

[5] Kalman, J.: Lattices with involution. Trans. Amer. Math. Soc. 87(2):485–491, 1958.

[6] McCune, W.: *Prover9 and Mace4*. Version Dec-2007. http://www.cs.unm.edu/~mccune/prover9

[7] Pynko, A.: Implicational classes of De Morgan lattices. Discrete Mathematics 205:171–181, 1999.

Classification of finite semigroups and categories using computational methods *

Wesley Fussner, Najwa Ghannoum, Tomáš Jakl, and Carlos Simpson

Laboratoire J.A. Dieudonné, CNRS, and Université Côte d'Azur
Nice, France
wesley.fussner@unice.fr, najwa.ghannoum@unice.fr, tomas.jakl@unice.fr,
carlos.simpson@unice.fr

Abstract

We report on our work in progress aimed at analyzing the structure of finite categories, with an eye to developing structure theorems for these. In this work, we rely on the use of McCune's Prover9/Mace4 to construct models, providing in particular a count of the number of categories with two non-isomorphic objects and such that all hom-sets have size 3.

The enumeration of associative structures, together with their classification, presents a deep challenge for algebraic combinatorics. The use of computer algebra systems and tools from artificial intelligence have been applied to great success for finite monoids [7], but these techniques have not been applied in the setting of finite categories. Each of the latter consists of a finite set of objects and a finite set of morphisms between them. As associative algebraic structures, these present an avenue for deepening our understanding of enumeration and classification problems in associative algebra. Additionally, a finite category leads to a derived category of modules, and hence to a moduli stack in algebraic geometry [9]. From the classification set, we obtain a range of examples of derived categories and geometric stacks. Thus, another long-term motivation for this work is to try to find examples having interesting geometrical properties by developing a full picture of the combinatorial structure of the classification question.

Our method of analyzing finite categories is to use computational resources to generate statistics regarding finite categories of a certain type, and then analyze this data in terms of certain features of the objects. This yields, *inter alia*, an exact count of the number of categories with two non-isomorphic objects and such that all hom-sets have size 3. The objects of such finite categories may be identified with their endomorphism monoids, and the structure of the categories may be analyzed by reference to properties of these endomorphism monoids.

This method relies on the use of McCune's Prover9/Mace4 [5] to construct models. We offer a representation of finite categories as semigroups equipped with zero and an additional unary predicate, which encodes the identity morphisms. This representation is a generalisation of the well-known Ehresmann-Schein-Nambooripad Theorem which expresses a fundamental connection between inverse semigroups and inductive groupoids (see e.g. [8]). Our representation allows us to restate the enumeration problem for finite categories in terms of these expanded semigroups. Using a program written in Python to generate optimized semigroup equations as an input for Mace4[1], we generate non-isomorphic algebraic models satisfying those equations. This provides a count of the corresponding finite categories.

This work follows the line of research began in [3, 6] and continued in [4], viewing categories as associated to certain square matrices. The entries of the matrix correspond to the size of

*This project received funding from the European Research Council (ERC) under the European Unions™ Horizon 2020 research and innovation program (grant agreement No. 670624).

[1] Mace4 is a program that searches for finite models of first-order theories.

hom-sets between every two objects, i.e. the matrix (a_{ij}) corresponds to categories where there are exactly a_{ij} morphisms from i to j. For example, the matrix

$$\begin{pmatrix} 3 & 3 \\ 3 & 3 \end{pmatrix}$$

corresponds to categories with two objects and exactly 3 morphisms between any pair of objects. Viewed from this perspective, our motivation is to study the structure of finite categories for a fixed matrix type. The coefficients on the diagonals constrain the structure and nature of the objects by giving restrictions on their endomorphism monoids, which, when considered as fixed parameters, give insight into the enumeration and classification problem. In particular, we obtain information about the number of categories that can be constructed when the endomorphism monoids of the objects are fixed. Our data shows that some types of endomorphism monoids, such as semilattices, give more options to build categories. This way we also discover which structural properties of finite monoids allow their combinations when realized as objects in categories of a given type.

Allouch and Simpson inaugurated the counting problem in [4], where they count the number of categories associated to matrices whose coefficients are all 2. The calculations in this work are performed by hand, up to matrices of size 3. Using our new methods, we extend the Allouch-Simpson count to the matrices of size 2 whose coefficients are all 3.

Ongoing work by Alfaya, Balzin, and Simpson seeks to apply techniques from neural networks to obtain approximate counts of the number of semigroups of a given cardinality. Because the most difficult combinatorial questions in the present work appear to be related to the constituent semigroups, we expect that similar techniques may further the program of enumeration and classification articulated here.

References

[1] S. Allouch, *Classification des Catégories Finies*, thèse dirigée par Carlos Simspon, Université de Nice - Sophia Antipolis, Laboratoire J. A. Dieudonné, soutenue le 22/03/2011.

[2] S. Allouch, *On the Existence of a Category with a given matrix*. Preprint arXiv:1007-2884, (2010).

[3] S. Allouch, C. Simpson, *Classification des Matrices Associées aux Catégories Finies*. Cahiers de topologie et de géomètrie differentielle catégoriques **55** (2014), 205-240.

[4] S. Allouch, C. Simpson, *Classification of Categories with Matrices of Coefficient 2 and order n*. Communications in Algebra **46** (2018), 3079-3091.

[5] W. McCune: *Prover9 and Mace 4*. Version Dec-2007. http://www.cs.unm.edu/~mccune/prover9.

[6] C. Berger, T. Leinster, *The Euler Characteristic of a Category as the Sum of a Divergent Series*. Homol., Homotopy Appl. **10** (2008), 41–51.

[7] A. Distler, T. Kelsey, *The monoids of orders eight, nine and ten*. Ann. Math. Artif. Intell. **56** (2009), 3–21.

[8] C. Hollings, *The Ehresmann-Schein-Nambooripad Theorem and its Successors*. Eur. J. Pure Appl. Math. **5** (2012), 414–450.

[9] B. Toën, M. Vaquié, *Moduli of objects in dg-categories*. Ann. Sci. E.N.S. **40** (2007), 387–444.

Quantum Interference Measurement with Physics Aware Machine Learning at the Large Hadron Collider at CERN

Aishik Ghosh, David Rousseau
Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405 Orsay, France

Introduction

Particle Physics has gone through a few waves of machine learning innovations and is giving back ideas to the machine learning research. Moving from the regime of shoehorning physics problems into forms where existing state-of-the-art machine learning solutions can be applied, particle physics is starting to marry machine learning tools with physics insight to create a new family of "physics-aware machine learning" algorithms where the objective of the tools more closely matches the actual objective of the physicist. This allows leveraging extra information from existing physics tools that can boost performances beyond the use of off-the-shelf machine learning algorithms.

We will compare a neural network model aware of the flexibility of a theoretical physics model (developed by [5]) and a traditional approach optimised at a single point in the phase space being probed with no explicit knowledge of the physics model, for a particular particle physics study very important for physics at the Large Hadron Collider (LHC), CERN. We will show that the former is better even at the particular point at which the traditional approach was optimised, simply because it "understands" the physics being studied better.

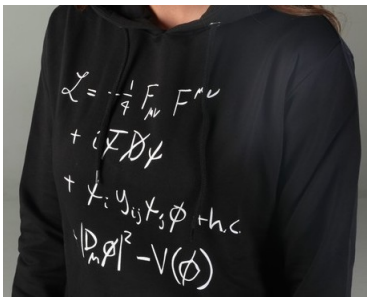
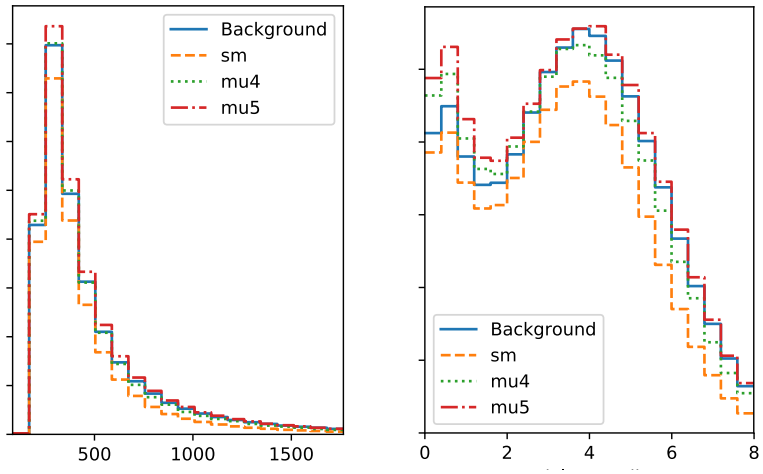


Figure 1: The Standard Model Lagrangian

The Problem of Quantum Interference of the Offshell Higgs

The Lagrangian of the Standard Model (SM) of particle physics is a mathematical formula (Figure 1) that condenses our current understanding of the universe from a quantum perspective, and is known to be incomplete (it does not explain gravity, neutrino mass or matter-antimatter dis-balance). There are several proposed mathematical extensions to the SM (Lagrangians with extra terms) but the most promising ones are already being excluded by data. The SM is continuously being tested at the LHC where the ATLAS experiment [2] collects a huge amount of data to perform precision measurements to find hints of a direction in which to expect new physics. The data is too complex to interpret without involved statistical techniques and a deep understanding of precisely what the SM predicts.

The predictions of a model (Lagrangian) using Quantum Field Theory (QFT) calculations is too expensive to compute analytically so an entire sub-field of particle physics develops Monte-Carlo based simulation techniques for it. At the end, physicists are interested in the inverse problem of going from the measured data to the value of the theory parameters (parameters in the Lagrangian) that best describe it. Re-doing the data analysis for each hypothesis (each new proposed Lagrangian) is impossible given the limited resources, and hence studies are selected that can have an impact on the assessment of multiple promising proposals. One such study is the interference between the



(a) High level feature constructed based on physics motivations

(b) A different feature

Figure 2: Distributions of (a) a physics motivated feature that is usually used for a "four lepton" analysis, but cannot differentiate between $\mu = 0$ and $\mu = 4$ (b) another physics motivated feature which can break the degeneracy between $\mu = 0$ and $\mu = 4$.

"offshell" Higgs boson processes and other "background" processes where four leptons are observed by the detector at the very end ("final state").

The "offshell" Higgs boson particle is "virtual", with a mass far away from the one described by special relativity's $E = MC^2$ (Heisenberg allowed particles to disobey Einstein as long as they do so for a very short period of time, through his famous uncertainty principle, $\sigma_E \sigma_t \geq \frac{\hbar}{2}$). Quantum mechanics also prescribes that given an initial and final state, all possible intermediate states can and will occur, and they might interfere with one another. For the ATLAS "Higgs to four leptons" study this implies that the observed physics could look different for small changes in the "Higgs Couplings", i.e. parameters in the theory that determine how strongly the Higgs interacts with other fields. In this project, these parameters are assumed to scale in similar ways and are represented together by the "signal strength" μ.

Quantum Interference is Problematic to Traditional Algorithms

Usually the signal and background quantum processes come from disjoint phase spaces and can thus be simulated separately in a particle physics simulation. However, in the presence of quantum inference between the signal and background processes, they need to be simulated together to model the probability distributions correctly. As a simplified example, the probability of having one particular sample X, denoted $P(X)$ (with $0 \leq P(X) \leq 1$) is a function of two complex functions from Quantum Field Theory, $M_s(X)$, $M_b(X)$ (with $M_s, M_b \in \mathbb{C}$), for the signal and background process respectively, is given by Eq. 1. If the third term was insignificant and could be ignored, the signal and background contributions could be simulated separately and combined when needed. However in this case, the contribution from the mixed term cannot be ignored.

$$P(X) = |M_s(X) + M_b(X)|^2 = |M_s(X)|^2 + |M_b(X)|^2 + 2Re(\overline{M_s(X)} M_b(X)) \tag{1}$$

This renders the notion of "true class labels" ill-defined, and thus the task cannot translate into a classification problem. Further, since the inference describes very different kinds of physics depending on the value of μ, any algorithm will have to be aware of the physics going on at various values of μ, not just the one at the SM (where $\mu = 1$). Figure 2 demonstrates how a high level, physics motivated feature can fail to distinguish between two very different kinds of physics, which happens due to the added complications of quantum interference.

A new family of machine learning algorithms [3, 4, 5, 6] have recently been in development that are at the intersection of machine learning, probabilistic programming, statistics and particle physics phenomenology. The techniques rely on the ability to simulate accurate samples and "cheat", i.e. extract additional information about the physics model from the simulator that would be unavailable in real data recorded at the LHC. The additional information allows to train neural networks that

are not just aware of "Signal" and "Background" classes but rather learn the flexibility of the Largrainians themselves. The authors are able to show on a toy particle physics data-sets that training on such augmented data allows to use neural networks as a tool for calculus of variations and arrive at the likelihood ratio between any two physics models.

The actual ATLAS Higgs to four leptons analysis [1] is more complicated, and the family of physics models is confined by extra assumptions from the inference strategy and also prior knowledge from other measurements. We investigate the possibility to adapt the technology presented in [5] to this problem. Further, particle physicists have found a way to study almost all possible alternatives to the SM that might be measurable at the LHC using an "Effective Field Theory Framework" (EFT Framework) [7]. This is possible because of some mathematical and physical properties any Lagrangian must satisfy, making the number of terms of the Lagrangian to study finite. A successful use of these new algorithms within the ATLAS experiment will pave the way for further investigation into them for ongoing studies within ATLAS in the context of EFT.

Results

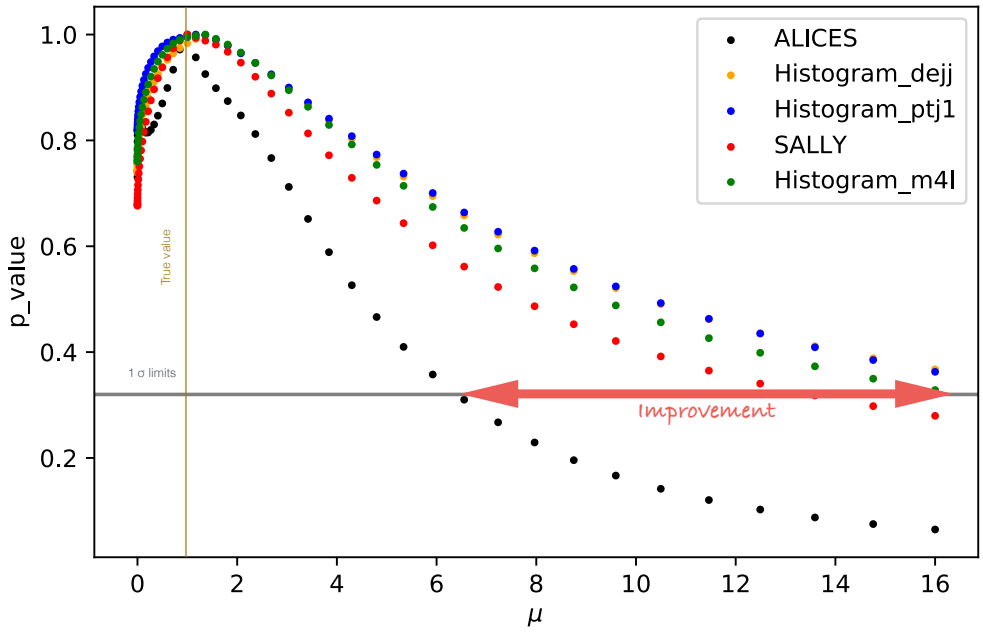


Figure 3: P-value scan using various Histogram techniques compared to SALLY and ALICES for a true value of $\mu = 1$ (sharper is better). The horizontal grey line indicates the p-value corresponding to a 1σ confidence interval

Some preliminary results (Figure 3) are shown to compare a traditional "histogram" approach of particle physics with more and more "physics-aware" algorithms to infer the true parameters of the Lagrangian. At inference time, the inputs of the neural network, for a given sample, are the features measured by the detector, as well as the hypothesis being tested (i.e. one particular value of μ). The output of the network is the likelihood ratio between the test hypothesis and the null hypothesis ($\mu = 1$). The output for all samples in the test dataset for a given test hypothesis is converted into a single p-value, as in standard statistics, and the entire process is redone for the same test dataset with a new test hypothesis (new value of μ). The p-values for the histogram techniques is calculated using multi-binned Poisson likelihood with the normalised histogram of particular physics motivated features. The "SALLY" (**S**core **A**pproximates **L**ikelihood **L**ocall**y**) model is aware of physics in the neighbourhood of the SM ($\mu = 1$) whereas "ALICES" (**A**pproximate **L**ikelihood with **I**mproved **C**ross-entropy **E**stimator and **S**core) is aware of physics in the entire range of μ, and shows better results (narrower peaks in a p-value scan, smaller 1σ margin of uncertainty for measuring μ), thus demonstrating the usefulness of a physics-aware model.

Conclusion

There is a long history of cross-pollination between particle physics and machine learning. A first study is performed to try to adapt a family of novel "physics-aware" machine learning algorithms to a realistic Higgs to four leptons analysis for the ATLAS experiment at CERN. Other efforts along similar lines such as probabilistic programming, graph networks, physics-aware generative models, adversarial networks, and so on also indicate the impending shift in the particle physics community from shoehorning physics problems into state-of-the-art machine learning algorithms to developing physics-aware algorithms that can leverage available physics insight as well as inject inductive biases to algorithms in a way that was not possible before.

Our initial studies indicate that a neural network aware of the theoretical physics model performs better inference than traditional physics-agnostic techniques, in the presence of severe quantum interference. Further studies need to be done taking into account all signal and background processes as well as simulating within the ATLAS software infrastructure to take into account the true detector effects. These machine learning models for the first time could be extended to also be aware of systematic uncertainties (when there is a known systematic difference between the simulated training data, and the real unlabeled data to which we will apply the model, but the amount and nature of the difference is unknown) that were difficult to incorporate in traditional machine learning techniques. Success with these techniques encourages the idea of extending this philosophy to other fields, such as "Maths-Aware Machine Learning".

References

[1] Morad Aaboud et al. Constraints on off-shell Higgs boson production and the Higgs boson total width in $ZZ \to 4\ell$ and $ZZ \to 2\ell 2\nu$ final states with the ATLAS detector. *Phys. Lett.*, B786:223–244, 2018.

[2] ATLAS Collaboration. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.

[3] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. A Guide to Constraining Effective Field Theories with Machine Learning. *Phys. Rev.*, D98(5):052004, 2018.

[4] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. Constraining Effective Field Theories with Machine Learning. *Phys. Rev. Lett.*, 121(11):111801, 2018.

[5] Johann Brehmer, Felix Kling, Irina Espejo, and Kyle Cranmer. MadMiner: Machine learning-based inference for particle physics. 2019.

[6] Johann Brehmer, Gilles Louppe, Juan Pavez, and Kyle Cranmer. Mining gold from implicit models to improve likelihood-free inference. 2018.

[7] D. de Florian et al. Handbook of LHC Higgs Cross Sections: 4. Deciphering the Nature of the Higgs Sector. 2016.

Make E Smart Again *

Zarathustra Amadeus Goertzel, Jan Jakubův, and Josef Urban

Czech Technical University in Prague, Prague

Making E Stupid and Then Smart Again The ENIGMA [4, 5, 6, 3] system with the XGBoost [1] implementation of gradient boosted decision trees has recently shown high capability to learn guiding the E [8] prover's inferences in real time. In particular, after several proving and learning iterations, its performance on the 57897 problems from the Mizar40 [7] benchmark improved by 70% (= 25397/14933) [6] over the good E strategy used for the initial proving phase. This good strategy uses many sophisticated *clause evaluation functions*, the Knuth-Bendix ordering (KBO6), and other E heuristics.

In this work we strip E to the bare bones: KBO6 is replaced with an identity relation as the minimal possible ordering (an addition to E), the strategy is replaced with the simple combination of the clause weight and FIFO (first in first out) evaluation functions, and literal selections are disabled. Literal selection is important because by limiting the literals used in inference, E can generate far fewer clauses and avoid redundant inferences. E is thus practically reduced to a basic resolution prover with some rewriting capabilities. We call this strategy E0:

```
--definitional-cnf=24 --prefer-initial-clauses -tIDEN --restrict-literal-comparisons
 -WNoSelection -H'(5*Clauseweight(ConstPrio,1,1,1),1*FIFOWeight(ConstPrio))'
```

E0 solves only 3872 of the Mizar40 problems in 5 seconds. The task for ENIGMA with this basic prover is to learn the ATP guidance completely on its own, i.e., we explore how smart E can become using machine learning in this *zero-strategy* setting. The more general related question is to what extent can machine learning replace the sophisticated human-invented theorem-proving body of wisdom used in today's ATPs for restricting advanced proof calculi.

Learning Experiments We evaluate ENIGMA with the basic strategy, E0, in several scenarios and over two datasets of different size. All experiments are run with 5 seconds per problem.[1] ENIGMA has so far been used in two ways: *coop* combines the learned advice with some standard E strategy equally (50 : 50) [2] while *solo* only uses the learned ENIGMA model for choosing the given clauses. The best results have been achieved by looping: that is, an ENIGMA model loop 0 is trained and run with E (loop 0), then the resulting data are added to the initial training data and a new ENIGMA model is trained (loop 1).

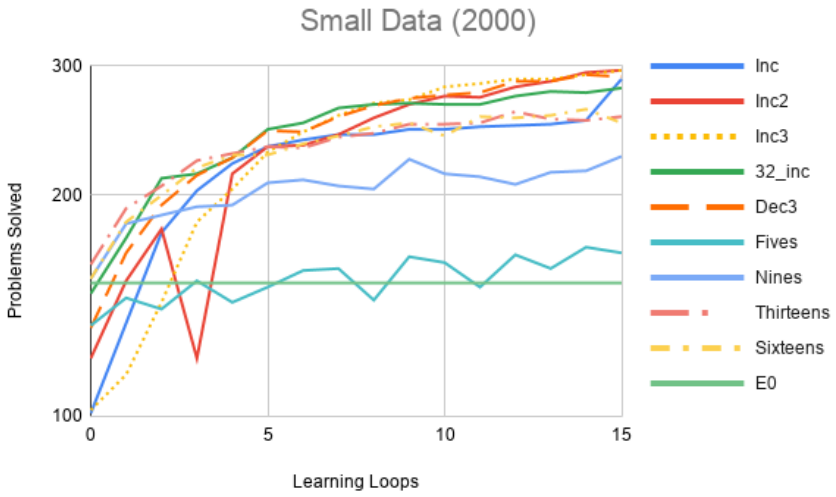
In this work we train with both *solo* and *coop*, and only present results from *solo* runs because they represent the most minimal setting.

Small Data (2000 problems): The E evaluations and XGBoost training can take a long time on the full Mizar40 dataset, so we randomly sampled 2000 problems to test hyper-parameters on. Each XGBoost model consists of T decision trees of depth D, the most important training meta-parameters. In previous work T and D were fixed for all loops of learning. Here we try vary the values during 15 loops. Let $S_{D,T}$ denote the experiment with specific T and D. The following results are included in the plot of solved problems (above right): *Fives* ($S_{5,100}$), *Nines* ($S_{9,100}$), *Thirteens* ($S_{13,200}$), *Sixteens* ($S_{16,100}$).

*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.

[1] Almost all the experiments are run on the same hardware: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz with 188GB RAM.

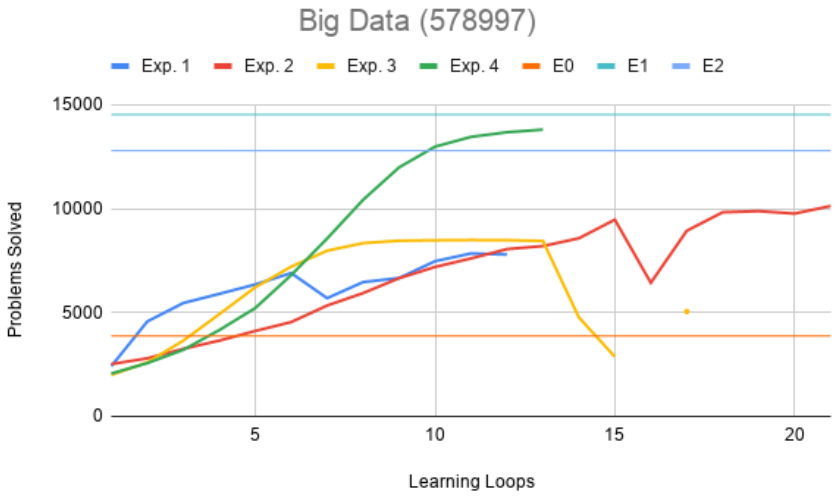
[2] This means that half of the given clauses are selected by the E strategy and half by ENIGMA's model.



We also experiment with adaptively setting the hyper-parameters as the number of training examples increases. Protocol *Inc* ($S_{[3,33],100}$) increases D by 2 from 3 to 33 and keeps $T = 100$ fixed. Protocol *32_inc* ($S_{32,[50,250]}$) fixes $D = 32$ and gradually increases T from 50 to 250. Protocol *Inc2* ($S_{[3,33],*}$) gradually decreases T from 150 to 50, varying the value intuitively. Protocol *Inc3* ($S_{[3,33],[50,250]}$) aims to be more systematic and steps T from 50 to 250, and Protocol *Dec3*($S_{[3,33],[250,50]}$) decreases T from 250 to 50.

At the 15th loop *Inc* is best solving 299 problems, doubling the performance of E0 (152). *Inc2* and *Inc3* solve 298 problems for second, and *32_inc* takes third place at 291 problems. The conclusion is that simple protocols work well so long as T or D is incremented adaptively rather then fixed.

Big Data (57897 problems): The experiments are done on a large benchmark of 57897



Mizar40 [7] problems from the MPTP dataset [9]. E1 and E2 are two strong E strategies solving 14526 and 12788 problems.

- **Experiment 1** is done with $D = 9$ and $T = 200$ and uses our previously trained model, which allowed us to solve solve 25562 problems when cooperating with E1 in our previous experiments [6]. This strong model, which hashes the features into 65536 (2^{16}) buckets [2, Sec. 3.4], is used with E0 now.
- **Experiment 2**'s parameters were intuitively toggled during the looping as in *Inc3*. Exp. 2 uses training data from E1 and E2 for additional guidance up to the 4th loop (and then stops including them in the training data on the assumption they may confuse learning).
- **Experiment 3** sets T and S according to protocol *Inc3*. Exp. 3 only learns from E run with E0 and trains on the GPU, which requires the feature size to be reduced to 256.
- **Experiment 4** mimics Exp. 3 but uses E1 and E2 data for training (up to the 4th loop).

The strong model does not help much in guiding E without ordering or selection in Exp. 1. Exp. 2 learns gradually and catches up with Exp. 1, but seems to plateau around $10,000$. Surprisingly the pure Exp. 3 learns fast with the small feature size, but plateaus and drops in performance (perhaps due to overfitting). Exp. 4 indicates that guidance is useful and surpasses E2 with 13805 in round 13. This is a great improvement over the 3872 problems solved by E0.

Conclusion ENIGMA can learn to guide the E prover effectively even without smart strategies and term orderings. The models confer a 256% increase over the naive E0 after 13 rounds of the proving/learning loop, and even trained without guidance data, a 121% increase. The experiments indicate that machine learning can be used to fully control an ATP's guidance, learning to replace orderings, heuristic strategies, and deal with the increase in generated clauses without literal selection. Given the large symmetry-breaking impact of these methods in classical ATP, future work includes, e.g., training the guidance in such a way that redundant (symmetric) inferences are not done by the trained model once it has committed to a certain path. This probably means equipping the learning with more history in the saturation-style setting.

Running ENIGMA without term ordering and other restrictions is important because it allows us to combine training data from different strategies. We aim at combining several strategies into one with a performance comparable to their parallel execution. Increasing parameters S and T with training loops also seems promising and as it outperforms static values we plan to investigate it further.

References

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. *CoRR*, abs/1903.03182, 2019.

[3] Zarathustra Goertzel, Jan Jakubuv, and Josef Urban. Enigmawatch: Proofwatch meets ENIGMA. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019.

[4] Jan Jakubuv and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[5] Jan Jakubuv and Josef Urban. Enhancing ENIGMA given clause guidance. 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 118–124. Springer, 2018.

[6] Jan Jakubuv and Josef Urban. Hammering Mizar by learning clause guidance. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[7] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.

[8] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[9] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

A Controlled Natural Language for Type Theory*

Thomas Hales

University of Pittsburgh
Pittsburgh, PA, USA

1 Introduction

This abstract describes the design and development of a controlled natural language for mathematics that has the Lean theorem prover as intended target. We call this language Colada (short for *Co*ntrolled *la*nguage *da*ta). Documents in our dialect are written in a specially prepared LaTeX file. Our aim is to capture definitions and theorem statements from the published mathematical literature in our dialect, but checking mathematical proofs is beyond the scope of our project.

Our design grows out of previous controlled natural languages for mathematics (specifically, Forthel-Naproche-SAD), as described in Peter Koepke's AITP 2019 talk, which exhibited some short proofs written in fluent English that can be read and checked by their software [FK19], [Pas07], [LPV04].

We use Forthel as the generic name for any of the dialects inspired by Forthel (a controlled natural language developed by various researchers starting with Glushkov's *Evidence Algorithm*, and implemented in Paskevich's PhD thesis), including Colada. We refer to the Colada language as *our dialect*. Our dialect differs from others in that our semantic target is the logical Calculus of Inductive Constructions (CiC) as implemented in the Lean theorem prover, instead of first-order logic [dMKA+15]. Our dialect can be viewed as a fusion of three different syntactic traditions: Forthel syntax, LaTeX syntax, and Lean theorem-prover syntax. From another perspective, our dialect might be viewed as a mountain of syntactic sugar for Lean.

2 Controlled Natural Languages (CNL)

By controlled natural language for mathematics (CNL), we mean an artificial language for the communication of mathematics that is (1) designed in a deliberate and explicit way with precise computer-readable syntax and semantics, (2) based on a single natural language (which for us is English), and (3) broadly understood at least in an intuitive way by mathematically literate speakers of the natural language.

CNLs can achieve a much higher degree of English fluency than other proof-checking languages.

Our basic aim is to develop a technology that lies somewhere between the current practice of research mathematicians and the current practice within the proof assistant community.

*I thank Peter Koepke for introducing me to the field and Jesse Han for his contributions. Research is supported by Sloan Foundation grant G-2018-10067. This work is part of the Formal Abstracts project, which aims to capture all the major definitions and theorems of mathematics in a format that is both human and computer friendly. Source code and examples are found at github.com/formalabstracts/CNL-CIC.

3 Research to Date

Our specific research contributions to date are as follows.

We have a design and specification of a controlled natural language. Like other Forthel dialects, our grammar is not context-free. However, it is similar to a context-free grammar by being specified through production rules on terminal and nonterminal symbols. Users may extend the grammar with new mathematical notation and constructs: the language contains syntax for the extension of its own syntax.

The lexical structure of our dialect is specified in sedlex, a lexical generator tool for OCaml. Our dialect has been specified in menhir, an OCaml-based parser-generator tool for LR(1) grammars. (Although our dialect is not an LR(1) grammar, which prevents menhir from automatically generating a parser, the software checks that our grammar is well-formed.)

We believe that some complexity is justified (and even required) to capture widespread mathematical idioms and formulas, the syntax of type theory, and their interactions. Our grammar is recursive to an extraordinary degree. The grammar has about 350 nonterminals and about 550 production rules. The grammar contains about 150 English words (such as *all, any, are, case, define, exists, if, iff, is, no, not, of, or, over, proof, the, theorem*, etc.) with a fixed grammatical function. User syntax extensions build on that base.

We keep most features of Forthel, such as its handling of synonyms, noun phrases, verbs, and adjectives; and its grammar extension mechanisms. We have added many additional features such as plural formation for nouns and verbs, operator precedence parsing (with user-specified precedence levels and associativities); scoping of variables; syntax for LaTeX macros; and dependent type theory including inductive and mutual inductive types, structures, and lambda terms.

A parser for our grammar has been implemented in OCaml, building substantially on the parser combinator library that John Harrison wrote to parse HOL Light.

Future work will transform parsed output to type-checked terms in Lean: our system will output Lean-pre-expressions, which will then be processed by Lean's elaboration and type-checking procedures. Another future project is syntax highlighting and auto-completion tools for our dialect in editors such as emacs and VSCode. We also plan to develop large mathematical libraries in our dialect.

We have written software that takes a specially prepared LaTeX file as input and strips away the non-semantic content (such as headers, spaces and other layout, graphics, remarks, and dollar signs) and outputs raw CNL. The key to beautifully typeset TeX documents is a dual expansion system for macros. The TeX engine expands macros in the usual way, but the CNL engine expands macros according to an independent semantic specification.

We believe our language will find novel applications to search, document analysis, and document transformation.

4 Example

Examples will be given during the AITP presentation to show that English fluency is obtained without loss of semantic content. The presentation will include a discussion of dependent types, structures, inductive types, type coercions, and implicit arguments.

Here is one example that assumes a context in which a binary relation $(R, \leq)$ has been defined.

4.1 pdf

Here is a sample text, as viewed by the mathematician reading the document.

Definition 1 (greatest element)**.** *We say that y is a* **greatest element** *in R iff for all x, $x \leq y$.*

Let $x < y$ stand for $x \leq y$ and $x \neq y$.

4.2 source

The LaTeX source file for the pdf is similar to documents prepared every day by mathematicians.

```
\def\deflabel#1{\begin{definition}[#1]\label{#1}}

\deflabel{greatest element} We say that $y$ is a
\df{greatest\~element} in $R$ iff for all\ $x,\ x \le y$.
\end{definition}

Let $x < y$ stand for $x \le y$ and $x \ne y$.
```

4.3 CNL

The CNL is generated from the source file by stripping formatting.

```
Definition Label_greatest_element . We say that y is a
greatestelement in R iff for all x , x \le y .

Let x < y stand for x \le y and x \ne y .
```

4.4 parse tree

This parse tree is generated from the CNL. We only display the first definition and have pruned the tree for simplicity. Nonterminals are typeset in smallcaps and literals are in teletype. At hierarchical levels above those shown in the outline, we have TEXT_ITEM → DECLARATION → DEFINITION. A further transformation not shown would produce a Lean pre-expression from the tree.

- DEFINITION__PREAMBLE
  - LIT__DEF ..... `Definition`
  - LABEL ..... `Label_greatest_element`
  - PERIOD ..... `.`
- list(ASSUMPTION) .....
- DEFINITION__AFFIRM
  - DEFINITION__STATEMENT → PREDICATE__DEF
    - OPT__SAY ..... `We say that`
    - PREDICATE__HEAD → PREDICATE__WORD__PATTERN → NOTION__PATTERN
      - TVAR ..... y
      - LIT__IS ..... `is`
      - LIT__A ..... `a`
      - WORD__PATTERN ..... `greatestelement in` R
    - IFF__JUNCTION ..... `iff`
    - STATEMENT ..... `for all` x`,` x `\le` y
  - PERIOD ..... `.`

References

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In *International Conference on Automated Deduction*, pages 378–388. Springer, 2015.

[FK19] Steffen Frerix and Peter Koepke. Making set theory great again: The Naproche-SAD project. 2019.

[LPV04] Alexander Lyaletski, Andrey Paskevich, and Konstantin Verchinine. Theorem proving and proof verification in the system SAD. In *International Conference on Mathematical Knowledge Management*, pages 236–250. Springer, 2004.

[Pas07] Andrei Paskevich. The syntax and semantics of the ForTheL language, 2007.

Towards Big Theory Exploration

Sólrún Halla Einarsdóttir[1] and Moa Johansson[1]

Chalmers University of Technology, Gothenburg, Sweden.
{slrn, moa.johansson}@chalmers.se

Abstract

QuickSpec is a system for theory exploration, able to automatically generate many interesting conjectures about mathematical functions. However, when exploring bigger theories containing many different functions (approx. $> 15 \sim 20$), the exponential blow-up of its search space can make it too slow to be useful. We present work in progress on a template-based extension, intended as a complement when dealing with big theories. We sacrifice broad search for more direction towards commonly occurring property patterns, sourced from e.g. mathematical libraries.

1 Introduction

Theory exploration is a method of automatically inventing interesting properties or candidate lemmas. For example, we might give our theory exploration system the functions *length* and *reverse* on lists, and discover the property $length\ (reverse\ xs) = length\ xs$. Knowing about this property could then be helpful in inventing or proving more complicated theorems.

QuickSpec [6] is a theory exploration tool which discovers equational properties about Haskell programs by generating all type-correct terms that can be formed using the given functions, up to a size limit, and then using the property-based testing tool QuickCheck [1] to test which terms are equivalent. A weakness of QuickSpec is that if it is given a large number of functions to explore at once it will cease to be quick as its name suggests and instead becomes too slow to be practically useful, while outputting an overwhelming amount of properties and struggling to prove away those that are uninteresting (see example in section 4.2 of [6]).

Determining what properties are "interesting" is a big challenge. We hypothesize that many properties that humans consider interesting (or useful) in fact often have similar shapes (some of these shapes have names, such as associativity, commutativity, distributivity, or appear as type-class laws etc.). Also, if considering using the theory exploration system in combination with a theorem prover, as with QuickSpec in the Hipster [4] system, a failed proof attempt might suggest shapes of potential missing lemmas [3]. We therefore propose a "quick-QuickSpec" using *property templates* to capture such shapes or patterns, while sacrificing some of the completeness in search. The templates are instantiated with available function symbols and results tested for counter-examples. Surviving conjectures are presented to the user or passed on to an automated prover. Similar techniques have been suggested in e.g. [5, 2], but we aim for a higher level of automation.

2 Template-based QuickSpec

We have implemented a prototype of a modified version of QuickSpec using templates. The user specifies each template they are interested in using an expression format where question marks denote holes, for example: $?F(?G(X,Y)) = ?F(?G(Y,X))$ describes the composition of two functions being commutative in two variables. Candidate properties are generated by

attempting to fill the holes in the template using the functions in the exploration scope, restricting the generated equations to be well typed. For example, filling the holes in the template above using functions *length*, *reverse*, and ++ on lists gives the candidate properties $length\,(xs\,+\!+\,ys) = length\,(ys\,+\!+\,xs)$ (cp1) and $reverse\,(xs\,+\!+\,ys) = reverse\,(ys\,+\!+\,xs)$ (cp2). The generated candidate properties are then tested using QuickCheck [1]. If no counterexamples are found the property is presented to the user as a conjecture. In our example, cp1 passes this phase and is presented as a conjecture, while counterexamples are found to cp2.

Evaluation

We have compared the performance of our extension to standard QuickSpec on some benchmark theories[1]. Using 12 templates describing basic properties of functions and operators, we first explored a few normal-sized theories of list functions, booleans and arithmetic. Most of the properties found by standard QuickSpec for these theories are in fact also found using these templates, and some of the properties not replicated are ones we consider redundant or uninteresting. We even find some nice properties that standard QuickSpec had pruned away (e.g. one of De Morgan's laws).

Secondly, we considered a stress-test (section 4.2 in [6]), where QuickSpec was used to find properties about a set of 33 Haskell functions on lists. This took standard QuickSpec 42 minutes and resulted in 398 properties when limited to terms of size 7 or less, and hit a time limit of 2 hours when the size was increased to 8, illustrating how running QuickSpec on larger theories scales poorly with regard to run-time and may produce an overwhelming amount of output. In contrast, running our new prototype on this big theory, using the same standard 12 templates as above, we discover 41 properties in under 2 seconds, including some properties containing terms of size 7. We can also discover properties containing even larger terms (e.g. terms of size 9) in under 1 second if we provide templates supporting those sizes. Theory exploration is now tractable, at the price of providing a stricter specification of the shape of desired properties.

3 Next steps

The next step is to automate the discovery of lemma templates. We will explore several options, e.g. machine learning to extract common patterns from proof libraries, learning common lemma shapes given properties of the theorem we want to prove (c.f. [2]), as well as exploiting type-class laws and other algebraic properties. We will also investigate extracting templates from failed proof attempts, similar to critics in proof planning [3]. We will conduct a larger experimental evaluation, comparing standard QuickSpec with our template-based algorithm to identify the "sweet-spot" for the respective approaches, and how they can be combined. Naturally, the template based approach might miss "unusually shaped" properties, but this could be a price worth paying for scalability. Perhaps the more exhaustive approach of standard QuickSpec should be used for small terms (of which there are fewer) and a template-based approach used for larger terms, with the exact split depending on the size of the theory being explored.

4 Acknowledgements

This work was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation.

[1] See https://github.com/solrun/quickspec/tree/master/template-examples

References

[1] K. Claessen and J. Hughes. QuickCheck: a lightweight tool for random testing of Haskell programs. In *Proceedings of ICFP*, pages 268–279, 2000.

[2] J. Heras, E. Komendantskaya, M. Johansson, and E. Maclean. Proof-pattern recognition and lemma discovery in ACL2. In *Proceedings of LPAR*, 2013.

[3] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16:79–111, 1996.

[4] M. Johansson. Automated theory exploration for interactive theorem proving: An introduction to the Hipster system. In *Proceedings of ITP*, volume 10499 of *LNCS*, pages 1–11. Springer, 2017.

[5] O. Montano-Rivas, R. McCasland, L. Dixon, and A. Bundy. Scheme-based theorem discovery and concept invention. *Expert systems with applications*, 39(2):1637–1646, 2012.

[6] N. Smallbone, M. Johansson, K. Claessen, and M. Algehed. Quick specifications for the busy programmer. *Journal of Functional Programming*, 27, 2017.

Learning cubing heuristics for SAT from DRAT proofs

Jesse Michael Han

University of Pittsburgh
jmh288@pitt.edu

We learn a variable selection heuristic for the cube-and-conquer paradigm in SAT solving by training the NeuroCore architecture to predict variable occurrence counts in DRAT proofs of unsatisfiable formulas. We evaluate our models by averaging CDCL runtimes on the subproblems produced by branching on their predictions, and also by their average scores in the Prover-Adversary game against a random adversary. As a baseline, we compare with Z3's implementation of the `march_cu` variable selection heuristic. Our results indicate that training to predict DRAT variable counts usually outperforms training to predict occurrence of a variable in an unsat core. On all three evaluation datasets, our best models outperform `march_cu` on solution time, and on two, they achieve superior performance on the game-based metric.[1]

Introduction *Cube-and-conquer* [6] is a relatively new SAT solving paradigm wherein a lookahead solver makes expensive, globally-informed decisions on how to partition (cube) a SAT problem into subproblems, which are then solved (conquered) in parallel by CDCL solvers. It has been used to prove the unsatisfiability of relatively small (but hard for CDCL) combinatorial SAT problems [5, 4, 3]. While previous approaches [14, 7, 10, 9] to improving SAT solvers with neural networks have tried integrating variable and literal selection directly into the run of a CDCL solver, we propose targeting cubing heuristics, which allow for fewer but more expensive and impactful decisions.

A cubing heuristic comprises a variable selection heuristic and a cutoff heuristic. After each variable selection, two new leaves are added to the search tree, corresponding to either assignment of the variable. After propagating the assignments, the cutoff heuristic examines the resulting formulas at the leaves and decides whether or not they are easy for CDCL. If it deems a leaf to be easy (or if it has exceeded a budget of cubes), the cutoff heuristic freezes it. This process is repeated on the hardest unfrozen leaf. This produces a truncated search tree whose leaves are the cubes. In our present work, we target only the variable selection heuristic by querying our models for the top K variables and producing 2^K cubes. In practice, this leads to poor parallelization [5] as K scales due to a few disproportionately hard problems near the root, so we use $K = 1$ and $K = 3$, and evaluate our models by averaged runtime on the leaves.

As observed in unpublished work by Selsam [13], the job of the cutoff heuristic is essentially to estimate the size of the DPLL search tree beneath a leaf. Knuth [8] showed that the size of a backtracking search tree can be estimated by the lengths of randomly sampled paths through the tree. We recast this in terms of playouts in a two-player zero-sum game, known in the literature as the *Prover-Adversary game* [12]. At each round, player 1 (Prover) picks an unassigned variable, and player 2 (Adversary) assigns it. The game ends when either all clauses are satisfied or some clause is unsatisfied by the trail of assignments. The terminal value of the game is the number of rounds divided by the number of variables; for an unsat formula, player 1 seeks to minimize this, and player 2 seeks to maximize this. We modify the game so that unit propagation occurs after every round; then every playout is a path through the DPLL search tree. Urquhart [16] proved that player 1 has a winning strategy in fewer than K rounds iff there is a resolution proof of unsat of depth $\leq K$. A good policy for player 1 will thus guide the

[1] https://www.github.com/jesse-michael-han/neurocuber-public/

game towards shallower parts of the search tree where conflicts occur relatively quickly. This is exactly the behavior we desire from the variable selection heuristic of a cuber. We additionally evaluate our models with the average terminal value of playouts against a random adversary, where our models queried at every round for Player 1's policy. This metric provides a more robust evaluation of our models' decisions, as they are queried dozens to hundreds of times per formula versus only once; also, unlike the timing-based metric, this is unaffected by resource contention.

DRAT proofs Resolution proofs emitted by SAT solvers quickly become enormous as problems scale. A DRAT proof [2] emitted by a modern CDCL solver is essentially an extremely compressed resolution proof: each line in the proof will typically be a learned conflict clause abbreviating dozens or hundreds of unit propagation steps. DRAT proofs can thus be roughly thought of as the SAT analogue of a tactic proof script for interactive theorem provers in higher logics: a list of high-level non-deterministic steps which can be formally linked together by more primitive automation (unit propagation). The intuition behind our approach is that DRAT proofs are a readily-available high-quality representation of resolution trees, and if a variable occurs frequently in a resolution tree, branching on it will correspondingly minimize the average size of the resolution trees (and proportionally the solving times) for the leaves.

Network architecture, data, and training Our implementation is based on the simplified NeuroSAT [15] architecture used by Selsam and Bjørner [14] to guide CDCL solvers through periodic refocusing of EVSIDS scores [11]. They trained a variable scoring head and a clause scoring head to predict the variables and clauses in a labelled unsat core. Besides minor modifications to the GNN embedding network, we add another variable scoring head, and train it to predict occurrence counts of variables in DRAT proofs. The loss function is calculated by softmaxing the true occurrence counts and logits and taking the forward KL-divergence. We train to minimize the sum of all three losses . All models are implemented in TensorFlow 2. We trained on a synthetic dataset `src` of 250000 problems, based on the problem distributions **SR** and **SRC** described in [15] as follows: first, we extract an unsat core C of size ≥ 20 and ≤ 100 from a problem in $\mathbf{SR}(20)$, modified to exclude binary clauses to increase the difficulty of the core, and then sample a formula from $\mathbf{SRC}(100, C)$ which is between 5 to 20 times larger than C. As a baseline, we also trained a separate model on another dataset `sr` of 250000 unsatisfiable problems drawn from $\mathbf{SR}(\mathbf{U}(10, 40))$. We obtain variable occurrence counts from DRAT proofs (excluding deletion clauses) emitted by the state-of-the-art CDCL solver `cadical` [1], and extract unsat cores by verifying the proofs with `drat-trim` [17].

Evaluations We evaluate on three datasets `ramsey`, `schur`, and `vdw` of 1000 random subproblems each (randomly assigning 5, 35, and 3 variables) of the hard combinatorial problems $\mathsf{Ramsey}(4, 4, 18)$, $\mathsf{Schur}(4, 45)$, and $\mathsf{vanderWaerden}(2, 5, 179)$. The problems range in size from ~ 3000 to ~ 7800 clauses. For timing evaluation, we query each variable selection head of each model once on the first 250 problems in each dataset, picking the top $K = 1, 3$ scored variables, then recording the solving time of the cubes in parallel with as many cores as cubes; we used `cadical` as the conqueror. We only queried `march_cu` for $K = 1$. The runs were performed serially on a 16-core machine with no other compute-intensive tasks. For random playout evaluation, we play 50 matches on all formulas on all datasets and record the average terminal values and average number of unit propagations after every round. We used the distribution framework `ray` to parallelize up to 16 playouts at once per run; all runs were done in parallel on the PSC Bridges cluster.

	sr_core	sr_drat	src_core	src_drat	random	march_cu
avg terminal value	0.193	0.162	0.144	0.14	0.192	0.146
avg unit props	1.065	1.278	1.415	1.471	1.064	1.873

Table 1: Average terminal values and unit propagations for all variable selection heuristics after 50000 playouts vs. a random Adversary on `ramsey`. Lower terminal values are better.

	sr_core	sr_drat	src_core	src_drat	random	march_cu
ramsey	5.017	3.811	4.345	4.025	5.248	4.83
schur	1.934	1.787	1.504	1.517	2.392	1.903
vdw	2.618	1.85	1.843	1.803	2.215	2.07

Table 2: Averaged wall clock runtimes (in seconds) for top-1 cubing of all variable selection heuristics. On average, our best heuristics produce an 18% speedup over `march_cu`.

Results We use the naming convention `neurocuber-<train_dataset>_<head>` to refer to our learned variable selection heuristics. On all test datasets, both `neurocuber-src_drat` and `neurocuber-sr_drat` outperform `march_cu` on top-1 timing evaluation (Table 2). Table 1 shows the result of random playout evaluation on 1000 subproblems of Ramsey$(4, 4, 18)$. Remarkably, even though `march_cu` finds significantly more unit propagations than our models, it is still outperformed by `neurocuber-src`. The same phenonenon occurs on the `schur` dataset. Only on `vdw` does `march_cu` achieve the best terminal value. Table 3 and Table 4 show the average percent change in performance of the DRAT-variable over core-variable scoring heads for both models. On `ramsey` and `vdw`, DRAT-variable heads outperformed core-variable heads across all models and metrics, with significant improvement for `neurocuber-sr` across the board.

Conclusions and future work The comparative timing performance of our models was tightly correlated with their terminal scores in the Prover-Adversary game, providing empirical evidence that good policies for player 1 translate to good variable selection heuristics for cube-and-conquer. Our experiments show that training to predict DRAT variable counts consistently yields better variable selections than training to predict the binary occurrence of variables in unsat cores. Remarkably, on some hard combinatorial problems squarely in the domain of cube-and-conquer, our strongest models outperform domain-specific heuristics without optimizing as much for unit propagation. While maximizing the number of expected unit propagations is an obvious short-range policy in the Prover-Adversary game (and far more sophisticated versions of this are currently state-of-the-art for cube-and-conquer), our experiments suggest that better policies can be learned, even through just supervised training on proxy targets. The natural next step is direct reinforcement learning of the policy and value functions. We will discuss steps in this direction during our talk.

	ramsey	schur	vdw
top1 timing	24.06	6.3	29.19
top3 timing	55.24	30.31	59.42
random playout	16.38	8.77	5.78

Table 3: Percent improvement of DRAT over core-var heads for `neurocuber-sr`.

	ramsey	schur	vdw
top1 timing	7.3	-1.4	2.06
top3 timing	17.42	0.1	9.96
random playout	2.91	-0.54	5.14

Table 4: Percent improvement of DRAT over core-var heads for `neurocuber-src`.

Acknowledgements We thank Daniel Selsam and Nikolaj Bjørner for initially suggesting the approach taken in this paper. This work benefited from conversations with Tom Hales, Emre Yolcu, Daniel Abolafia, and Ruben Martins. This work was supported by XSEDE start up grant TG-DMS190028 and grant G-2018-10067 from the Sloan Foundation.

References

[1] Armin Biere. CaDiCaL simplified satisfiability solver. http://fmv.jku.at/cadical/.

[2] Marijn J. H. Heule. The DRAT format and DRAT-trim checker. *CoRR*, abs/1610.06229, 2016.

[3] Marijn J. H. Heule. Avoiding triples in arithmetic progression. *Journal of Combinatorics*, 8(3):391–422, 2017.

[4] Marijn J. H. Heule. Schur number five. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606, 2018.

[5] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, pages 228–245, 2016.

[6] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-conquer, a hybrid SAT solving method. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4864–4868, 2017.

[7] Sebastian Jaszczur, Michał Łuszczyk, and Henryk Michalewski. Neural heuristics for SAT solving. In *Representation Learning on Graphs and Manifolds Workshop at ICLR 2019*, 2019.

[8] Donald E Knuth. Estimating the efficiency of backtrack programs. *Mathematics of Computation*, 29(129):122–136, 1975.

[9] Vitaly Kurin, Saad Godil, Shimon Whiteson, and Bryan Catanzaro. Improving SAT solver heuristics with graph networks and reinforcement learning. *arXiv preprint arXiv:1909.11830*, 2019.

[10] Gil Lederman, Markus N Rabe, Edward A Lee, and Sanjit A Seshia. Learning heuristics for automated reasoning through deep reinforcement learning. *arXiv preprint arXiv:1807.08058*, 2018.

[11] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 530–535, 2001.

[12] Pavel Pudlák. Proofs as games. *The American Mathematical Monthly*, 107(6):541–550, 2000.

[13] Daniel Selsam. Neurocuber: training NeuroSAT to make cubing decisions for hard SAT problems.

[14] Daniel Selsam and Nikolaj Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, pages 336–353, 2019.

[15] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[16] Alasdair Urquhart. The depth of resolution proofs. *Studia Logica*, 99(1-3):349, 2011.

[17] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, pages 422–429, 2014.

Deductive Support for Automated Argument Maintenance

Robert C. Kahlert[1], Bettina Berendt[1,2], and Benjamin P. Rode[3]

[1] Department of Computer Science, KU Leuven, Belgium*
[2] Department of Computer Science, TU Berlin, Germany
[3] Cycorp Inc, Austin, TX, USA

Abstract

Automated theorem provers (= ATPs) employ truth maintenance systems (= TMS) to maintain deductive consistency in the face of changing knowledge. However, with broadening the scope from normative branches of knowledge like mathematics into the digital humanities, where abductive and inductive inferences are necessary for knowledge discovery and gap filling, TM strategies have to be broadened beyond the deductive cases as well. Using the example of contemporary art interpretation of Albrecht Altdorfer's *Alexanderschlacht* of 1529, we enumerate some key forms of argumentation found in digital art history; their relationship to C.S. Peirce's three modes of inference; the way changing evidence refutes arguments; and the contribution that deductive theories of argument maintenance can make to alert digital art historians to problems automatically.

1 Introduction

This research is part of an ongoing effort to check the argumentation needs of the humanities against the representational capabilities of common-sense reasoning using ATP.[1] By identifying and filling as far as possibly any gaps, we aspire to advance the state of the digital humanities. As such, this paper has some of the characteristics of a progress report.

2 The logical Needs of Digital Humanities

John Sowa gives the following description of truth maintenance systems and their benefits for nonmonotonic systems of reasoning:

> In effect, a TMS [i.e. a truth maintenance system] is a bookkeeping system for metaleval reasoning about logical dependencies among propositions. For nonmonotonic systems, it can improve efficiency by replacing global consistency checks with local tests in a limited region of the network. [16, 381]

Nonmonotonicity describes the digital humanities to a tee: the predominant mode of historical reasoning in academia is *underdetermined historiography* [17], meaning that discovering new information can prune proposed interpretations, without however bringing the number of interpretations commensurate with the existing evidene to one. Sciences such as art history therefore interpret their works in a context of competing proposals supported by evidence.

At the same time, such an "inference to the best explanation" [11] is not only deductive, but an argument that involves both abductive and inductive steps, to use the terminology of Peirce [16, 389-392]. Thus, other than in Sowa's description, it is not the dependency between propositions, but the dependency between arguments that keeps the overall claims of an interpretation in line with the best evidence.

*Corresponding email: robert.kahlert@gmail.com

[1] At AITP 2019, we presented how to use coordinated scripts [15] to model the narratives explaining symbolic actions, using the World War I centennial commemorations as our exemplar.

3 Arguing about Altdorfer's Alexanderschlacht

Some of the arguments given in analyses of Albrecht Altdorfer's famous painting *Die Alexanderschlacht* (the Battle of Alexander; see p.5) from 1529 [13, 68, Item 290] illustrates how argumentation and argument revision figure in the practice of art history.

Koselleck [9, 19] interprets the concurrent visibility of sun and moon as an indication of the cosmic importance of the battle. Oberstadler [12] views it as an accurate depiction of the sky on that day at that time, courtesy of Altdorfer's astrologer friend Joseph Grünpeck. Goldberg [6, 14] takes it to sympobilze the duration of the battle as lasting a day. Goldberger's case is strengthened by another Altdorfer painting from 1518, the *Battle of Regensburg*, uses that same symbolism, a concurrent sun and moon, to indicate a duration of three days of Charlemagne fighting the Avars. Thus, a single (!) example induces a pattern in Altdorfer's style repertoire for the interpretation of the (later) *Alexander*-painting.

The group of women in festive clothes fleeing the battle field[2] also needs explanation. One historical source that Altdorfer used, Quintus Curtius Rufus, describes (3.9.6b) [14, 40] how in accordance with Persian customs, the womenfolk of Darios III are placed onto the battlefield at the start but later captured during the Macedonian sacking of the Persian camp (3.11.24-26) [14, 44]. The depicted flight from the battle field, however, is implied by the account but not actually narrated and has to be abduced for an argument.

The problem of the women is compounded by their rendering. Though Altdorfer paints the Persians as Renaissance Turks and the Macedonians as German landsknechts [9, 19], the Persian women wear German fashion. The abduced scene and the unclear depiction has led some art historians to argue that the women cannot represent Darius III's womenfolk [8, 234].

Alternatively, Altdorfer may have lacked sources for depicting Turkish women. The military garb of Turkish soldiers were known from fly-leafs of the First Siege of Vienna in 1528. The earliest known illustrations of Turkish women's attire first appeared in Western Europe in the 1580s.[3] Of course, this argument is—like many *terminus post quem* arguments—nonmonotonic and overturned by the discovery of an appropriate new source.

The interpretation is complicated by the painting's history: it was trimmed and the originally German inscription overpainted with a Latin one. There is a bill for restoration work done in 1658 by Johann de Pey on the *Alexanderschlacht* in the Bavarian duchy's archives (*Hofzahlamt*) [2, 14], but only an abductive leap can connect Pey's restoration to these specific modifications.

4 Modeling Argumentation: A Progress Report

In forensic situations, Aristotle observed (*Rhetoric* 1.2.8; 13f; et al. [5]), people will not give syllogistic proofs, but rather enthymemes: proof fragments that elide shared assumptions and skip "obvious" steps in the proof sequence. Art historic argumentation is similarly succinct. In addition, as the suspicion of de Pey overpainting the German inscription as part of his "restoration" indicates, the understood parts are often not directly supported by historical evidence, but require abducing new entities—events, visual references or similar—to bridge the inferential gaps.

Though the art historical claims need to discharge eventually as FOL sentences for ATP to find deductive footing, this is not the appropriate level of modeling. Rather, one has to

[2] The scene takes place to the left and below Darius III's scythe chariot.

[3] Cf. the Italian codex *I Turchi* (Codex Vindobonensis 8626, 1585-1591), the drawings by Johannes Lewenklaw (Codex Vindobonensis 8615, 1585) or the *Turkish Book of Manners* (1595), now in Kassel.

model the claims which then expand into the appropriate sentences, together with any needed scaffolding entities as well.

Furthermore, we model arguments like the ones above in ResearchCyc [10] [3] using microtheories as contexts [7]. Contexts allow to resuse the shared assumptions as well as isolate the actual interpretation differences. Modeled claims can be analyzed in their description context and discharged as FOL sentences in a dependent context, akin to conversational implicature. Though such a separation of context allows some deductive TMS at the projected level, this separation is no substitute for maintence of the argument claims.

That separation comes at a price, however: At the level of FOL TMS, the clash between contradictory information can be detected syntactically: If p is already asserted, then $\neg p$ cannot be added (and vice versa). The connection is far less direct at the level of claims in argumentation. As noted above, perhaps Altdorfer drew Persian women in German dress because he lacked references. One way in which such references could be lacking—and weaker conditions may also suffice—is that no information about Turkish female clothing was available in Western Europe before 1529 at all.

The problem is not reasoning about the absence of illustrations depicting items of certain types from before the 1580s,[4] but recognizing the need to trigger reconsideration cheaply. Lacking the syntactic similarity between p and $\neg p$ as a simple tip-off, such an AMS has to validate the assumptions under its management by running queries, in the limit after every new fact added—an expensive solution that is a stop-gap measure at best. At least, due to the separation between the claims and the representation, it is possible to notate which queries need to be run.

The situation is very similar for inductive knowledge, such as distributions of various features over a population. In [18], we used WEKA's J45 [4] as an external decision tree inducer to generate the *modus operandi* probabilities, which Cyc accessed via removal-module at inference time. Efficiently detecting that such information has gone stale and needs to be re-exported and re-generated remains a desideratum.

5 Outlook

In a deductive truth-maintenance system, the repair strategy of the bookkeeping system is either not to accept the new information or to retract the conflicting propositions as new information arrives. In a argumentation maintenance system, it is not clear that retraction is the correct behavior. However, there remains a significant body of problems that can be flagged deductively, a form of conceptual spell-checking for digital humanities researchers that would already be a significant improvement over the current state of the art, the *Zettelkasten*, whether implemented in paper or document files.

[4] ResearchCyc supports reasoning about unavailable information via the relation `unknownSentence` in queries; cf. [1].

References

[1] Alan Belasco, Jon Curtis, Robert C. Kahlert, Charles Klein, Corinne Mayans, and Pace Reagan. Representing knowledge gaps effectively. In Dimitris Karagiannis and Ulrich Reimer, editors, *Practical Aspects of Knowledge Managment: 5th International Conference*. Springer Verlag, 2004.

[2] Ernst Buchner. *Albrecht Altdorfer und sein Kreis. Gedächtnisausstellung zum 400. Todestag*. Wolf & Sohn, Universitäts-Buchdruckerei, München, 2nd edition, 1938.

[3] Cycorp. Researchcyc, December 2018.

[4] Eibe Frank, Mark A. Hall, and Ian H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Ann Arbor, MI, 4th edition, 2016.

[5] J.H. Freese. *Aristotle: Rhetoric*, volume 22 of *Aristotle in 23 Volumes*. Harvard University Press, Cambridge – London, 1922.

[6] Gisela Goldberg. *Die Alexanderschlacht und die Historienbilder Herzog Wilhelms IV von Bayern*. Bayrische Staatsgemäldesammlung, München, 2002.

[7] Ramanathan V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford, 1991.

[8] Rose-Marie Hagen and Rainer Hagen. *Bildbefragungen: 100 Meisterwerke im Detail*. Bibliotheca Universalis. Taschen, Cologne, 2019.

[9] Reinhart Kosselleck. *Vergangene Zukunft: Zur Semantic geschichtlicher Zeiten*. Number 757 in Suhrkamp Taschenbuch Wissenschaft. Suhrkamp, Frankfurt am Main, 1st edition, 1989.

[10] Douglas B Lenat and R. V Guha. *Building large knowledge-based systems: representation and inference in the Cyc project*. Addison-Wesley Pub. Co., Reading, Mass., 1990.

[11] Peter Lipton. *Inference to the Best Explanation*. International Library of Philosophy. Routledge, London — New York, 2nd edition, 2004.

[12] Margit Oberstadler. *Der Wald in der Malerei und der Graphik des Donaustils*. Ars Viva. Böhlau, Vienna, 2006.

[13] Franz von Reber. *Katalog der Gemälde-Sammlung der Königlichen Älteren Pinakothek in München*. Bruckmann, München, 1904.

[14] Quintus Curtius Rufus. *The History of Alexander*. Penguin Classics. Penguin, London, 3 edition, 2004.

[15] R.C. Schank and R.P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry Into Human Knowledge Structures*. The Artificial Intelligence Series. Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.

[16] John F. Sowa. *Knowledge Representation. Logical, Philosophical and Computational Foundations*. Brooks-Cole, Pacific Grove, CA, 1 edition, 2000.

[17] Aviezer Tucker. *Our Knowledge of the Past*. Cambridge University Press, Cambridge, UK, 2004.

[18] Michael Witbrock, Elizabeth Coppock, and Robert C. Kahlert. Uniting a priori and a posteriori knowledge : A research framework. 2009.

Figure 1: Albrecht Altdorfer, Die Alexanderschlacht (1529), now in the Pinakothek in Munich.

ForTheL for Type Theory

Adrian De Lon, Peter Koepke, and Anton Felix Lorenzen

University of Bonn

Naproche-SAD [Nap] accepts mathematical input texts in the ForTheL language, translates them into first-order logic and checks for logical correctness. The controlled natural language ForTheL allows statements close to the language of mathematical textbooks. The system supports natural structurings of proofs, using strong ATPs to deal with tedious details. Altogether, it is possible to write proof documents which are *natural* and immediately readable by mathematicians. Naproche-SAD has been presented at the two previous editions of AITP.

Other proof assistants have realized that broader adoption in the mathematical community would require a *readable* input language. Mizar and Isabelle have declarative proof languages that mimic ordinary proof structures and provide some readability. It seems, however, that even more naturality is required for wider acceptance [Hal19].

Since most popular and successful proof systems are based on type theories this calls for a CNL for type theory, with additional automation to reach acceptable proof structures and granularities. Our presentation will be based on the project described in the final section below.

1. The ForTheL language. ForTheL is a (subset of) the natural language of mathematics and therefore amenable to a type-theoretical analysis [Gan13,Ran94]. Common nouns like *man* or *number* can be viewed as types, and proper nouns like *Aristotle* or 15 as inhabitants of those types. Predicates or adjectives like *mortal* or *odd* can be used to modify types. Natural language quantification commonly uses nouns with implicit variables as in *"Every man is mortal"*.

Similarly, ForTheL's parser categorizes phrases as nouns, verbs, and adjectives and provides subparsers to identify specific phrases (= patterns) in the input. The ForTheL language centers around *notions*, which can be viewed as weak types. Notions are introduced as noun phrases by `Signature` and `Definition` commands:

```
Signature. A natural number is a notion.
Definition. Assume that n is a natural number.
A divisor of n is a natural number m such that ...
```

2. First-order parsing in Naproche-SAD. Linguistically, this ForTheL text introduces a type `naturalNumber` and a dependent type `divisorOf(x)`. The patterns `[natural, number]` and `[divisor, of, ?]` are stored as assignments `[natural, number] → naturalNumber(x)` and `[divisor, of, ?] → divisorOf(x,y)` of word patterns to first-order predicate symbols and are used in the further parsing of the text. A parser `notion` then looks for previously defined notions, allowing us to parse `0 is a natural number` as its first-order equivalent `naturalNumber(0)`. Similarly, the statement `Every natural number has a divisor` leads to `forall x (naturalNumber(x) -> exists y divisorOf(x,y)`.

3. Towards a type-theoretic parsing. First-order parsing views notions as unary predicates and dependent notions as n-ary predicates with $n \geq 2$, corresponding to standard first-order renderings of type-theoretic formulas. By a simple syntactic reification we turn the predicates into a type `naturalNumber` and a dependent type `divisorOf(x)`. Now `0 is a natural number` can be translated to the judgment `0 : naturalNumber` and `Every natural number has a divisor` to `(forall x : naturalNumber) (exists y : divisorOf(x))`.

4. Type-theoretic foundations and axiomatic style. Type theory ideally builds up the mathematical edifice from first principles. This creates types modelling common mathematical structures in an intuitionistic and constructive framework. Many mathematical properties and

proofs can however be understood axiomatically, proceeding from assumed given types and their properties to theorems and proofs. Locally this is also true of type-theoretic libraries like LEAN's mathlib [Com].

5. ForTheL, mathlib and fabstracts. We have reformulated typical entries of mathlib in ForTheL: the introduction of groups and commutative groups, e.g., in the mathlib file `groups.lean` reads as follows:

```
class group (A : Type u) extends monoid A, has_inv A :=
    mul_left_inv : forall a : A, inv a * a = 1
class comm_group (A : Type u) extends group A, comm_monoid A
```

and can be reformulated in ForTheL, using some LaTeX pretty printing:

Definition 1. *A* group *is a monoid* α *such that* α *is a type with inverses and for all* $a : \alpha$ $a^{-1,\alpha} *^{\alpha} a = 1^{\alpha}$.

Definition 2. *A* commutative group *is a group that is a commutative monoid.*

The ForTheL translation of such statements can straightforwardly be modified to generate valid LEAN input. Interestingly, the ForTheL text describing some elementary lemmas on groups and similar structures immediately proof-checks in the first-order Naproche-SAD system with E as an external ATP.

A CNL for CiC.

We have recently started working on a new ForTheL-like CNL as a front-end for the calculus of inductive constructions (CiC). We have reconsidered some architectural decisions, which should allow for faster performance, more flexibility and maintainability, and improved error messages. Unlike the current implementation of Naproche-SAD, we will have a clear data structure representing the parse tree, so that the translation process becomes more transparent. This CNL will feature new phrases that will allow the user to interact with the underlying type theory in a natural manner, departing from the mostly first-order semantics of ForTheL. One of our goals is to write documents that translate to LEAN code equivalent to some fragments of mathlib and fabstracts.

Our presentation will discuss relevant mathematical linguistics and demonstrate a prototype implementation of the new CNL. We expect to be able to show a new parser and a translator from a mostly declarative fragment of natural mathematical language to CiC. We will also discuss details of implementation and design of such a language, and show some mathematical examples.

References

[Com] LEAN Community. mathlib. https://github.com/leanprover-community/mathlib.

[Gan13] Mohan Ganesalingam. *The language of mathematics.* Springer, 2013.

[Hal19] Thomas Hales. Mathematical Definitions, Formally Speaking. `https://www.icms.org.uk/downloads/bigproof/Hales.pdf`, 2019.

[Nap] Naproche-SAD in Isabelle-jEdit. https://files.sketis.net/Isabelle_Naproche-20190611/.

[Ran94] Aarne Ranta. Type theory and the informal language of mathematics. In *Types for Proofs and Programs. Types 1993*, pages 352–365. Springer, 1994.

Isomorphism Revisited

David McAllester

December 6, 2019

Abstract

Isomorphism is central to the structure of mathematics and has been formalized in various ways within dependent type theory. All previous treatments have done this by replacing quantification over sets with quantification over groupoids of some form — categories in which every morphism is an isomorphism. Quantification over sets is replaced by quantification over standard groupoids in the groupoid model, by quantification over infinity groupoid in homotopy type theory, and by quantification over morphoids in the morphoid model. Our treatment in [6] is based on the intuitive notion of sets as collections without internal structure. Quantification over sets remains as quantification over sets. Isomorphism and groupoid structure then emerge from simple but subtle syntactic restrictions on set-theoretic language. This approach more fully unifies the classical ZFC foundations with a rigorous treatments of isomorphism, symmetry, canonicality, functors, and natural transformations. This is all done without reference to category theory.

1 Introduction

Isomorphism is central to the structure of mathematics. Mathematics is organized around concepts such as graphs, groups, topological spaces and manifolds each of which is associated with a notion of isomorphism. Each concept is associated with a classification problem — can we enumerate the instances of a given concept *up to isomorphism*. We also have the related notions of symmetry and canonicality. There is no canonical point on a geometric circle — any point can be mapped to any other point by rotating the circle. A rotation of the circle is an isomorphism of the circle with itself — a symmetry or automorphism. Similarly, there is no canonical basis for a finite dimensional vector space. For any basis there is a symmetry (automorphism) of the vector space which moves the basis to a different basis — a situation precisely analogous to a point on a circle. Isomorphism is also central to understanding representation. A group can be represented by a family of permutations. Different (non-isomorphic) families of permutation can represent the same group (up to isomorphism).

At first isomorphism seems simple. The notion of isomorphic graphs, and the intuition that two isomorphic graphs are "the same", seems intuitively clear to essentially anyone who encounters the concept. Indeed, for a broad class of concepts the notion of isomorphism is easily defined. More specifically we can consider concepts defined by a carrier set of "points" plus predicates, relations and functions providing structure on those points. A graph consists of a set of nodes (points) plus an edge relation on the nodes. For concepts defined by a carrier set plus structure, two instances are isomorphic if there exists a bijection between their points which identifies their structure — which "carries" the structure of one to the structure of the other. This notion of isomorphism is easily formalized for concepts defined as the models of a given (higher order) signature where a signature is a set of predicate symbols, relation symbols and function symbols operating over a carrier set of points.

But general mathematics is carried out in a language richer than that defined by a single higher order signature. Mathematical statement typically involve several different instances of several different concepts. For example, we can abstract a document — a sequence of words — to a multiset (bag) of words. When we do this we understand that structure has been lost. It is more subtle than simply noting that different sequences can map to the same bag. A bag fundamentally has less structure than a sequence. The grammatical (well typed) statements about a bag are more restricted than the grammatical statements about a sequence. We cannot talk about the first element of a bag.

Dependent type theory [1] is a formal system for specifying interfaces to objects and can be used as a formal foundation for mathematics [2]. Unlike set theory, type theory handles concepts (types) with a specified interface to the instances of each concept. Type theory allows for statements relating concepts and their instances in a way that mirrors natural mathematical language.

Isomorphism has been formalized in dependent type theory using the groupoid model [3]. The groupoid model replaces quantification over sets with quantification over groupoids — categories in which every morphism is an isomorphism. Homotopy type theory replaces quantification over sets with quantification over a form of infinity groupoid related to algebraic topology. The Morphoid model achieves compositionality by replacing quantification over sets with quantification over "morphoids" [5].

In [6] we achieve a treatment of isomorphism which preserves the intuitive concept of a set as a collection without internal structure — quantification over sets remains as quantification over sets. Isomorphism and groupoid structure then emerge naturally from simple but subtle syntactic restrictions on set-theoretic language. Functors and natural transformations also emerge naturally without any explicit introduction of groupoids or category theory.

Kevin Buzzard, in his talk at AITP 2018, described the understanding of canonical isomorphisms as a human superpower. Hopefully the approach to isomorphism given in [6] will facilitate the automation of this superpower.

References

[1] Per Martin Löf. An intuitionistic theory of types: predicative part. In *Logic Colloquium '73 (Bristol, 1973)*, volume 80 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1975.

[2] B. Barras, S. Boutin, C. Cornes, J. Courant, J.C. Filliatre, E. Gimenez, H. Herbelin, G. Huet, C. Munoz, C. Murthy, et al. The coq proof assistant reference manual: Version 6.1. INRIA Research Report, 1997.

[3] Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*. Oxford Univ. Press, New York, 1998.

[4] HoTT-Authors. Homotopy type theory, univalent foundations of mathematics. http://hottheory.files.wordpress.com/2013/03/hott-online-611-ga1a258c.pdf, 2013.

[5] David McAllester. Set-theoretic type theory. *CoRR*, abs/1407.7274, 2018.

[6] David McAllester. Isomorphism Revisited. Scheduled for announcement on ArXiv 1:00 GMT, Monday December 9, 2019.

Learning Semantic Annotations for LaTeX Documents

Dennis Müller*and Cezary Kaliszyk

Department of Computer Science
University of Innsbruck, Austria

1 Introduction

In the last decades, the formalization of mathematical knowledge, and the verification and automation of formal proofs, has become increasingly popular. Formal methods nowadays are not just used by computer scientists to verify software and hardware as well as in program synthesis, but ave also drawn the interest of an increasing number of research mathematicians. By now, there is a plurality of systems available, each with its own growing library of formalized mathematics.

However, many mathematicians complain that

- formal systems are difficult to learn and use, even if one is well acquainted with the (informal) mathematics involved,
- they require a level of detail in proofs that is prohibitive even for "obvious" conclusions,
- their libraries are difficult to grasp without already being familiar with the system's language, conventions and functionalities.

Consequently, the utility of formalizing mathematical results can be too easily (and too often *is*) dismissed in light of the additional time and work required for non-experts. This is despite the fact that many services available for formal mathematics are already enabled by *semi*-formal (or *flexiformal*) representations, such as semantic annotations in natural language texts, or formal representations containing opaque informal expressions (see e.g. [Koh13], [Lan11a], [Ian17], [Koh+17b], [CS17], [Deh+16]). Therefore, we need to invest into methods for bridging the gap between informal mathematical practice and (semi-)formal mathematics.

We want to contribute to such a bridge between informal and (semi-)formal documents, by **developing a framework** using symbolic and machine learning techniques that

1. **automatically adds formal semantic annotations** to informal mathematics where possible, and
2. **highlights ambiguities** where not, in order to encourage clarification from a user.

Michael Kohlhase developed the sTeX package [Koh08] for LaTeX, specifically for annotating mathematical documents with structural and formal semantics. In particular, sTeX is based on an OMDoc [Koh06] ontology, which is foundation-agnostic in the sense that it does not favor a specific foundation (such as type or set theories) over any other. This approach is consequently best suited for semantifying informal documents, where foundations are often unspecified, left implicit or switched fluently. Furthermore, sTeX allows markup both on the level of mathematical expressions as well as on a structural level, such as declarations, definienda/definientia and theorems. Consequently, sTeX can serve as an ideal target for this goal.

*The first author and this work are supported by a postdoc fellowship of the German Academic Exchange Service (DAAD)

As a first approach, we will use the *SMGloM* [Koh14] semantic glossary of mathematics, which contains hundreds of sTEX-annotated concepts and definitions, providing LATEX-macros for their symbolic *notations* (i.e. presentation as pure LATEX) as well as introducing logical identifiers for semantically referencing concepts in natural language texts.

Individual entries in the glossary are collected in individual, .tex-files, which can be compiled into (disambiguated) OMDoc. The individual files are connected via a module system provided by the sTEX-package using the logical identifiers.

Consequently, the *SMGloM* library can serve as an ideal data set for supervised learning to 1. disambiguate formal expressions in LATEX using SMGloM macros, and 2. automatically reference SMGloM entries in natural language paragraphs.

sTEX declaration	% equality as a flexary infix operator \symdef[name=equal,gfc=N2]{eqFN}{\mathrel{=}} \symdef[name=equal,assocarg=1]{eq}[1]{\assoc[p=300]\eqFN{#1}}
sTEX references	We call two mathematical objects a and b \trefi{equal}, (written $\eq{a,b}$), iff there are no properties that discern them.
OMDoc for \eq{a,b}	<OMA> <OMS cd="http://mathhub.info/smglom/mv/equal.omdoc?equal" name="equal"/> <OMV name="a"/> <OMV name="b"/> </OMA>

\symdef introduces a new mathematical concept with globally unique identifier (see third row), \trefi allows for referencing it, the formal expression $a = b$ is disambiguated in the resulting OMDoc.

sTEX itself is integrated, and shares an underlying OMDoc ontology, with the Mmt system [RK13; HKR12; Rab17] – a foundation-independent meta-framework and API for knowledge management services. This integration makes the generic services provided by MMT available to informal mathematical texts. As a next step, we will explore the possibility of using Mmt's generic type checking component to formally verify the disambiguated expressions obtained from informal mathematical texts in the step above. This would result in a rudimentary type checker integrated into LATEX, similar to *Naproche* [Cra+09] and related systems.

Additionally, several theorem prover libraries have been translated to OMDoc and integrated in the Mmt system, e.g. [Koh+17a; MRS19] (for a detailed overview, see [Mül19] and [KR20]). This allows extending our training data to existing data sets for automated formalization (e.g. [KUV17a; KUV17b; WKU18]), potentially extending the SMGloM automatically, and provides an attractive avenue for subsequent research by using *alignments* [Mül19; Mül+17] between SMGloM and formal libraries to verify informal mathematics using several state-of-the-art theorem prover systems.

We expect the work to result in a deeper integration of formal methods in the workflows of working mathematicians (e.g. via proper integration in LATEX-IDEs), making formal methods and their advantages accessible to non-experts in STEM fields. Hopefully, this will vastly increase both their ubiquity outside the formal mathematics community and the general amount of formal mathematics available, thus also benefiting e.g. the formal abstracts and related projects.

References

[Cra+09] M. Cramer, B. Fisseni, P. Koepke, D. Kühlwein, B. Schröder, and J. Veldman. "The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts". In: *Controlled Natural Language*. Ed. by N. Fuchs. Springer, 2009, pp. 170–186.

[CS17] J. Corneli and M. Schubotz. "math.wikipedia.org: A vision for a collaborative semiformal, language independent math(s) encyclopedia". English. In: *AITP 2017. The Second Conference on Artificial Intelligence and Theorem Proving*. 2017, pp. 28–31.

[Deh+16] P.-O. Dehaye et al. "Interoperability in the OpenDreamKit Project: The Math-in-the-Middle Approach". In: *Intelligent Computer Mathematics 2016*. Conferences on Intelligent Computer Mathematics (Bialystok, Poland, July 25, 2016–July 29, 2016). Ed. by M. Kohlhase, M. Johansson, B. Miller, L. de Moura, and F. Tompa. LNAI 9791. Springer, 2016. URL: https://github.com/OpenDreamKit/OpenDreamKit/blob/master/WP6/CICM2016/published.pdf.

[Geu+17] H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke, eds. *Intelligent Computer Mathematics*. Conferences on Intelligent Computer Mathematics. LNAI 10383. Springer, 2017. DOI: 10.1007/978-3-319-62075-6.

[HKR12] F. Horozal, M. Kohlhase, and F. Rabe. "Extending MKM Formats at the Statement Level". In: *Intelligent Computer Mathematics*. Ed. by J. Campbell, J. Carette, G. Dos Reis, J. Jeuring, P. Sojka, V. Sorge, and M. Wenzel. Springer, 2012, pp. 64–79.

[Ian17] M. Iancu. "Towards Flexiformal Mathematics". PhD thesis. Bremen, Germany: Jacobs University, 2017. URL: https://opus.jacobs-university.de/frontdoor/index/index/docId/721.

[Koh06] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2)*. Lecture Notes in Artificial Intelligence 4180. Springer, 2006.

[Koh08] M. Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304.

[Koh13] M. Kohlhase. "The Flexiformalist Manifesto". In: *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. Ed. by A. Voronkov, V. Negru, T. Ida, T. Jebelean, D. Petcu, S. M. Watt, and D. Zaharie. Timisoara, Romania: IEEE Press, 2013, pp. 30–36. URL: http://kwarc.info/kohlhase/papers/synasc13.pdf.

[Koh14] M. Kohlhase. "A Data Model and Encoding for a Semantic, Multilingual Terminology of Mathematics". In: *Intelligent Computer Mathematics 2014*. Conferences on Intelligent Computer Mathematics (Coimbra, Portugal, July 7, 2014–July 11, 2014). Ed. by S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban. LNCS 8543. Springer, 2014, pp. 169–183. URL: http://kwarc.info/kohlhase/papers/cicm14-smglom.pdf.

[Koh+17a] M. Kohlhase, D. Müller, S. Owre, and F. Rabe. "Making PVS Accessible to Generic Services by Interpretation in a Universal Format". In: *Interactive Theorem Proving*. Ed. by M. Ayala-Rincón and C. A. Muñoz. Vol. 10499. LNCS. Springer, 2017. URL: http://kwarc.info/kohlhase/submit/itp17-pvs.pdf.

[Koh+17b] M. Kohlhase, T. Koprucki, D. Müller, and K. Tabelow. "Mathematical models as research data via flexiformal theory graphs". In: *Intelligent Computer Mathematics (CICM) 2017*. Conferences on Intelligent Computer Mathematics. Ed. by H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke. LNAI 10383. Springer, 2017. DOI: 10.1007/978-3-319-62075-6. URL: http://kwarc.info/kohlhase/papers/cicm17-models.pdf.

[KR20] M. Kohlhase and F. Rabe. "Experiences from Exporting Major Proof Assistant Libraries". 2020. URL: https://kwarc.info/people/frabe/Research/KR_oafexp_20.pdf.

[KUV17a] C. Kaliszyk, J. Urban, and J. Vyskočil. "Automating Formalization by Statistical and Semantic Parsing of Mathematics". In: *Interactive Theorem Proving*. Ed. by M. Ayala-Rincón and C. A. Muñoz. Cham: Springer International Publishing, 2017, pp. 12–27.

[KUV17b] C. Kaliszyk, J. Urban, and J. Vyskocil. "System Description: Statistical Parsing of Informalized Mizar Formulas". In: 2017. DOI: 10.1109/synasc.2017.00036.

[Lan11a] C. Lange. "Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration". Also available as a book [Lan11b]. PhD thesis. Jacobs University Bremen, 2011. URL: https://svn.kwarc.info/repos/swim/doc/phd/phd.pdf.

[Lan11b] C. Lange. *Enabling Collaboration on Semiformal Mathematical Knowledge by Semantic Web Integration*. Studies on the Semantic Web 11. Heidelberg and Amsterdam: AKA Verlag and IOS Press, 2011. URL: http://www.semantic-web-studies.net.

[MRS19] D. Müller, F. Rabe, and C. Sacerdoti Coen. "The Coq Library as a Theory Graph". accepted at CICM 2019. 2019.

[Mül+17] D. Müller, T. Gauthier, C. Kaliszyk, M. Kohlhase, and F. Rabe. "Classification of Alignments between Concepts of Formal Mathematical Systems". In: *Intelligent Computer Mathematics (CICM) 2017*. Conferences on Intelligent Computer Mathematics. Ed. by H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke. LNAI 10383. Springer, 2017. DOI: 10.1007/978-3-319-62075-6. URL: http://kwarc.info/kohlhase/papers/cicm17-alignments.pdf.

[Mül19] D. Müller. "Mathematical Knowledge Management Across Formal Libraries". PhD thesis. Informatics, FAU Erlangen-Nürnberg, Oct. 2019. URL: https://kwarc.info/people/dmueller/pubs/thesis.pdf.

[Rab17] F. Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* 27.6 (2017), pp. 1753–1798.

[RK13] F. Rabe and M. Kohlhase. "A Scalable Module System". In: *Information and Computation* 230.1 (2013), pp. 1–54.

[WKU18] Q. Wang, C. Kaliszyk, and J. Urban. "First Experiments with Neural Translation of Informal to Formal Mathematics". In: *CoRR* abs/1805.06502 (2018). arXiv: 1805.06502. URL: http://arxiv.org/abs/1805.06502.

LiFtEr: Language to Encode Induction Heuristics

Yutaka Nagashima[1,2]

[1] CIIRC, Czech Technical University in Prague,
Prague, Czech Republic
[2] Department of Computer Science, University of Innsbruck,
Innsbruck, Tyrol, Austria

Abstract

Proof assistants, such as Isabelle/HOL, offer tools to facilitate inductive theorem proving. Isabelle experts know how to use these tools effectively; however, there is a little tool support for transferring this expert knowledge to a wider user audience. To address this problem, we present our domain-specific language, `LiFtEr`. `LiFtEr` allows experienced Isabelle users to encode their induction heuristics in a style independent of any problem domain. `LiFtEr`'s interpreter mechanically checks if a given application of induction tool matches the heuristics, thus automating the knowledge transfer loop.

1 Induction in Isabelle/HOL

Isabelle offers the `induct` proof method to handle inductive problems. Proof methods are the Isar syntactic layer of LCF-style tactics. For example, consider the following reverse functions, `rev` and `itrev`, from literature [3]:

```
primrec rev::"'a list =>'a list" where
  "rev  []      = []"
| "rev (x # xs) = rev xs @ [x]"
fun itrev::"'a list =>'a list =>'a list" where
  "itrev  []    ys = ys"
| "itrev (x#xs) ys = itrev xs (x#ys)"
```

where `#` is the list constructor, and `@` appends two lists into one. One can prove the equivalence of these reverse functions in multiple ways using the `induct` method:

```
lemma prf:"itrev xs ys = rev xs @ ys" apply(induct xs ys rule:itrev.induct) by auto
```

`prf` applies functional induction on `itrev` by passing an auxiliary lemma, `itrev.induct`, to the `rule` field. There are other lesser-known techniques to handle difficult inductive problems using the `induct` method, and sometimes users have to develop useful auxiliary lemmas manually; however, for most cases the problem of how to apply induction boils down to the the following question: *what arguments do you pass to the* ***`induct`*** *method?*

Isabelle experts often apply induction heuristics to answer this question and decide what arguments to pass to the `induct` method; however, they did not have a systematic way to encode such heuristics, which made it difficult for new users to learn how to apply induction effectively.

2 LiFtEr: Language to Encode Induction Heuristics

We address this problem with our domain-specific language, `LiFtEr`. `LiFtEr` allows experienced Isabelle users to encode their induction heuristics in a style independent of problem domains.

LiFtEr's interpreter mechanically checks if a given application of induction is compatible with the induction heuristics written by experienced users.

We designed LiFtEr to encode induction heuristics as assertions on invocations of the induct method in Isabelle. An assertion written in LiFtEr takes a triple of a proof goal at hand, its underlying proof state, and the arguments passed to the induct method to prove the goal. When one applies a LiFtEr assertion to an invocation of the induct method, LiFtEr's interpreter returns a boolean value as the result of the assertion applied to the triple.

The goal of a LiFtEr programmer is to write assertions that implement reliable heuristics. A heuristic encoded as a LiFtEr assertion is reliable when it satisfies the following two properties: first, the LiFtEr interpreter is likely to evaluate the assertion to true when the arguments of the induct method are appropriate for the given proof goal. Second, the interpreter is likely to evaluate the assertion to false when the arguments are inappropriate for the goal.

The following is an example assertion written in LiFtEr:

```
  ∃ r1 : rule. True
→
  ∃ r1 : rule.
    ∃ t1 : term.
      ∃ to1 : term_occurrence ∈ t1 : term.
          r1 is_rule_of to1
        ∧
          ∀ t2 : term ∈ induction_term.
            ∃ to2 : term_occurrence ∈ t2 : term.
              ∃ n : number.
                  is_nth_argument_of (to2, n, to1)
                ∧
                  t2 is_nth_induction_term n
```

As a whole this LiFtEr assertion checks if the following holds: if there exists a rule, $r1$, in the rule field of the induct method, then there exists a term $t1$ with an occurrence $to1$, such that $r1$ is derived by Isabelle when defining $t1$, and for all induction terms $t2$, there exists an occurrence $to2$ of $t2$ such that, there exists a number n, such that $to2$ is the nth argument of $to1$ and that $t2$ is the nth induction terms passed to the induct method.

prf is compatible with this heuristic: there is an argument, itrev.induct, in the rule field, and the occurrence of its related term, itrev, in the proof goal takes all the induction terms, xs and ys, as its arguments in the same order.

3 Conclusion

We presented LiFtEr and its example assertion. LiFtEr is a domain-specific language in the sense that we developed LiFtEr to encode induction heuristics; however, heuristics written in LiFtEr are usually not specific to any problem domain, because LiFtEr's language construct is not specific to any variable names, types, or constants. This absence encourages LiFtEr users to encode heuristics that are not specific to any problem domains but are applicable to many domains. To the best of our knowledge, LiFtEr is the first domain-specific language that allows us to encode induction heuristics as programs. We released a working prototype of the LiFtEr interpreter and six example assertions at GitHub [2]. And a more comprehensive explanation of LiFtEr's grammar is provided in our paper [1].

Acknowledgments

This work was supported by the European Regional Development Fund under the project AI & Reasoning (reg. no.CZ.02.1.01/0.0/0.0/15_003/0000466).

References

[1] Yutaka Nagashima. LiFtEr: Language to encode induction heuristics for Isabelle/HOL. In *Programming Languages and Systems - 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, pages 266–287, 2019.

[2] Yutaka Nagashima et al. data61/PSL. https://github.com/data61/PSL/releases/tag/v0.1.3-alpha, 2019.

[3] Tobias Nipkow and Gerwin Klein. *Concrete Semantics - With Isabelle/HOL*. Springer, 2014.

Property Invariant Neural Network for Embedding Formulas in CNF*

Miroslav Olšák[1], Cezary Kaliszyk[1], and Josef Urban[2]

[1] University of Innsbruck, Innsbruck, Austria
[2] Czech Technical University in Prague, Czechia

Abstract

Most approaches for embedding formulas use learned embeddings for the predefined function symbols and constants. That is however not well suited for handling new definitions such as Skolem constants. We present a network that does not take into account global names of functions, considering names always local inside a single query. This approach is suitable for the classification of enough clauses at the same time so that the network can infer the features of the symbols only from their usage.

1 Architecture Outline

The network receives a set of clauses and outputs an embedding vector for every clause, every function and relational symbol, and every literal and every subterm occurring in the clauses. To produce them, the input set of clauses is first encoded into a graph in which every object of the three types above is represented by s node. Od particular interest is the type (3) of subterms and literals since we use perfect sharing among the subterms, that is, if two terms or literals are identical (e.g. same variables), they are represented by a single node.

The graph is provided with two types of edges. There are binary edges between clauses and literals describing what literals belong to what clauses. The second type of edges is 4-ary, containing one symbol node and three term nodes. These edges represent the structure of the terms including the argument orders of functions.

We initialize the graph based on basic node properties only – the origin of the clause, whether a term is a variable, etc. and then we perform several (constant number) message-passing layers on the graph. The output of the network is the output of the last message-passing layer.

By design of the graph, the network is invariant under symbol renaming, reordering the clauses, or reordering the literals in a clause. The invariance under negation is achieved by carefully handling the embeddings of symbols. If a vector (embedding) e represents a relational symbol R, then $-e$ represents the relational symbol $\neg R$. The message passing is designed so that this property is preserved through the layers.

2 Experiments

We have conducted three experiments with the network. The first one is for guiding a tableaux connection prover leanCoP [3], in a reinforcement learning setup similar to rlCoP [2]. The prover obtains a set of clauses on the input, and it tries to form a case analysis tree that would prove a contradiction. The network obtains all the axioms from the input (that do not change), and

*MO and CK were supported by the ERC Project *SMART* Starting Grant no. 714034. JU was funded by the *AI4REASON* ERC Consolidator grant nr. 649043, the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

also the current tree, and outputs an evaluation of the current state (value) and probability distribution (policy) of available actions; the actions are represented by literals in the axioms. First, we ran leanCoP with random guidance on Miz40 dataset to get training data, then, we trained the network's value and policy on a training part of the solved problems (90% of them), and then we ran leanCoP guided by Monte Carlo Tree Search (MCTS) based on the network's estimation. In the MCTS we expand each node 200 times before making a bigstep, and the prover has a limit of 200 steps to prove the theorem.

The random prover solved 4595 training and 510 testing problems. The MCTS prover with network guidance then solved 11978 training and 1322 testing problems in the first iteration and 12648 training and 1394 in the second one. This does not reach the results of the original rlCoP, however, it seems to be rather caused by stricter limits despite better predictions.

Our second experiment is premise selection on DeepMath dataset [1], in this experiment, we consider a whole premise with all its candidate premises as a single query and the network can, therefore, use the other premise candidates to determine whether a particular candidate is positive or negative. The network achieved around 80% testing accuracy on this task.

The third experiment is also performed on DeepMath dataset with an atypical objective. Since our network uses only the structure of the formulae and returns also embeddings of the symbols, we trained it to predict the symbol names in the formula and then tested it on the testing part of the dataset. The network achieved around 65% testing accuracy on this task.

3 Related work

Graph Neural Networks for formula embedding invariant under variable renaming were previously used in the FormulaNet [5] mainly for experiments with higher order logic. A different invariance property was proposed in a network for propositional calculus in the NeuroSat [4]. This network is invariant under negation, order of clauses, and order of literals in clauses, however, this is restricted to propositional logic, where no quantifiers and variables are present.

References

[1] Alexander A. Alemi, François Chollet, Niklas Eén, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike V. Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2235–2243, 2016.

[2] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olšák. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 8836–8847, 2018.

[3] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.

[4] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[5] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information*

Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, pages 2783–2793, 2017.

Learning theorem proving through self-play

Stanisław Purgał

University of Innsbruck, Innsbruck, Tirol, Austria
stanislaw.purgal@uibk.ac.at

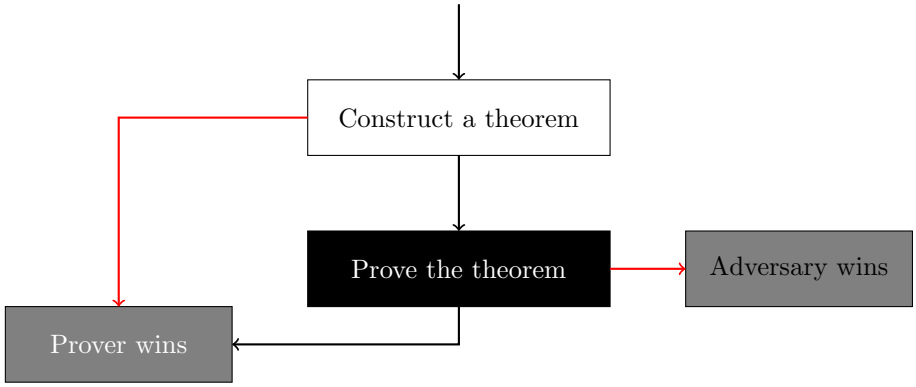
1 Introduction

This work attempts to apply the AlphaZero algorithm [4] to theorem proving. Following the philosophy of learning without using any human–generated datasets we attempt to learn to prove theorems without using any database of proofs or theorems. This is different from other attempts at ML-guided theorem proving in [2], [3], [1].

The only input we expect before starting training is the set of axioms we can use in our proofs — no theorems or conjectures.

2 The theorem-construction game

In the game we are using to learn theorem proving, one player constructs a provable theorem and the other player tries to prove it:



The goal of the adversary is to construct such a theorem, that the prover will fail to prove it. Because of the way the construction works, this theorem will have to be provable.

In the game we use prolog-like *terms*, where a term can be either a *variable*, or a pair of an atom and a list of subterms. In the examples we use the convention of marking variables with capital letters, and denoting compound terms and an atom name followed by a list of subterms in brackets (skipped when the list is empty).
Eg. `node(A, leaf)`.

The construction game is defined for a given set on *inference rules*. An inference rule is a pair of a term and a list of terms, that can share variables.
Eg. `tree(node(A, B))` ← `tree(A)`, `tree(B)`.

A state here is a pair consisting of a list of terms that need to be proven and an information about which player is now in control. During its move a player can choose one of the given inference rules, and apply it to the first term of the list. The left side of the rule is then unified

with that term. If the unification fails, the player making the move loses. If it succeeds, the term is removed from the list, and the right side of the rule (after unification) is added.

The first player (called *adversary*) starts the game with a list consisting of a single variable term. It then proceeds to "prove" it using the inference rules. As it is a variable, to begin with any inference rule can be applied. When the list is empty (meaning that the theorem was proven), the variable we started with will be unified with some theorem. This theorem is then given to the other player, after replacing every remaining variable with a fresh ground atom.

The second player then tries to prove the theorem, winning when the list is empty.

To ensure termination of the game, during every move there is a small chance that the player making the move will immediately lose, so that every game will end with probability 1.

3 Monte Carlo tree search modification

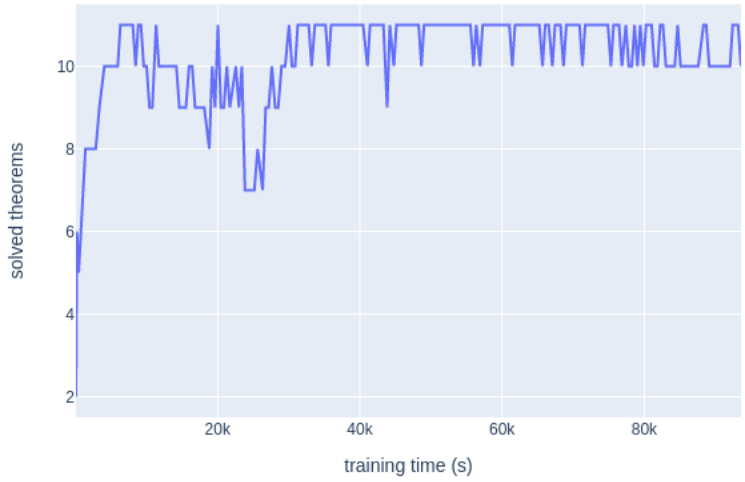
The AlphaZero [4] algorithm utilizes the Monte Carlo Tree Search (MCTS) to estimate state values and policies. As it is used there it works well, when getting a sure value of a game state is almost impossible. However, when players don't take turns, and instead can make several moves in a row, it's possible to find a path to a winning state, and prove with certainty the value of state without searching infeasibly large state space.

To allow propagation of sure state values in our implementation of MCTS we keep track of upper and lower bound for every state. In a non-final game state these are simply (1) and (-1) (as the reward is always somewhere between -1 and 1), but in a final state they are both equal to the outcome of the game. These bounds are then propagated up the tree, in accordance with state ownership (with which player is making a move in which state). This assures that if the tree search finds a certain way for one player to win in state s, the value of this state will become exactly 1.

It is worth pointing out that finding a winning path in the MCTS doesn't necessarily mean further search is pointless. Eg., the player constructing the theorem can avoid building theorems, for which the MCTS already found a proof.

4 Preliminary investigation

When training our model on a toy problem (involving reversing a list) we observed that the performance does improve in time, although it does not achieve a stable high result. Although we do not use any set of theorems during training, we do require it to measure the performance.



For estimating value and policy we currently use a variant of a Graph Attention Network [5], but we plan on experimenting with different architectures, as well as different axiom sets and hyperparameters.

References

[1] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *ArXiv*, abs/1905.10501, 2019.

[2] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, 2018.

[3] Michael Rawson and Giles Reger. A neurally-guided, parallel theorem prover. In *FroCos*, 2019.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *ArXiv*, abs/1712.01815, 2017.

[5] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *ArXiv*, abs/1710.10903, 2017.

Autoencoding TPTP

Michael Rawson and Giles Reger

University of Manchester — Manchester, UK
michael@rawsons.uk, regerg@cs.man.ac.uk

Abstract

Extracting features from problem files is a prerequisite in learning systems for automatic theorem proving, notably for strategy creation and scheduling. Such manually-designed features are crucial in enabling machine learning algorithms to help solve otherwise-difficult problems. We propose a neural autoencoder approach for problem sets (allowing automatic feature extraction), and aim to show that the learned features are complementary to human-designed problem features. Learned features may also shed some light on the structure and behaviour of problem sets frequently-used in the community. The TPTP problem set is used as a well-known running example.

1 Background

Given a problem p in a set P, many machine-learning techniques and existing applications require n real-valued features supplied by a feature extraction mapping $f : P \to \mathbb{R}^n$. Learning to predict good prover options ("strategies") is one example of such a system. Previous approaches have often utilised manual feature engineering [2], but this is labour-intensive, and it is not clear in general which features are useful for a given task. Autoencoders [4] learn to reconstruct the input they are given, but must pass data through a "bottleneck" layer which is typically smaller than the input, thereby learning a compressed representation at the bottleneck. Representing the input/output problem set in our application — collections of first-order formulae — is non-trivial, but recent advances in neural network techniques make this more tractable. In this work we use a directed-graph representation of formulae [7], along with graph neural network techniques [1] for encoder and decoder networks.

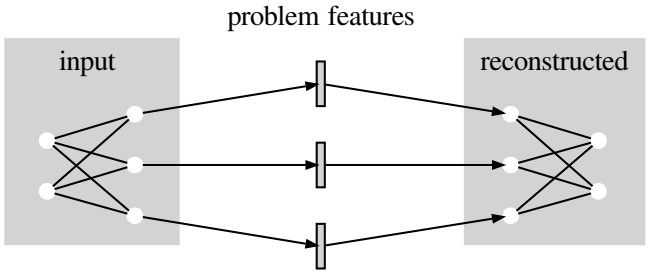


Figure 1: Information flow in the autoencoder. Each problem node receives its own feature vector based on its local formula graph P, and other nodes are then discarded. This vector is then used to try and recover the original information in the reconstructed graph $\hat{P}$.

2 Task

We represent a problem set P as a directed graph. A subset of the nodes in the directed graph are "problem nodes", representing a single problem with constituent axioms and conjecture as immediate children — in TPTP [5] this is a natural construction. An encoder network is allowed to produce a feature vector in $\mathbb{R}^n$ for each node, then all nodes except the problem nodes are discarded in a bottleneck, after which a decoder network attempts to recreate the input graph's node data. A graphical representation of this approach is shown in Figure 1. The level of accuracy in reconstruction and the degree of compression achieved is a useful test of network representation, while also providing a means of producing learned feature vectors from problems in an end-to-end fashion. This transductive task is also interesting from a machine-learning perspective: it is both a node-embedding and autoencoding task. A moderately-deep variational autoencoder model produces reconstruction results significantly better than chance on the first-order problems of TPTP.

3 Future Work

While an obvious next step is to experiment with better neural encoder/decoder pairs, there are many directions for future work. We aim to investigate and present:

1. The effects of different representations and architectures on the performance and on the learned embedding.
2. Conclusions from and visualisations of the learned embedding. Techniques such as t-SNE [3] are expected to be helpful here.
3. Performance of the learned representation on tasks such as strategy scheduling. Are the learned features complementary to existing designed features?
4. Transfer learning: do learned encoders/decoders generalise well to new problem sets? If not, how much training is needed to re-specialise?
5. Comparison of TPTP and other datasets, such as MPTP [6], when viewed under the lens of this new tool.

References

[1] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[2] Daniel Kühlwein and Josef Urban. MaLeS: A framework for automatic tuning of automated theorem provers. *Journal of Automated Reasoning*, 55(2):91–116, 2015.

[3] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[4] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[5] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337, 2009.

[6] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006.

[7] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In *Advances in Neural Information Processing Systems*, pages 2786–2796, 2017.

Developing a Concept-Oriented Search Engine for Isabelle Based on Natural Language: Technical Challenges

Yiannos A. Stathopoulos, Angeliki Koutsoukou-Argyraki, and Lawrence C. Paulson*

Department of Computer Science and Technology, University of Cambridge, UK
[yas23,ak2110,lp15]@cam.ac.uk

The Isabelle libraries and the Archive of Formal Proofs (AFP) contain thousands of formally checked facts (theorems, lemmata, corollaries, propositions, definitions etc.). Current efforts for indexing and searching collections of facts revolve around two approaches. The first approach is mathematical knowledge management (MKM), which involves abstracting mathematical knowledge in the libraries using a semantic markup language, such as OMDoc [6, 2] or a formal meta language, such as MMT [10, 8]. The second approach is *online search* (i.e., searching libraries loaded in the active session in real-time) of Isabelle libraries using symbolic pattern matching of strings. For instance, the Isabelle command `find_theorems` [19] takes a set of criteria (e.g., keywords that must be present in fact names) as input and returns a list of facts that explicitly match these criteria.

In certain cases, `find_theorems` may be limiting for the users. Inexperienced users might have an idea of what kind of material is needed to complete their proof but not enough knowledge of the Isabelle library organisation and naming conventions to construct effective queries for `find_theorems` [7]. This limitation is exacerbated by the fact that new users are more familiar with search interfaces akin to Google's search box: they expect their search query to be a "bag-of-words" describing in natural language the concepts or topic of their enquiry. Furthermore, in response to their query, users expect to be presented with a list of results ordered by relevance.

The aforementioned user expectations are presently not always fulfilled by `find_theorems`. First, it is not straightforward to rank by relevance results produced using strict pattern matching: many facts may match the input criteria exactly. Second, `find_theorems` only matches queries to facts in libraries and theories loaded in the active session. This may be counter-intuitive to new users who might be looking for facts in unloaded theories and are accustomed to searching the entire web in fractions of a second. At the same time, users may not know in which theory the material they are searching for is located; note that classifying mathematical knowledge is non-trivial in principle. Third, as `find_theorems` is based on pattern matching (and is even case-sensitive) it does not find results that are associated conceptually if their names do not exactly match.

We have been investigating a new approach to indexing and searching Isabelle libraries based on natural language. In our approach, each fact is represented by a "bag-of-words" and a set of textual "mathematical concepts" [15, 13] (natural language phrases that refer to mathematical objects, structures and ideas) rather than formal abstractions. Our goal is to develop and evaluate a search engine that (1) enables efficient, *offline search* (search is performed on an index with pre-computed representations so that it does not depend on the loaded theories at each session) of facts in the Isabelle libraries and the AFP; (2) allows Isabelle users to search the libraries using a search box (3) supports "conceptual search" by allowing Isabelle users to search the libraries for desired facts or definitions by describing them using a bag-of-words

*The authors were supported by the ERC Advanced Grant ALEXANDRIA (Project 742178) funded by the European Research Council and led by Professor Lawrence Paulson at the University of Cambridge, UK.

and associated textual mathematical concepts; (4) presents results in order of relevance so that Isabelle users can quickly assess the usefulness of each listed fact or definition. In this presentation we focus on the technical challenges encountered while working towards the above goals and introduce promising solutions.

The first challenge is offline indexing of Isabelle theories. Isabelle users interact with the theorem prover using Isabelle's rich syntax, which includes outer syntax commands, structured Isar proofs and an inner syntax term language [19]. An important step in offline indexing of Isabelle theories is extracting information from the syntax and internal state of the theorem prover. This task is complicated for two reasons. First, it is non-trivial to write an external parser of Isabelle's syntax mainly because the syntax is ambiguous and valid parse trees can only be selected after type-checking [19]. Second, useful information about facts in an Isabelle session, such as types, can only be retrieved from the internal state of the prover, which is not easily achieved using external tools. In order to produce an offline index of the Isabelle libraries we developed an information extraction pipeline for Isabelle. The first stage of our pipeline involves interpreting the PIDE [16, 17] message exchange between Isabelle and jEdit (obtained from `isabelle-dump` [18]). Next, the interpreted messages are transformed into a sequence of tokens representing Isabelle commands. The sequence of tokens is then chunked into constructs such as theorems (and proofs), lemmata and definitions. Our pipeline supports extraction of arbitrary feature sets from Isabelle theories using an interface akin to Map-Reduce [4].

The second challenge is that of automatically modelling mathematical knowledge by assigning concepts (keyword and phrase clouds) to Isabelle facts. Mapping mathematical concepts to Isabelle facts enables linking natural language descriptions of the mathematical knowledge being sought by the user to facts in the Isabelle libraries. Constructing this mapping automatically at scale is challenging because mathematical knowledge in the libraries is almost exclusively expressed in Isabelle's formal language. Our approach is to construct this mapping by linking Isabelle facts to Wikipedia articles that describe mathematical results, structures and objects. We represent each Isabelle fact using two vectors extracted from linked Wikipedia articles. The first representation is a vector of associated words constructed from the body and title of linked Wikipedia articles. The second representation is a vector of associated mathematical concepts discovered in linked articles. We discover mathematical concepts in linked Wikipedia articles using a dictionary of 1.23 million phrases that name mathematical concepts [13]. Searching and ranking facts and definitions using natural language representations enables us to use the Vector Space Model (VSM) [12] to approximate topical similarity to bag-of-words queries. The VSM is an established model of topical similarity in natural language that is known to produce reliable rankings of search results [11, 3].

The third challenge is evaluating the effectiveness of our search engine at retrieving Isabelle facts. The main challenge for evaluation is building a *test collection* for Isabelle search composed of real-life Isabelle queries, complete with expert decisions on which facts in the libraries are relevant to each query (also referred to as *relevance judgements*). In Mathematical information retrieval (MIR), evaluation resources such as the Cambridge University MathIR Test Collection (CUMTC) [14] and the NTCIR math track test collection [1] have facilitated comparisons between systems [5].

We have implemented some promising solutions to the above challenges in the form of a prototype search engine (SErAPIS: Search Engine by the Alexandria Project [9] for ISabelle) and performed a preliminary evaluation. It is our intention to make our search engine publicly available online[1], and procure real-life search queries and relevance judgements from the Isabelle community to produce a resource much like the CUMTC and NTCIR test collections.

[1] The search engine will be available online at `behemoth.cl.cam.ac.uk/serapis`

References

[1] Akiko Aizawa, Michael Kohlhase, and Iadh Ounis. Ntcir-10 math pilot task overview. In *Proceedings of the 10th NTCIR Conference*, June 2013.

[2] Jonas Betzendahl and Michael Kohlhase. Translating the IMPS Theory Library to MMT/OMDoc. In *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, pages 7–22, 2018.

[3] Chris Buckley and Ellen M. Voorhees. Evaluating Evaluation Measure Stability. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 33–40, New York, NY, USA, 2000. ACM.

[4] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, pages 137–150, San Francisco, CA, 2004.

[5] Ferruccio Guidi and Claudio Sacerdoti Coen. A Survey on Retrieval of Mathematical Knowledge. *CoRR*, abs/1505.06646, 2015.

[6] Michael Kohlhase. *OMDoc - An Open Markup Format for Mathematical Documents [version 1.2]*, volume 4180 of *Lecture Notes in Computer Science*. Springer, 2006.

[7] Angeliki Koutsoukou-Argyraki. Formalising Mathematics – in Praxis ; A Mathematician's First Experiences with Isabelle/HOL and the Why and How of Getting Started (submitted preprint), 07 2019.

[8] Dennis Müller, Thibault Gauthier, Cezary Kaliszyk, Michael Kohlhase, and Florian Rabe. Classification of Alignments Between Concepts of Formal Mathematical Systems. pages 83–98, 06 2017.

[9] Lawrence Paulson. ALEXANDRIA: Large-Scale Formal Proof for the Working Mathematician. https://www.cl.cam.ac.uk/~lp15/Grants/Alexandria/.

[10] Florian Rabe. The MMT Language and System.

[11] S. E. Robertson. The Probability Ranking Principle in IR. *The Journal of Documentation*, 33:294–304, 1977.

[12] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[13] Yiannos Stathopoulos, Simon Baker, Marek Rei, and Simone Teufel. Variable Typing: Assigning Meaning to Variables in Mathematical Text. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 303–312, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[14] Yiannos Stathopoulos and Simone Teufel. Retrieval of research-level mathematical information needs: A test collection and technical terminology experiment. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers*, pages 334–340, 2015.

[15] Yiannos Stathopoulos and Simone Teufel. Mathematical information retrieval based on type embeddings and query expansion. In *Proceedings of the 26th International Conference on Computational Linguistics, Coling 2016, December 11-16, 2016, Osaka, Japan*, pages 334–340, 2016.

[16] Makarius Wenzel. PIDE as front-end technology for Coq. *CoRR*, abs/1304.6626, 2013.

[17] Makarius Wenzel. Isabelle/PIDE after 10 years of development. In *UITP 2018 (International Workshop on User Interfaces for Theorem Provers 2018).*, 2018. https://sketis.net/wp-content/uploads/2018/08/isabelle-pide-uitp2018.pdf.

[18] Makarius Wenzel. The Isabelle System Manual. 2019. https://isabelle.in.tum.de/doc/system.pdf.

[19] Makarius Wenzel. The Isabelle/Isar Reference Manual. 2019. https://isabelle.in.tum.de/doc/

isar-ref.pdf.

Learning Strategy Design: First Lessons

Martin Suda[1]*and Sarah Winkler[2]†

[1] Czech Technical University in Prague, Prague, Czech Republic
martin.suda@gmail.com
[2] Università degli Studi di Verona, Verona, Italy
sarahmaria.winkler@univr.it

Abstract

Automatic theorem provers typically offer a wide variety of parameters to control their search strategy. However, the strategy range is vast, and a suitable strategy for a given input problem hard to predict.

Here we sketch an experiment analyzing a large data set, obtained by running all 801 of Vampire's CASC mode 2018 strategies on first-order problems in TPTP: (1) We build random forest regression models predicting strategy performance, using different feature sets to investigate feature importance. (2) Each strategy is defined by a set of parameter values. We investigate the correlation between problem features and successful parameter values to work towards learning how to construct suitable strategies for a given problem. (3) We analyze correlations between problem features and the success of tools run in CASC.

1 Strategy Prediction

Data set. Vampire [6] was run for 60 s on all 17 574 FOL problems in TPTP library [9] (version 7.2.0) using all the 801 strategies used in CASC-27 (a total running time of ∼16 years on a single core).

Problem features. We use the problem properties specified in the TPTP files (e.g., number of axioms, terms, variables, etc) and a property set determined by Vampire (e.g., has extensionality, linear integer or group problem). Moreover we experimented with the three TPTP *pseudo-features*: domain, rating, and source,[1] where "source" refers to a token combining the author and year of the TPTP submission, like 'Sla93'. In addition, three hand-crafted features were used which estimate the number of unifiable positive and negative literals and the number of terms matching and unifying with (non-variable) equation sides. In total, we obtained 98 features in this way.[2]

Regression models. For each of the 801 strategies (set $\mathcal{S}$), we built a random forest regressor to predict the runtime on a problem using our feature set [8]. Hyperparameters were determined by a grid search. We built rating-balanced test and training sets (ratio 1:4), and trained all 801 regressors on the latter. (If a strategy did not solve a problem, a *timeout penalty* of 300 s was assigned.) In the test phase, for every problem in the test set, the strategy with the lowest predicted run time was recommended, and we counted how many problems can be solved by the recommended strategy. The lessons we learned from that include the following:

*Supported by the ERC Consolidator grant AI4REASON no. 649043 under the EU-H2020 programme and the Czech Science Foundataion project 20-06390Y.

†Supported by FWF project T789.

[1] One can question whether these are "legitimate" features as they are not obtained from the problem itself viewed as a logical formula.

[2] For the feature list and all further details see http://profs.scienze.univr.it/winkler/learn_strat/.

- *Tell me the source, I tell you the strategy.* When predicting a strategy in $\mathcal{S}$ from a *single* feature, the source works best: 2342 of 3515 test problems can be solved (2180 from the number of terms, 2241 from the domain, 2166 from the rating). From all features, successful predictions can be made for 2583 problems, without TPTP features 2548.
- *Interaction matters.* All three handcrafted features about unifiable and matching terms are in the top 10 of the most important features. In total, they contribute 11.6%. Other top 10 features are numbers of terms (6.2%), variables (4.3%), atoms (3.8%), connectives (3.6%), functions (3.5%), unit clauses (3%), and constants (3%) (without TPTP features). If included, rating is *the* most important feature (23%) and source is in the top 10, too.
- *Regression quality ≠ prediction power.* The *coefficient of determination* (r^2-value) is a common measure for the amount of variance explained by a regression model. When using all features it amounts to $r^2 = 0.71$, for source only $r^2 = 0.28$, for rating only $r^2 = 0.41$. Obviously, the rating can correctly predict many timeouts, but as the numbers of solved problems above show, not so many suitable solving strategies.

2 Correlation

In Section 1 we tried to predict a strategy from the fixed set $\mathcal{S}$. Next we investigate correlations between problem features and strategy components (i.e., Vampire options). To that end, we clustered problems according to features and compared the probability that a problem from some cluster C can be solved by a strategy with option o set to a particular value v to the probability that (1) an arbitrary strategy solves a problem from C, and (2) a strategy with $o = v$ solves a problem on average. For example, on EPR problems, age-weight ratio (-awr; used for controlling clause selection) values of 1:50, 1:64, 1:128 are 11.0%, 8.6%, 9.6% better than strategies are on EPR on average, and 15.0%, 16.8%, 18.0% better than these option values usually are. As another example, for the sources 'Sla93', 'WM89', 'Bau99', 'Sta09', problems are 60% more likely to be solved by a strategy with the finite model builder (-sa fmb; which replaces a saturation algorithm). We add two more general observations (for more examples and complete data see the website):

- *The strongest correlations appear with sources.* Even for sources which occur in at least 20 problems, we found 389 correlations where problems from a particular source are solved at least 30% more likely with a certain option.
- *Correlations identify fragile options.* For options like nwc[3] and awr, often a certain range is beneficial (as for EPR above), but others are *fragile*, i.e. only one value works.
- *EPR and UEQ show correlations for many option values.*

We also correlated feature properties with the success rate of different CASC tools, using TPTP2T data to check which tool is (a) the most appropriate for a cluster, and (b) more powerful on a cluster than on TPTP in general. For (a), though Vampire is almost always the best choice, we found some clusters where this is not the case. For instance, in the presence of reals CVC4 1.7 is superior, with list axioms Leo-III 1.3 and Isabelle 2016 prevail, and for problems with source 'Hoe08' or 'Sta08', versions of E solve most problems. For (b), we discovered many cases of "overperformance" of a tool on a certain problem cluster. For instance on EPR, iProver, Z3, and Zipperposition win by overperformance, but Vampire solves more problems.

[3]The non-goal-weight coefficient: it penalises clauses not derived from the conjecture by artificially increasing their weight (by multiplying it by a given coefficient) before it is used for clause selection.

Related Work. Given the numerous parameters offered by state-of-the-art theorem provers and the hence vast number of search strategies, automatic tuning by machine learning techniques is a natural approach. Early work in this direction was done for the equational theorem prover Discount [2, 3] using a syntactic feature set and a nearest-neighbor strategy. A similar approach was also pursued for E using features related to symbol counts, evaluated both a priori and after some proof steps [1]. Similar features, together with the TPTP properties, were exploited by the strategy tuning framework MaLeS [7]. The strategy design tools BliStr and BliStrTune [5] predict strategies for E and were evaluated on the Mizar Mathematical Library. Related work for iProver was also presented at AITP 2019 [4].

Conclusion. In next steps, we want to repeat our experiments with a more extensive strategy set, take the solving time for correlations into account, and play with dimensionality reduction. In the future, we plan to work on predicting good *schedules* [10] for a given problem, which is a potentially very rewarding endeavour but has received relatively little attention so far.

References

[1] J. P. Bridge, S. B. Holden, and L. C. Paulson. Machine learning for first-order theorem proving - learning to select a good heuristic. *Journal of Automated Reasoning*, 53(2):141–172, 2014.

[2] M. Fuchs. Automatic Selection of Search-Guiding Heuristics. In D. D. II, editor, *Proc. of the 10th FLAIRS, Daytona Beach*, pages 1–5. Florida AI Research Society, 1997.

[3] M. Fuchs. *Learning Search Heuristics for Automated Deduction.* Number 34 in Forschungsergebnisse zur Informatik. Verlag Dr. Kovač, 1997. Accepted as a Ph.D. Thesis at the Fachbereich Informatik, Universität Kaiserslautern.

[4] E. K. Holden and K. Korovin. SMAC and XGBoost your theorem prover. In *Proc. 4th Conference on Artificial Intelligence and Theorem Proving (AITP 2019)*, pages 93–95, 2019.

[5] J. Jakubuv and J. Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Y. Bertot and V. Vafeiadis, editors, *Proc. 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)*, pages 43–52. ACM, 2017.

[6] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *Proc. 25th International Conference on Computer Aided Verification (CAV 2013)*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[7] D. Kühlwein and J. Urban. MaLeS: A framework for automatic tuning of automated theorem provers. *Journal of Automated Reasoning*, 55(2):91–116, 2015.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[10] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

Neural Architectures for Tactic-Based Automated Theorem Proving

Christian Szegedy, Sarah M. Loos, Aditya Paliwal, Markus Rabe, and Kshitij Bansal

Google Research

1 Introduction

In this talk, we compare various neural network architectures for tactic-based neurally guided proof search for higher order logic in HOL-Light [4] interactive theorem prover. It was first demonstrated in the TacticToe [3] prover that learned guidence for tactic based interactive proof search could yield superior results for automated higher order theorem proving compared to hammers based on first order logic based ATPs [5]. Here we focus on a deep learning based solution. We will be addressing two kinds of tasks: the selection of a tactic out of 41 possible tactics and the ranking of tactic arguments from all the usable tactic arguments from a theorem database.

Our experiments are conducted on the HOList [2] benchmark, which comprises a standardized set of theorems sorted such that later theorems can be proved solely by earlier theorems and definitions in the database. Our main metric is the number of proofs successfully closed on a held out set of theorems. In our imitation learning setup, we train models using our database of human proofs, logged from the HOL-Light libraries. We also experiment with a reinforcement learning setup, allowing the model to control the proof search with tactic and tactic argument selection, while simultaneously training on human proofs. Finally, we perform reinforcement learning without imitation learning (i.e. "from zero" human proofs); in this setting we additionally measure the cumulative number of proofs closed over a fixed number of proof attempts.

2 Architectures Tested

Our theorem prover is based on a simple breadth first search based backward prover augmented by a neural network for premise selection and tactic prediction. The neural network is a two-tower architecture without weight sharing. The two towers produce a fixed dimensional embedding, one for the goal and one for the premise. The two embeddings are combined by a cheap three-layer network to produce a ranking score for the premise. This architectural choice is essential for fast ranking of a large number of premises in relatively short time, since the embeddings for the potential premises can be shared. However, we have a lot of freedom for choosing the architecture for the individual embedding towers that incur the most computational cost. Here we worked with two types of networks: those that consider the input as a sequence of tokens and those that take a graph representation of the formulas. In the latter case we also employ subexpression sharing.

Our experiments focus on various base neural network architectures which all share the common feature that they produce a feature vector for each input token. In order to use the produced features efficiently for ranking the premises, this set (or sequence) of output feature vectors needs to be reduced to a single, relatively short, fixed dimensional feature vector that

can be used in a nearest neighbor look up. The choice of this reduction method is also explored in detail here. For the base architectures, we have evaluated the following variants:

- simple convolutional networks,
- dilated convolutional networks (a.k.a WaveNets [7]),
- transformer network architectures [8],
- graph neural networks (GNNs [6]),
- graph attention networks [9].

We additionally evaluate a variety of pooling mechanisms:

- maximum pooling,
- average pooling,
- expanding the dimension of output features before pooling,
- attention based pooling
- and transformer layers (with self-attention).

3 Evaluation Methodologies and Metrics

These architectures were trained with imitation learning (learning from human proof-logs) and the best models were tested in the context of the reinforcement learning from scratch without utilizing any of the human proofs (in the context of DeepHOL-Zero [1]). We also report several proxy metrics for tactic selection and premise ranking and their evolution during the training process. We study which metrics are most indicative of the final end-to-end prover performance.

References

[1] Kshitij Bansal, Sarah M Loos, Markus N Rabe, and Christian Szegedy. Learning to reason in large theories without imitation. *arXiv preprint arXiv:1905.10501*, 2019.

[2] Kshitij Bansal, Sarah M Loos, Markus N Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher-order theorem proving. *ICML 2019. International Conference on Machine Learning*, 2019.

[3] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with hol4 tactics. In *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46, pages 125–143, 2017.

[4] John Harrison. HOL Light: A tutorial introduction. In *FMCAD*, pages 265–269, 1996.

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

[6] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *arXiv preprint arXiv:1905.10006*, 2019.

[7] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Learning Clause Deletion Heuristics with Reinforcement Learning

Pashootan Vaezipoor[1], Gil Lederman[3], Yuhuai Wu[1,2], Roger Grosse[1,2] and Fahiem Bacchus[1]

[1]University of Toronto
[2]Vector Institute
[3]UC Berkeley

Abstract

We propose a method for training of clause deletion heuristics in DPLL-based solvers using Reinforcement Learning. We have implemented it as part of a software framework `SAT-Gym` which we plan to release as an OpenAI Gym compatible environment. We present experiments and preliminary results for the clause deletion heuristic in Glucose.

1 Introduction

Solvers for difficult combinatorial problems such as SAT, QBF, etc., rely on heuristics that are used in many different phases of their computation. Years of human experience and experimentation have lead to very effective heuristics for many different types of solvers. However, these achievements have been quite painstaking and leave open the question of whether other heuristics could yield better performance. The focus of this work, similar to [5, 6], is to use *Machine Learning (ML)* to automatically learn those heuristics. In particular, there has been recent interest in framing the learning process in a *Reinforcement Learning (RL)* setting [4, 3, 7]. We argue that this is a desirable design choice for several reasons: First, a solver is a dynamic process and changes to the heuristics directly affect the landscape of the future observations. Hence a model that is trained offline in a supervised manner might fail to capture this inherently non-stationary behaviour; Second, an RL-based approach, allows for training the heuristic directly towards optimizing the desired metric (e.g., number of decisions, running time, etc.). This is again in contrast to the supervised setting in which one often needs to substitute that desired metric with a surrogate labeling mechanism. Under these considerations, we propose a RL architecture to learn a *clause deletion* policy to improve the running time of SAT solvers.

Why Clause Deletion? In order to use *Deep Neural Networks (NN)* in the loop of a modern solver we need to address the time-scale mismatch: In the time it takes the NN to make one informed decision, the solver can take thousands. Consequently, we need to make less frequent queries to the NN oracle for it to be feasible. Clause deletion in *Conflict-Driven Clause Learning (CDCL)* SAT solvers naturally fits this criteria as it is executed much less frequently than other heuristics.

Clause deletion is an integral component of CDCL solvers, as the solver accumulates a large number of clauses and as a result starts to slow down. Deleting the learned clauses periodically has proven to be effective in speeding up the process.

2 Clause Deletion in SAT-Gym

In (episodic) RL, we consider an agent that interacts with an environment over finite discrete time steps. The agent gets observations from the environment, takes actions, and accumulates reward. In our setting the environment is Glucose, a time step is a garbage collection, and the agent takes the decision of which learned clause to keep.

Observation: The observation is a set of solver state features that includes: 1. The ratio of learned to original clauses, 2. A histogram of the LBD scores of recently learned clauses along with their average, 3. Moving averages of both the recent trail size and recent decision levels.

Action: We use a policy gradient algorithm that directly optimizes a stochastic policy, which at each time step outputs a distribution over the set of actions. In each time step (garbage collection) the agent has to decide for each clause out of N whether to keep or to drop it. A naïve implementation results in a discrete action space of size 2^N, where N is on the order of 2000. In order to overcome this curse of dimensionality we use the *Literals Block Distance (LBD)* [1] value of a clause, which is the standard metric for the "usefulness of clauses" (clauses with lower LBD values are more useful). We constrain our policy to output as action an integer *LBD threshold*, and delete all clauses with LBD values above the threshold.

Reward: Our goal is to improve the running time of the SAT solver, however due to volatility of CPU time, we count the number of "logical operations" *op* that the solver performs, as a more stable and deterministic replacement. These logical operations consist of the number of times the solver accesses the clauses clause during unit propagation. We have observed a high correlation between the *op* and the wall-clock solving time of an instance, which makes *op* a viable surrogate.

Episodes are rolled until solved, or are *aborted* if they accumulate more than 10^9 logical operations. We define the reward of a successfully solved episode to be $200 - op \times 10^{-7}$. The reward of an aborted episode is set to 0.

3 Training

We provide an OpenAI Gym [2] compatible environment that includes a "solver framework" that allows the user to replace different parts of a standard solver (Glucose in our case). The framework currently supports learning of branching and clause deletion heuristics in SAT and 2QBF solvers.

For our dataset, we used the Cellular Automata benchmark which is part of SATCOMP-2018. We chose this particular benchmark because it allowed us to generate a large dataset with tunable difficulty level.

We have preliminary results indicating we can train a policy to improve its reward on 500 formulas from this dataset using our framework (Figure 1). We are actively working on improving our feature set as well as more elaborate hyper-parameter optimization for our training in order to achieve a viable heuristic that can improve the state of the art.

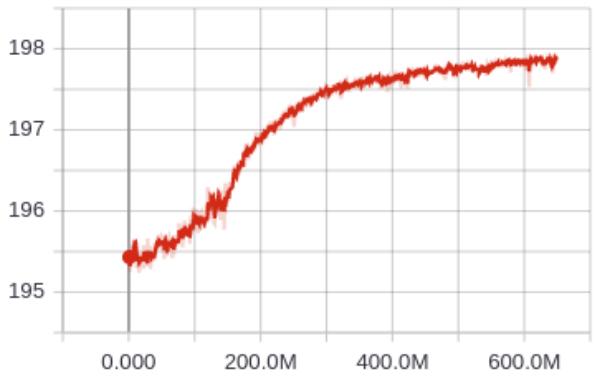


Figure 1: The average episode reward of the agent on 500 formulas from Cellular Automata benchmark over the course of training.

References

[1] G. Audemard and L. Simon. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Twenty-first International Joint Conference on Artificial Intelligence*, 2009.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[3] V. Kurin, S. Godil, S. Whiteson, and B. Catanzaro. Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning, 2019.

[4] G. Lederman, M. N. Rabe, and S. A. Seshia. Learning Heuristics for Automated Reasoning Through Deep Reinforcement Learning. *CoRR*, abs/1807.08058, 2018.

[5] D. Selsam and N. Bjørner. NeuroCore: Guiding CDCL with Unsat-Core Predictions. *arXiv preprint arXiv:1903.04671*, 2019.

[6] M. Soos, R. Kulkarni, and K. S. Meel. CrystalBall: Gazing in the Black Box of SAT Solving.

[7] E. Yolcu and B. Poczos. Learning Local Search Heuristics for Boolean Satisfiability. In *Advances in Neural Information Processing Systems*, pages 7990–8001, 2019.

Reinforcement Learning for Interactive Theorem Proving in HOL4

Minchao Wu[1], Michael Norrish[12], Christian Walder[12], and Amir Dezfouli[2]

[1] Research School of Computer Science
Australian National University, Canberra, ACT, Australia
[2] Data61, CSIRO, Canberra, ACT, Australia

We present an interface for reinforcement learning for interactive theorem proving in HOL4. The interface supports treating HOL4 as an interactive environment for agents to learn to prove theorems in a tactic style. We also describe in detail our reinforcement learning settings for the task, including the design of states, rewards and policy networks. We then give preliminary results demonstrating that theorem proving in HOL4 can be learned with our baseline approach of reinforcement learning using the interface.

Learning systems for interactive theorem proving have started to appear in recent years. Among them there are systems for special purposes such as premise selection [2][16] or algebraic rewriting [11]. There are supervised learning systems designed for general proof search such as TacticToe [8] for HOL4 [14] and CoqGym [18] for Coq [4]. There are also systems using deep reinforcement learning for general proof search such as DeepHOL [3] for HOL Light [9]. Our system is designed for general proof search in HOL4. Unlike TacticToe, which learns from human proof scripts without using deep learning, we use deep reinforcement learning to train policy networks to predict tactics as well as their arguments. Our system is also different from DeepHOL in the following aspects.

- The arguments of a tactic can be not only names of theorems, but also HOL4 terms. Like DeepHOL, predictions are made based on the embedded statements (i.e., expressions) of theorems, not their names.
- For tactics that can take more than one argument, an argument is predicted not only depending on the tactic and the context, but also the previously predicted arguments of the same tactic application. This is because some tactics, such as `simp` and `fs`, are sensitive to such dependence.
- The system does not assume a fixed set of tokens in advance. Once the agent is trained, it should be able to handle newly introduced definitions and theorems which are likely to contain new tokens invented by a user.

Another related implementation of deep reinforcement learning in HOL4 is given by Gauthier [7] recently. The implementation supports reinforcement learning inside HOL4 by implementing basic learning algorithms in standard ML. On the other hand, our interface supports interaction with HOL4 from within Python and manages proofs on the Python side. The interface is designed in a way that HOL4 theorem proving could be integrated as an actual Gym environment[5]. The environment provides information that can be directly processed by popular machine learning frameworks such as PyTorch [12] or TensorFlow [1].

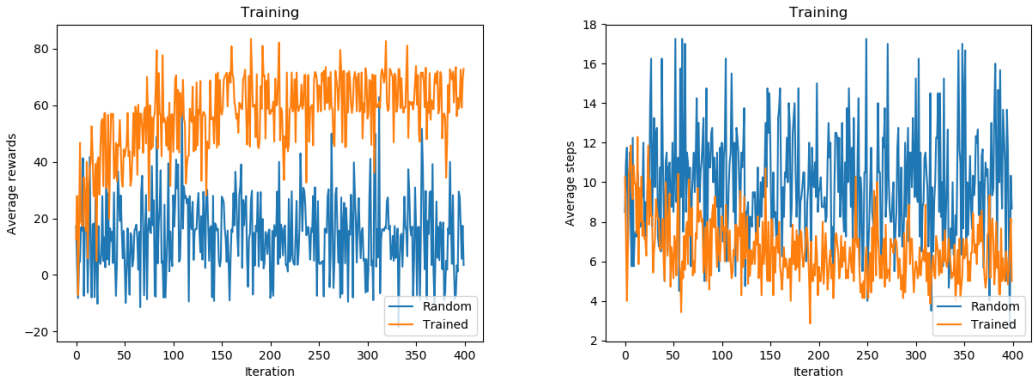
Reinforcement learning formulation A proof attempt in HOL4 can be treated as a game. A state of the game is what we call a fringe. A fringe contains all the remaining goals of a proof attempt, along with their corresponding local context. If one thinks of proof search

as a tree with edges being tactic applications and nodes being the resulting set of goals with their contexts, then the fringe is the union of the unexplored nodes at some stage. A game is won if the fringe becomes empty within a fixed number of timesteps. The action space can be arbitrarily large, as we consider a set of selected tactics as well as their arguments, which can possibly be all the definitions and theorems available in HOL4 or those provided by a user. During proof search, if a theorem is proved, then it is also added to the candidate pool from which an argument is chosen. We distinguish certain resulting states of a tactic application for reward shaping. An action is called ineffective if the tactic application does not change the goal nor its corresponding context. For an inapplicable or ineffective action, we penalize the agent by giving it a reward -2. If an action times out, then the agent receives a reward -1. If the agent managed to prove the main goal within a fixed number of timesteps, then we give it a positive reward that is sufficient to compensate the damage due to the penalization so that the accumulated reward of the episode would end up positive. In other situations, it receives a reward 0.

Policies Actions are predicted by a combination of three policy networks – a tactic policy for choosing a tactic, an argument policy for choosing a list of theorem names as the arguments of the tactic and a term policy for choosing a term if the tactic expects a HOL4 term as its argument. The tactic policy takes a state as an input, and returns a probability distribution π_{tactic} over the possible tactics. The agent then samples one tactic to apply according to π_{tactic}. The argument policy takes additionally the previously predicted argument and a hidden variable, and returns the scores s of the candidate theorems and a hidden variable h. An argument t is then chosen by sampling Softmax(s). Then t and h are passed to the same policy again to predict next argument. The hidden variables are computed by a LSTM [10]. The term policy is similar to the argument policy, but the candidates are currently restricted to the tokens occurring in the goal being handled. In our basic settings, the predicted action is applied to the first element in the fringe by default. Backtracking is also not explicitly treated as an action in the basic settings, as it can be expected that the policy networks should learn to avoid unpromising applications by itself. However, more sophisticated approaches are always possible. For example, we can have an additional value network that scores the states for pruning unpromising actions.

Learning algorithms The policies are trained by policy gradient methods [15]. In our baseline approach, the policy networks are trained jointly using the REINFORCE [17] algorithm. We also describe the possibility of adding Monte-Carlo Tree Search [6] based on the learned policies as a policy improvement operator [13].

Preliminary Experiments We implement the baseline approach in PyTorch. Preliminary results are obtained based on the following settings. We train the agent to prove 10 theorems from the list theory of HOL4. Tactics allowed to be used in the proofs are `simp`, `fs`, `metis_tac`, `Induct_on`, `irule`, and `strip_tac`. For tactics that take theorems as arguments, we only allow the 56 definitions in the list theory to be chosen as the arguments. The length of the argument list is fixed to 5. Reuse of proved theorems is disabled. That is, the agent always tries to prove a theorem from scratch. For induction, the argument can be an arbitrary variable occurring in the goal. One iteration of training contains 10 episodes. Each episode is a proof attempt of one of the 10 theorems. If a theorem is proved, then the agent gains a reward of 100. Othewise, the rewards are as described in the above reinforcement learning formulation. The timeout limit for a single tactic is set to be 0.2 seconds. The timestep limit for a single



(a) Average total rewards received in each iteration. (b) Average steps to find a proof in each iteration.

Figure 1: Peformances in terms of total rewards and timesteps.

proof attempt is 20. We train the agent for 400 iterations and compare its performance against random rollouts with the same settings. It can be seen from Table 1 and Figure 1 that the agent is able to prove more theorems as the training goes on, and is guessing less to find a proof.

	average rewards	average steps	successful proofs	success rate
Overall	56.8	6.6	2771	69.2%
Last 100 episodes	65.7	5.9	75	75%
Random	14.2	10	1533	38.3%

Table 1: Performances of training and random rollouts on the same settings. Average steps refer to the number of timesteps needed to find a proof.

Improvements In our baseline approach, each formula in the fringe is given as a sequence of a finite number of tokens in Polish notation. The tree struture of a formula is not fully reflected in the representation which uses integer encoding, and the models are sequence-based. We plan to replace the current representation by more sophisticated ones such as learned embeddings using RNN as proposed in GamePad [11] or TNN for HOL4 terms as proposed in [7]. With better and deeper networks for both representation and policies, we hope that the performance of preliminary experiments generalizes to a larger scale. Other improvements include pre-training the policies on easy problems to accelerate training, or learning a supervised policy in advance [13] to help with proof exploration. We may also model backtracking by considering a proof graph as a state and allow the agent to choose fringes to work on.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike

Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Alexander A. Alemi, François Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 2243–2251, USA, 2016. Curran Associates Inc.

[3] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[4] Pierre Boutillier, Stephane Glondu, Benjamin Grégoire, Hugo Herbelin, Pierre Letouzey, Pierre-Marie Pédrot, Yann Régis-Gianas, Matthieu Sozeau, Arnaud Spiwack, and Enrico Tassi. Coq 8.4 Reference Manual. Research report, Inria, July 2014. The Coq Development Team.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[6] 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, March 2012.

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

[8] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Learning to reason with HOL4 tactics. *CoRR*, abs/1804.00595, 2018.

[9] John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 60–66, Munich, Germany, 2009. Springer-Verlag.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *CoRR*, abs/1806.00608, 2018.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.

[13] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of GO without human knowledge. *Nature*, 550(7676):354, 2017.

[14] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics*, pages 28–32, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[16] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2786–2796. Curran Associates, Inc., 2017.

[17] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforce-

ment learning. *Machine Learning*, 8(3):229–256, May 1992.

[18] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, 2019.

Neural Theorem Proving on Inequality Problems

Yuhuai Wu*, Albert Jiang*, Roger Grosse, Jimmy Ba

1 Introduction

We present an attempt to prove rudimentary inequality theorems as solving sequential decision making problems. Our contributions include 1., designing an inequality theorem generator that can sample non-trivial inequality theorems of arbitrary length, given a set of axioms 2. demonstrating various degree of generalizations of a graph neural network based learning agent. We believe the proposed dataset can be used as a testbed for machine learning methods on theorem proving.

2 Proving Inequalities as a Markov Decision Process

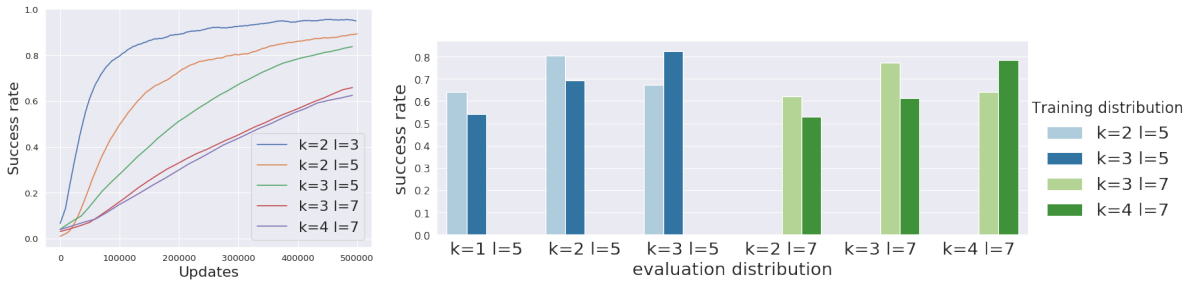
Similar to Huang et al. [2018], Bansal et al. [2019], Yang and Deng [2019], we model proving inequality theorems as a Markov Decision Process. Different from the approach by Fawzi et al. [2019] which models polynomial inequalities as optimization problems, our framework is more general and can be easily extended to other kinds of mathematical theorem proving tasks. We chose inequality problems because of their simplicity and analytic transparency, while embodying the general difficulty of theorem proving. The state space consists of a set of known facts (premises and proved sentences), and a set of goals (the formulae to prove). The action space is a tuple of an axiom and its arguments. The axiom set has all the axioms to define an ordered ring, plus a few composed axioms to reduce redundancy. All axioms are listed in Appendix A.

Our proof system allows the agent to prove in both forward and backward directions. Given the axiom and entities chosen, the proof system first constructs a formula p representing the premise, and a formula c representing the conclusion. If the premise p exists in the current set of ground truth, then the conclusion c is proved and c will be added to the set of known facts in the next state; this is proving in the forward direction. On the other hand, if the goal set contains the conclusion, this suggests we can convert the goal from c to p. We hence delete the c from goal set and add p to it; this is proving in the backward direction. When the goal set is a subset of the set of facts, the proof is finished and the agent succeeds.

3 Dataset

One way to generate theorems is to randomly sample a sequence of actions and apply them to the initial state s_0, transforming it to a new state s. Each of the proved facts in s with the minimal premises needed to prove it can be treated as a new theorem. However, to generate a new conclusion, the premise needs to be satisfied a priori. It is therefore not easy to sample a long and interesting theorem from randomly applied actions. The set of theorems generated this way are skewed towards using axioms whose premises are easy to satisfy, and usually require not more than 2 steps to prove.

Inequality theorem generator Hence, one of the main contribution of this paper is to design a theorem generator that is able to sample non-trivial and long theorems. We overcome those previously mentioned challenges by writing a production rule for each axiom. This not only ensures that new conclusions can be generated by constructing premises on the fly, but also to chain up previous proof steps to form longer proofs. Our strategy is to start with a simple formula (e.g. $a = a$) and iteratively transform it to a more complicated one to prove. The synthesis algorithm takes in a list of axioms, and two integer parameters k and l, and generates a theorem distribution. Each theorem in the distribution can be proved using k unique



(a) (b)

Figure 1: (a) shows learning curves evaluated on the same theorem distribution. (b) shows generalization performances across various k while keeping l fixed.

axioms from the list of axioms, and within total proof steps l. We provide the pseudo-code for the synthesis algorithm in Algorithm 1. We also constrain the length of the sampled expressions to be short, so as to make the theorems more natural.

Six Dimensions of Generalization Our motivation of creating the dataset is to use it as a tool to answer research questions for learning methods on theorem proving. The most essential challenge in theorem proving learning is to be able to prove new theorems that has not been seen during training, i.e., the problem of generalization. Specifically, given a fixed set of axioms, we consider testing agent's generalization ability on unseen theorems whose variations are resulted from 1. the randomness from theorem distribution, 2. varying the complexity of initial conditions 3.varying the orders in which axioms are applied, 4. varying the combinations of axioms used, 5. varying the number of unique axioms k, and 6. varying the length of the proof l. Notice that the first generalization is still within *i.i.d.* regime, namely, the unseen theorems are sampled from the same theorem distribution as in training. In contrast, the second and the third generalization considers non *i.i.d.* generalization, an essential characterization of the challenge of in learning theorem proving.

4 Experiments

We now present our results on generalization along some generalization dimensions. We converted each formula to its computational graph, and used a graph neural network based agent to encode the state, and an autoregressive model to propose actions. We used the proposed inequality theorem generator to generate theorems as well as their proofs as training data for imitation learning. We generated data in an online fashion to reduce overfitting. All of these agents were trained for 500000 updates. For offline evaluation, we calculated the success rate by running the agents for 10 steps on 1000 theorems sampled from a particular distribution.

The generalization results over the first dimension when fixing k and l is demonstrated in Fig 1 (a). We observe that our learning algorithm was able to improve performances over all pairs of k and l, indicating generalization over the same distribution is successful.

To test the agent's generalization ability over k, we evaluated our agents on theorem distribution with varying number of axioms k, while the length of the proof l was kept unchanged. The results are presented in Fig 1 (b). We observe that the agent that was trained on theorem distribution $k = 2$, $l = 5$ was able to solve to unseen theorems from distribution $k = 1, 3$ and $l = 5$, with a $13 - 16\%$ performance degradation. Similarly for the other three agents, all suffered from $10 - 30\%$ performance degradation. The largest drop in performance happened in evaluating the agent trained on $k = 3$, $l = 5$ and evaluated on $k = 1$, $l = 5$. It is surprising because $k = 1$, $l = 5$ may seem to be an easier theorem distributions than $k = 3, l = 5$, but the agent did worse on 29% worse on easier theorems, indicating poor non-*i.i.d.* generalization.

Lastly, to test agent's ability to prove longer proofs, we evaluated our agents on all theorem distribution shown in the Table.(1). We first observe that all agents suffered performance degradation when the length of the proof is increased. Agents trained on simple theorem distribution such as $k = 2$ $l = 3$ could not

Table 1: Generalization performances of agents trained on various theorem distributions.

Training on	Evaluation								
	k=1 l=1	k=1 l=3	k=2 l=3	k=2 l=5	k=3 l=5	k=3 l=7	k=3 l=9	k=4 l=7	k=4 l=9
k=2 l=3	92.9%	84.7%	**98.0%**	0	4.1%	0	0	0	0
k=2 l=5	94.9%	84.7%	93.9%	73.5%	68.4%	28.6%	5.1%	22.4%	4.1%
k=3 l=5	86.7%	86.7%	92.9%	66.3%	83.7%	38.8%	4.1%	43.9%	6.1%
k=3 l=7	95.9%	80.6%	90.0%	84.7%	84.7%	**70.4%**	**58.2%**	72.4%	**55.1%**
k=4 l=7	96.9%	85.7%	90.8%	64.3%	82.7%	56.1%	40.8%	**75.5%**	46.9%
k=4 l=9	91.8%	80.6%	78.6%	59.2%	62.2%	50.0%	41.8%	68.4%	42.9%
curriculum	**98.0%**	**95.9%**	94.9%	**92.9%**	**86.7%**	66.3%	21.4%	63.3%	28.6%

generalize to any problems beyond length larger than 3. In contrast, agents trained on more difficult theorem distributions, such $k = 4$, $l = 9$, can generalize to easier theorem distributions with good performances. Most remarkably, the agent trained on $k = 3$ $l = 7$ was able to achieve quite impressive generalization on longer problems $k = 3$ $l = 9$ and $k = 4$ $l = 9$, even surpassing the performances of those agents that are trained on these training distributions. This indicates a curriculum agent that learns easier theorem distribution could help prove harder theorems. Therefore, in additional to agents trained on single theorem distribution, we also trained an agent on all theorem distributions consisting of $k = 1$ $l = 3$, $k = 1$ $l = 5$, $k = 2$ $l = 3$, $k = 2$ $l = 5$, $k = 3$ $l = 5$, named **curriculum** in the results. We can see that it is indeed able to beat most of the agents in the distribution it was trained on, and almost matched the $k = 3$ $l = 7$ agent on $k = 3$ $l = 7$, demonstrating the benefits brought by curriculum learning.

References

Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/bansal19a.html.

Alhussein Fawzi, Mateusz Malinowski, Hamza Fawzi, and Omar Fawzi. Learning dynamic polynomial proofs. In *Advances in Neural Information Processing Systems*, pages 4181–4190, 2019.

Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A Learning Environment for Theorem Proving. *arXiv preprint arXiv:1806.00608*, 2018.

Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6984–6994, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/yang19a.html.

A Inequality Theorem Generator

For each step of the synthesis program, we will sample an axiom from the given list, and we initiate three procedures, `NEW`, `EXT` and `SUB` associated to the sampled axiom. The `NEW` procedure produce any new premise so that the given axiom can be applied in the synthesized proof. The `EXT` procedure then proceeds to produce the entities as input to the axiom. Lastly, in order to chain up the previous proving steps and the new step, one may be required to do an extra substitution step called `SUB` procedure. The resulting new formula then is used to replace l.

Algorithm 1 Inequality Theorem Generator

Input: Set of all available axioms A, cardinality of axioms to use k, desired length of proof l
Output: Synthesized goal g, a set of premises H

```
Set of theorems to use A_k ← sample(A, k)
Initialize set of used theorems A' ← ∅, premise set H ← ∅
Initialize fact f ~ Uniform({a = a, b = b, c = c})
Initialize length counter c ← 0
while c < l do
  E ← set of all entities in current state.
  if A' ≠ A_k then
    #IF THERE ARE AXIOMS UNUSED, SAMPLE AN UNUSED AXIOM
    a ~ Uniform(A_k \ A')
  else
    #IF ALL AXIOMS ARE USED, SAMPLE A RANDOM AXIOM
    a ~ Uniform(A_k)
  end if
  A' ← A' ∪ {a}
  h_new ← NEW(f, E)
  o ← EXT(f, E)
  Apply theorem a with operands o; obtain conclusion f'.
  c ← c + 1
  o_sub ← SUB(f, f', E)
  Apply substitution axiom with operands o_sub; obtain conclusion f_new.
  if f_new ≠ f' then
    #ONLY INCREMENT COUNTER IF SUBSTITUTION IS NOT REDUNDANT
    c ← c + 1
  end if
  f ← f_new
  H ← H ∪ h_new
end while
return f, H
```

t =AdditionCommutativity: 2 inputs: a and b Assumptions: $\emptyset$ Conclusions: $[a+b=b+a]$	$e \sim Uni(E)$ `NEW`(f,E) : **return** $\emptyset$ `EXT`(f,E) :, **return** $[LHS(f), e]$ `SUB`(f,f',E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =AdditionAssociativity: 3 inputs: a, b and c Assumptions: $\emptyset$ Conclusions: $[a+(b+c)=(a+b)+c]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$ `NEW`(f,E) : **return** $\emptyset$ `EXT`(f,E) : **return** $[LHS(f), e_1, e_2]$ `SUB`(f,f',E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =AdditionSimplification: 2 inputs: a and b Assumptions: $[a=0]$ or $[b=0]$ Conclusions: $[a+b=b]$ or $[a+b=a]$	`NEW`(f,E) : **return** $\emptyset$ `EXT`(f,E) :, **return** $[LHS(f), 0]$ `SUB`(f,f',E) : **return** $[RHS(f'), RHS(f)]$

t =MultiplicationCommutativity: 2 inputs: a and b Assumptions: $\emptyset$ Conclusions: $[a * b = b * a]$	$e \sim Uni(E)$ NEW(f, E) : **return** $\emptyset$ EXT(f, E) : **return** $[LHS(f), e]$ SUB(f, f', E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =MultiplicationAssociativity: 3 inputs: a, b and c Assumptions: $\emptyset$ Conclusions: $[a * (b * c) = (a * b) * c]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$ NEW(f, E) : **return** $\emptyset$ EXT(f, E) : **return** $[LHS(f), e_1, e_2]$ SUB(f, f', E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =MultiplicationSimplification: 2 inputs: a and b Assumptions: $[a = 0]$ or $[b = 0]$ or $[a = 1]$ or $[b = 1]$ Conclusions: $[a * b = 0]$ or $[a * b = 0]$ or $[a * b = b]$ or $[a * b = a]$	NEW(f, E) : **return** $\emptyset$ EXT(f, E) :, **return** $[LHS(f), 1]$ SUB(f, f', E) : **return** $[RHS(f'), RHS(f)]$
t =AdditionMultiplicationLeftDistribution: 3 inputs: a, b and c Assumptions: $\emptyset$ Conclusions: $[(b + c) * a = b * a + c * a]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$ NEW(f, E) : **return** $\emptyset$ EXT(f, E) : **return** $[LHS(f), e_1, e_2]$ SUB(f, f', E) : **return** $[RHS(LHS(f')), RHS(f)]$
t =AdditionMultiplicationRightDistribution: 3 inputs: a, b and c Assumptions: $\emptyset$ Conclusions: $[a * (b + c) = a * b + a * c]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$ NEW(f, E) : **return** $\emptyset$ EXT(f, E) : **return** $[LHS(f), e_1, e_2]$ SUB(f, f', E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =SquareDefinition: 1 input: a Assumptions: $\emptyset$ Conclusions: $[a^2 = a * a]$	NEW(f, E) : **return** $\emptyset$ EXT(f, E) :, **return** $[LHS(f)]$ SUB(f, f', E) : **return** $[LHS(LHS(f')), RHS(f)]$
t =SquareGEQZero*: 1 input: a Assumptions: $\emptyset$ Conclusions: $[a^2 \geq 0]$	$e \sim Uni(E)$ NEW(f, E) : **return** $\emptyset$ EXT(f, E) : **return** $[e]$ SUB(f, f', E) : **return** $\emptyset$
t =EquivalenceTransitivity: 3 inputs: a, b and c Assumptions: $[a = b,\ b = c]$ Conclusions: $[a = c]$	$e \sim Uni(E)$, new independent variable z, NEW(f, E) : **return** $[e + z = LHS(f)]$ EXT(f, E) :, **return** $[e + z, LHS(f)RHS(f)]$ SUB(f, f', E) : **return** $\emptyset$
t =EquivalenceSymmetry: 2 inputs: a and b Assumptions: $[a = b]$ Conclusions: $[b = a]$	NEW(f, E) : **return** $\emptyset$ EXT(f, E) :, **return** $[LHS(f), RHS(f)]$ SUB(l, E, G_n, c) : **return** $\emptyset$
t =PrincipleOfEquality: 4 inputs: a, b, c and d Assumptions: $[a = b,\ c = d]$ Conclusions: $[a + c = b + d]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$, new independent variable z, NEW(f, E) : **return** $[e_1 + z = e_2]$ EXT(f, E) : **return** $[LHS(f), RHS(f), e_1 + z, e_2]$ SUB(f, f', E) : **return** $\emptyset$
t =EquMoveTerm: 3 inputs: a, b and c Assumptions: $[a + b = c]$ Conclusions: $[a = c + (-b)]$	$e \sim Uni(E)$, new independent variable z, NEW(f, E) : **return** $[LHS(f) + z = e]$ EXT(f, E) : **return** $[LHS(f), z, e]$ SUB(f, f', E) : **return** $[LHS(f'), RHS(f)]$
t =IneqMoveTerm: 3 inputs: a, b and c Assumptions: $[a + b \geq c]$ Conclusions: $[a \geq c + (-b)]$	$e \sim Uni(E)$, new independent variable z, NEW(f, E) : **return** $[LHS(f) + z \geq e]$ EXT(f, E) : **return** $[LHS(f), z, e]$ SUB(f, f', E) : **return** $[LHS(f'), RHS(f)]$
t =EquivalenceImpliesDoubleInequality: 2 inputs: a and b	NEW(f, E) : **return** $\emptyset$ EXT(f, E) :, **return** $[LHS(f), RHS(f)]$

Assumptions: $[a = b]$ Conclusions: $[a \geq b,\ a \leq b]$	$\texttt{SUB}(f, f', E)$: **return** $\emptyset$
t =InequalityTransitivity: 3 inputs: a, b and c Assumptions: $[a \geq b,\ b \geq c]$ Conclusions: $[a \geq c]$	$e \sim Uni(E)$, new independent variable z, $\texttt{NEW}(f, E)$: **return** $[e + z \geq LHS(f)]$ $\texttt{EXT}(f, E)$: **return** $[e + z, LHS(f), RHS(f)]$ $\texttt{SUB}(f, f', E)$: **return** $\emptyset$
t =FirstPrincipleOfInequality: 4 inputs: a, b, c and d Assumptions: $[a \geq b,\ c \geq d]$ Conclusions: $[a + c \geq b + d]$	$e_1 \sim Uni(E), e_2 \sim Uni(E)$, new independent variable z, $\texttt{NEW}(f, E)$: **return** $[e_1 + z \geq e_2]$ $\texttt{EXT}(f, E)$: **return** $[e_1 + z, e_2, LHS(f), RHS(f)]$ $\texttt{SUB}(f, f', E)$: **return** $\emptyset$
t =SecondPrincipleOfInequality: 3 inputs: a, b and c Assumptions: $[a \geq b,\ c \geq 0]$ Conclusions: $[a * c \geq b * c]$	$e \sim Uni(E)$, new independent variable z, $\texttt{NEW}(f, E)$: **return** $[e + z \geq 0]$ $\texttt{EXT}(f, E)$: **return** $[LHS(f), RHS(f), e + z]$ $\texttt{SUB}(f, f', E)$: **return** $\emptyset$
t =EquivalenceReflexivity: 1 input: a Assumptions: $\emptyset$ Conclusions: $[a = a]$	$\texttt{NEW}, \texttt{EXT}, \texttt{SUB}$ undefined as the theorem is not in any theorem set T used in the NES synthesis.
t =EquivalenceSubstitution: 2 inputs: a and b Assumptions: $[f(a),\ a = b]$ Conclusions: $[f(b)]$	$\texttt{NEW}, \texttt{EXT}, \texttt{SUB}$ undefined as the theorem is not in any theorem set T used in the NES synthesis.

*: This theorem is special as it requires an application of the first principle of inequality after the application of itself.

B Example problems

Equality theorems

Theorem 1
Goal: $((0 \cdot 1) \cdot ((-(a^2)) \cdot c)) = (((-(a^2)) \cdot ((a \cdot a) + (-(a^2)))) \cdot c)$

Theorem 2
Goal: $(((((0 + c) + a) \cdot a) \cdot 1) \cdot (b \cdot (0 + c))) = ((((c \cdot a) + (a \cdot a)) \cdot (0 + c)) \cdot b)$

Theorem 3
Goal: $0 = ((((c + 0) \cdot (a + a)) \cdot (\frac{1}{((c \cdot a) + (c \cdot a))})) + (-(0 + 1)))$

Theorem 4
Premises: $(b + d) = b$
Goal: $(1 + (-((b + b) \cdot (\frac{1}{((b + (b + d)) \cdot 1)})))) = (0 + 0)$

Theorem 5
Premises: $(a + d) = b$
Goal: $1 = (((d \cdot ((a + d) + ((c + (a + d)) + 0))) \cdot ((d \cdot (a + d)) + (d \cdot (c + b)))) \cdot (\frac{1}{((d \cdot ((a + d) + ((c + (a + d)) + 0)))^2)}))$

Theorem 6
Premises: $((b \cdot b) + d) = (b \cdot b)$
Goal: $(0 + ((b \cdot b) + d)) = (((1 \cdot ((b + b) \cdot b)) + (-(((b \cdot b) + (b \cdot b)) \cdot 1))) + (b \cdot b))$

Theorem 7
Goal: $((a \cdot (a + 0)) + ((-(0 + a)) \cdot (a + 0))) = ((a \cdot 0) + (0 \cdot 0))$

Theorem 8
Goal: $(((c \cdot c) + c) \cdot ((c^2) \cdot 1)) = (((c \cdot c) \cdot (0 + (c \cdot c))) + (c \cdot (0 + (c \cdot c))))$

Theorem 9
Goal: $1 = ((((a \cdot c) + ((b \cdot (a \cdot b)) \cdot c)) \cdot (a + (a \cdot c))) \cdot (\frac{1}{((((a + ((b \cdot a) \cdot b)) \cdot c) \cdot (a \cdot c)) + (((a + ((b \cdot a) \cdot b)) \cdot c) \cdot a))}))$

Theorem 10
Goal: $((((b \cdot c) + (c \cdot c)) + (-(0 + ((b + c) \cdot c)))) \cdot (c \cdot c)) = ((c^2) \cdot 0)$

Theorem 11
Goal: $(1 \cdot (b + a)) = ((0 + (a + b)) + 0)$

Theorem 12
Goal: $(((-c) \cdot (-c)) + (((-c) \cdot c) + ((-c) \cdot (-c)))) = (((-c) \cdot (-c)) + (0 \cdot (-c)))$

Theorem 13
Goal: $(((a^2) \cdot (a \cdot (a + 0))) + (a \cdot (a \cdot (a + 0)))) = ((((a^2) \cdot (a^2)) + (a \cdot (a^2))) + 0)$

Theorem 14
Goal: $((((b \cdot 1) \cdot (a \cdot c)) \cdot (b \cdot a)) + (((b \cdot 1) \cdot (a \cdot c)) \cdot (b \cdot a))) = ((((((b \cdot a) \cdot c) \cdot (b \cdot a)) + (((b \cdot a) \cdot c) \cdot (b \cdot a))) \cdot 1)$

Theorem 15
Goal: $1 = ((\frac{1}{((\frac{1}{(b + 0)}) \cdot b)}) \cdot 1)$

Theorem 16

Goal: $0 = ((0 + (-((a \cdot b) + (-(b \cdot a))))) + (-(0 \cdot 1)))$

Theorem 17
Premises: $(a + d) = c$; $((b + c) + e) = (a + d)$
Goal: $(((b \cdot d) + (b \cdot (b + (a + d)))) + ((b + c) + e)) = ((((b \cdot d) + (b \cdot (b + c))) \cdot 1) + (a + d))$

Theorem 18
Goal: $((((\frac{1}{b}) \cdot b) \cdot b) \cdot 1) = ((b \cdot 1) \cdot 1)$

Theorem 19
Goal: $(((1 \cdot (b \cdot (c + a))) + (b \cdot a)) + 1) = (1 \cdot ((1 \cdot ((b \cdot c) + (b \cdot a))) + ((b \cdot a) + 1)))$

Theorem 20
Premises: $(b + d) = c$; $((1 \cdot a) + e) = a$
Goal: $(((a + (b + d)) \cdot (\frac{1}{((1 \cdot a) + c)})) + ((1 \cdot a) + e)) = ((1 \cdot 1) + a)$

Theorem 21
Goal: $((((c^2) \cdot ((c^2) \cdot c)) + (-(((c \cdot c) \cdot (c^2)) \cdot c))) + (b + b)) = ((1 \cdot ((0 + b) + b)) + (-0))$

Theorem 22
Premises: $(b + d) = (a \cdot b)$
Goal: $(1 \cdot ((((c + c) \cdot (((a \cdot b) \cdot c) + (c + c))) + ((c + c) \cdot (c + c))) + (a \cdot b))) = ((((((c + c) \cdot ((((a \cdot (b \cdot c)) + c) + c) + (c + c))) + (b + d)) \cdot 1) + 0)$

Theorem 23
Premises: $((0 \cdot 1) + d) = (1 \cdot 0)$
Goal: $(((((a + (0 \cdot 1)) \cdot (1 \cdot 0)) + (-b)) + (1 \cdot 0)) + (1 \cdot 0)) = ((((((a \cdot (1 \cdot 0)) + ((b + (-b)) \cdot (1 \cdot 0))) + ((-b) + (1 \cdot 0))) + ((0 \cdot 1) + d)) + 0)$

Theorem 24
Premises: $(a + d) = (1 + c)$
Goal: $(((((1 \cdot b) + (c \cdot b)) + (1 + c))^2) \cdot ((1 + c) \cdot b)) = ((((((1 \cdot b) + (c \cdot b)) + (1 + c)) \cdot (((b \cdot (1 + (1 \cdot c))) + (a + d)) \cdot 1)) \cdot (1 + c)) \cdot b)$

Theorem 25
Premises: $(a + d) = (b \cdot 1)$
Goal: $0 = ((b + (a + d)) + (-((b \cdot 1) + (b \cdot 1))))$

Theorem 26
Premises: $(c + d) = a$
Goal: $(0 + ((((a + a) \cdot 1) + a) \cdot 1)) = (1 \cdot ((1 \cdot ((a + (c + d)) + 0)) + (1 \cdot a)))$

Theorem 27
Premises: $(c + d) = (b + c)$
Goal: $(1 \cdot ((((b + c) \cdot c) + (b \cdot (b + c))) + (c + d))) = ((((b + c)^2) + (b + c)) \cdot 1)$

Theorem 28
Premises: $((1 \cdot b) + d) = b$
Goal: $(((((1 \cdot b) + b) \cdot (a \cdot 1)) \cdot (((b + ((1 \cdot b) + d)) \cdot a) \cdot 1)) + 0) = ((((b + (1 \cdot b)) \cdot (a \cdot 1))^2) \cdot 1)$

Theorem 29
Goal: $(((b \cdot 1) + 0) \cdot (1 \cdot 0)) = (((b \cdot 1) \cdot ((-(0 + b)) + (1 \cdot b))) + (0 \cdot ((-(0 + b)) + (1 \cdot b))))$

Theorem 30

Goal: $(1 \cdot 1) = (((((a \cdot (c + c)) + 0) \cdot (b \cdot (c + c))) \cdot (\frac{1}{((((a \cdot c) + (a \cdot c)) \cdot b) \cdot (c + c))})) + 0)$

Theorem 31
Goal: $((1 \cdot (b \cdot b)) \cdot b) = (1 \cdot (0 + (((0 + b) \cdot b) \cdot b)))$

Theorem 32
Goal: $(((c \cdot (c \cdot 1)) + 0) \cdot 1) = (((c \cdot c) + 0) \cdot 1)$

Theorem 33
Goal: $1 = (1 \cdot (\frac{1}{((1+0) \cdot (\frac{1}{((b \cdot (\frac{1}{b})) + 0)}))}))$

Theorem 34
Goal: $(((((((c+a) \cdot a) \cdot (c+a)) \cdot c) \cdot (a+c)) \cdot (c+a)) \cdot (c+a)) = (((((((a+c) \cdot (c+a)) \cdot a) \cdot c) \cdot (a+c)) \cdot (c+a)) \cdot (c+a))$

Theorem 35
Goal: $0 = ((-(1 \cdot 0)) + ((-(c + c)) + ((1 \cdot c) + c)))$

Theorem 36
Goal: $1 = (1 \cdot (\frac{1}{(a \cdot (\frac{1}{(((a+c)+a)+(-(c+a)))}))}))$

Theorem 37
Premises: $(a + d) = a$; $((\frac{1}{c}) + e) = b$
Goal: $(((1 \cdot (1 \cdot (\frac{1}{(c \cdot (\frac{1}{c}))}))) + a) + b) = (1 \cdot (((1 \cdot 1) + (a + d)) + ((\frac{1}{c}) + e)))$

Theorem 38
Goal: $0 = ((b \cdot (b + (-b))) + (-(((0 + 0) \cdot b) + 0)))$

Theorem 39
Goal: $(((1 \cdot c) + (-(1 \cdot (c \cdot 1)))) \cdot 1) = ((0 \cdot 1) \cdot 1)$

Theorem 40
Goal: $((a + b) \cdot (1 \cdot ((b \cdot c) + (c \cdot c)))) = ((a \cdot ((c \cdot c) + (b \cdot c))) + (b \cdot ((c \cdot c) + (b \cdot c))))$

Theorem 41
Goal: $(0 + ((0 + ((c + c) \cdot c)) \cdot (a \cdot b))) = (0 + ((((c \cdot c) \cdot a) + ((c \cdot c) \cdot a)) \cdot b))$

Theorem 42
Premises: $(0 + d) = 1$
Goal: $((((1 \cdot 0) + (a + (a \cdot 1))) + 0) + d) = (((((1 \cdot a) + (-(a \cdot 1))) + a) + (a \cdot 1)) + 1)$

Theorem 43
Premises: $(b + d) = 0$
Goal: $0 = ((((((0 + b) \cdot 0) + ((0 + b) \cdot b)) \cdot 1) + 0) + (-((((b \cdot 0) + (b \cdot b)) + (b + d)) \cdot 1)))$

Theorem 44
Goal: $((0 + c) \cdot ((-c) + (((c \cdot 1) + 0) + (-c)))) = (((0 + c) \cdot (-c)) + ((0 + c) \cdot 0))$

Theorem 45
Goal: $0 = (0 + (-(((0 \cdot 0) + (a \cdot 0)) + (-(((((((a \cdot b) + (a \cdot b)) + ((b + b) + b)) + (-(((a \cdot (b + b)) + (b + b)) + b))) + a) \cdot 0) \cdot 1)))))$

Theorem 46

Premises: $((a + b) + d) = (a + b)$; $(b + e) = a$
Goal: $(a \cdot a) = (1 \cdot (a \cdot a))$

Theorem 47
Premises: $(c + d) = c$
Goal: $((b \cdot (1 + 0)) + (b \cdot (c + d))) = (0 + (b \cdot ((0 + (1 \cdot (\frac{1}{((b+(c+d))\cdot(\frac{1}{(b+c)}))}))) + (c + d))))$

Theorem 48
Goal: $((b + ((((a + a) \cdot 1) \cdot a) + 0)) \cdot a) = ((b \cdot a) + ((((a \cdot a) \cdot 1) + (a \cdot (a \cdot 1))) \cdot a))$

Theorem 49
Goal: $(((1 + b) \cdot (((a \cdot ((c \cdot 1) + (c^2))) + 1) + b)) + ((1 + b) \cdot (((a \cdot ((c \cdot 1) + (c^2))) + 1) + b))) = ((((1 + b) + (1 + b)) \cdot ((((a \cdot (c \cdot 1)) + (a \cdot (c \cdot (c \cdot 1)))) + (1 + b)) \cdot 1)) \cdot 1)$

Theorem 50
Goal: $0 = (((((0+(((c\cdot c)+(c\cdot c))+((c+c)+(c\cdot c))))+0)+c)\cdot a)+(-(((0+((((c+c)\cdot c)+(c+c))+(c\cdot c)))\cdot a)+(c\cdot a))))$

Inequality theorems

Theorem 1
Premises: $(1 + d) \geq 0$; $(b + e) \geq 0$
Goal: $((((1 + 1) \cdot (a \cdot (\frac{1}{a}))) \cdot (1 + d)) + (b + e)) \geq ((((1 \cdot 1) + (1 \cdot 1)) \cdot (1 + d)) + 0)$

Theorem 2
Goal: $(b^2) \geq (0 + (b \cdot (1 \cdot b)))$

Theorem 3
Premises: $((c + 0) + d) \geq 0$; $(d + e) \geq b$
Goal: $((c \cdot ((c + 0) + d)) + (d + e)) \geq (((0 + c) \cdot ((c + 0) + d)) + b)$

Theorem 4
Goal: $(b + 0) \geq ((((0 + b) + c) + c) + (-(c + c)))$

Theorem 5
Premises: $(1 + d) \geq 0$
Goal: $((((c \cdot c) + c) + a) \cdot (1 + d)) \geq ((((c^2) + (c + a)) \cdot 1) \cdot (1 + d))$

Theorem 6
Premises: $(b + d) = b$
Goal: $1 \geq ((((a + b) + (-(b + d))) \cdot ((a + b) + b)) \cdot (\frac{1}{((a\cdot(a+b))+(a\cdot b))}))$

Theorem 7
Premises: $((0 + a) + d) = 0$
Goal: $(((0 + a) \cdot a) + ((0 + a) + d)) \geq ((a^2) + 0)$

Theorem 8
Premises: $(b + d) = a$
Goal: $((c \cdot b) + (b \cdot b)) \geq (1 \cdot ((((c + a) + (-(b + d))) + b) \cdot b))$

Theorem 9
Goal: $(1 \cdot ((((b \cdot (\frac{1}{b})) \cdot a) \cdot (1 \cdot 1)) + a)) \geq ((1 \cdot ((1 \cdot 1) \cdot (a \cdot (1 \cdot 1)))) + (1 \cdot a))$

Theorem 10
Premises: $(c + d) \geq 0$
Goal: $(b \cdot (c + d)) \geq ((((b + b) + 0) + (-b)) \cdot (c + d))$

Theorem 11
Goal: $(((b + 0) + (b + c)) + 0) \geq (((b + b) + c) + 0)$

Theorem 12
Goal: $((c \cdot (c + 0)) + 0) \geq ((c^2) + 0)$

Theorem 13
Goal: $(1 \cdot (b \cdot 1)) \geq ((1 \cdot b) \cdot 1)$

Theorem 14
Goal: $1 \geq ((((b \cdot (\frac{1}{b})) + (\frac{1}{b})) + 1) \cdot (\frac{1}{((1+(\frac{1}{b}))+1)}))$

Theorem 15
Goal: $1 \geq ((\frac{1}{((c \cdot a) \cdot (\frac{1}{(a \cdot c)}))}) \cdot 1)$

Theorem 16
Goal: $((c \cdot (a \cdot a)) + (((a \cdot a) + (c \cdot a)) \cdot (a \cdot a))) \geq (0 + ((c + (0 + ((a + c) \cdot a))) \cdot (a \cdot a)))$

Theorem 17
Goal: $(((c \cdot b) + a) \cdot ((c \cdot b) + (c \cdot b))) \geq ((a \cdot ((c \cdot b) + (c \cdot b))) + ((c \cdot b) \cdot ((c \cdot b) + (c \cdot b))))$

Theorem 18
Goal: $((a \cdot b) \cdot 1) \geq ((((a \cdot 1) \cdot b) \cdot 1) \cdot 1)$

Theorem 19
Goal: $a \geq ((a + c) + (-c))$

Theorem 20
Goal: $((c \cdot b) \cdot b) \geq (b \cdot (b \cdot c))$

Theorem 21
Premises: $(a + d) = a$; $((a + d) + e) \geq 0$; $(b + f) \geq (0 \cdot 0)$
Goal: $(((((((c \cdot 0) + (0 \cdot 0)) + (a + d)) \cdot ((0 + ((c + 0) \cdot (a + (-a)))) + a)) \cdot ((a + d) + e)) + (b + f)) \geq ((0 \cdot ((a + d) + e)) + (0 \cdot 0))$

Theorem 22
Premises: $(c + d) \geq 0$; $((0 + 0) + e) \geq (0 + 0)$
Goal: $((((((0 + (c + (-c))) \cdot (-c)) \cdot (\frac{1}{((0 \cdot (-c)) + (0 \cdot (-c)))})) \cdot (0 + 1)) \cdot (c + d)) + ((0 + 0) + e)) \geq ((0 \cdot (c + d)) + (0 + 0))$

Theorem 23
Premises: $((a^2) + d) \geq 0$
Goal: $((((a \cdot a) + c) \cdot (0 + (1 \cdot (a \cdot a)))) \cdot ((a^2) + d)) \geq ((((a \cdot a) \cdot ((a^2) + 0)) + (c \cdot ((a^2) + 0))) \cdot ((a^2) + d))$

Theorem 24
Premises: $(c + d) = c$; $((0 + a) + e) \geq a$

Goal: $((((a+b)\cdot(((a+(-a))+(a+b))+(c+d)))\cdot((((((0+a)+b)+c)\cdot(a+b))\cdot 1))+((0+a)+e))\geq(0+a)$

Theorem 25
Goal: $1\geq((a\cdot(c+b))\cdot(\frac{1}{((a\cdot c)+(a\cdot b))}))$

Theorem 26
Premises: $(a+d)\geq b$
Goal: $((0\cdot((((((a+c)+a)\cdot(a\cdot c))\cdot(a\cdot c))+((a\cdot c)\cdot(a\cdot c)))+(-((((((a+(c+a))\cdot a)\cdot c)+(a\cdot c))\cdot(a\cdot c))+0))))+(a+d))\geq(0+b)$

Theorem 27
Premises: $((c\cdot b)+d)=(b\cdot b)$; $((b\cdot b)+e)\geq a$
Goal: $((((b+b)+(b+b))\cdot((((c\cdot(b\cdot b))+b)+b)+(b^2)))+((b\cdot b)+e))\geq((((b+b)\cdot((((c\cdot b)\cdot b)+(b+b))+((c\cdot b)+d)))+((b+b)\cdot((((c\cdot b)\cdot b)+(b+b))+((c\cdot b)+d))))+a)$

Theorem 28
Premises: $((b\cdot 0)+d)\geq c$
Goal: $((((b+(((0+c)+(0+c))+0))\cdot 0)\cdot((b\cdot 0)+(((0+c)+(0+c))\cdot 0)))+((b\cdot 0)+d))\geq(0+c)$

Theorem 29
Premises: $(a+d)\geq 0$
Goal: $((0\cdot((((c\cdot c)+(c\cdot 0))\cdot a)+(-(((c+0)\cdot((c+0)\cdot a))\cdot 1))))+(a+d))\geq(0+0)$

Theorem 30
Premises: $(a+d)\geq c$
Goal: $(((b\cdot(b\cdot 1))+(b\cdot c))+(a+d))\geq((0+(b\cdot((b\cdot 1)+c)))+c)$

Theorem 31
Goal: $(0+(0+(c+b)))\geq(0+((b+c)+0))$

Theorem 32
Goal: $(a+(a+0))\geq((((0+a)+0)+a)+0)$

Theorem 33
Premises: $((c+c)+d)\geq a$; $(d+e)\geq 0$; $((c+c)+f)\geq(0+a)$; $(b+g)\geq 0$
Goal: $(((((((c+c)+(c+c))\cdot((c+c)+(c+c)))+((c+c)+d))+(d+e))+((c+c)+f))+(b+g))\geq((((0+a)+0)+(0+a))+0)$

Theorem 34
Goal: $(((0+b)+c)+a)\geq(0+(0+(b+(c+a))))$

Theorem 35
Premises: $(a+d)\geq 0$; $(a+e)\geq(c\cdot c)$; $(e+f)\geq 0$; $(c+g)\geq 0$; $(c+h)\geq(c+g)$; $(c+i)\geq 0$
Goal: $(((((((c\cdot c)\cdot(a+d))+(a+e))\cdot(e+f))\cdot(c+g))+(c+h))\cdot(c+i))\geq((((((0\cdot(a+d))+(c\cdot c))\cdot(e+f))\cdot(c+g))+(c+g))\cdot(c+i))$

Theorem 36
Goal: $(1\cdot(1\cdot(1\cdot a)))\geq(1\cdot((a+0)+0))$

Theorem 37
Premises: $(b+d)\geq b$; $((c+b)+e)\geq c$; $(b+f)\geq a$; $(e+g)\geq(b+f)$
Goal: $(((c+(b+d))+(b+f))+(e+g))\geq((((((c+b)+c)+(-((c+b)+e)))+a)+(b+f))$

Theorem 38
Goal: $((a + (((b + c) \cdot (b + c)) + ((c + b) \cdot b))) \cdot ((c + b) + (c + b))) \geq ((((((b + c) \cdot (c + b)) + ((b + c) \cdot b)) + a) \cdot (c + b)) + ((((((b + c) \cdot (c + b)) + ((b + c) \cdot b)) + a) \cdot (c + b)))$

Theorem 39
Premises: $(c + d) = b$; $((c + b) + e) = (c + d)$; $(a + f) \geq 0$; $(0 + g) \geq 0$; $(g + h) \geq 0$; $(d + i) \geq 0$
Goal: $((((((c+(c+d))+((c+b)+e))\cdot(a+f))\cdot(0+g))\cdot(g+h))\cdot(d+i)) \geq ((((((c+b)+(c+d))\cdot(a+f))\cdot(0+g))\cdot(g+h))\cdot(d+i))$

Theorem 40
Goal: $((((c + a) \cdot b) \cdot b) + (a + c)) \geq ((a + c) + (((a + c) \cdot b) \cdot b))$

Theorem 41
Goal: $(((c + b) + (a + (c + b))) \cdot (\frac{1}{((((1\cdot c)+b)+a)+(c+b))})) \geq (1 \cdot 1)$

Theorem 42
Premises: $(c + d) = b$
Goal: $(((((c\cdot b)+(c^2))\cdot((b+c)\cdot(c\cdot b)))+(c+d))\cdot((((((c\cdot(b+c))\cdot(b+c))\cdot c)\cdot b)+b)) \geq ((((((c\cdot b)+(c^2))\cdot((b+c)\cdot(c\cdot b)))+(c+d))^2)$

Theorem 43
Premises: $(a + d) = b$; $(d + e) = a$; $(c + f) \geq 0$; $((b + b) + g) \geq 0$
Goal: $((1 \cdot (c + f)) \cdot ((b + b) + g)) \geq (((((b + b) + a) \cdot (\frac{1}{(0+((b+(a+d))+(d+e)))})) \cdot (c + f)) \cdot ((b + b) + g))$

Theorem 44
Goal: $(((((a \cdot 1) \cdot a) \cdot 1) \cdot b) + (((a \cdot 1) \cdot (a \cdot 1)) \cdot (a \cdot a))) \geq (1 \cdot ((((a \cdot a) \cdot 1) \cdot b) + (((a \cdot a) \cdot 1) \cdot (a \cdot a))))$

Theorem 45
Premises: $((c + 0) + d) \geq b$; $(1 + e) \geq a$
Goal: $((0 + ((c + 0) + d)) + (1 + e)) \geq (((0 + (-((c \cdot 1) + (-(c + 0))))) + b) + a)$

Theorem 46
Premises: $(c + d) \geq (a \cdot c)$
Goal: $(((1 \cdot (1 \cdot (a \cdot (a \cdot c)))) \cdot ((1 \cdot ((a \cdot a) \cdot c)) + 0)) + (c + d)) \geq (0 + (a \cdot c))$

Theorem 47
Premises: $(c + d) \geq c$
Goal: $((c \cdot (0 + c))^2) \geq (((0 + ((c \cdot (0 + c)) \cdot (c^2))) + c) + (-(c + d)))$

Theorem 48
Premises: $(a + d) = b$
Goal: $(1 \cdot ((b + b) + (-(1 \cdot (b + (a + d)))))) \geq (1 \cdot (0 \cdot 1))$

Theorem 49
Premises: $((c \cdot b) + d) = a$; $((c \cdot b) + e) \geq b$
Goal: $(((b \cdot b) \cdot (a \cdot (c \cdot b))) + ((c \cdot b) + e)) \geq ((((b \cdot b) \cdot a) \cdot (c \cdot b)) + b)$

Theorem 50
Goal: $(((a + c) \cdot (c + a)) + ((a \cdot (c + a)) + ((c \cdot c) + (c \cdot a)))) \geq (((a + c) \cdot ((c + a) + (c + a))) \cdot 1)$

Update on FLoP, a Reinforcement Learning based Theorem Prover *

Zsolt Zombori[1], Adrián Csiszárik[1], Henryk Michalewski[3], Cezary Kaliszyk[2], and Josef Urban[4]

[1] Alfréd Rényi Institute of Mathematics, Budapest
[2] University of Innsbruck
[3] University of Warsaw, Google
[4] Czech Technical University in Prague

1 Introduction

The FLoP system was built to allow for experimenting with advanced reinforcement learning (RL) methods applied to guide theorem proving. Its particular focus is to enable learning from and generalizing to long proofs, which is a largely unsolved challenge in theorem proving. The system is very flexible in terms of what it can learn from: even a single training environment (proof) can result in meaningful generalization. On the other hand, FLoP is simplistic in several ways: 1) it learns from manually extracted features, 2) it can overfit in some learning scenarios and 3) its merits have so far been demonstrated only on a very simple dataset. Here we only address 1), the problem of feature extraction.

We present ongoing work that aims to use graph neural networks (gnn) [9] for feature extraction. Gnns have been used to learn features of logic formulae on several supervised tasks, e.g. [3, 7, 8, 2]. However, there are very few experiments with such extractors in a reinforcement learning setting. RL models are typically convolutional and dense networks. Related exceptions are [6] and [5] that use graph extractors. However, while these systems use intertwined iterations of proof search and supervised learning, FLoP uses a pure reinforcement learning loop.

We consider learned formula embedding as a stepping stone for more involved projects that combine machine learning and theorem proving. In Appendix A and B we briefly present two such project proposals planned as future work.

2 Feature extraction

Machine learning models require inputs embedded into some Euclidean space $\mathbb{R}^n$. However, when it comes to learning to guide a theorem prover, states and actions are given as logical formulae and it is highly unclear how to turn them into fixed length vectors. An often used approach is to do some manual feature extraction. Currently, FLoP extracts triples of adjacent nodes in the formula trees as features. These features convey some statistically relevant

*ZZ and AC were supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008. HM was supported by the Polish National Science Center grant UMO-2018/29/B/ST6/02959. CK was supported by ERC grant no. 714034 *SMART*. JU was funded by the *AI4REASON* ERC Consolidator grant nr. 649043, the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

information, however, a large part of the semantics is lost. Another approach that is gaining popularity is to represent formulae as graphs and use graph neural networks to produce an embedding vector. Their promise is to adapt feature extraction both to the data and the problem, i.e., to produce an embedding that best fits the current learning task.

3 Embedding with Graph Neural Networks

A gnn takes a labelled graph as input. Each node has some *inital embedding* vector. The initial embedding is refined in multiple iterations using a learnable *updater model*: the new embedding is calculated from previous embeddings of its neighbourhood. Hence, it exploits the structure of the graph, allowing information to propagate along the edges. We perform a fixed number of update operations, called *hops* to obtain the final embedding of the nodes. Finally, some *aggregation* operation creates a single embedding of the graph from that of its nodes.

Projects using gnns show a large variance with respect to how the input is turned into a graph. We present our proposed approach by comparing it to two recent variants: [7] and [8, 6].

NeuroSAT [7] embeds propositional formulae in conjunctive normal form (CNF). The resulting graph has 2 kinds of nodes (clauses, literals) and 2 kinds of edges (from literals to their containing clauses, between negated literal pairs). Thanks to the small number of node/edge labels, each kind of interaction is represented by separate neural networks in the update step.

FormulaNet and Graph Embeddings for HOList [8, 6] embed formulae of higher order logic. The graph is the abstract syntax tree of the formula. The number of different symbols that can occur in the input is not bounded, so a single update operation is performed on all nodes. Node type information is preserved in a learnable initial embedding. Function application is curried in the syntax tree, so each node has at most two children, i.e., we only need two types of edges. Identical subexpressions are merged. A major complication, that was not present in [7] is the representation of variables. [8, 6] collapse all variables into a single "VAR" symbol.

FLoP In FLoP, we embed first order formulae in CNF. Our implementation is very similar to that of [6], we start from the syntax tree, with two differences: 1) The initial embedding is a fixed random vector. 2) Variables are not collapsed into a single node. Rather, they are wrapped into a "VAR" function and are normalised to ensure that they are renaming invariant. This setup ensures that the formula is recoverable from the graph: the initial embedding vector of the nth variable (according to a preorder traversal) is the same for all inputs.

4 Graph Embedding in FLoP

FLoP is built on the leanCoP connection tableau calculus, so its current state is given by the set of valid actions and the partial tableau tree, with the following main components: the current goal, the branch leading to the current goal, remaining open goals and the currently applicable lemmas. At each step, a policy network computes a probability distribution over the valid

actions. Each component of the input is currently a hand crafted feature vector, which can be easily replaced with the embedding network described above.

This is an ongoing effort and in our talk we will present first results using graph embeddings in FLoP. As a proof of concept, we have done some supervised experiments that use our embedding network: we collected theorem proving attempts from FLoP training and trained to predict if a (state, action) pair can lead to success. We achieved 100% training accuracy on a training set of 20000 entries. We are working to see how well it generalizes.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017.

[2] Karel Chvalovsky. Top-down neural model for formulae. In *International Conference on Learning Representations*, 2019.

[3] Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. Enigma-ng: Efficient neural and gradient-boosted inference guidance for e. In *CADE*, 2019.

[4] The MPTP Challenge. http://www.tptp.org/Seminars/MizarVerification/TheMPTPChallenge.html. Accessed: 2019-05-20.

[5] Miroslav Olšák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *CoRR*, 2019.

[6] Aditya Paliwal, Sarah M. Loos, Markus N. Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. *CoRR*, abs/1905.10006, 2019.

[7] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. *CoRR*, abs/1802.03685, 2018.

[8] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2783–2793, 2017.

[9] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks, 2019. cite arxiv:1901.00596Comment: updated tables and references.

Appendix A Project Plan: Bolzano-Weierstrass Theorem

The MPTP Challenge [4] consists of the Bolzano-Weierstrass theorem and its 252 auxiliary lemmas, constituting a relatively small, consistent problem domain. One part of the challenge is to prove the theorem and all lemmas from scratch using in each derivation only basic axioms, hence forcing long proofs. In this setup, we believe that curriculum learning can be very useful and we intend to try to tackle the challenge with FLoP.

Appendix B Project Plan: Backward Hindsight Experience Replay

Hindsight Experience Replay (HER) [1] is a clever approach to alleviate reward sparsity problems in RL environments. Its core idea is to take an unsuccessful exploration trajectory, observe state S that it reached (as opposed to target state T) and then replay the same trajectory, while pretending that the target state is now S. During the replay, the agent is rewarded for reaching the new target.

HER is directly applicable to a theorem proving environment that performs forward reasoning: each theorem proving attempt some valid consequences of the axioms, even if not the target conjecture, so it makes sense to assume the new target in the replay. However, it is not obvious how to do it for backward reasoning.

The aim of our project is to redesign HER for the setting of a backward theorem prover.

B.1 Setup

We want to train a backward theorem prover, i.e., one that starts from a target formula (goal) and at each inference step reduces the current goal to a list of other goals. Once a goal is identical to some axiom or previously known lemma, the goal is closed and we can proceed to try to prove the remaining goals. The proof is complete when all goals have been closed.

B.2 Core Idea

We use Hindsight Experience Replay to provide denser reward to the guidance model. Consider a single theorem proving attempt. If all goals are closed, then we have obtained a proof of the target and we can give positive reward to the policy. If there are some open goals, then we can pretend that those goals were among the initial axioms and give positive reward in this modified setting.

B.3 Components

The system has four components:

1. **Embedder e**: takes a formula and maps it into a vector in $\mathbb{R}^n$. This is most likely a graph neural network.
2. **Aggregator c**: takes a set of formula (axiom) embeddings $e(a_1), e(a_2) \dots e(a_k)$ and maps it into a single aggregate embedding, which represents the conjuction of the formulae. This could be a recurrent neural network, though some permutation invariant solution would be best.
3. **Policy p**: takes a goal embedding **and an aggregate axiom embedding** and returns an action probability distribution.
4. **Value v**: takes a goal embedding **and an aggregate axiom embedding** and returns a scalar value of the goal, **given the axioms**.

These components are trained together, end-to-end.

B.4 Training

We iterate the steps below:

1. Select a problem with goal g_0 and axioms (lemmas) $a_1, a_2, \dots a_k$
2. Compute initial goal embedding $e_{g_0} = e(g_0)$ and aggregate axiom embedding $e_a = c(e(a_0), e(a_1), \dots e(a_k))$
3. Try to prove the goal based on the current policy $p(e_{g_i}, e_a)$
4. Perform a gradient step based on the proving attempt for the value and policy, propagating the gradients all the way to the aggreagator and embedder as well.
5. If the proof attempt failed, i.e., we are left with open goals $og_0, og_1, \dots og_l$, then
   (a) Compute new aggregate embedding $e_{\hat{a}} = c(e(a_0), e(a_1), \dots e(a_k), e(og_0), e(og_1), \dots e(og_l))$
   (b) Replay the same inference steps with the new aggregate axiom embedding in the policy $p(e_{g_i}, e_{\hat{a}})$
   (c) The open goals (og_i) are now axioms, so the proof is complete, hence we give positive reward and perform a gradient step.

B.5 Benefits

- We provide positive reward for every single theorem proving attempt.
- The policy receives a representation of the axiom set (knowledge base) and can make more informed decisions.
- This works with any RL algorithm. There is no need of a DAGGER like setup, with separate phases of data collection and supervised learning.

B.6 Difficulties

- Building a meaningful aggregate embedding of the available knowledge base (all axioms and lemmas) might be hard and might be very slow. Some ideas to address this:
  - Use premise selection to restrict the aggregator to a handful of lemmas.
  - Precompute the aggregate embedding of all the lemmas and only "incorporate" the embeddings of the axioms to the aggregate lemma embedding for each problem
- Different open goals might be related due to sharing some variables. When we add the open goals as new axioms in HER, we have to make sure that the axioms are consistent. E.g. when we have two open goals $f(X)$ and $\neg f(X)$, there is no way to add axioms that satisfy both.

Learning Complex Actions from Proofs in Theorem Proving*

Zsolt Zombori[1] and Josef Urban[2]

[1] Alfréd Rényi Institute of Mathematics, Budapest
[2] Czech Technical University in Prague

Introduction We propose to develop procedures that extend simple theorem provers with complex actions that are the result of learning. Learning is based on traces of successful proof attempts: we will use inductive logic programming (ILP) [5] to learn Prolog programs that can reproduce (some repetitive parts of) the proofs found. Such programs are then incorporated into the next iteration of the prover as new actions, resulting in a hierarchy of more advanced provers. This approach has several motivations:

1. Long, repetitive parts of the proofs can be delegated to algorithms that are not burdened with search, resulting in shorter proof search. This is a well established approach used in current proof assistants and SMT solvers that implement a number of *manually* designed algorithms, tactics and decision procedures combined in various ways with the proof search.
2. We believe such algorithms are gradually learned by humans who generalize over the proofs found so far. Our goal is to emulate this process, starting from simple actions and generating more and more complex ones, in a similar way as when human mathematicians develop more and more advanced proving methods. The generated programs are typically much shorter and easier to understand than the proof traces, hence they foster human understanding.
3. With more and more recent work related to theorem proving using learned guidance, we argue that theorem provers should start moving focus from simple statistical learning and guidance of the primitive actions to setups that also learn symbolically new (more complicated) actions (executable symbolic programs) that are added to the portfolio of statistically guided proof actions. Compared to standalone statistical learning, symbolic programs enjoy a number of interesting properties - among others the possibility to formally prove their correctness for certain classes of inputs.

Learning Setup We use the rlCoP [4] system, based on Monte Carlo Tree Search (MCTS) guiding the leanCoP [6] connection tableau prover. We have back-ported the RL (reinforcement learning) extensions to the Prolog language in which leanCoP was originally implemented. We call this system plCoP. Prolog was chosen because both prover actions and ILP-learned programs are easy to represent as Prolog clauses and it is easy to incorporate a learned program into the set of valid inference steps. rlCoP learning consists of iterations of proof search using MCTS (data collection) and model fitting (training XGBoost [1] policy and value regressors), following the approach in [8].

One can very naturally interweave this process with occasional ILP-style program generation. Once the prover has generated some proof traces, we try to generate a program that reproduces

*ZZ was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008. JU was funded by the *AI4REASON* ERC Consolidator grant nr. 649043, the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

the proofs. This program is incorporated into the prover as a new action: selecting this action corresponds to executing the program on the current goal. The program is very similar to ITP style tactics or SMT style decision procedures, with the important novelty that it was learned from proof traces.

Program Generation using Simple Inductive Logic Programming A proof trace in leanCoP is a sequence of goal-action pairs. A goal is a literal and an action is an ordered formula in clausal normal form, called *contrapositive*. For each step there is some substitution σ that unifies goal G and the negation of the first literal of contrapositive $A = A_1 \vee A_2 \vee \ldots \vee A_n$, i.e. $G\sigma = \neg A_1\sigma$. Such a pair can be easily turned into the following Prolog clause:

$A_1\sigma$:- $\neg A_2\sigma, \neg A_3\sigma \ldots \neg A_n\sigma$.

This rule is an instance of A specialized to goal G and it is typically too specific to be used in the final generated program. However, if we consider several instantiations of the contrapositive A (coming from one or more proofs), then we can compute the Least General Generalization (lgg) [7] of the instances. For each contrapositive that occurs in the proof traces, we create a new clause using the lgg operator. Next, the clauses are ordered (we illustrate program generation in Appendix A):

1. If the head of clause C_1 is more specific than that of clause C_2, then C_1 comes before C_2.
2. If two clauses are incomparable with respect to head specificity, then the one that occurs more frequently in the proof traces comes first.

Experiments Our experiments are preliminary. We ran plCoP on simple arithmetic equalities of the form $N_1\{\cdot,+\}N_2 = N_3$ with N_i nonnegative integers, using the axioms of Robinson Arithmetic, described in Appendix B. Proofs of these problems have a strong shared structure, however, they can get very long as numbers increase. The training set consisted of all 200 problems with $N_1, N_2 < 10$. We ran two experiments: one using standard leanCoP and another using leanCoP extended with paramodulation. Adding paramodulation to leanCoP makes it a more powerful prover when it comes to equational reasoning. However, the presence of new valid actions makes it harder to navigate in the search space. The addition of paramodulation is so far manual, but it is also an example of a more complicated action that we can (given enough background knowledge) try to learn automatically from many proof traces that use equality and congruence axioms.

The programs generated from plCoP proof traces can be found in Appendix C (leanCoP only) and Appendix D (leanCoP plus paramodulation). In both cases, the generated programs can fall into an infinite loop by repeatedly applying some unproductive rules. However, such loops can be avoided by enforcing regularity, i.e., the proof search is restricted to proofs where no literal occurs more than once in the active path. This is a well known optimization that does not affect completeness. We can ensure regularity by writing a simple Prolog interpreter that keeps track of all literals on the current branch. We provide the interpreter code in Appendix E. Once regularity is ensured, the programs in both experiments can generate proofs for all problems, irrespective of the numbers inside. In this example, the learned program solves a very specific class of problems, so its benefit is limited, however, plCoP can be extended with it as an extra action and the system can learn when it is worth using it.

Conclusion and Future Work Our project aims to extend theorem provers with complex actions that are learned from proof traces using ILP. We believe that this approach is suitable to

delegate large parts of the proof task to deterministic algorithms, allowing proof search to focus on parts that truly require search. So far, we have back-ported rlCoP to Prolog and experimented with simple ILP for learning arithmetic from the proof traces. The next steps include addition of more advanced ILP learning over richer domains with larger background knowledge and better statistical guidance of the ILP search for suitable Prolog programs analogous to our existing efficient statistical guidance of ATPs [4, 2, 3].

References

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In Pascal Fontaine, editor, *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2019.

[3] Jan Jakubuv and Josef Urban. Hammering mizar by learning clause guidance. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[4] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, pages 8836–8847, 2018.

[5] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

[6] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36:139–161, 2003.

[7] Gordon D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[8] 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–, October 2017.

Appendix A Example Program Generation

Suppose our proof traces use the following contrapositives:

- $X + s(Y) \neq s(X + Y)$ used on goals $0 + s(0) = X$ and $s(0) + s(0) = X$.
- $X = Y \vee X \neq Z \vee Z \neq Y$ used on goals $X + s(0) = s(s(0))$ and $s(0) + X = s(0)$.

The corresponding rule instances and their least general generalizations are

Rule1: $0 + s(0) = s(0 + 0)$.
Rule2: $s(0) + s(0) = s(s(0) + 0)$.
Lgg: $X + s(0) = s(X + 0)$.

Rule1: $X + s(0) = s(s(0))$:- $X + s(0) = Z, Z = s(s(0))$.
Rule2: $s(0) + X = s(0)$:- $s(0) + X = Z, Z = s(0)$.

Lgg: $X + Y = s(V)$:- $X + Y = Z, Z = s(V)$.

We obtained two clauses with heads $X + s(0)$ and $X + Y$. The former is more specific, so that comes first. The resulting program is:

```
X+s(0) = s(X + 0).
X+Y = s(V):-
    X + Y = Z, Z = s(V)
```

Appendix B Axioms of Robinson Arithmetic

Our experiments with Robinson Arithmetic use the following axioms:

- $\forall X : \neg(o = s(X))$
- $\forall X, Y : (s(X) = s(Y)) \Rightarrow (X = Y)$
- $\forall X : plus(X, o) = X$
- $\forall X, Y : plus(X, s(Y)) = s(plus(X, Y))$
- $\forall X : mul(X, o) = o$
- $\forall X, Y : mul(X, s(Y)) = plus(mul(X, Y), X)$

The axioms are automatically extended in leanCoP with rules for handling equality:

- $\forall X : X = X$
- $\forall X, Y : (X = Y) \Rightarrow (Y = X)$
- $\forall X, Y, Z : (X = Y) \wedge (Y = Z) \Rightarrow (X = Z)$
- $\forall X, Y : (X = Y) \Rightarrow (s(X) = s(Y))$
- $\forall X_1, X_2, Y_1, Y_2 : (X_1 = X_2) \wedge (Y_1 = Y_2) \Rightarrow plus(X_1, Y_1) = plus(X_2, Y_2)$
- $\forall X_1, X_2, Y_1, Y_2 : (X_1 = X_2) \wedge (Y_1 = Y_2) \Rightarrow mul(X_1, Y_1) = mul(X_2, Y_2)$

Appendix C Program Generation from Proof Traces in leanCoP

After running plCoP on the set of arithmetic equalities, we used the successful proof traces to generate the following program:

```
eq(mul(A,o),o).
eq(plus(A,s(B)),s(plus(A,B))).
eq(s(A),s(B)):-
    eq(A,B).
eq(plus(A,o),A).
eq(plus(s(s(s(A))),s(s(B))),plus(s(s(s(A))),s(s(B)))):-
    eq(s(s(s(A))),s(s(s(A)))),eq(s(s(B)),s(s(B))).
eq(A,A).
eq(mul(A,s(B)),plus(mul(A,B),A)).
eq(A,B):-
    eq(A,C),eq(C,B).
eq(A,B):-
    eq(B,A).
```

```
eq(A,B):-
    eq(s(A),s(B)).
```

This program – when proof regularity is enforced – can prove any of the arithmetic problems, irrespective of the numbers inside.

The program contains some unnecessary rules, which can be iteratively removed, making sure that the coverage does not change. After pruning, we obtain the following program:

```
eq(mul(A,o),o).
eq(plus(A,s(B)),s(plus(A,B))).
eq(s(A),s(B)):-
    eq(A,B).
eq(plus(A,o),A).
eq(mul(A,s(B)),plus(mul(A,B),A)).
eq(A,B):-
    eq(A,C),eq(C,B).
```

Appendix D Program Generation from Proof Traces in leanCoP Extended with Paramodulation

After running plCoP using paramodulation on the set of arithmetic equalities, we used the successful proof traces to generate the following program:

```
eq(plus(A,s(B)),s(plus(A,B))).
eq(plus(s(A),s(B)),plus(s(A),C)):-
    eq(s(A),s(A)),eq(s(B),C).
eq(plus(A,s(B)),C):-
    eq(s(plus(A,B)),C),true.
eq(s(A),plus(B,s(C))):-
    eq(s(A),s(plus(B,C))),true.
eq(plus(A,o),A).
eq(plus(A,o),B):-
    eq(A,B),true.
eq(A,plus(B,o)):-
    eq(A,B),true.
eq(A,A).
eq(mul(s(s(s(A))),s(s(s(B)))),plus(mul(s(s(s(A))),s(s(B))),s(s(s(A))))).
eq(mul(s(o),s(s(o))),mul(s(o),plus(s(s(o)),o))):-
    eq(s(o),s(o)),eq(s(s(o)),plus(s(s(o)),o)).
eq(mul(A,s(B)),C):-
    eq(plus(mul(A,B),A),C),true.
eq(A,mul(s(B),s(C))):-
    eq(A,plus(mul(s(B),C),s(B))),true.
eq(mul(A,o),o).
eq(s(plus(A,B)),C):-
    eq(plus(A,s(B)),C),true.
eq(s(A),s(B)):-
```

```
    eq(A,B).
eq(A,B):-
    eq(B,A).
eq(A,B):-
    eq(s(A),s(B)).
eq(A,B):-
    eq(A,C),eq(C,B).
```

This program – when proof regularity is enforced – can prove any of the arithmetic problems, irrespective of the numbers inside.

The program contains some unnecessary rules, which can be iteratively removed, making sure that the coverage does not change. After pruning, we obtain the following program:

```
eq(plus(A,s(B)),C):-
    eq(s(plus(A,B)),C),true.
eq(s(A),plus(B,s(C))):-
    eq(s(A),s(plus(B,C))),true.
eq(plus(A,o),B):-
    eq(A,B),true.
eq(A,plus(B,o)):-
    eq(A,B),true.
eq(A,A).
eq(mul(A,s(B)),C):-
    eq(plus(mul(A,B),A),C),true.
eq(A,mul(s(B),s(C))):-
    eq(A,plus(mul(s(B),C),s(B))),true.
eq(mul(A,o),o).
eq(s(A),s(B)):-
    eq(A,B).
```

Appendix E Prolog Interpreter

We provide a small Prolog interpreter (written in Prolog) that extends Prolog by adding reduction step and restricting proof search to regular proofs.

```
execute(true, _, []):- !.
execute((G1, G2), Path, Proof):- !,
    execute(G1, Path, Proof0),
    execute(G2, Path, Proof1),
    append(Proof0, Proof1, Proof).
execute(Goal, Path, Proof):-
    ( has_loop(Goal, Path) -> fail
    ; reduction(Goal, Path, Proof)
    ; extension(Goal, Path, Proof)
    ).

negate(neg(Goal), Goal):- !.
negate(Goal, neg(Goal)).
```

```
has_loop(Goal, Path):-
    member(G, Path), G == Goal, !.

reduction(Goal, Path, [red(Clause)]):-
    negate(Goal, NegGoal),
    member(NegGoal, Path),
    copy_term(Goal-NegGoal, Clause).
```