AITP 2021

Sixth Conference on
Artificial Intelligence and Theorem Proving

Abstracts of the Talks

September 5–11, 2021, Aussois, France

Preface

This volume contains the abstracts of the talks presented at AITP 2021: Sixth Conference on Artificial Intelligence and Theorem Proving held September 5–11, 2021 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 2021 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 2021. 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 project SMART (no. 714034), 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!

September 2021

Michael Douglas
Cezary Kaliszyk
Ramana Kumar
Stephan Schulz
Josef Urban

Program Committee

Jasmin Blanchette	Vrije Universiteit Amsterdam
Ulrich Furbach	University of Koblenz
Thibault Gauthier	Czech Technical University in Prague
Thomas Hales	University of Pittsburgh
Sean Holden	University of Cambridge
Mikolas Janota	Czech Technical University in Prague
Michael Kinyon	University of Denver
Peter Koepke	University of Bonn
Michael Kohlhase	FAU Erlangen-Nürnberg
Konstantin Korovin konstantin.	The University of Manchester
Adam Pease	Infosys
Geoff Sutcliffe	University of Miami
Christian Szegedy	Google
Sarah Winkler	Free University of Bozen-Bolzano
Yuhuai Wu	University of Toronto
Zsolt Zombori	Hungarian Academy of Sciences

Additional Reviewers

Jan Frederik Schaefer	Anne Baanen
Miroslav Olšák	Jonas Betzendahl
Stanisław Purgał	Jannis Limperg
Jannis Limperg	

Table of Contents

Invited Talk: Interpretable Deep Learning for Physics via Symbolic Regression 7
Miles Cranmer

Invited Talk: ML applications to string theory 8
Fabian Ruhle

Invited Talk: Mathematics in the Scholarly Literature 9
Lucy Lu Wang

Project Proposal: Model-Based Optimization of Strategy Schedules 10
Filip Bártek and Martin Suda

ITP Automation in Practice: A User Study on Tactician 13
Lasse Blaauwbroek and Herman Geuvers

Characteristic Subsets of TPTP Benchmarks 16
Karel Chvalovský and Jan Jakubův

Learning Reasoning Components 20
Karel Chvalovský, Jan Jakubův, Miroslav Olšák, and Josef Urban

Creation of a modular proof assistant engine for a logic e-tutor 25
Jakub Dakowski, Aleksandra Draszewska, Barbara Adamska, Dominika Juszczak, Łukasz Abramowicz, and Robert Szymański

Dealing with Soft Types in Naproche's Logical Backend 28
Adrian De Lon, Peter Koepke, and Anton Lorenzen

Handling Cryptomorphism in Proof Assistants and Automated Concept Formation 30
Floris van Doorn, Michael R. Douglas and David McAllester

Computer Verification for Historians of Philosophy 32
Landon D. C. Elkind

Mining counterexamples for wide-signature algebras with an Isabelle server 36
Wesley Fussner and Boris Shminke

Synthesis of Recursive Functions from Sequences of Natural Numbers 39
Thibault Gauthier

Parental Guidance in E 42
Zarathustra Amadeus Goertzel, Jan Jakubův, and Josef Urban

Contrastive finetuning of generative language models for informal premise selection 46
Jesse Michael Han, Tao Xu, Stanislas Polu, Arvind Neelakantan, and Alec Radford

Representing First-order Problems for Heuristic Selection 50
Edvard K. Holden and Konstantin Korovin

Towards Graph Neural Networks for SMT Portfolios 54
Jan Hůla, David Mojžíšek, and Mikoláš Janota

Minimal Generating Sets in Magmas 57
Mikoláš Janota, António Morgado, and Petr Vojtěchovský

Learning SMT Enumeration 60
Mikoláš Janota, Jelle Piepenbrock, and Bartosz Piotrowski

LISA: Language models of ISAbelle proofs 63
Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu

Ordering Subgoals in a Backward Chaining Prover 66
Gergő Csaba Kertész, Gergely Papp, Péter Szeredi, Dániel Varga, and Zsolt Zombori

Designing a Theorem Prover for Reinforcement Learning and Neural Guidance 70
Jonathan Laurent and André Platzer

(Auto)Complete this Proof: Decentralized Proof Generation via Smart Contracts 74
Jin Xing Lim, Barnabé Monnot, Georgios Piliouras, and Shaowei Lin

Faster Smarter Proof by Induction in Isabelle/HOL with Definitional Quantifiers 78
Yutaka Nagashima

A Corpus of Spatial Reasoning Problems 81
Adam Pease, Paulo Santos, and Alexandre Rademaker

Learning Equational Theorem Proving 84
Jelle Piepenbrock, Tom Heskes, Mikoláš Janota, and Josef Urban

Deep Learning for Temporal Logics 87
Frederik Schmitt, Christopher Hahn, Jens U. Kreber, Markus N. Rabe, and Bernd Finkbeiner

A Closer Look at Successful Clause Derivations Through the Lens of Recursive Neural Networks 92
Martin Suda

Dreaming to Prove 96
Kristóf Szabó and Zsolt Zombori

Retrieval-Augmented Proof Step Synthesis 100
Christian Szegedy, Markus Rabe, and Henryk Michalewski

NaturalProofs: Mathematics meets Natural Language 103
Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho

Latent Action Space for Efficient Planning in Theorem Proving 107
Minchao Wu and Yuhuai Wu

Decision Trees for Tactic Prediction in Coq 113
Liao Zhang, Lasse Blaauwbroek, Bartosz Piotrowski, Cezary Kaliszyk, and Josef Urban

Invited Talk: Interpretable Deep Learning for Physics via Symbolic Regression

Miles Cranmer

Princeton University

Abstract

If we train a neural network on some dynamical system in some region of phase space, and it learns a way to execute the dynamics more efficiently than a handwritten code, how do we distill physical insight from the learned model? In this talk, I will argue that symbolic learning should play a major role in the process of interpreting a machine learning model for physical systems. I will discuss our method for converting an MLP-based deep neural network - which has been trained on a physical system - into a symbolic model. Our method makes use of our genetic algorithm-based symbolic regression tool, "PySR", applied to each latent space of the network. One of the problems with this process is working with the fact that neural networks have high-dimensional latent spaces, and genetic algorithms scale poorly with the number of features. To work around this issue, I'll then introduce our "Disentangled Sparsity Network", which encourages a neural network to learn an easy-to-interpret representation. I will then share several recent applications of our techniques to real physical systems, and the various insights we have discovered and rediscovered.

Invited Talk: ML applications to string theory

Fabian Ruhle

CERN

Abstract

String theory has evolved into a complex framework used to address and advance open problems in physics and mathematics. Recently, machine learning techniques have been applied to address problems arising in this context. In this talk, I will provide an overview of these recent developments. In more detail, many properties of string theory are topological in nature and can hence be described by discrete mathematical data. This often results in computationally hard combinatorial problems or in Diophantine equations, which have been successfully addressed using Reinforcement Learning, genetic algorithms, and SAT solvers. Other properties lead to differential equations that depend on continuous properties of space-time. These have been addressed using fast optimizers and differentiation provided by ML libraries.

Invited Talk: Mathematics in the Scholarly Literature

Lucy Lu Wang

Allen Institute for AI

Abstract

Formulas, equations, proofs, and symbols are used throughout the scholarly literature, in mathematics papers, but also throughout engineering, the sciences, and beyond. Modeling the interplay between scientific text and mathematics is an important part of document understanding. In this talk, I will introduce corpora and tools released by Semantic Scholar that can be used to study mathematics in context. Specifically, I will discuss S2ORC, a structured corpus of scientific papers created to support scientific NLP and text mining. S2ORC contains over 700K mathematics papers, along with many other papers in math-heavy domains such as Computer Science and Physics, a treasure trove of mathematics content. I will also briefly discuss several ongoing initiatives at Semantic Scholar that aim to understand and explain math equations using the paper context around these equations.

Project Proposal: Model-Based Optimization of Strategy Schedules*

Filip Bártek and Martin Suda

Czech Technical University in Prague

1 Introduction of the main task

Automated theorem provers (ATPs) such as Vampire [4] are parameterized by a large number of options that control the proof search heuristics. We call each valid combination of option values of a given prover a strategy.

Given a prover A, a problem $p \in \mathcal{P}$, and a strategy $s \in \mathcal{S}$, the outcome of running A on p using s often varies greatly depending on s. For example, an execution of Vampire with the option `--saturation_algorithm fmb` (finite model building) [4] searches for a model of the problem and thus is only suitable for satisfiable problems, while the saturation algorithm `discount` searches for a refutation by iterative inference and is especially suitable for unsatisfiable problems. On a problem set that contains both satisfiable and unsatisfiable problems, strategies with the saturation algorithms `fmb` and `discount` are complementary.

Strategy scheduling [6] is a common way to utilize the complementarity of various strategies: Given a problem, the prover runs a sequence of independent time-limited solution attempts, each with a different strategy. A question naturally follows: Given a prover, a problem, and a runtime budget, how can we find a strategy schedule that optimizes the expected performance of the prover on the problem?

Several successful attempts have been explored to answer this and other closely related questions. Vampire offers a so-called CASC mode [4], which chooses a schedule based on the syntactic features of the input problem. MaLeS [5] first initializes a set of useful strategies and then dynamically constructs a schedule of these strategies for an input problem. Given a constraint satisfaction problem (CSP), CPHYDRA [2] schedules eight solvers optimally with respect to a model trained in advance. Hydra [8] constructs a portfolio of strategies using an arbitrary algorithm configurator (for example, Sequential Model-based Algorithm Configuration (SMAC) [3]) and a portfolio builder (for example, SATzilla [9]).

In this proposed project, we intend to design and evaluate a method for data-parsimonious construction of a strategy schedule: Given a prover and a distribution of problems, the method should propose a schedule with a high expected probability of solving a problem from the input distribution, while keeping the number of evaluations of the prover during the training small.

2 Solution outline

We propose to search for a good schedule by Sequential Model-Based Optimization (SMBO) [3], that is, by iterating between fitting a regression model of runtime for a given problem and strategy, and gathering additional training data.

*This work is supported by the Czech Science Foundation project no. 20-06390Y (JUNIOR grant), and the Grant Agency of the Czech Technical University in Prague, grant no. SGS20/215/OHK3/3T/37.

The training data consists of triples of the form (p, s, t), where $p \in \mathcal{P}$ is a problem, $s \in \mathcal{S}$ is a strategy, and $t \in \mathbb{R}$ is a PAR10[1] [1] score of solving problem p with strategy s. Each problem is represented by a vector of syntactic features, and each strategy is defined as a vector of numeric and categorical values. Thus, it is possible to fit an empirical performance model $\hat{\pi}(T|P, S)$ that predicts the PAR10 score T (as a random variable) for problem P and strategy S. We will model the score by a random forest or a Gaussian process, which will allow us to capture the uncertainty of the model. We denote the ground truth score distribution by $\pi(T|P, S)$.

A schedule $f : \mathcal{S} \to \mathbb{R}_{\geq 0}$ allocates a time slice to each strategy. For practical reasons, we assume that f allocates a nonzero time slice to only a finite number of strategies. When the prover runs with a strategy s on problem p, the probability of failure to solve the problem in time limit $f(s)$ is $\pi(T > f(s)|P = p, S = s)$. The probability that the schedule f fails to solve problem p is $\prod_{s \in \mathcal{S}, f(s)>0} \pi(T > f(s)|P = p, S = s)$. An optimum schedule $f^*_{\pi,B}$ minimizes the expected number of unsolved problems $C = \sum_{p \in \mathcal{P}} \prod_{s \in \mathcal{S}, f(s)>0} \pi(T > f(s)|P = p, S = s)$ under the runtime budget constraint $\sum_{s \in \mathcal{S}, f(s)>0} f(s) \leq B$.

To sample a new data point, we choose a problem p and a strategy s in a way that combines exploration and exploitation in the context of strategy scheduling using our current score model $\hat{\pi}$. The choice of the sampling scheme is the central research question of this project.

Once the training has finished, it is necessary to extract a schedule from the performance model $\hat{\pi}$. We plan to construct a complete schedule by splitting the time budget into slices, and iteratively assigning each slice either to a new strategy found by local search, or to one of the strategies that have already been assigned at least one slice, minimizing the expected number of unsolved problems C greedily.

To assess the practical utility of the proposed system and especially the schedule-directed sampling scheme, the system will be implemented and empirically compared with a combination of the greedy schedule constructor Hydra [8] and the general-purpose algorithm configurator SMAC [3], which uses a sampling scheme oriented at finding a single best strategy.

3 Discussion

The space of valid strategies is vast, which makes the search for useful strategies difficult compared to the task of algorithm selection. The training problem set is typically large and contains groups of similar problems. Generalizing across strategies and problems becomes crucial to modeling the runtime in a data-efficient manner. Care must be taken not to overfit to the training problem set.

The trained model may overfit to a particular first-order logic representation of the input problem, while many equivalent representations of the problem exist. E.g., equivalent representations that differ only in the order of clauses, the order of literals in clauses, the orientation of equalities, or the names of the symbols. Probabilistic modeling may allow us to capture this ambiguity in a rigorous fashion.

The choice of the schedule quality criterion depends on the intended use. As an alternative to the expected number of problems solved within the runtime budget, the expected time to solve an arbitrary problem could be optimized.

The task of budgeted strategy scheduling is parameterized by the runtime budget B. A question of potential interest is: Can a schedule construction scheme be trained without reference to a particular runtime budget value?

[1] For a solution attempt, the penalized average runtime with a penalty factor of 10 (PAR10) is the actual runtime in case the solution is found within the time limit, or 10 times the time limit in case the attempt times out. Alternatively, survival analysis [7] may allow a more rigorous handling of timeouts.

References

[1] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237:41–58, aug 2016.

[2] Derek Bridge, Eoin O'Mahony, and Barry O'Sullivan. Case-based reasoning for autonomous constraint solving. In *Autonomous Search*, volume 9783642214349, pages 73–95. Springer-Verlag Berlin Heidelberg, oct 2012.

[3] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6683 LNCS, pages 507–523. Springer, Berlin, Heidelberg, 2011.

[4] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

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

[6] Tanel Tammet. Towards efficient subsumption. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1421, pages 427–441. Springer Verlag, 1998.

[7] Alexander Tornede, Marcel Wever, Stefan Werner, Felix Mohr, and Eyke Hüllermeier. Run2survive: a decision-theoretic approach to algorithm selection based on survival analysis. In *Asian Conference on Machine Learning*, pages 737–752. PMLR, 2020.

[8] Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, page 210–216. AAAI Press, 2010.

[9] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, jul 2008.

ITP Automation in Practice: A User Study on Tactician

Lasse Blaauwbroek[1]*and Herman Geuvers[2]

[1] Czech Institute for Informatics, Robotics and Cybernetics, Czech Republic
Radboud University Nijmegen, the Netherlands
lasse@blaauwbroek.eu
[2] Radboud University Nijmegen, the Netherlands
Technical University Eidhoven, the Netherlands
herman@cs.ru.nl

We present the initial results of a study on the real-world usage patterns of the Tactician automation plugin for Coq. Telemetry was collected from students of two courses on type theory and proof assistants (specifically Coq).

Many new automation techniques for interactive theorem proving through machine learning and other methods have been presented in recent times. These techniques are usually experimentally evaluated by attempting to automatically re-prove existing mathematical developments. This is an important metric to judge automation by. However, due to the interactive nature of proof assistants, this does not tell the whole story. In practice, automation tools are used in a feedback loop, where users continually refine lemmas, theorems and their proofs, until they "go through". We wish to know in what way different types of automation tools can assist in this. Existing work on this is hard to come by, and often consists of (valuable) user testimonies, such as for Sledgehammer [3]. The only study with hard data we are aware of is REPLica [4], which aims to study the general usage patterns of Coq users. Because many members of our community are the creators, users and teachers of their tools, we have an opportunity to obtain more direct and scientific evidence by collecting telemetry from real-life usage situations.

Here, we present initial work in this direction, where we instrument the Tactician [2, 1] plugin for Coq while used in a pedagogical setting. Tactician is a machine learning plugin that adds two new functions to Coq: The `Suggest` command provides a list of tactics to the users that might advance the current proof, and the `search` tactic attempts to autonomously finish the current proof. We test Tactician over the run of two introductory Master level courses on type theory and proof assistants (specifically Coq). Since these courses serve as a first introduction to proof assistants, our study mainly concerns the ability of Tactician to aid in a pedagogical setting. During the courses, we provided the students with a distribution of Coq with a special version of Tactician installed that collects data about their usage of the two commands described above and automatically transmits it to a remove server. All data was collected anonymously and it was made clear to students that participation was entirely voluntary and would not have any bearing on their course grade. In addition to automated data collection, we also asked students to complete a short survey about their experience at the end of the course. To introduce students to the automation, an simple tutorial Coq file was used that showcased the new commands. In addition, some attention was given on the subject during lectures.

Each time a student executed either the `Suggest` command or `search` tactic, data was collected, including the date and time, an anonymous user identifier, the operating system, the size

*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)

of Tactician's learned model and the proof state the command was executed on. For `Suggest` we additionally recorded the suggestions presented to the users. For `search` we recorded the amount of time it was run by the users, whether or not it was successful, the amount of inference steps performed, a witness for the found proof, and the trace through the search space of that proof.

Results Due to space limits we will only present some initial results and conclusions here. For both courses together, a total of 74 students were enrolled. The final exam was attended by 60 students, and 56 students turned in a project (in one course students were allowed to do their projects in pairs, which 14 pairs did). Telemetry was received at least once from 29 students. Most of these students were part of the second course. During the first course there were some hiccups with the software and the way it was introduced to students that caused adoption to be low.

After filtering out testing telemetry, telemetry generated by teachers, and telemetry originating from the tutorial file, we obtained data from 1605 unique usages of the commands. Of these, 1100 originated from `Suggest` and 505 from `search`. Of the searches, 225 (48.9%) were solved, while 225 (44.6%) were aborted. In the remaining 33 cases, the entire search space was exhausted. Adoption of the automation by students was highly irregular. Below, we show usage counts from individual students from high to low.

total:	404	222	156	134	125	110	73	56	56	43	26	26	18	17	16	16	...
`Suggest:`	403	222	134	56	37	35	34	19	16	16	15	14	13	13	11	10	...
`search:`	121	95	88	39	37	37	20	16	15	9	8	7	4	2	1	1	...

The `Suggest` command saw quite some usage, at least by some students. Our survey indicates that it is especially useful in the beginning, when students need to be reminded of the available tactics often. It is easier to run this command than to look tactics up in Coq's manual. However, for educational purposes, this comes with a caveat: To increase students understanding of proof assistants, it is desirable for them to understand the most basic building blocks (tactics) of a proof (`intro`, `apply`, `rewrite`,..). However, suggestions will often come in the form of more high-level tactics that are not suitable for pedagogical reasons. For this reason, the decision was made to introduce `Suggest` only after this initial learning phase was complete.

Adoption of the `search` command was much lower, which is somewhat surprising given that it solved the students goal almost half of the time. We think that two factors may have contributed to this: (1) Due to the Covid pandemic, all teaching and tutoring happened remotely which resulted in less contact between students and teachers and therefore fewer opportunities to expose students to the automation. (2) Our survey indicates that several students (understandably) mistook the `search` tactic for Coq's built-in `Search` command (which helps users find lemmas). This once again gives credence to the famous quote *"there are only two hard things in Computer Science: cache invalidation and naming things."* As a result of this new insight we are in the market for a new name for `search`.

The median amount of time it took for `search` solve a proof was 0.03 seconds with a maximum of 115 seconds. In case of failure, students waited to abort `search` for a median of 14 seconds and a maximum of 940 seconds. This indicates that the time automation has to do its work is limited, but should be enough to pick most of the low-hanging fruit. The number of tactics executed in the current lemma before `search` was executed was a median of 2 for in case `search` was successful and 3 when aborted. One might expect to see a longer trace in case of success (because arguably the remaining proof would be easier), but we are unable to observe this here.

References

[1] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. Tactic learning and proving for the coq proof assistant. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 138–150. EasyChair, 2020.

[2] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician - A seamless, interactive tactic learner and prover for coq. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 271–277. Springer, 2020.

[3] Lawrence C. Paulson. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In Renate A. Schmidt, Stephan Schulz, and Boris Konev, editors, *Proceedings of the 2nd Workshop on Practical Aspects of Automated Reasoning, PAAR-2010, Edinburgh, Scotland, UK, July 14, 2010*, volume 9 of *EPiC Series in Computing*, pages 1–10. EasyChair, 2010.

[4] Talia Ringer, Alex Sanchez-Stern, Dan Grossman, and Sorin Lerner. Replica: REPL instrumentation for coq analysis. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 99–113. ACM, 2020.

Characteristic Subsets of TPTP Benchmarks*

Karel Chvalovský and Jan Jakubův

Czech Technical University in Prague, Prague, Czech Republic

1 Motivation: ATP Evaluation over Large Benchmarks

When developing an Automated Theorem Prover (ATP), like E [21], Vampire [16], or Prover9 [18], one often needs to evaluate the prover over a large set of benchmark problems. This is typically done to empirically evaluate a new implementation when a new feature or a new proof search strategy are implemented in the prover.

There are several standardized and well-established libraries of first-order problems maintained with the goal to help developers to evaluate the generality of their provers. First of all, there is the TPTP library of problems in the TPTP language [22], which became widely adopted by the ATP community. Next, a large number of problems are being translated from large mathematical libraries of interactive theorem provers like Mizar [10], Coq [7], and Isabelle [19], by dedicated systems [4] like MPTP [24], CoqHammer [8], and SledgeHammer [5]. Moreover, an annual automated theorem prover competition (CADE) [23] introduces special tracks with additional interesting problems, like problems coming from the AIM project [15].

Evaluating a single prover strategy over all first-order problems in TPTP with a standard time limit (like 5 minutes) can easily take several hours, even with the help of massive parallelization. The situation gets even worse when more than one strategy, or a parametric strategy with many arguments, needs to be evaluated. This is usually approached by restricting the evaluation time limit or by selecting a random benchmark subset to obtain the evaluation results in a reasonable time.

On the other hand, within large problem libraries, many of the problems can be syntactically similar or even duplicate. Since similar prover performance can be expected with similar problems, the identification of similar problems can help us to speed up the evaluation. Here we propose a method to construct the *benchmark characteristic subset* by employing standard machine learning methods for clustering [20]. The desired property of this characteristic subset is that it faithfully characterizes all benchmark problems. That is, that any development, like parameter tuning or scheduler construction, performed on this subset yields similar results as the same development performed on all benchmark problems, but faster.

2 Benchmark Clustering and Characteristic Subsets

Our proposed method to construct the *benchmark characteristic subset* is to employ clustering algorithms. Clustering algorithms can partition a set of entities into classes, called *clusters*, such that similar entities appear in the same cluster. As the benchmark characteristic subset should not contain similar problems, we propose to cluster all benchmark problems and to construct the characteristic subset by selecting one or a few problems from each cluster.

*Supported by the ERC Project *SMART* Starting Grant no. 714034, the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902, and the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

Problem Features. To employ clustering algorithms, such as the k-means [17] algorithm, benchmark problems need to be represented by numeric *feature vectors*. We propose two methods to compute feature vectors.

(1) To use the ENIGMA features [12, 13, 6, 11] based on symbol-independent clause syntax. The ENIGMA features are designed to represent first-order clauses by encapsulating various syntactic properties, like clause length, variable count, and various encodings of clause syntax trees. We plan to represent each problem by accumulating ENIGMA features of the problem clauses.

(2) To compute *performance features* obtained from the statistics of short probing prover runs. We perform short resource-limited runs with several E Prover strategies, and we extract several runtime statistics, like the number of processed/generated clauses, the count of performed rewriting steps, and so on.

Clustering. We employ several clustering algorithms, such as the k-means [17] algorithm and the density-based DBSCAN [9]. From each cluster, we select one or a few problem representants, and we add them to the benchmark characteristic subset. Thus, the desired size of the characteristic subset can be controlled by the count of clusters. Hence, we can compute characteristic subsets of various sizes.

Benchmarks. We propose two separate experiments. The first is on the first-order problems from TPTP [22]. The second is on Mizar40 [14] problems coming from translation of the Mizar Mathematical Library [10] to first-order logic.

Evaluation. Once the benchmark characteristic subsets of different sizes are computed, we need to evaluate their quality. The first evaluation method is based on the construction of the *best cover*. Suppose we evaluate the set of prover strategies S on all benchmark problems P. We can then construct the strategy subset $S_0 \subseteq S$ (of a given size) which maximizes the performance on a given benchmark characteristic subset $P_0 \subseteq P$. The subset S_0 is called the *best cover* on P_0. We can compare the performance of S_0 on all problems P with the best cover S' computed on all problems P and measure their relative error. Moreover, we can compute the relative error of the best covers constructed on the random benchmark subsets. Then we can verify that our benchmark characteristic subsets provide a better benchmark characterization than random subsets of the same size.

Another method of evaluation is based on a parameter search performed on a problem subset. For example, we can perform a parameter grid search on the parameters of a parametrized prover strategy. Then we can compare the performance of the best parameters found on a benchmark characteristic subset with the best parameters found on a random subset of the same size.

Related Work. A similar approach to construct a benchmark characteristic subset, limited to the use of performance features and to the evaluation on the best cover construction, has been recently[1] successfully experimented with using the SMT-LIB library [1, 2] with the help of the CVC4 [3] solver to compute performance features. Here we shift our attention to the TPTP problems, we experiment with syntactic problem features, and we provide additional clustering and evaluation methods.

[1] Currently under review.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[2] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

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

[5] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Jürgen Giesel and Reiner Hähnle, editors, *Proc. of the 5th IJCAR, Edinburgh*, volume 6173 of *LNAI*, pages 107–121. Springer, 2010.

[6] Karel Chvalovský, Jan Jakubův, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In *CADE*, volume 11716 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2019.

[7] The Coq Proof Assistant. `http://coq.inria.fr`.

[8] Lukasz Czajka, Burak Ekici, and Cezary Kaliszyk. Concrete semantics with coq and coqhammer. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 53–59. Springer, 2018.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231. AAAI Press, 1996.

[10] Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

[11] Jan Jakubův, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[12] Jan Jakubův 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.

[13] Jan Jakubův and Josef Urban. Enhancing ENIGMA given clause guidance. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

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

[15] 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.

[16] 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.

[17] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[18] William McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[19] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[20] Mahamed Omran, Andries Engelbrecht, and Ayed Salman. An overview of clustering methods. *Intell. Data Anal.*, 11:583–605, 11 2007.

[21] 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.

[22] 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.

[23] Geoff Sutcliffe. The CADE-27 automated theorem proving system competition - CASC-27. *AI Commun.*, 32(5-6):373–389, 2019.

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

Learning Reasoning Components

Karel Chvalovský[1], Jan Jakubův[1], Miroslav Olšák[2], and Josef Urban[1]

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

Introduction We describe two iterative algorithms that combine high-level proof state evaluation and strategic reasoning decisions with guided low-level saturation-style proof search. For each part, we learn the tasks using an efficient logic-aware graph neural network [14] (GNN) recently integrated [8] into the ENIGMA [9, 4] guidance system of E [16, 17]. The general motivation is to explore and develop more human-like reasoning architectures, i.e., systems combining various (Malarious [20]) AI components and learning/reasoning feedback loops, which are (preferably) also competitive with ATPs in resource-controlled large-theory settings.[1]

GNN-ENIGMA The novelty previously introduced by GNN-ENIGMA compared to other saturation provers is that the generated clauses are not ranked immediately and independently of other clauses. Instead, they are judged by the GNN in larger batches and with respect to a large number of already selected clauses – the *context*. The GNN estimates collectively the most useful subset of the context and new clauses (*queries*) by several rounds of message passing, which sees the connections between symbols, terms, literals, and clauses. The GNN is trained on many previous proof searches, estimating which clauses will work together best.

Leapfrogging Our first method implements the idea that the graph-based evaluation of a particular clause may significantly change as new clauses are produced and the context changes. It corresponds to the human-based mathematical exploration, in which initial actions can be done with relatively low confidence and following only uncertain hunches. After some amount of initial exploration is done, clearer patterns often appear, allowing re-evaluation of the approach, focusing on the most promising directions, and discarding of less useful ideas. In tableau-based provers such as leanCoP [15] with a compact notion of a *state*, such methods can be approximated in a reinforcement learning setting by the notion of *big steps* [12] in the Monte-Carlo tree search (MCTS), implementing the standard *explore/exploit* paradigm [7]. In the saturation setting, our proposed algorithm uses short standard saturation runs in the (low-level) exploration phase, after which the set of processed (selected) clauses is re-evaluated and a (high-level, strategic) decision on its most useful subset is made by the GNN. These two phases are iterated in a procedure that we call *leapfrogging*.

In more detail, given a problem consisting of a set of initial clauses $S = S_0$, a saturation-style search (E/ENIGMA) is run on S with an abstract time limit. We may use a fixed limit (e.g., 1000 nontrivial processed clauses) for all runs, or change (e.g. increase) the limits gradually. If the initial run results in a proof or saturation within the limit, the algorithm is finished. If not, we inspect the set of created clauses. We can inspect all generated clauses, or a smaller set, such as the set of all processed clauses. So far, we used the latter because it is typically much smaller and better suits our training methods. This (*large*) set is denoted as L_0. Then we apply a trained graph-based predictor to L_0, which selects a smaller *most promising* subset of L_0, denoted as S_1. We may or may not automatically include also the initial negated conjecture clauses or the whole initial set S_0 in S_1. S_1 is then used as an input to the next limited saturation run of E/ENIGMA. This process is iterated, producing gradually sets S_i and L_i.

[1] Examples of such fair AI-style settings are the global resource limits used in the MPTP Challenge [13] and CASC LTB [18]. Similarly for large ITP benchmarks, e.g., Mizar [11], Flyspeck [10], HOL4 [3], and Isabelle [2].

Table 1: Four leapfrogging runs with different GNN-ENIGMAs

GNN-strategy	original-60s-run	leapfrogging (300-500-60s)	union	added-by-lfrg
G_1	2711	2218	3370	659
G_2	2516	2426	3393	877
G_3	2655	2463	3512	857
G_4	2477	2268	3276	799

Learning Reasoning Components Mathematical problems often have well-separated reasoning and computational components. Examples include numerical calculations, computing derivatives and integrals, performing Boolean algebra in various settings, sequences of standard rewriting and normalization operations in various algebraic theories, etc. Such components of the larger problem can be often solved mostly in isolation from the other components, and only their results are then used together to connect them and solve the larger problem. Human-designed problem solving architectures addressing such decomposition include, e.g., SMT systems, systems such as MetiTarski [1], and a tactic-based learning-guided proof search in systems such as TacticToe [6]. In all these systems, the component procedures or tactics are however *human-designed* and (often painstakingly) human-implemented, with a lot of care both for the components and for the algorithms that merge their results. This seems hard to scale to the large number of complex algorithms and heuristics used in research-level mathematics.

We instead want to *learn* such *targeted components*, expressed as sets of clauses that perform targeted reasoning and computation within the saturation framework.[2] We also want to learn the merging of the results of the components automatically. This is ambitious, but there seems to be growing evidence that such targeted components are being learned in many iterations of GNN-guided proving followed by retraining of the GNNs in our recent large iterative evaluation over Mizar.[3] In these experiments we have significantly extended our previously published results [8],[4] eventually automatically proving 73.5% (more than 40k) of the Mizar theorems. In particular, there are many examples on the project's Github page showing that the GNN is learning to solve more and more involved computations in problems involving differentiation, integration, boolean algebra, algebraic rewriting, etc. Our proposed *Split and Merge* algorithm is therefore to (i) use the GNN to learn to identify interacting reasoning components, (ii) use graph-based and clustering-based algorithms to split the set of clauses into components based on the GNN predictions, (iii) run saturation on the components independently, (iv) possibly merge the most important parts of the components, and (v) iterate. In more detail, we use a modified version of our GNN to predict the graph of future clause interactions in (i), experiment with several (soft) clustering methods for (ii), and again use the GNN to implement (iv).

Experiments The first leapfrogging experiment uses increasing limits on the set of processed clauses (300 and 500) with the final run limited by CPU time (60s). This is done over 28k hard Mizar problems with four differently parameterized GNNs ($G_1, \ldots, G_4$). The methods indeed achieve high complementarity to the original GNN strategies (Table 1), likely thanks to the different context in which the GNNs see the clauses in the subsequent runs. In the first Split and Merge experiment, we get 111 newly solved problems in the Split (component) phase out of 3000 hard problems unsolved in the initial standard saturation run (both using a limit of 1000 processed clauses). Then the Merge phase yields (with the same limit) additional 66 problems.[5] Many of the new proofs indeed show frequent computational patterns (see Appendix A).

[2] Note that this is also *fuzzier* (and possibly easier) than related splitting by *crisp* (neural) conjecturing [5, 19].
[3] https://github.com/ai4reason/ATP_Proofs
[4] The publication of this large evaluation is in preparation.
[5] The full experimental details will be given in the talk.

References

[1] Behzad Akbarpour and Lawrence C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *J. Autom. Reasoning*, 44(3):175–205, 2010.

[2] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for isabelle/hol. *J. Autom. Reason.*, 57(3):219–244, 2016.

[3] Chad E. Brown, Thibault Gauthier, Cezary Kaliszyk, Geoff Sutcliffe, and Josef Urban. GRUNGE: A grand unified ATP challenge. 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 123–141. Springer, 2019.

[4] 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.

[5] 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.

[6] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 125–143. EasyChair, 2017.

[7] John C Gittins. Bandit processes and dynamic allocation indices. *J. the Royal Statistical Society. Series B (Methodological)*, pages 148–177, 1979.

[8] Jan Jakubuv, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[9] 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.

[10] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.

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

[12] 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.

[13] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Machine learner for automated reasoning 0.4 and 0.5. In Stephan Schulz, Leonardo de Moura, and Boris Konev, editors, *4th Workshop on Practical Aspects of Automated Reasoning, PAAR@IJCAR 2014, Vienna, Austria, 2014*, volume 31 of *EPiC Series in Computing*, pages 60–66. EasyChair, 2014.

[14] Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.

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

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

[17] 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.

[18] Geoff Sutcliffe. The 4th IJCAR automated theorem proving system competition - CASC-J4. *AI Commun.*, 22(1):59–72, 2009.

[19] Josef Urban and Jan Jakubuv. First neural conjecturing datasets and experiments. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2020.

[20] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In *IJCAR*, pages 441–456, 2008.

Appendix A Interesting Frequent Computational Patterns

Here we show three of the computationally looking Mizar proofs found automatically only by the Split and Merge algorithm for theorems `T16_FDIFF_5`,[6] `T48_NEWTON`,[7] and `T10_MATRIX_4`.[8]

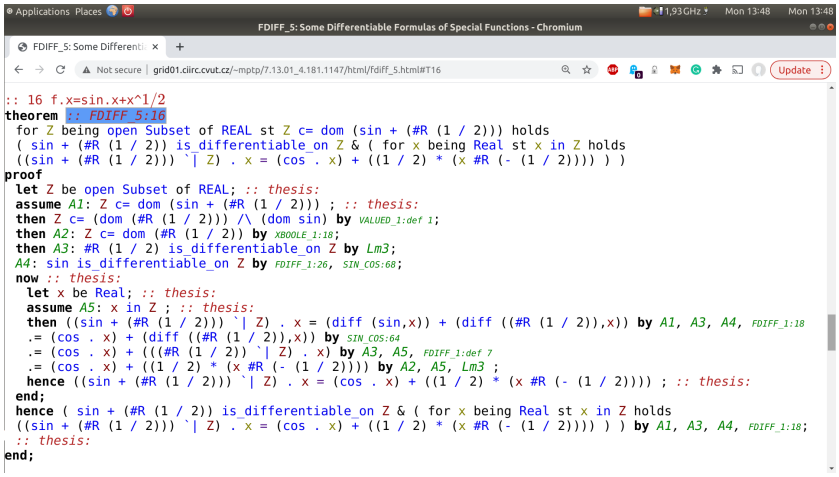


Figure 1: Differentiation – T16_FDIFF_5

[6] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/fdiff_5.html#T16
[7] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/newton.html#T48
[8] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/matrix_4.html#T10

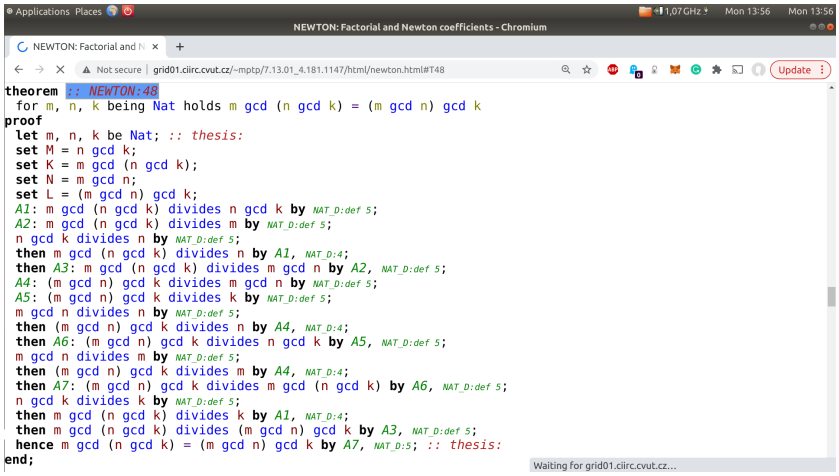


Figure 2: Associativity of gcd by many rewrites – T48_NEWTON

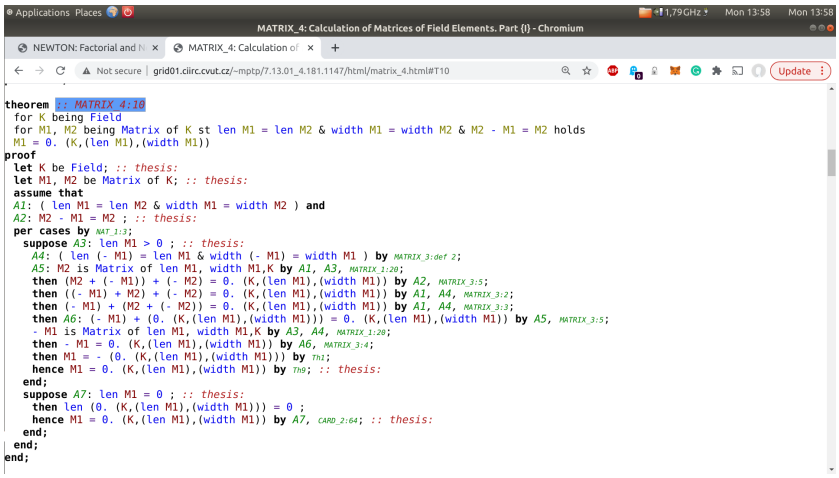


Figure 3: Matrix manipulation – T10_MATRIX_4

Creation of a modular proof assistant engine for a logic e-tutor

Jakub Dakowski[1], Aleksandra Draszewska[1], Barbara Adamska[1], Dominika Juszczak[1], Łukasz Abramowicz[1], and Robert Szymański[1]

Adam Mickiewicz University, Poznań, Poland
larch.amu@gmail.com

1 Background

There have been several attempts at creating Intelligent Tutoring Systems (i.e. applications that provide intelligent teaching support for their users) for several proof methods in formal logic. Huertas [6] counts 8 e-tutors created in the first decade of this century. Nowadays such software usually can give demonstrations and in some cases finish proofs that have already been started [4]. Such programs use many different strategies for obtaining hints. Unfortunately, most of these programs tend to implement only one formal system with a fixed syntax and static output form.

2 Aim

This project, called *Larch*, aims at improving these aspects of Intelligent Tutoring Systems. It's an application designed in hopes of creating a middleman between the researchers trying to implement new hint generation mechanisms and users who usually are unaccustomed to the complicated notation used in formal logic and hard to use interfaces. It was originally created with Analytic Tableaux for Classical Propositional Logic [8] in mind, but recently efforts were made to introduce sequent calculus to this system. Such software created the need for a system that encapsulates the architecture of a typical logic e-tutor in the form of interchangeable plugins.

3 Method

The discussed program was written in Python in line with its versatility and popularity. Python ensures a low entry threshold for creating Larch plugins and at the same time, it doesn't create any major boundaries in terms of possible solutions.

Larch's architecture was inspired by the *Plugin Oriented Programming* paradigm, in which the codebase splits into modular and independent plugin subsystems with a central hub [7]. The plugin system used in *Larch* was created specifically for this project using the built-in `importlib` module. Its central concept is a *Socket*, to which different plugins (these can be both singular files and packages) can be connected via specific functions that are defined in a plugin template. Plugins can still use core libraries and they also have their own socket libraries (internally called `utils` files).

Five sockets have already been created: User Interface, Output generator, a tokenizer (internally called a Lexicon), a Formal System with a solver and a hint generator (called Assistant). Here the last three components of this system are discussed.

The most basic of the three is the Lexicon socket which specifies the alphabet used in the program. It allows for simple customization of the used symbols. For example users can quickly implement a notation which uses LaTeX commands instead of the built-in method. This is implemented using a class provided by the software. This class is in fact a wrapper for the *SLY* package [1], which is a Python library used to create tokenizers and parsers. Such implementation allows the users to define tokens which will be used only while certain formal systems are plugged in. This wrapper also alleviates the hurdle of generating new tokens, thanks in part to the *exrex* package [9].

The Formal socket's role is to perform operations on the proof, check its correctness and finish the proof if needed. As the software was designed to teach Analytic Tableaux for Propositional Logic, one of the biggest dilemmas in this project was to create a system which will not end up as a simple solver. Because of this a notion of strict and naive versions of rules was introduced. When used incorrectly (for example using a rule for implications on a conjunction), naive rules will produce *a* conclusion and strict rules will fail. In a way, strict rules are adequate and their naive counterparts are complete, but not sound. Naive rules are used by the user, while the strict rules are used by the checker and the solver (rule priority is implemented by storing the rules in a tree). These structures are however completely optional and, if desired, this can be implemented from the ground up.

The Formal socket interacts with the Assistant socket to provide the necessary hints and point out user mistakes. In the beginning the Assistant socket also provided a solver, however, while it can still use its own solver, this burden was shifted onto the Formal socket. Assistant socket was given a more didactic role - it provides a knowledge base and generates feedback based on Formal sockets activities. While it has a lot of freedom when it comes to generating hints, commenting on user mistakes is a simple act of interpreting `UserMistake` objects generated by the Formal socket.

Besides the plugin system, there are also different aspects allowing for this amount of freedom in implementation. There is a context management system that ensures both proper information for the user and the data needed to operate on the proof. *Larch* relies on context definition objects created in the language's plugin. These contain information about arguments (both technical, such as type, and user-oriented, such as a readable name), which need to be provided to a rule in order for it to work. To improve the readability of code this plugin management system is connected directly to Python type hinting system. This way the context can be defined with the naive rule functions. The data structure of the whole project consists of two elements. The formulas are stored in a tree structure (implemented using *anytree* [2]) alongside their history and the closedness of their branch. The second part consists of the proof's metadata - right now this is mainly rule history.

4 Applications

As Analytic Tableaux for Classical Propositional Logic was the original purpose of this tool, it has already been implemented. Besides detecting and commenting on mistakes made by the user, the program also produces High Level Hints [3] regarding proof length optimisation and operation precedence. If need be it is also possible to implement Next Step Hints, however these are not recommended — the software can be reduced to a solver when their usage is not controlled.

There are also ongoing works on the sequent calculus for the Intuitionistic Propositional Logic (based on the Swiss calculus by [5]). The Formal plugin has been mostly implemented, but it still lacks a proper solver and a syntax checker. A custom Assistant plugin would also

be beneficial. However, this still shows that, despite the major differences, it is possible to introduce other formal systems in the form of plugins. Implementing the sequent calculus for the Intuitionistic Propositional Logic will also allow for a better presentation of the hint and improved error detection mechanisms.

5 Discussion and future work

The unique value of Larch comes from its ability to encapsulate other tools. With that being said one should ask about the possibilities of such a tool. As of now, it was only tested on a relatively straightforward proof method. This created a situation in which the hint generation couldn't be fully explored, both in the case of different formal systems and Intelligent Tutoring System algorithms. In the future, these perspectives should be explored. The ongoing implementation of the sequent calculus also shows that plugin systems should include diverse built-in libraries. While it certainly is possible to implement new formal methods using what is available now, better tools would facilitate this task.

This work shows a possible application of *Plugin Oriented Programming* in Automated Theorem Proving, but it is not the only one. Similar techniques could be used to modularize other systems and might someday produce a universal standard for logic-related libraries, which would allow almost seamless interchangeability and code reusability. Even in the case of *Larch*, its code might someday be used to create other software, either by reusing the plugins or by reusing the engine gluing them together.

References

[1] David Beazley. Writing parsers and compilers with ply. *PyCon'07*, 2007.

[2] c0fec0de. anytree, Dec 2019.

[3] Christa Cody, Behrooz Mostafavi, and Tiffany Barnes. Investigation of the influence of hint type on problem solving behavior in a logic proof tutor. In *International Conference on Artificial Intelligence in Education*, pages 58–62. Springer, 2018. https://link.springer.com/chapter/10.1007/978-3-319-93846-2_11.

[4] Cristiano Galafassi, Fabiane FP Galafassi, Eliseo B Reategui, and Rosa M Vicari. Evologic: Intelligent tutoring system to teach logic. In *Brazilian Conference on Intelligent Systems*, pages 110–121. Springer, 2020. https://link.springer.com/chapter/10.1007/978-3-030-61377-8_8.

[5] Jacob M Howe. Two loop detection mechanisms: a comparison. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 188–200. Springer, 1997.

[6] Antonia Huertas. Ten years of computer-based tutors for teaching logic 2000-2010: Lessons learned. In *International Congress on Tools for Teaching Logic*, pages 131–140. Springer, 2011. https://link.springer.com/chapter/10.1007/978-3-642-21350-2_16.

[7] Tobias Macey and Thomas Hatch. Making complex software fun and flexible with plugin oriented programming. Podcast.__init__, 2019. https://www.pythonpodcast.com/plugin-oriented-programming-episode-240/.

[8] Raymond M. Smullyan. *First-order logic*. Dover, 1995.

[9] Adam Tauber. exrex, Jun 2017.

Dealing with Soft Types in Naproche's Logical Backend

Adrian De Lon, Peter Koepke, and Anton Lorenzen

University of Bonn, Germany, https://www.math.uni-bonn.de/ag/logik/

In formal mathematics there is a demand for readable or natural input languages. The Formal Abstracts project of Thomas Hales [3] proposes to

- give a statement of the main theorem of each published mathematical paper in a language that is both human- and machine-readable,
- link each term in theorem statements to a precise definition of that term (again in both human- and machine-readable form).

Ideally, the language of Formal Abstracts is part of the common natural language for mathematics, including symbolic terms, and – at the same time – is fully formal with respect to an efficiently implementable formal grammar. The language of Formal Abstracts should thus be a controlled natural language (CNL) for mathematics. Even though the Formal Abstracts project does not demand proofs of main theorems, Formal Abstracts have to be well-formed with respect to the CNL grammar and well-posed mathematically. The latter may involve non-trivial or even substantial proving: the simple well-definedness of the Euclidean norm $\sqrt{a^2+b^2}$ on $\mathbb{R}^2$, e.g., involves checking that the term a^2+b^2 is always non-negative, which requires a proof from the axioms for ordered fields.

The Naproche (Natural Proof Checking) system [2] uses the mathematical CNL ForTheL [5]. Before proving a ForTheL statement, the reasoner checks for its well-definedness in what is called an ontological check which corresponds to type-checking. ForTheL as a natural language employs a "soft" type system where types may have non-trivial definitions. So far, Naproche translates soft types into first-order type guards which are dealt with by an external ATP like E [6]. Although ontological checks are typically shallow in comparison to the proof of the statement itself, the number of ontological checks often surpasses the number of logical checks. This may lead to a high proof burden or even failure of overall checking.

A main motivation for our present work is to internally employ adequate and efficient type mechanisms to model ForTheL's soft types. This requires a redesign of Naproche's logical backend which transforms parse trees into formulae of the internal logic. We have made experiments with the many-sorted logic TFF0 [7] together with automatic coercions for subtypes. This covers, e.g., number systems with types for integers, rationals and reals, or set theories with sets and classes where one has obvious direct embeddings between types.

Coercions are often implemented using type-classes and are inferred either implicitly (for example in Coq [1]) or explicitly by a call to a function like `coe` (in Lean). Type-classes can also be used for subtyping and are used to implement algebraic structures like groups or metric spaces in Lean's mathlib. This technique is very powerful, but requires a worst-case exponential time algorithm.

In the case of direct embeddings (which form a transitive partial order) we can instead use a data structure for transitive closures [4], which can infer a coercion in $O(1)$ time when using arrays. Because we expect the graph of coercions to be relatively sparse we use a binary tree instead for storing the coercions. The asymptotic lookup time is then $O(\log n)$, where n is the number of introduced types. This makes it feasible to check for coercions at every function application. Furthermore, the total time consumed by insertions into the data structure is bounded by $O(nm)$, where n is the number of types and m the number of different coercions

between two types. These coercions yield TFF0 formulae that can be processed directly by E or Vampire without introducing type guards. We now have parsing and checking algorithms that are cubic in the worst case and linear for several interesting formalizations. The ontological checking of those formalizations is accelerated by orders of magnitude and provides informative error messages in case of failure.

The new Naproche backend includes a variety of other enhancements like caching of proof results and automatic translations to Lean. The backend also serves as basis for a future web interface to Naproche.

References

[1] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Yann Coscoy, David Delahaye, Daniel de Rauglaudre, Jean-Christophe Filliâtre, Eduardo Giménez, Hugo Herbelin, et al. The Coq proof assistant reference manual. *INRIA, version*, 6(11), 1999.

[2] Adrian De Lon, Peter Koepke, Anton Lorenzen, Adrian Marti, Marcel Schütz, and Makarius Wenzel. The Isabelle/Naproche natural language proof assistant (to appear). In *CADE-28*, 2021.

[3] Thomas Hales. Formal Abstracts, 2020. URL: https://formalabstracts.github.io/.

[4] G.F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48:273–281, 1986.

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

[6] Stephan Schulz. The E theorem prover. URL: https://eprover.org.

[7] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP typed first-order form with arithmetic. In Nikolaj Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 406–419, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

Handling Cryptomorphism in Proof Assistants and Automated Concept Formation

Floris van Doorn, Michael R. Douglas and David McAllester*

May 11, 2021

Abstract

A mathematical concept can be defined in different ways. For example, a group can be defined as a set together with a group operation, an inverse operation and identity element such that the group axioms hold, or a group can be defined as a set together with a group operation such that the inverse operation and identity element exist. The term cryptomorphism was coined by Birkoff and popularized by Rota as a name for this phenomenon. Cryptomorphism is important in proof assistants and in systems for exploratory mathematics where we want to introduce new concepts while avoiding redundancy. We develop formal definitions of concept and cryptomorphism within a dependent type theory and describe tactics for establishing cryptomorphisms between different concept definitions.

Mathematical structures are at the core of mathematics. Structures are exemplified by number systems (natural numbers, integers, rationals, reals, *etc.*), by algebraic systems (semigroups, groups, modules, *etc.*), and by topological and geometric structures (topological spaces, manifolds, *etc.*). A common approach to group these structures in classes, used for example by Bourbaki, is the signature-axiom class (SAC). A signature-axiom class can be defined by a type expression of dependent type theory whose instances are the structures in that class. These structures consist of one or more carrier sets (sorts) together with data over the sorts — typically constants, functions, and predicates. The class expression (the type expression defining the class) also specifies axioms that the data must satisfy.

Here we are interested in defining and automatically recognizing equivalence between concepts — equivalence between type expressions that define signature-axiom classes. An example of such an equivalence is the two equivalent definitions of a group as a four-tuple of a set, a group operation, an inverse operation and an identity element or the equivalent definition of a group as a pair of a set and a group operation such that the inverse operation and identity element exist. Equivalent concept definitions are called cryptomorphic, a term coined

*All three authors contributed equally to this work.

by Birkoff and popularized by Rota. In modern treatments cryptomorphism is typically defined in terms of category theory. However, we do not wish to require a human user or automated mathematical exploration system to specify a category for each concept. Rather each concept, by virtue of being a type in a dependent type theory, is naturally associated with groupoid structure. In a model of type theory in which all lambda expressions denote functors over this inherent groupoid structure we can define cryptomorphism to simply mean that there exists a pair of lambda expressions establishing a bijection between the two classes. In an appropriate type theory it is possible to show that every class expression is cryptomorphic to a signature-axiom expression where the sorts, data, and axioms are explicitly segregated [1].

A library of concepts (signature-axiom class expressions) naturally forms a concept graph whose nodes are the concepts and whose edges include containment between concepts, for example an Abelian group is a subclass of groups, and functors mapping one concept into another. We wish to avoid redundancy in this concept graph — we want cryptomorphic concepts to be represented by a single node. We consider various types of edges in the concept graph, depending on the relation between two concepts.

Establishing a cryptomorphism between two concepts involves finding a pair of functors establishing a bijection between them. This is a special case of the problem of finding nontrivial functors between concepts. Finding useful functors is fundamental to automated mathematical exploration and we also consider this more general problem. Another important feature of automated mathematical exploration is to recognize that two structures are *not* cryptomorphic. We can do this by showing that the automorphism groups the structures are different.

We also consider the problem of defining an objective function for automated mathematical exploration. For this we consider a richer graph including theorems, proofs and tactics. We consider an objective defined in terms of coverage and size. Coverage is defined in terms of the fraction of automatically generated questions that have "obvious" answers where obviousness is defined by an automated tactic. Size is simply the size of the library used in computing obvious answers. Shorter proofs using more general concepts are considered superior. Automated library expansion can be done by improving coverage at the cost of size and then reducing size for a given coverage. We will propose Reinforcement learning methods for training a value function that predicts the utility (eventual out degree in the library graph) of new conjectures, concepts and tactics.

We have implemented some of these proposals in the Lean ITP system. This includes tactics for finding functors, and a concept graph extraction tool. The latter is derived from a tool developed by Patrick Massot, adapting it to the problem of extracting concepts as we have defined them.

References

[1] David McAllester, *Dependent type theory as related to the Bourbaki notions of structure and isomorphism*, 2021.

Computer Verification for Historians of Philosophy

Landon D. C. Elkind

Western Kentucky University

In this paper I discuss the place of some computational formal methods in doing history of philosophy. Specifically, I describe how to apply interactive theorem provers in textual interpretation and argument reconstruction, to the benefit of both researchers and their broader scholarly community. More concretely, such applications involve formalizing key notions in the argument or text in a manner that the program can read and understand, compiling the file and fixing any runtime errors, and then identifying the philosophical results of such program executions. This last step usually involves producing a human-readable writeup of what was done, problems found, solutions implemented, and lessons learned, perhaps with some excerpted code from the program.

All of this can occur alongside and as a supplement to research produced using more traditional, informal, or non-computational methods in history of philosophy, that is, interactive theorem provers are not a replacement or substitute for critically thinking about a text as historians of philosophy have done for thousands of years, nor are they a replacement for longstanding methods in history of philosophy. Rather, deploying interactive theorem provers can complement and support the usual sort of activity, especially by automating for readers much of the mental labor of verifying arguments and spotting informal and formal fallacies. Perhaps most importantly, utilizing interactive theorem provers can spare others the labor of rewriting formalized arguments again once it has been done once because the source code from argument reconstructions in interactive theorem provers can be open-source. Thus, such code can be downloaded, modified, retooled, and fit to new purposes.

For any historian of philosophy, particularly if one is unfamiliar with interactive theorem provers, the natural question to ask at this stage is, 'Why would I do all of that?' What I described sounds like much more work for not terribly much payoff. After all, arguments of past philosophers can be reconstructed and even formalized on paper, as they have been for over a century, without the need to translate them into some interactive theorem prover's system. Such translations might even negatively effect the work: whichever formal system is used within some interactive theorem prover may have a distorting effect on the past philosopher's argument. So such applications of interactive proof assistants appear at first blush to involve much work with little or no gain, and, as a historian of philosophy sensitive to issues of translation is well-positioned to notice, could even be a substantial step backwards.

In this paper I address these concerns. My view is that interactive theorem provers have already been, and stand to continue being, useful to historians of philosophy. So much may seem obvious after reviewing some research in history of philosophy that leverages interactive theorem provers, which I do below. The novelty in my argument here is to indicate the untapped potential of interactive theorem provers to historians of philosophy. Interactive theorem provers cannot do philosophy for us, or, to make a more modest claim, nothing in my argument hinges on claiming that they can. But the manners in which interactive theorem provers can assist research in history of philosopher are about as plenitudinous as the ways in which computer verification and sharing code assists software development. That is what I argue for here. If this conclusion is true, then interactive theorem provers can be very useful tools indeed.

This paper builds on work by other philosophers in a similar vein, especially those applying interactive theorem provers in philosophy. There have been at multiple applications of the contemporary metaphysical and epistemological notions in philosophy. For example: Fitelson and Zalta (2007) have done axiomatic metaphysics in the interactive theorem prover

Prover9. Benzmüller et al. (2015) have formalized various modal systems and the relations between them in the interactive theorem prover *Isabelle/HOL*. Novak (2015) has used the computer proof-assistant MetaPRL to formalize certain epistemological notions and then used that formalization in MetaPRL to analyze well-known puzzles like the Surprise Examination Paradox. Blumson (2021) has axiomatized classical mereology in *Isabelle/HOL*.

Additionally, interactive theorem provers have been applied to the texts of past figures, including philosophers. For example: Fleuriot (2001) has formalized arguments in Isaac Newton's *Principia Mathematica* using interactive theorem provers. Lokhorst (2011) has formalized Mally's deontic logic and meta-ethical principles in the interactive theorem prover *Prover9*. Alama et al. (2015) have formalized (an interpretation of) Leibniz's theory of concepts in *Prover9*. Benzmüller and Paleo (2015) and Fuenmayor and Benzmüller (2017) have formalized multiple readings of Gödel's ontological argument for the existence of God in *Isabelle/HOL*. Building on the informal work in (Smith, 2020), Koutsoukou-Argyraki (2019) has formalized in *Isabelle* some of Aristotle's proofs and meta-theoretical results concerning his syllogistic.

Citing all these developments, Kirchner et al. (2019, §4) have defended the "benefit from interdisciplinary studies in which computational techniques are applied" and shown some use for interactive theorem provers in metaphysics. Fuenmayor and Benzmüller (2018) discuss the use of interactive theorem provers in formalizing natural language arguments and describe their approach as "computational hemeneutics." As yet, though, philosophers have not considered the general applicability of interactive theorem provers in doing history of philosophy, especially by reference to the scholarly activities of historians of philosophy and to the specific issues raised in applying formal methods, including computational ones like interactive theorem provers, in doing history of philosophy. This lacunae exists in the literature despite the fact that answers to some significant methodological issues are implicitly assumed in some applications of interactive theorem provers just noted, especially in the formalizations of Leibniz's theory of concepts and Gödel's argument for the existence of God. Hence, there is a real need for the present essay.

I organize the paper as follows. First I briefly describe what interactive theorem provers are (§2). The purpose of doing that will be to show how these programs can be used in the philosophical historian's practice of formalizing arguments. Those already familiar with interactive theorem provers might skip this section, referring back to specific details as needed. Next I discuss the metaphilosophical issues raised by formalizing arguments in doing history of philosophy (§3). There I argue that what is commonly called rational reconstruction of arguments can benefit from formalization using interactive proof assistants, and further, that such argument formalization can serve as a helpful complement to the other kinds of investigation undertaken by historians of philosophy. Then I discuss some examples of applying interactive theorem provers in history of philosophy (§4). Considering these applications will support my claim in §5 that formalizing arguments using interactive theorem provers can benefit the practice of doing history of philosophy. Finally I tie all of this discussion together to offer a prospective view of what interactive theorem provers can assist historians of philosophy in doing (§6). To give away the ending, computationally verifying argument reconstructions using such programs offers definite benefits to philosophers working in history of philosophy. Thus interactive theorem provers can be a useful tool to an important activity, rational reconstruction, in doing history of philosophy.

References

Jesse Alama, Paul E. Oppenheimer, and Edward N. Zalta. Automating leibniz's theory of concepts. In *International Conference on Automated Deduction*, pages 73–97. Springer, 2015.

Christoph Benzmüller and Bruno Woltzenlogel Paleo. Interacting with Modal Logics in the Coq Proof Assistant. In L. D. Beklemishev and D. V. Musatov, editors, *Computer Science – Theory and Applications*, volume CSR 2015 of *Lecture Notes in Computer Science, Vol. 9139*, pages 398–411, Switzerland, 2015. Springer. doi: 10.1007/978-3-319-20297-6.

Christoph Benzmüller, Maximilian Claus, and Nik Sultana. Systematic verification of the modal logic cube in isabelle/hol. *arXiv preprint arXiv:1507.08717*, 2015.

Ben Blumson. Mereology. *Archive of Formal Proofs*, March 2021. ISSN 2150-914x. `https://isa-afp.org/entries/Mereology.html`, Formal proof development.

Branden Fitelson and Edward N. Zalta. Steps toward a computational metaphysics. *Journal of Philosophical Logic*, 36(2):227–247, 2007.

Jacques Fleuriot. *A Combination of Geometry Theorem Proving and Nonstandard Analysis with Application to Newton's* Principia. Distinguished Dissertations. Springer-Verlag, 2001. ISBN 978-1-85233-466-6. doi: 10.1007/978-0-85729-329-9.

David Fuenmayor and Christoph Benzmüller. Automating emendations of the ontological argument in intensional higher-order modal logic. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 114–127. Springer, 2017.

David Fuenmayor and Christoph Benzmüller. Computational hermeneutics: An integrated approach for the logical analysis of natural-language arguments. In *International Conference on Logic and Argumentation*, pages 187–207. Springer, 2018.

Daniel Kirchner, Christoph Benzmüller, and Edward N. Zalta. Computer science and metaphysics: A cross-fertilization. *Open Philosophy*, 2(1):230–251, 2019.

Angeliki Koutsoukou-Argyraki. Aristotle's assertoric syllogistic. *Archive of Formal Proofs*, October 2019. ISSN 2150-914x. `https://isa-afp.org/entries/Aristotles_Assertoric_Syllogistic.html`, Formal proof development.

Gert-Jan C. Lokhorst. Computational meta-ethics. *Minds and machines*, 21(2):261–274, 2011.

Natalia Novak. Practical Extraction of Evidence Terms from Common-Knowledge Reasoning. *Electronic Notes in Theoretical Computer Science*, 312(24):143–160, April 2015. doi: https://doi.org/10.1016/j.entcs.2015.04.009.

Robin Smith. Aristotle's Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2020 edition, 2020.

Mining counterexamples for wide-signature algebras with an Isabelle server *

Wesley Fussner[1] and Boris Shminke[1]

Laboratoire J.A. Dieudonné, CNRS, and Université Côte d'Azur, France
wfussner@unice.fr
Boris.SHMINKE@univ-cotedazur.fr

Abstract

We propose an approach for searching for counterexamples of statements about algebraic structures with a medium-sized signature using the Isabelle proof assistant in an efficient, parallel manner. We contribute a Python client Isabelle server and other scripts implementing our approach, and provide results of our computational experiments. In particular, our experiments yield counterexamples that resolve a previously open question regarding the interdependencies between distributive-like identities in residuated binars.

In partnership with automated theorem proving, finite model builders have been applied highly effectively in studies of algebraic structures (e.g., for quasigroups and loops [9]). However, the more fundamental operations there are appearing in an algebraic language, the more expensive computations become. Most successful applications of computational tools concern semigroups, quasigroups, and other algebraic structures with only a few fundamental operations, and algebras of even slightly wider signatures pose considerable challenges (but see, e.g., [11, 6]).

This contribution offers a case study in computer-assisted counterexample construction for algebras of wider signature. Our case study concerns *residuated binars* [4], each of which consists of an algebra $\mathbf{A} = (A, \wedge, \vee, \cdot, \backslash, /)$ such that $(A, \wedge, \vee)$ is lattice, $(A, \cdot)$ is a set with a binary operation, and for all $x, y, z \in A$,

$$x \cdot y \leq z \iff y \leq x\backslash z \iff x \leq z/y. \tag{1}$$

In this context, [4] studies implications among the distributivity identities

$$x \cdot (y \wedge z) = (x \cdot y) \wedge (x \cdot z) \tag{2}$$
$$(x \wedge y) \cdot z = (x \cdot z) \wedge (y \cdot z) \tag{3}$$
$$x\backslash(y \vee z) = (x\backslash y) \vee (x\backslash z) \tag{4}$$
$$(x \vee y)/z = (x/z) \vee (y/z) \tag{5}$$
$$(x \wedge y)\backslash z = (x\backslash z) \vee (y\backslash z) \tag{6}$$
$$x/(y \wedge z) = (x/y) \vee (x/z) \tag{7}$$

in the presence of *lattice distributivity*

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z). \tag{8}$$

*This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 670624). This work has also been supported by the French government, through the 3IA Côte d'Azur Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-19-P3IA-0002. The authors would like to thank Makarius Wenzel for commenting on an early version of the Python client code.

The identities (2)–(7) have proven important in obtaining normal forms for terms in residuated structures. This has found applications everywhere from establishing non-trivial categorical equivalences [5] to obtaining decidability results for models of program execution [13]. These identities are interdependent, and [4] establishes the following:

Theorem 1 (Theorem 2.3 of [4])**.** *Let* **A** *be a residuated binar satisfying* (8)*. Then:*

1. (5),(6) *implies* (4).
2. (4),(7) *implies* (5).
3. (2),(5) *implies* (7).
4. (3),(4) *implies* (6).
5. (6),(2) *implies* (3).
6. (7),(3) *implies* (2).

Furthermore, by exhibiting some small countermodels (of size 4 and 5), [4] shows that the implications announced in the previous theorem completely characterize all interdependencies among (2)–(7) in the presence of lattice distributivity. [4] mentions the case without lattice distributivity as an open question.

This contribution resolves the aforementioned question by the computer-assisted construction of finite countermodels witnessing the failure of the previous theorem in the general (i.e., non-lattice-distributive) case. We obtain:

Theorem 2. *In general, none of the distributivity laws* (2)–(7) *follows from any combination of the others.*

Thanks to its accessibility and amenability to proof simplification strategies (see [7]), working algebraists tend to favor McCune's Prover9/Mace4 [8] for automated work. However, Prover9/Mace4 is now regarded as somewhat out-of-date among researchers in automated deduction. Its model search capability has been gradually surpassed by a series of improvements in the field. The Paradox [3] system introduced so-called static symmetry reduction, a technique reducing the number of isomorphic models (see [1] for Mace4 and Paradox comparison). Later, Kodkod (see [12] for realization details and comparison with Paradox) brought sparse representation of binary relations and even more symmetry-breaking schemes to the process of translating a model-search task into a SAT problem.

We obtain Theorem 2 with the help of Nitpick, a highly efficient tool for the construction of finite counterexamples packaged with the Isabelle proof assistant [14]. Nitpick serves as a translator from Isabelle language to Kodkod, which currently relies on Jingeling ([2], the winner of SAT 2020 competition). Our work takes advantage of the fact that the Isabelle server implementation can run Nitpick tasks in parallel, yielding an environment for countermodel search with large computational advantages. Nevertheless, we observed that model search for residuated binars in cardinalities higher than 14 did not finish even after a week of running an Isabelle server. For Mace4 the boundary was lower, and even models of size larger than 5 became intractable.

To our best knowledge, there was no established way to communicate with the Isabelle server from Python. Thus we created a Python client to the Isabelle server [10] and a parser of Isabelle server logs (from TCP-communicated JSON parcels to LaTeX code for Cayley tables and Hasse diagrams). We conducted our computational experiments yielding Theorem 2 on three Linux machines, the largest having 180 CPU cores (Intel® Xeon® Gold 6254 3.10GHz) and 832 GB of RAM, totaling to about two weeks of wall-clock time. The conclusion of these experiments resulted in mined counterexamples (the largest having the underlying set of 10 elements) supporting the proof of Theorem 2. These counterexamples and the code for getting them are available as supplementary material for this paper.[1]

[1] https://github.com/inpefess/residuated-binars

References

[1] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing finite models by reduction to function-free clause logic. *J. Appl. Log.*, 7(1):58–74, 2009.

[2] A. Biere. CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017. In T. Balyo, M. Heule, and M. Järvisalo, editors, *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*, volume B-2017-1 of *Department of Computer Science Series of Publications B*, pages 14–15. University of Helsinki, 2017.

[3] K. Claessen and N. Sörensson. New techniques that improve MACE-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications*, pages 11–27. Citeseer, 2003.

[4] W. Fussner and P. Jipsen. Distributive laws in residuated binars. *Algebra Universalis*, 80.54, 2019.

[5] N. Galatos and J.G. Raftery. A category equivalence for odd Sugihara monoids and its applications. *J. Pure Appl. Algebra*, 216:2177–2192, 2012.

[6] P. Jipsen and M. Kinyon. Nonassociative right hoops. *Algebra Universalis*, 80.47, 2019.

[7] M. Kinyon. Proof simplification and automated theorem proving. *Philos. Trans. Roy. Soc. A*, 377.20180034, 2019.

[8] W. McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[9] J.D. Phillips and D. Stanovský. Automated theorem proving in quasigroup and loop theory. *AI Communications*, 23:267–283, 2010.

[10] B. Shminke. Python client for isabelle server. https://pypi.org/project/isabelle-client/.

[11] M. Spinks and R. Veroff. Constructive logic with strong negation is a substructural logic I. *Studia Logica*, 88:325–348, 2008.

[12] E. Torlak and D. Jackson. Kodkod: A relational model finder. In *In Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, pages 632–647. Wiley, 2007.

[13] S. van Gool, A. Guatto, G. Metcalfe, and S. Santschi. Time warps, from algebra to algorithms. 2021. Preprint. Available at arXiv:2106.06205.

[14] M. Wenzel. *The Isabelle System Manual*. https://isabelle.in.tum.de/doc/system.pdf.

Synthesis of Recursive Functions from Sequences of Natural Numbers*

Thibault Gauthier

Czech Technical University, Prague

Abstract Given a sequence of natural numbers, we lay the foundations for automatically synthesizing a program that generates this sequence.

Techniques for synthesizing programs have been developed in the course of the last thirty years [3] in the domain of inductive logic programming. The subjects studied cover recursion, higher-order programs, optimal programs and library building. In parallel, program synthesis tasks have been attempted by reinforcement learning. Most recent developments have implemented additional features on top of the learning loop such as a library building mechanism [4] or inputs from a deductive reasoning system [2]. We as well have investigated how to synthesize mathematical objects, that could be considered programs such as: combinators [5], Diophantine equations [5] and set theory formulas [1].

The aim of our project is to automatically construct a program that generates a given sequence of natural numbers. Mastering this task would have important implications. Indeed, finding a "small" program matching a partial sequence would produce a powerful predictive model. Such automation could be applied any time a scientist is confronted with some unknown data representable with natural numbers. To train the automation, we will rely on a set of natural number sequences collected by mathematicians available in the online encyclopedia of integer sequences (OEIS)[6]. Indeed, there are currently 343338 integer sequences in the OEIS, 325191 of which only contain natural numbers. Some of the sequences in this dataset are simple such as the sequence of squares $A000290$. Others are related to open problems. For example, if the sequence $A073101 : [1, 1, 2, 5, 5, 6, 4, 9, 7, 15, 4, 14, 33, \ldots]$ contains 0 then the Erdős-Strauss conjecture is false. For this reason, this problem set is ideal as it provides a gradual learning curve and a large pool of interesting objectives. If a significant portion of the problems is solved, this will indicate that the automation has acquired an understanding of the programming task.

Design of the Programming Language Our language is inspired by the language of recursive functions defined in computability theory. It contains the projections represented by variables in the examples and the basic operators $0, 1, +, -, \times$, *modulo*, *division*, a conditional operator $if(a, b, c)$ that tests if a is equal to 0 and returns b and otherwise returns c, and recursive calls. This language is designed to be expressive enough to construct functions generating interesting sequences in a natural way. By synthesizing a function f from $\mathbb{N}$ to $\mathbb{N}$, we can generate the sequence of elements $f(n)$ for $n \in \mathbb{N}$. For example, a program generating the Fibonacci sequence $A000045$ can be written as:

$$f(n) := \text{ if } n = 0 \text{ then } 0 \text{ else if } n - 1 = 0 \text{ then } 1 \text{ else } f(n-1) + f((n-1)-1)$$

Using a subprogram g, we can also get a program with linear time complexity:

$$g(a, b, c) := \text{ if } a = 0 \text{ then } b \text{ else } g(a-1, c, b+c) \qquad f(n) := g(n, 0, 1)$$

*Supported by the Czech Science Foundation project 20-06390Y

Synthesis Strategy There are many ways to explore the space of programs just defined. Since programs can be represented as trees, we propose to use a bottom-up approach relying on a stack of program trees as intermediate synthesis steps. The synthesis process starts with an empty stack and updates the program stack depending on the operator chosen as illustrated on the following example:

$$[\,] \to [n] \to [n-1] \to [f(n-1)] \to [n, f(n-1)] \to [n + f(n-1)] \to$$

$$[0, n + f(n-1)] \to [n, 0, n + f(n-1)] \to [\text{if } n = 0 \text{ then } 0 \text{ else } n + f(n-1)]$$

After each synthesis step that ends with a single program tree t in the stack, we check if the program $f(n) := t$ generates the targeted sequence. When executing $f(i)$ we limit the number of execution steps to $30 \times (i+1)^3$. This bound was chosen to allow the program to run longer for larger input but only in a cubic manner. Therefore, in general programs with exponential complexity will not be synthesized.

Programs in the stack are complete trees and therefore can be defined by their behavior on the inputs which may facilitate learning their embeddings. In contrast, in a top-down approach the partially synthesized program trees have open branches and thus more complex semantics.

Reinforcement Learning We rely on the deep reinforcement learning framework developed in [5] to train a machine learning model (tree neural networks) on how to select the right synthesis step given learned embeddings for the current program stack and the a prefix of the targeted sequence (first 16 elements). At each generation, a pool of 200 sequences is targeted. The TNN learns from each attempt and re-uses this knowledge for the next generation.

Test Run The framework was tested on a set of 2000 sequences generated from random programs of size less or equal to 30. It was run for 200 generations. After that time, the system had synthesized programs for most of the sequences (first 16 elements).

Conclusion For the described synthesis task, we have designed a small but expressive programming language, have chosen a suitable synthesis strategy and have started testing a reinforcement learning framework. In the future, we would like to scale this approach on the OEIS and improve it by incorporating data augmentation techniques, library building mechanisms and deductive reasoning abilities.

References

[1] Chad E. Brown and Thibault Gauthier. Self-learned formula synthesis in set theory. *CoRR*, abs/1912.01525, 2019.

[2] Yanju Chen, Chenglong Wang, Osbert Bastani, Isil Dillig, and Yu Feng. Program synthesis using deduction-guided reinforcement learning. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 587–610. Springer, 2020.

[3] Andrew Cropper, Sebastijan Dumancic, and Stephen H. Muggleton. Turning 30: New ideas in inductive logic programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4833–4839. ijcai.org, 2020.

[4] Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020.

[5] Thibault Gauthier. Deep reinforcement learning for synthesizing functions in higher-order logic. In Elvira Albert and Laura Kovács, editors, *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, volume 73 of *EPiC Series in Computing*, pages 230–248. EasyChair, 2020.

[6] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. *Electron. J. Comb.*, 1, 1994.

Parental Guidance in E *

Zarathustra Amadeus Goertzel, Jan Jakubův, and Josef Urban

Czech Technical University in Prague, Prague

Parents and Clause Selection in E State-of-the-art automated theorem provers (ATP), such as E [13,14], Prover9 [10], and Vampire [11], are based on the saturation loop paradigm and the *given clause algorithm* [12]. The input problem in first-order logic is translated into a refutationally equivalent set of clauses. The ATP's search for a contradiction maintains two sets of clauses: *processed* (initially empty) and *unprocessed* (initially the input clauses). At each step, one unprocessed clause is selected (*given*), and all of the possible inferences with all the processed clauses are generated (typically using resolution, paramodulation, etc.), extending the unprocessed clause set. The selected clause is then moved to the processed clause set. An important invariant is that all mutual inferences among the processed clauses have been computed at each step.

The selection of the "right" given clause is known to be an important choice-point vital to the success of the proof search. E's strategies consist of *clause evaluation functions* that weigh and prioritize clauses for selection based on their symbols and properties. The ENIGMA systems [2–7] apply various machine learning methods to learn how to select effective given clauses from corpora of previous successful proof searches. Given clause selection does not give the ENIGMA system complete control over the inferred clauses because all inferences between the given clause and clauses in the processed clause set are computed. One reason this can be important is that the ENIGMA systems tend to perform best when run in cooperation with a strong E strategy where each selects half the clauses.

This talk discusses the implementation and experimentation of an ENIGMA system that can "judge children by their parents" to filter out unnecessary inferences between the given clause and processed clauses[1] [2]. It is hoped that pruning the children of irresponsible parents can improve E's performance by allowing clause selection ENIGMA models and E strategies greater focus.

Implementation There have been many versions of ENIGMA, and the latest is ENIGMA Anonymous [4], which uses as the underlying machine learning method either Graph Neural Networks or Gradient Boosted Decision Trees (GBDTs, implemented by LightGBM here) [1,9]. For the GBDTs, clauses are represented by fixed-length numerical vectors based on clause syntax trees and names are anonymized by replacing symbol names with their arities. The goal clauses and theory clauses (which include axioms) are merged to create the goal and theory fixed-length vectors, which represent the clause's context. The three are then concatenated to create the *feature vector*.

*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] The code can be found at https://github.com/zariuq/eprover/tree/parentalguidance_frozen.

[2] And a pre-print of the full paper can be found at https://arxiv.org/abs/2107.06750

Parental guidance can be generally defined as clause evaluation based on (the features of) the parents of the clause (and possibly also on the clause itself). Two methods are evaluated:

1. $\mathcal{P}_{\mathsf{fuse}}$ merges the feature vectors of the parent clauses into one vector, typically by simply adding the feature counts.

2. $\mathcal{P}_{\mathsf{cat}}$ concatenates the feature vectors of the parent clauses to preserve their information in full.

The resulting parent feature vector is concatenated with the goal and theory vectors to create the feature vector for parental guidance.

A GBDT based filter is inserted into E's given clause algorithm so that clauses generated by parents whose scores are below a chosen threshold do not get evaluated by E's clause selection heuristics. Because not all clauses are compatible to mate together, the clause's parents are only sent to the GBDT for evaluation after the clause has been generated and before simplifications are performed. This leverages E's efficient indexing. Because they have two parents, only clauses generated by paramodulation (which implements resolution in E) are evaluated by the parental guidance model. Filtered clauses are stored in the *freezer* set so that E can restore them if the unprocessed clause set becomes empty, which avoids impairing the completeness of the proof search.

Training The experiments are performed[3] on a large benchmark of 57 880 problems[4] originating from the Mizar Mathematical Library (MML) [8] exported to first-order logic by MPTP [15]. The data are split into 3 subsets[5]: (1) 52k problems for *training*, (2) 2896 problems for *development*, and (3) 2896 problems for final evaluation (*holdout*).

First, the *baseline* in this work, called $\mathcal{D}_{\mathsf{large}}$, is a clause selection ENIGMA Anonymous model that is trained over a dataset consisting of at most 3 proofs from ca. 36k problems in the *training* set. The model consists of 150 decision trees of depth 40 with 2048 leaves and is the model that performed best in some prior experiments. The training data for $\mathcal{D}_{\mathsf{large}}$ consists of clauses processed during a proof search: clauses appearing in the proof are labeled *positive* and other clauses are *negative*. When run on the *training* set, $\mathcal{D}_{\mathsf{large}}$ proves 28 495 problems with 30 seconds per problem.

To train parental guidance models, the parents of all generated clauses from the $\mathcal{D}_{\mathsf{large}}$ run on the *trraining* set are used. Two methods of classifying the good pairs of parents are considered:

1. $\mathcal{P}^{\mathsf{proof}}$ classifies parents of only the proof clauses as *positive* and all other generated clauses as *negative*.

2. $\mathcal{P}^{\mathsf{given}}$ classifies parents of all processed (selected) clauses as *positive* and the unprocessed generated clauses as *negative*.

The reasoning behind (2) is that if a clause is selected by a well-trained strategy, then it probably should not be filtered: the aim is to remove only the worst of the children.

In the $\mathcal{P}^{\mathsf{proof}}$ data, the *pos-neg ratio*, the ration of positive to negative clauses, is 1:192. This is is experimentally reduced.

[3] On a server with 36 hyperthreading Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz cores and 755 GB of memory.

[4] http://grid01.ciirc.cvut.cz/~mptp/1147/MPTP2/problems_small_consist.tar.gz

[5] http://grid01.ciirc.cvut.cz/~mptp/Mizar_eval_final_split

Evaluation The parental guidance models are evaluated in combination with $\mathcal{D}_{\mathsf{large}}$ for clause selection with 30 seconds per problem. First a series of grid searches is done on a 300 problem subset of the development set and then the results are compared with the baseline $\mathcal{D}_{\mathsf{large}}$ on the full *development* and *holdout* sets.

The following parameters are tested:

1. the *pos-neg reduction ratio* of negative to positive clauses,
2. the threshold by which to filter clauses,
3. the positive data classification scheme ($\mathcal{P}^{\mathsf{proof}}$ vs $\mathcal{P}^{\mathsf{given}}$),
4. the LightGBM parameters (number of trees, maximum leaves per tree, and maximum tree depth)
5. the parental feature vector creation method ($\mathcal{P}_{\mathsf{fuse}}$ vs $\mathcal{P}_{\mathsf{cat}}$)

The final results can be seen in Table 1. Only considering proof clauses as positive examples ($\mathcal{P}^{\mathsf{proof}}$) outperforms considering all selected clauses ()$\mathcal{P}^{\mathsf{given}}$), which is probably because the data is cleaner and includes no confusing clauses. The low thresholds among the best models indicate that parental guidance works best when only the most obviously irresponsible parents are filtered. The cost of mistakenly filtering a necessary clause is high. Concatenating parent clause features ($\mathcal{P}_{\mathsf{cat}}$) seems far superier to merging them ($\mathcal{P}_{\mathsf{fuse}}$). The improvement of 11.7%, num163 more problems than the baseline, seems highly promising.

model	threshold	solved (D)	solved (H)
$\mathcal{D}_{\mathsf{large}}$	-	1397	1390
$\mathcal{P}^{\mathsf{given}}_{\mathsf{fuse}}+\mathcal{D}_{\mathsf{large}}$	0.05	1411 (+1.0%)	1417 (+1.9%)
$\mathcal{P}^{\mathsf{proof}}_{\mathsf{fuse}}+\mathcal{D}_{\mathsf{large}}$	0.1	1489 (+6.6%)	1486 (+6.9%)
$\mathcal{P}_{\mathsf{cat}}+\mathcal{D}_{\mathsf{large}}$	0.05	1571 (+12.4%)	1553 (+11.7%)

Table 1: Final 30s evaluation on development (D), and holdout (H) compared with $\mathcal{D}_{\mathsf{large}}$.

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] 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, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th*

International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[5] 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.

[6] 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.

[7] 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.

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

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.

[10] 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.

[11] 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.

[12] Ross A. Overbeek. A new class of automated theorem-proving algorithms. *J. ACM*, 21(2):191–200, April 1974.

[13] 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.

[14] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. 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 495–507. Springer, 2019.

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

Contrastive finetuning of generative language models for informal premise selection

Jesse Michael Han[1,2], Tao Xu[1], Stanislas Polu[1], Arvind Neelakantan[1], and Alec Radford[1]

[1] OpenAI
[2] University of Pittsburgh

Introduction

Premise selection [6] is a classic problem in automated theorem proving (ATP) which asks how to select the most relevant lemmas useful for proving a given theorem. As such, it is firmly situated in the domain of formal mathematics and has long been a target for machine learning methods in ATP [9, 3, 4, 5, 10, 1]. In this work, we consider *informal* premise selection, where the statements of premises and theorems are in natural language and labels are given by references to premises in ground truth informal proofs. The NaturalProofs dataset, recently introduced in [12], frames informal premise selection as an information retrieval task.

We explore the applications of pretrained generative language models finetuned on a CLIP-style [8] contrastive objective for retrieval over informal mathematics corpora. We show that WebMath pretraining [7] leads to significant performance gain compared to pretraining only on the same data as GPT-3 [2]. We achieve a new state-of-the-art on the NaturalProofs dataset [12], improving on the previous state-of-the-art by up to 80% while using causal rather than bidirectional transformers and fewer parameters overall.

Methodology

We use decoder-only transformers similar to GPT-3 [2] with $n_{\text{layers}} = 12$, $d_{\text{model}} = 768$, $n_{\text{head}} = 12$, and $d_{\text{head}} = 64$, totalling to $125M$ trainable parameters. After pre-training on the autoregressive language modelling task, we adapt our models for embedding-based retrieval as follows. Given a query/document $\mathbf{x}$, we compute an embedding $\widehat{\mathbf{x}} \in \mathbb{R}^{d_{\text{model}}}$ for $\mathbf{x}$ by taking $\widehat{\mathbf{x}}$ to be the activations for the end-of-text (EOT) token. We finetune our models using an InfoNCE loss [11] exactly analogous to the objective used by CLIP [8]. That is, given a batch of N positive (query, document) pairs, we train the encoder to maximize the cosine similarity of the N positive examples while minimizing the cosine similarity of the $N^2 - N$ negative examples. At test time, we retrieve documents for a given query by maximizing the cosine similarity of their embeddings. We test our methodology on the NaturalProofs dataset [12], which comprises (theorem, premise) pairs extracted from proofs of theorems on ProofWiki. We use the same theorem-wise train/test split in this work.

Unlike CLIP [8] or the BERT-based model studied in NaturalProofs [12], we use the same encoder to embed both queries (theorems) and documents (premises). Since "X is useful to prove Y" is an asymmetric relation and we use a CLIP-style symmetric cross-entropy loss, the encoder must be allowed to distinguish between theorems and references. We do this by simply formatting the inputs to the transformer as

```
Theorem title: <title> <newline> Theorem statement: <statement>

Reference title: <title> <newline> Reference statement: <statement>.
```

During contrastive finetuning, we sample batches of $N = 2048$ pairs by first sampling N theorems from the NaturalProofs train split, and then further sampling a positive reference from the proof of each theorem in the batch. All our models are trained for approximately 7000 steps with the Adam optimizer, using 32 V100 GPUs.
We study three pretraining regimes for the NaturalProofs informal premise selection task:

- **No pretraining.** The model is randomly initialized and only learns theorem/premise representations through contrastive training.
- **GPT-3 style pretraining.** The model is pretrained for 300B tokens on the same data (a mix of filtered CommonCrawl, WebText, books, and Wikipedia) as GPT-3 [2].
- **WebMath pretraining.** Starting from the final snapshot of the previous model, we train for another 72B tokens on the WebMath dataset [7], comprising a mix of math arXiv, Python, Math StackExchange, Math Overflow, and PlanetMath.

We refer to our methodology for informal premise selection as contrastive theorem-premise training (CTPT) and denote the three models above by `ctpt-no-pretrain`, `ctpt-webtext`, and `ctpt-webmath`.

Results and discussion

	recall@10	recall@100	avgp@100	full@100	full@1K
BERT	**20.27**	**59.44**	**14.01**	**27.39**	**70.52**
ctpt-no-pretrain	23.76	54.01	11.91	23.75	56.32
ctpt-webtext	34.39	65.45	17.97	34.76	64.51
ctpt-webmath	**36.92**	**70.39**	**21.53**	**39.49**	**73.52**

Table 1: Our models' performance on the NaturalProofs test set alongside results from [12].

Our main results are displayed in Table 1. The model `ctpt-webmath` outperforms the previous state-of-the-art on all metrics. Our models also utilize 43% fewer parameters since the BERT-based model embeds theorems and references with separate copies of `bert-base-cased` (110M params). It is possible that the `webtext` data contains ProofWiki, but WebMath does not and we consider the significant performance gap between `ctpt-webtext` and `ctpt-webmath` to be of primary interest. We speculate that the models studied in [12] are severely undertrained due to using only 200 randomly sampled negatives for each positive example.

Future directions The results discussed in this extended abstract are preliminary, albeit promising. We plan to ablate the effect of including various components of the pretraining (e.g. Python vs informal math in WebMath, the necessity of `webtext`), as well as the zero-shot performance of our models (i.e. no contrastive finetuning) and potential methods for unsupervised retrieval. We consider the applications of our methodology to premise selection in the formal setting (e.g. inside an ITP or ATP) to also be a promising future direction.

Acknowledgements We thank Raul Puri, Harrison Edwards, Yuhuai Wu, Sean Welleck, and Christian Szegedy for helpful discussions.

References

[1] Kshitij Bansal, Sarah M. Loos, Markus N. 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, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 2019.

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[3] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike von 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.

[4] Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[5] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: Machine learning for sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2013.

[6] Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 378–392. Springer, 2012.

[7] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021.

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

[10] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jirí Vyskocil. Malarea SG1- machine learner for automated reasoning with semantic guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2008.

[11] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018.

[12] Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun

Cho. Naturalproofs: Mathematical theorem proving in natural language. *CoRR*, abs/2104.01112, 2021.

Representing First-order Problems for Heuristic Selection

Edvard K. Holden and Konstantin Korovin

The University of Manchester, U.K.

Abstract

Machine learning (ML) has been applied to Automated Theorem Provers (ATPs) with much success in recent years. However, representing first-order problems effectively as feature vectors remains a major challenge. The performance of an ML approach is bounded by how well the features represent the problems and relates to the task at hand. In this paper, we investigate the effectiveness of different problem representations for heuristic selection embeddings.

Heterogeneous Heuristic Selection and Optimisation

Heuristics are crucial for the success of ATPs, but finding good heuristics is challenging due to their vast parameter space. Additionally, problems are heterogeneous which means that different heuristics are required to solve different problems. Although there have been substantial work on automatically discovering heuristics for first-order reasoning [9, 12, 15, 17], they do not consider the heterogeneous nature of first-order problems.

To tackle this challenge, we developed a system for *heterogeneous heuristic optimisation and scheduling* using machine learning (HOS-ML) [4], illustrated in Figure 1. The key idea behind HOS-ML is to dynamically partition heterogeneous problem sets into homogeneous problem clusters and optimise heuristics for each cluster separately using Bayesian hyper-parameter optimisation. While there is no obvious way of grouping problems into homogeneous clusters, in [4] we proposed to compute clusters through a combination of heuristics evaluation clustering and problem embedding.

HOS-ML consists of three phases. The first phase discovers new heuristics by interleaving Bayesian hyper-parameter optimisation for discovering promising heuristics and dynamic re-clustering into homogeneous problem clusters. The problems are iteratively re-clustered into increasingly finer-grained clusters using heuristics evaluation feature vectors, which are extended as we discover new heuristics. In the second phase, we use constraint programming to construct effective schedules for each homogeneous cluster based on the discovered heuristics. The final phase deploys cluster schedules on unseen problems.

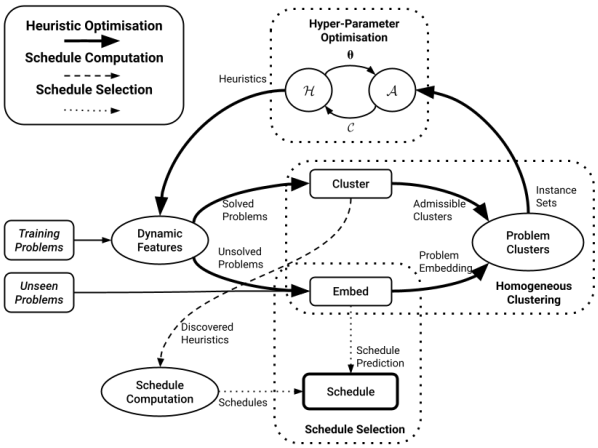


Figure 1: HOS-ML: heuristic optimisation and selection for heterogeneous problems.

A core component of HOS-ML, used in the final phase, is the *heuristics embedding model*, as illustrated by the node "Embed" in Figure 1. The heuristics embedding model maps unseen problems into heuristics evaluation vectors which in turn are used to assign unseen problems into clusters and the corresponding cluster schedules. The heuristics embedding model relies on problem representation using feature vectors. Such problem representations are useful in many applications but present a major challenge. In the following we experimented with different problem representations in the context of HOS-ML.

Problem Representation

A significant challenge is that many ML algorithms and models operate on numerical vectors known as feature vectors, while ATPs deal with sets of formulas. Therefore, applying ML on the formula level requires a feature vector representation of the formulas. However, there is no natural way for mapping tree-structured first-order formulas into one-dimensional numerical vectors.

This impediment becomes even more apparent at the problem-level for tasks such as heuristic selection. Problems consists of sets of formulas and the feature vector must be able to create a representation of this set. These challenges has typically been addressed by using more attainable feature sets at the cost of information loss about the problem's structure [1, 3, 5, 6, 14], or complex graph-based embeddings [8, 13] which can be difficult to train. The features have to contain some information related to the task at hand. For embedding evaluation properties, this means that the features should reveal some information about the behavioural properties of the problem, such that we can predict how a heuristic is going to perform on it. In our experiments we considered three different feature types for this task:

- **Syntactic Features:** These include syntactic properties such as the number equational, Horn, EPR, ground formulas, etc. We used 14 of such features. Such features create a good representation of the problem encoding but do not always reflect the algorithmic properties of formulas.
- **Solver State Features:** Syntactic features do not always reflect solver performance, additionally, the original structure of a problem does not always correspond to the internal representation of the problem in the ATP. To overcome this issues we consider solver state features. These features consist of 155 solver statistics on the solvers' key function calls during a run of the prover including a range of simplification counts and timings. These features are computed by attempting a problem with a single heuristic for a low-timelimit and extracting the solver statistics after termination. The advantage of solver state features is that they directly represent performance of different components of the solver on the problem, in contrast to the syntactic features.
- **Abstract Features:** Symbol based representations can be effective [7, 11] but have a drawback of being sensitive to symbol renaming. One approach to this problem is to use embeddings based on sophisticated graph neural networks [8, 13]. In this paper we investigate a simpler approach using *signature abstraction* by collapsing all signature symbols of the same type into an abstract symbol of this type. Abstract symbols are shared across all problems resulting in common features even when problems have different signatures. Signature abstraction preserve variable dependencies and fragmentic structure of the formulas, e.g., of being EPR, Horn, ground, monadic, etc. After creating an abstraction of the problem on the term, literal and clause level we represent the abstracted problem as a bag-of-words. In our experiments we only used the abstract features computed from the conjectures of the problems.

Problem Embedding

Let H be a set of heuristics, and $\mathbf{s_p}$ the feature vector of p. The admissible embedding model $\mathcal{E}$ learns the mapping between $\mathbf{s_p}$ and the evaluation vector $\mathbf{e_p}$, which encodes the evaluation of the heuristics in H on p. We use multi-label classification to construct the embedding model $\mathcal{E}$. We separate the multi-label classification task into $|H|$ binary classification tasks, where a separate binary classification model $\mathcal{M}_\theta$ is trained for each heuristic θ in H. The final embedding model is defined as $\mathcal{E}(\mathbf{s_p}) = (\mathcal{M}_{\theta_1}(\mathbf{s_p}), \cdots, \mathcal{M}_{\theta_{|H|}}(\mathbf{s_p}))$, and can be used to predict the evaluation vector of any given problem. In this paper, we predict which of the heuristics can solve a problem within a given time limit. We have experimented with different binary classification models such as neural networks, SVMs and tree-based model. In our experiments, the random forest model XGBoost [2] yielded the best performance.

	Syntactic	Solver	Abstract	All
F1-Score	0.76	0.80	0.72	0.81
Geometric Accuracy	0.68	0.75	0.65	0.76
Hamming Loss	0.71	0.76	0.67	0.76

Table 1: The embedding performance of different problem representations on testing problems.

Evaluation

In this experiment we evaluated the quality of the three types of problem representations in the context of HOS-ML: using syntactic, solver state and abstract features. The experiment consists of training the heuristic embedding model for each feature set and evaluating it on a set of unseen test problems. To obtain the experiment problems, we randomly sampled 4000 FOF and CNF problems from the TPTP library (v7.4.0) [16]. Next, we extracted the features of each problem and removed all problems that were solved during feature extraction or did not parse within a 1-second time limit.

To obtain the evaluation vectors we evaluated seventeen iProver [10] heuristics on the experiment problems with a 300 second time limit. Further, we removed the problems that were unsolved by all the heuristics as well as problems with all solutions below five seconds. This results in a problem set consisting of challenging yet solvable problems. Finally, we divided the remaining problems into training and testing sets with a 70-30 split.

We trained heuristic embedding models using XGBoost and different problem representations as described in the previous section. From the results shown in Table 1, we observe that heuristic embedding models can predict heuristic evaluation vectors with high accuracy. The solver state features outperform the two other feature types. We also observe that the performance is slightly increased by combining all three feature sets.

The heuristic embedding model was integrated in our HOS-ML implementation and we are currently evaluating HOS-ML on the full TPTP.

References

[1] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, 53(2):141–172, Aug 2014.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[3] 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.

[4] Edvard K. Holden and Konstantin Korovin. Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. CICM 2021. Submitted.

[5] Edvard K. Holden and Konstantin Korovin. Experiments with selection of theorem proving heuristics. In *Proc. Automated Reasoning Workshop 2019 Bridging the Gap between Theory and Practice*, 2019.

[6] Edvard K. Holden and Konstantin Korovin. SMAC and XGBoost your theorem prover. In *Proc. 4th Conference on Artificial Intelligence and Theorem Proving*, 2019.

[7] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike von 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.

[8] Jan Jakubuv, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In Nicolas Peltier and

Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[9] Jan Jakubuv and Josef Urban. Blistrtune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France*, pages 43–52. ACM, 2017.

[10] Konstantin Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In *IJCAR 2008. Proceedings*, pages 292–298, 2008.

[11] A. S. Kucik and K. Korovin. Premise selection with neural networks and distributed representation of features. *ArXiv e-prints, Abs/1807.10268*, July 2018.

[12] Daniel Kühlwein, Stephan Schulz, and Josef Urban. E-males 1.1. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, 2013. Proceedings*, volume 7898 of *LNCS*, pages 407–413. Springer, 2013.

[13] Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.

[14] Michael Rawson and Giles Reger. Dynamic strategy priority: Empower the strong and abandon the weak. In Boris Konev, Josef Urban, and Philipp Rümmer, editors, *6th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, number 2162 in CEUR Workshop Proceedings, pages 58–71, Aachen, 2018.

[15] Simon Schäfer and Stephan Schulz. Breeding theorem proving heuristics with genetic algorithms. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *GCAI 2015. Global Conference on Artificial Intelligence*, volume 36 of *EPiC Series in Computing*, pages 263–274. EasyChair, 2015.

[16] 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.

[17] Josef Urban. Blistr: The blind strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. EasyChair, 2015.

Towards Graph Neural Networks for SMT Portfolios

Jan Hůla[1,2], David Mojžíšek[1], and Mikoláš Janota[2]

[1] University of Ostrava
{jan.hula21,mojzisek.work}@gmail.com
[2] Czech Technical University in Prague
mikolas.janota@gmail.com

Solvers for *Satisfiability Modulo Theories (SMT)* are the driving force behind software verification, software testing, or software synthesis [1, 2, 3, 4]. These applications often require repeated queries to an SMT solver. This means that the quick response time of the solver is paramount.

An SMT solver receives as input a formula and responds if it is satisfiabile or not. In the case of satisfiability, a satisfying interpretation (model) of the formula is also produced. Since the problem is generally undecidable, solvers often time out or give up. Due to this hardness of the problem, different heuristics may show very different per-instance behaviour in terms of runtime and ability to find a successful solution in SMT solving. This can lead to the scenario in which there is a single best solver on average, but an algorithm selecting the best solver for a specific instance of a problem can yield a better result [5, 6]. This per-instance behaviour is hard to understand for a human, but in accordance with the current trend [7, 8, 9], we aim to predict the best solver using ML methods.

We develop an approach to solver selection in the domain of SMT using a Graph Neural Network (GNN). In contrast to related methods, GNNs do not require manual feature design as they enable discovering relevant features in the raw data. We compare several architectural choices of GNNs which are trained to predict the performance of individual solvers in the chosen benchmarks. Rather than choosing only one solver with the best prediction, we choose n best solvers, order them by the predicted score, and delegate part of the time budget to each of them. We compare our approach to a baseline, which uses bag-of-words as a representation of each formula and gradient boosted trees as a predictor. In the selected benchmarks, we show an improvement over this baseline in terms of the number of solved problems and overall solving time.

GNNs are neural networks that process inputs structured as a graph. This makes them different from other types of neural networks such as Multi-layer Perceptrons, Recurrent Neural Networks, Convolutional Neural Networks, or Transformers, which do not assume any special structure of the input. For this reason, GNNs became popular for processing all kinds of formal structures such as logical expressions, which are naturally represented as trees or directed acyclic graphs.

Most often, additional meta-information for nodes within an input graph is available. For a specific node, this information is encoded as a feature vector of a fixed size. In our case, we use the mapping from symbols corresponding to a given node to one-hot vectors.

Each layer of a GNN updates the feature vectors of all nodes by transforming and aggregating the feature vectors of its neighbour nodes. After several steps of such feature vector transformation, the final single vector is obtained by pooling, see also Figure 1.

The advantage of using a GNN is that the trained transformations are applied locally to each node and the final aggregation operator does not require a specific number of inputs. It allows the graphs to have different number of nodes and structure. Therefore, the trained GNN is applicable to arbitrary graphs. GNN architectures differ in how they achieve the layer-level node feature vector transformation and aggregation. In this contribution, we compare a

Figure 1: The steps conducted during the creation of the input graph from a given SMT formula.

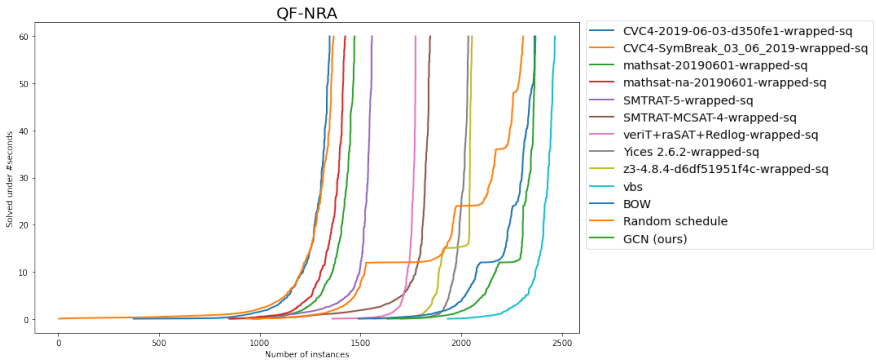


Figure 2: A cactus plot of one of our results on QF_NRA benchmark. The y-axis represents time and the x-axis the number of problems solved under the corresponding time. Virtual best solver is denoted by vbs and represents the upper bound of what could be achieved.

simple Graph Convolutional Network (GCN) [10], Graph Attention Network (GAT) [11], Graph Transformer [12] and Principal Neighbourhood Aggregation (PNA) [13].

We use GNNs for the regression task to predict different solvers runtimes for a specific instance of a problem. Rather than choosing only one solver with the best prediction, we choose n best solvers, order them by the predicted score, and delegate part of the time budget to each of them.

We test our GNN on 4 representative benchmarks: QF_NRA, UFNIA, UFNIA-CONF[1] and TPTP [14]. Figure 2 shows results for one of the considered families (non-linear arithmetic without quantifiers).

To summarize, our work has the following main contributions:

- It applies GNN to rank a portfolio of SMT solvers on a given instance according to suitability. To the best of our knowledge, this is the first time that GNN is applied in the context of SMT.
- The proposed approach schedules n best solvers rather than just picking the best one, which further improves the robustness of the approach.
- The experimental evaluation compares several GNN architectures. This will also be of use to other researchers that wish to apply GNN in the context of SMT for other tasks.

[1]We have collected the different strategies that the solver CVC5 uses to solve UFNIA formulas in the competition so we do not select the solver but the solving strategy.

Acknowledgment

This scientific article is part of the RICAIP project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306. The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902.

References

[1] Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009.

[2] Andrew Reynolds, Viktor Kuncak, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. Refutation-based synthesis in SMT. *Formal Methods Syst. Des.*, 55(2):73–102, 2019.

[3] Leonardo de Moura and Nikolaj Bjørner. Applications and challenges in satisfiability modulo theories. In *Workshop on Invariant Generation (WING)*, volume 1, pages 1–11. EasyChair, 2012.

[4] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. SAGE: whitebox fuzzing for security testing. *Commun. ACM*, 55(3):40–44, 2012.

[5] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.

[6] Serdar Kadioglu, Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann. Algorithm selection and scheduling. In *International Conference on Principles and Practice of Constraint Programming*, pages 454–469. Springer, 2011.

[7] Joseph Scott, Aina Niemetz, Mathias Preiner, Saeed Nejati, and Vijay Ganesh. MachSMT: a machine learning-based algorithm selector for SMT solvers. *Tools and Algorithms for the Construction and Analysis of Systems*, 12652:303, 2020.

[8] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[9] Chris Cameron, Rex Chen, Jason Hartford, and Kevin Leyton-Brown. Predicting propositional satisfiability via end-to-end learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3324–3331, 2020.

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

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

[12] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[13] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *arXiv preprint arXiv:2004.05718*, 2020.

[14] Geoff 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.

Minimal Generating Sets in Magmas*

Mikoláš Janota[1], António Morgado[2], and Petr Vojtěchovský[3]

[1] Czech Technical University, Prague
[2] INESC-ID, Lisboa
[3] University of Denver

A subset S of an algebra A *generates* A if the smallest subalgebra $\langle S\rangle$ of A that contains S is all of A. A generating subset of A of smallest possible cardinality is called a *minimal generating set*. In other words, calculating the closure of S, i.e., applying exhaustively the multiplication operation of A, S generates all the elements of A. The *rank* of an algebra is the cardinality of its minimal generating set.

Finding small and minimal generating sets is of importance in algebra, both theoretically and for the purposes of computations. For instance, a vector space is completely characterized by it rank (that is, dimension) and the underlying field. Groups with a single generator are very easy to understand while groups with two generators can be in some sense arbitrarily complicated. Alternative algebras with two generators are associative and hence relatively easy to understand compared to general alternative algebras. In computational group theory, the efficiency of algorithms often depends heavily on the number of generators given.

We present a method for calculating minimal generating sets in *magmas* (sets with a single binary operation) by means of SAT solvers and integer linear programs. This method cannot compete with specialized algorithms in highly structured magmas, such as groups, but it appears to be efficient in the general case. We focus on *loops*, that is, magmas M with identity element in which all translations $y \mapsto yx$ and $y \mapsto xy$ are bijections.

The main idea is as follows: Let M be a finite magma of size n and let S be any nonempty subset of M. If $\langle S\rangle = M$ then S is a generating set. Otherwise $\langle S\rangle < M$ and *every* generating set of M must contain an element from the complement $M \setminus \langle S\rangle$. (Indeed, if A is a generating set of M such that $A \cap (M \setminus \langle S\rangle) = \emptyset$ then $A \subseteq \langle S\rangle$ and $\langle A\rangle \leq \langle S\rangle < M$, a contradiction.)

Every subset $S \subseteq M$ with $\langle S\rangle < M$ therefore yields a restriction $t(S)$ on every generating set of M, in particular on every minimal generating set of M. This restriction can be expressed as a condition suitable for SAT solvers, namely $t(S) = \bigvee_{x \in M \setminus \langle S\rangle} x$, and can be readily translated into a constraint of an integer linear program (see below).

Given a collection $\{S_i : i \in I\}$ of subsets of M, any generating set must satisfy the conjunction $t(I) = \bigwedge_{i \in I} t(S_i)$. Finding a candidate for a minimal generating set is equivalent to solving the corresponding minimal hitting set problem (which is in general NP-complete).

To prove that M has rank larger than k, it suffices to find a collection $\{S_i : i \in I\}$ of subsets of M for which the following integer linear program is infeasible (unsatisfiable).

$$x \in \{0, 1\} \text{ for every } x \in M, \tag{1}$$

$$\sum_{x \in M - \langle S_i\rangle} x \geq 1 \text{ for every } i \in I, \tag{2}$$

$$\sum_{x \in M} x \leq k. \tag{3}$$

*The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902. This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020, the project INFOCOS with reference PTDC/CCI-COM/32378/2017.

If the above formulation becomes unsatisfiable, k must be increased. If the formulation is satisfiable, we obtain a set of elements that represent a *candidate* $S = \{x \in M : x = 1\}$ for the generator. If $\langle S \rangle = M$, we are done because the candidate is an actual generator; because k was increased only when needed, it is also guaranteed that this generator is minimal. Otherwise, if $\langle S \rangle < M$, we add S to our collection of subsets of M. Effectively, this means adding the restriction $\sum_{x \in M \setminus \langle S \rangle} x \geq 1$ to Equation 1.

This approach can be seen as an instantiation of the framework proposed by Saikko et al. [4], which shows that a class of problems can be tackled by iterative generation of the minimal hitting set problem. We remark that the problem can be directly encoded as a single SAT problem; this formulation is cubic, which has proven prohibitive. Applying SAT technology on the minimal hitting set instances obtained from this process (Equation 1) exhibited poor results. The integer linear programming solver gurobi [2] proved to be far more adequate.

We ran experiments on groups and on *Moufang loops*. Moufang loops are loops satisfying the identity $x(y(xz)) = ((xy)x)z$ and are closely related to the alternative algebras mentioned above.

`GAP` [1] contains extensive libraries of groups. There exist very efficient algorithm for $r(G)$ if G is a solvable group. We calculated $r(G)$ for all nonsolvable groups of order less than 2048. In all cases, we verified the rank $r(G)$ as posted in `GAP`. (In some instances the generating set of G stored in `GAP` is larger than $r(G)$ but then our $r(G)$ can be verified heuristically by methods of `GAP`.)

The package `LOOPS` [3] of `GAP` contains all nonassociative Moufang loops of order $n \leq 64$ and of orders $n = 81$ and $n = 243$. For instance, there are 4262 such loops of order 64 and 5 of order 81. No efficient methods for calculating the rank of Moufang loops are known. We calculated $r(M)$ for all Moufang loops M of the form $M = A \times G$, where A is a Moufang loop from the library of `LOOPS` and G is the cyclic group of order 8. Here, $r(A) > 2$ due to Moufang theorem and $r(G) = 1$. In future experiments we want to include products with all groups of order 8 (there are 5 of them).

All the instances were solved with the average time of 1.75 s. The calculated r remains small for the considered instances, typically 3, 4, 5. Interestingly, the largest considered loops of order $1944 = 243 \times 8$ have all rank 3; only several loops of order $512 = 256 \times 8$ have the maximal rank found 6. The number of iterations needed, i.e., size of Equation 1, is also typically small, in the range of hundreds.

The following remarks are specific to generating sets in (Moufang) loops and groups and play a role in the search.

- If $S \leq M$ and M is a finite Moufang loop then $|S|$ divides $|M|$. (This is false in general loops.)
- If $S \leq M$ and $x, y \in M$ then the cosets xS, yS might interest nontrivially (that is, $xS \cap yS \neq \emptyset$ and $xS \neq yS$) but the cosets xS, S either coincide or are disjoint.
- If $S < M$ then $|S| \leq |M|/2$. Consequently, $|M \setminus S| \geq n/2$ and the number of variables in every term $t(S)$ is large, resulting in a difficult hitting set problem that SAT solvers struggle with.
- The rank $r(M)$ of M is at most $\lfloor \log_2(|M|) \rfloor$.
- If A, B are finite loops then $r(A \times B) \leq r(A) + r(B)$. It is not well understood when the equality holds.

References

[1] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.11.1*, 2021.

[2] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021.

[3] GP Nagy and P Vojtěchovský. LOOPS, a package for gap 4.3. *Download GAP at https://www.gap-system.org/ GAP.*, 2006.

[4] Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In Chitta Baral, James P. Delgrande, and Frank Wolter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*, pages 104–113. AAAI Press, 2016.

Learning SMT Enumeration *

Mikoláš Janota[1], Jelle Piepenbrock[1,2], and Bartosz Piotrowski[1,3]

[1] Czech Technical University, Prague,
[2] Radboud University Nijmegen, The Netherlands
[3] University of Warsaw, Poland

Introduction Even though satisfiability-modulo-theories (SMT) solvers were mainly focused on solving quantifier-free first-order problems, many of them currently support quantified formulas. The main technique used in SMTs to handle quantifiers is *quantifier instantiation*. In this approach the solver intersperses generating ground instances of the quantified formulas and attempts to find a model for the ground formulas. Whenever a model is found, new ground instances are generated. This process continues until a contradiction is found at the ground level (which refutes the problem) or computational resources are exhausted.

A large majority of the instantiations produced during the search are not used in the final proof. The challenge is to produce *useful* instantiations, i.e., instantiations which are likely to contribute to the proof. One of the strategies to find these instances is *enumerative instantiation* [5]. Here, quantified variables in formulas are substituted with candidate ground terms. The set of these terms is induced by the ground model found in the last iteration of the main loop of the algorithm. This set is ordered using some predefined term ordering. In CVC5 this order is determined by *age*, i.e., the terms that exist longer in the formula are used first.

In our work we implement a machine-learning (ML) guidance for term-selection for the instantiations. More precisely, we use a machine-learned, formula-dependent term-scoring function in place of the predefined term ordering.

This is closely related to the work presented at AITP 2019 [2]. There, the authors also experimented with machine-learned guidance for term-selection for instantiations. However, their setting was substantially different. In particular, their machine-learned function served as a binary filter on a set of terms, not as a scoring function inducing an ordering. Moreover, their implementation was impractically slow. We show that the ML-guided SMT solver has improved proving performance when compared to the unguided solver with the same time limit.

Implementation As a basis for our experiments we use a well-established and efficient SMT-solver – CVC5 [1]. To model the term-scoring function we use the LightGBM toolkit [3]. It efficiently implements a versatile and powerful ML algorithm – the gradient boosted trees. The scoring function $S\colon \mathcal{F} \to [0,1]$ takes as its argument features $F(\phi, t)$ of a pair of the quantified formula ϕ and the candidate term t which may be used for instantiation. The returned score is intended to reflect how likely it is that ϕ instantiated with t will be used in the final proof.

As features of (ϕ, t) pairs we use information extracted from CVC5. For every symbol appearing in terms and formulas CVC5 determines its *kind*. These kinds include, e.g., *variable*, *skolem*, *not*, *and*, *plus*, *forall*, etc. We use these syntactic *kinds* to define a *bag-of-words*-type featurizer $\mathrm{BOW}(x)$, where x is a term or a formula, and the information returned by BOW consists of counts of kinds of symbols appearing in x[1]. Additionally, we use 6 numerical features describing the candidate terms: *varFrequency*, *age*, *phase*, *relevant*, *depth*, and *tried*.

*The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902.

[1] For example, $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) = \{\mathit{forall} : 1, \mathit{variable:}\ 1, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$.

varFrequency represent the number of times the variable occurs in the quantifier; *age* and *phase* measure how long the term has been in the candidate pool, with *age* being a more fine-grained measure. *depth* indicates the tree depth of the term. These, together with a disjoint union of BOW(ϕ) and BOW(t) constitute the features $F(\phi, t)$ being an input for the scoring function.

Experiments For evaluation we use SMT-LIB problems from the UFNIA/sledgehammer category. Problems solvable by CVC5 without doing any instantiations are filtered out.

First, we grid-search hyper-parameters for training LightGBM model on a random split. The selected ones are: *num_trees* = 50, *learning_rate* = 0.1, *num_leaves* = 32, *max_depth* = 10.

Then a looping-style evaluation is run (similar to [4, 6]). In the first iteration, an unguided SMT-solver is run on the benchmark and data extracted from the solved problems is used to train the ML model. Then, the model is used to guide the solver in the next iteration of solving the benchmark. The success rate is recorded and examples extracted from the newly solved problems are added to the training set. This solving-training procedure is repeated 20 times. The time limit for each solving attempt is limited to 120 s.

We use three metrics of a success: (1) a number of problems solved in the current iteration, (2) a cumulative number of problems solved so far, (3) an average number of instantiations generated by the solver in the current iteration (it may be seen as an abstract running time).

An ablation study is done by comparing to the standard solver with random perturbations, where the terms ordered by the predefined ordering are randomly swapped with probability 0.1.

The results of the looping evaluation in terms of the metrics (1-3) are presented in Figure 1. The ML-guided solver at the end of the loop is better than the randomized one with respect to all the metrics. Importantly, we see a growing trend in the number of problems solved by the ML-guided solver in individual iterations. However, there is quite high variance in this statistic.

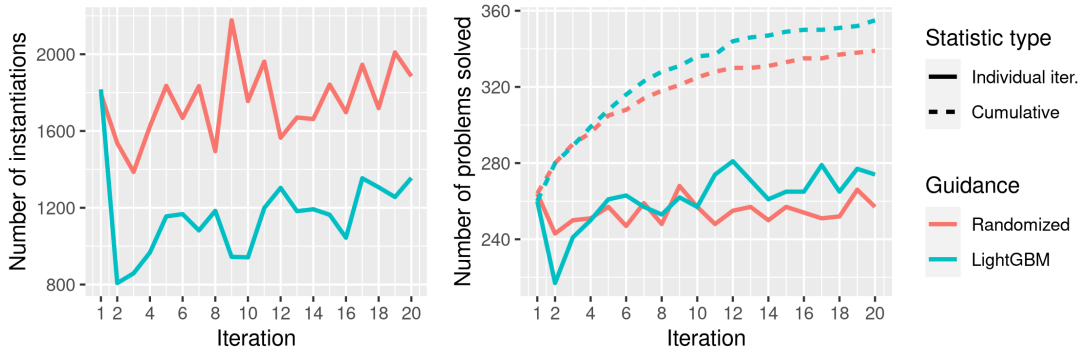


Figure 1: Statictics from the looping evaluation on the UFNIA Sledgehammer problems. The ML-guided solver (blue) *vs* the standard solver with randomly perturbated term orderings (red).

Conclusions and future work We show that ML guidance can be effectively used to guide instantiation within CVC5. There are many possible directions for future work. One of the most important areas for improvement is the treatment of cases where multiple variables in one formula need to be instantiated. Currently we handle each quantifier independently, while a more sophisticated method that takes into account the relations between quantifiers is more natural for the problem. For this, a way to score *tuples* of terms to instantiate with instead of single terms is needed. This will require training a tuple-scoring function and implementing a search procedure guided by this function, like A* search algorithm or a beam-search.

References

[1] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[2] Jasmin Christian Blanchette, Daniel El Ouraoui, Pascal Fontaine, and Cezary Kaliszyk. Machine learning for instance selection in smt solving. In Thomas C. Hales, Cezary Kaliszyk, Ramana Kumar, Stephan Schulz, and Josef Urban, editors, *4th Conference on Artificial Intelligence and Theorem Proving, AITP*, 2019.

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[4] Bartosz Piotrowski and Josef Urban. Atpboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018.

[5] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2018.

[6] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jirí Vyskocil. Malarea SG1- machine learner for automated reasoning with semantic guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2008.

LISA: Language models of ISAbelle proofs

Albert Qiaochu Jiang
University of Oxford
albert594250@gmail.com

Wenda Li
University of Cambridge
wl302@cam.ac.uk

Jesse Michael Han
OpenAI
jessemichaelhan@gmail.com

Yuhuai Wu
University of Toronto
ywu@cs.toronto.edu

ABSTRACT

We introduce an environment that allows interaction with an Isabelle server in an incremental manner. With this environment, we mined the Isabelle standard library and the Archive of Formal Proofs (AFP) and extracted 183K lemmas and theorems. We built language models on this large corpus and showed their effectiveness in proving AFP theorems.

1 INTRODUCTION

There has been a surge of interests recently in applying machine learning models for theorem provers. Examples include [3, 6–8, 12, 14], all of which demonstrate great promises of machine learning models in proving new theorems. In this work, we propose to mine the libraries used by the Interactive Theorem Prover (ITP) Isabelle, namely, the Isabelle standard library and the Archive of Formal Proofs. The libraries have been mined previously for proof method recommendations based on hand-crafted features [9, 10].

Contributions

- We built an environment where agents can interact with the Isabelle theorem prover in an incremental manner. This enables learning-based agents to conjecture in the Isar language.
- We mined the Archive of Formal Proofs and the standard library of Isabelle. We extracted 183K theorems and 2.16M proof steps. This is one of the largest proof corpora for interactive theorem provers.
- We trained large language models on this corpus and obtained the first results of using such models to prove theorems in this new dataset.

2 ENVIRONMENT AND DATASET

We created an environment where theorem proving is modelled as a sequential decision process. Initially, the environment will load a selected theorem and we have access to the top level state. At each time-step, the agent produces a proof step of arbitrary length. The environment then applies the proof step to the top level state and iterates the process if the theorem has not been proved. We show the proof process of a simple theorem in Figure 1. The theorem declaration initialises the first proof state. The proof states in the middle row represent the stage of the proof progress and the proof steps in the bottom row are what the agent should produce. We support three different kinds of inputs: with proof states only, with previous steps only, and with both proof states and previous steps. For example, the previous steps when the agent should produce

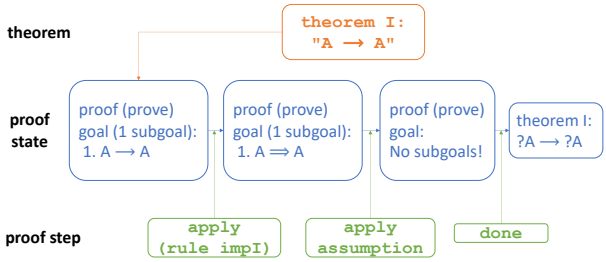


Figure 1: An illustration of the relationship between theorems, proof states, and proof steps.

"done" consist of "apply (rule impI)" and "apply assumption". Because Isabelle provides a Partially Observable Markov Decision Process (POMDP) with the proof states being the observation, conditioning on the previous steps of the proof helps the agent to reconstruct the state of the proof.

The unique feature that Isabelle enables in our system is that we can execute proofs token by token. The benefits brought by this feature include that we can make copies of a certain proof state and try multiple different methods very conveniently. This also allows us to change the order in which a proof is written, which makes proof sketching possible: we can potentially first sketch a proof skeleton containing the keyword “sorry”, which assumes that the given statement before it can be proven. Then, by saving all the states before the “sorry” command and attempting them after the skeleton has been completed, we allow a machine to write proofs in the same order a human sometimes would.

With this environment, we mined a total of 183K theorems from the Isabelle standard library [11] and the Archive of Formal Proofs (AFP) [1]. We then extracted a total of 2.16 million pairs of inputs and proof steps. This forms a dataset useful for theorem proving: if an agent can produce the correct proof step when prompted with an arbitrary proof state, it will be able to prove the theorem. We used a 95%/1%/4% random split to divide the proof corpus into the train/valid/test sets. We show some dataset statistics in Table 1.

3 EXPERIMENTS

3.1 Setup

We started by taking a language model pre-trained on the WebMath dataset for 72B tokens, similar to the GPT-f models applied to Metamath [12] and Lean [5]. We then fine-tuned the language

	Source length				Target length			
	min	max	mean	median	min	max	mean	median
With proof states only	7	227831	379.6	187.0	2	6522	34.2	19.0
With previous steps only	17	138581	3223.6	980.0				
With both proof states and previous steps	60	229885	3612.2	1328.2				

Table 1: Sequence length in characters

models only on the AFP part of the dataset, due to time constraints. The architecture we chose was a decoder-only transformer similar to GPT-3 [4]. All models have 163M non-embedding parameters. We use the same BPE encoding as GPT-3 [4]. For fine-tuning, we used a batch size of 2048, a learning rate of 0.005, a 100-step ramp-up, and decayed the learning rate according to a cosine schedule over 64B tokens; we early-stopped according to validation perplexity after 35B elapsed tokens.

3.2 Evaluation

We used a best-first search strategy at evaluation, similar to that of [5, 12]. We initialise and maintain a priority queue of top level states, sorted by their cumulative log probability. The cumulative log probability of a top level state is the sum of log probabilities of all the previous proof steps the agent takes to arrive at the current state. Initially, the priority queue contains only the top level state right after the declaration of the theorem, with a cumulative log probability of 0. At each search step, we pop the head of the priority queue to retrieve the top level state with the highest probability. We then query the language model for a set of 16 proof step candidates, with a temperature of 1.0. For each of the candidates, we duplicate the top level state, apply the candidate to it, and calculate the updated cumulative log probability. If the application of the candidate is successful, we add the resulted top level state to the queue. The queue has a length of 16 (i.e. it only maintains 16 entries with the highest cumulative log probabilities). If one of the resulted top level state shows that the proof is complete, we consider the proof attempt successful. If the queue is empty, or a timeout of 120s is spent on one attempt, or the number of queries exceeds 100, we consider the attempt a failure.

3.3 Results

We evaluated our language model with the best-first search strategy on a test set of 4000 theorems. 33.2% of the theorems were successfully proved. We analysed the failure causes of the rest of the theorems. 59.1% of the attempts failed because of the time limitation, 0.2% of the attempts failed because of the query number limitation and 7.6% of the attempts failed because the priority queue was empty at some point in the proving process. We show two successful proofs generated by our language model, and contrast them with the proofs in the AFP.

Theorem 1 is a lemma in *Utility.thy* from the AFP entry *Executable Matrix Operations on Matrices of Arbitrary Dimensions* [13]. Our proof is a one-liner and much simpler than the original proof. We checked the validity of some generated proofs manually by writing them in Isabelle with the same dependency as the original proofs.

Theorem 1 lemma foldr_foldr_concat:
"foldr (foldr f) m a = foldr f (concat m) a"
Original proof
```
proof (induct m arbitrary: a)
case Nil show ?case by simp
next
case (Cons v m a)
show ?case
unfolding concat.simps foldr_Cons o_def Cons
unfolding foldr_append by simp
qed
```
Our proof
```
by (induct m arbitrary: a) simp_all
```

Theorem 2 is a lemma in *Word_Lemmas.thy* from the AFP entry *Finite Machine Word Library* [2]. Although our proof is longer than the original, it utilises a different set of lemmas to finish the proof, and is written in a very different style compared to the original. This demonstrates that our proof search agent with language models is capable of discovering novel and interesting proofs.

Theorem 2 lemma scast_ucast_1:
"[[is_down (ucast :: 'a word ⇒ 'b word);
is_down (ucast :: 'b word ⇒ 'c word)]] ⟹
(scast (ucast (a :: 'a::len word) :: 'b::len word) :: 'c::len word) =
ucast a"
Original proof
```
by (metis down_cast_same ucast_eq ucast_down_wi)
```
Our proof
```
using unat_ucast
apply -
apply (simp add:ucast_def unat_ucast)+
apply (subst down_cast_same[symmetric])
apply (simp add: is_down)+
apply (rule word_eql)
apply (simp add: nth_ucast)
apply safe
apply simp
done
```

As a baseline, we also considered using greedy search. This is equivalent to best-first search with the queue length = 1. This agent, as a consequence, only proved 28.3% of the theorems.

4 CONCLUSIONS AND FUTURE WORK

We extracted a large corpus from Isabelle proofs and examined the performance of language models in proving theorems on the dataset. We showed that a non-trivial proportion of problems on AFP can be solved by the application of a language model and a

best-first search. The successful proofs demonstrated the language model's ability to compose succinct, or novel proofs.

The proof assistant Isabelle provides a very convenient command that allows users to conjecture ("have"). With our environment that interacts with the proof assistant in a very flexible manner, and our rich dataset, we can set out to further study how machines can learn to conjecture, and to reason about the proof construction more generally. Specifically, by learning from human conjectures, computer-assisted theorem provers are endowed with the ability to sketch proofs. This can be organically integrated with symbolic methods such as "nitpick" and "sledgehammer".

REFERENCES

[1] AFP 2021. Archive of Formal Proofs. Retrieved Feb 11, 2021 from https://www.isa-afp.org/index.html

[2] Joel Beeren, Matthew Fernandez, Xin Gao, Gerwin Klein, Rafal Kolanski, Japheth Lim, Corey Lewis, Daniel Matichuk, and Thomas Sewell. 2016. Finite Machine Word Library. *Archive of Formal Proofs* (June 2016). https://isa-afp.org/entries/Word_Lib.html, Formal proof development.

[3] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. 2014. Machine Learning for First-Order Theorem Proving - Learning to Select a Good Heuristic. *J. Autom. Reason.* 53, 2 (2014), 141–172. https://doi.org/10.1007/s10817-014-9301-5

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[5] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. 2021. Proof Artifact Co-training for Theorem Proving with Language Models. *CoRR* abs/2102.06203 (2021). arXiv:2102.06203 https://arxiv.org/abs/2102.06203

[6] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. 2016. DeepMath - Deep Sequence Models for Premise Selection. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett (Eds.). 2235–2243. https://proceedings.neurips.cc/paper/2016/hash/f197002b9a0853eca5e046d9ca4663d5-Abstract.html

[7] Dennis Lee, Christian Szegedy, Markus N. Rabe, Sarah M. Loos, and Kshitij Bansal. 2020. Mathematical Reasoning in Latent Space. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=Ske31kBtPr

[8] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence Paulson. 2021. IsarStep: a Benchmark for High-level Mathematical Reasoning. (2021).

[9] Yutaka Nagashima. 2020. Simple Dataset for Proof Method Recommendation in Isabelle/HOL. In *International Conference on Intelligent Computer Mathematics*.

[10] Yutaka Nagashima and Yilun He. 2018. PaMpeR: Proof Method Recommendation System for Isabelle/HOL. *CoRR* (2018). http://arxiv.org/abs/1806.07239

[11] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Vol. 2283. Springer Science & Business Media.

[12] Stanislas Polu and Ilya Sutskever. 2020. Generative Language Modeling for Automated Theorem Proving. *CoRR* abs/2009.03393 (2020). arXiv:2009.03393 https://arxiv.org/abs/2009.03393

[13] Christian Sternagel and René Thiemann. 2010. Executable Matrix Operations on Matrices of Arbitrary Dimensions. *Archive of Formal Proofs* (June 2010). https://isa-afp.org/entries/Matrix.html, Formal proof development.

[14] Josef Urban, Jirí Vyskocil, and Petr Stepánek. 2011. MaLeCoP Machine Learning Connection Prover. In *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings (Lecture Notes in Computer Science)*, Kai Brünnler and George Metcalfe (Eds.), Vol. 6793. Springer, 263–277. https://doi.org/10.1007/978-3-642-22119-4_21

Ordering Subgoals in a Backward Chaining Prover

Gergő Csaba Kertész[1], Gergely Papp[2], Péter Szeredi[1], Dániel Varga[2], and Zsolt Zombori[2,3]

[1] Budapest University of Technology and Economics, Budapest
[2] Alfréd Rényi Institute of Mathematics, Budapest
[3] Eötvös Loránd University, Budapest

Introduction Many automated theorem provers are based on *backward chaining*: reasoning starts from a goal statement which we aim to prove and each inference step reduces one of the goals to a (possibly empty) set of new subgoals. We thus maintain a set of open goals that need to be proven and the proof is complete once the open goal set becomes empty. For each goal, there can be several valid inferences, resulting in different successor goal sets and selecting the right inference constitutes the core problem of such theorem provers which has been thoroughly studied in the past half century.

There is, however, another decision to make during proof search, which has been largely underappreciated in the theorem proving community: this is the order in which we select goals from the open goal set. When goals do not share variables, their ordering is irrelevant as their proofs do not influence each other. However, variables establish connections between goals and a certain proof of one goal can result in variable instantiations that make it impossible to prove another goal.

The importance of subgoal ordering is recognised in the SETHEO [7] system. The authors give heuristic justifications for starting with goals that are less likely to succeed and use various manually crafted features to approximate this ordering. [2] use this heuristic ordering and show that when one proof attempt of one first goal fails, it is sometimes better to proceed with another goal before trying alternative attempts of the same goal.

In this paper, we look at the leanCoP [8] connection tableau calculus and explore how goal ordering influences theorem proving performance. leanCoP is a first order theorem prover which translates the problem into clausal normal form and builds a proof tree whose nodes are goals, using two kinds of inference steps: *extension* steps add children to a leaf node while *reduction* steps close leaf nodes. In leanCoP both steps are applied on the leftmost open node by default.

Goal Ordering Database When an extension step adds children to a leaf node, their order is determined and fixed by the order of literals in the input clauses. We start by building a database that reflects the effect of permuting the newly added goals. We introduce two modifictions to leanCoP: 1) we keep track of the number of inference steps for each goal and 2) after each extension, we attempt to finish the proof using **all** possible permutations of the new goals. Suppose our input clause set is C, the current goal is G, we select clause $(\neg G, H)$ where $H = H_1, H_2, \ldots, H_n$ to extend G and permutation $\sigma(H)$ yields a complete proof of G in I steps. Then we add the following tuple to our database:

$$< C, G, H, \sigma, \sigma(H), I >$$

When all permutations of a given (G, H) pair yield the same inference count, then the corresponding tuples are omitted. To avoid infinite branches, we impose a time limit T on the proof of each subgoal after depth D.

Heuristic Goal Ordering Our database aims to provide experimental evidence for the sensibility of goal ordering. We demonstrate its benefit by constructing a heuristic goal ordering strategy upon manual inspection of the data.

We run data extraction on a small set of 131 problems (referred to as the *training set*) from the Mizar40 [4] dataset, extracted from the Mizar Mathematical Library [1]. We employ a time limit of 10 sec for each subgoal. Upon observing the output, we indentify the following simple heuristics:

1. Negative literals should be tried before positive ones.
2. Equality predicates should be proven before other predicates.
3. Equality predicates with variables on both sides should be proven after any other predicates.

We evaluate this heuristic ordering using original leanCoP (without data extraction) and compare it with three baselines: **original**, **random** ordering and **reverse** ordering. Besides the small training set that we used for constructing the ordering, we evaluate on the larger *M2k* [3] benchmark introduced in [5], which consists of 2003 problems from Mizar40.

Table 1: Performance on the training and M2k datasets using various goal orderings: **heuristic**, **original**, **random**, **reverse**. We enforce a 1 sec time limit per problem. *Succ* is the number of problems proven and *Inf* is the average number of inferences in successful proofs.

	heuristic		**original**		**reverse**		**random**	
	Succ	**Inf**	**Succ**	**Inf**	**Succ**	**Inf**	**Succ**	**Inf**
training	**53**	4900	46	2504	47	1173	46	2553
M2k	**824**	2938	712	3103	783	3041	707	2388

Table 1 shows that our simple ordering brings a significant improvement of 16%, in the number of problems proven, on both datasets. The performance of original leanCoP is at par with random ordering, as expected. However, it is rather surprising that simply reversing the goals improves performance. These results demonstrate that ordering does make a big impact on the performance of the system.

The number of inferences required by the various orderings is harder to compare as they are averaged on different sets of problems. However, we can compare two orderings on the set of problems that are solved by both. Heuristic ordering requires less inferences than original, reverse ordering and random ordering in 60%, 65%, 52% (training set) and 60%, 69%, 65% (M2k) of these problems, respectively.

Learning Guided Goal Ordering Our simple heuristic ordering is the result of a superficial human glance over the extracted data. It is certainly not optimal and certainly not universal for all datasets. It was merely meant to demonstrate that there are useful patterns to be extracted from goal ordering statistics. The next logical step is to use machine learning and use a trained model for goal ordering. We experiment with various machine learning frameworks available

in Python and expose the trained models to the Prolog implementation of leanCoP via the *pyswip* [9] package. We have implemented the full pipeline of data generation, model training and evaluation and very recently started running learning assisted experiments.

Predicting a permutation of input elements is not a typical machine learning task and we considered three approaches of modeling it:

1. Build a model that gets a sequence of goals and returns a score. This is what our database provides directly, however, model evaluation involves running the model on all permutations of the input and selecting the one that maximizes the output.
2. Build a model that takes a single goal and returns a score. The goals are then evaluated separately and ordered based on the output. This approach is faster to evaluate then the previous one, however, it is not guaranteed that there exists a proper goal scoring function that is consistent with the scores assigned to the permutations in the dataset.
3. Build a sequence to sequence model that returns the optimal permutation of the input sequence. This is the easiest to evaluate, however, it requires the most sophisticated functionality from the model.

So far, we have experimented with the first approach, i.e., training a model that turns the input sequence into a single score. For embedding of goals, we used the features introduced in [6]. These features do not take the current goal into account, only the structure of the clause that the goal is resolved with, hence, we can precompute the clause orderings before proof search, resulting in a constant computational overhead.

Our experimental results are preliminary, but in the current state, we find that it is rather easy to obtain great (above 90%) accuracy on the training data with shallow (2-3 layer) neural networks. However, it is much harder to see any improvement in terms of problems solved. We are currently working to find the best training architecture and parameters.

Conclusion and Future Work Our work explores the effect of changing the order in which goals are proven in a backward chaining theorem prover. We modify the leanCoP connection tableau calculus to extract statistics about different orderings and then use a manually designed heuristic ordering to demonstrate the potential of the extracted data in designing ordering guidance. We believe that even stronger guidance is achievable using machine learning, which is the focus of our ongoing work.

Acknowledgments This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008 and by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

References

[1] Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

[2] Ortrun Ibens and Reinhold Letz. Subgoal alternation in model elimination. In Didier Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 201–215, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[3] Cezary Kaliszyk and Josef Urban. M2K dataset.

[4] Cezary Kaliszyk and Josef Urban. Mizar40 dataset.

[5] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, pages 8836–8847, 2018.

[6] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3084–3090. AAAI Press, 2015.

[7] Reinhold Letz, Johann Schumann, Stefan Bayerl, and Wolfgang Bibel. SETHEO: A high-performance theorem prover. *J. Autom. Reason.*, 8(2):183–212, 1992.

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

[9] Yüce Tekol and contributors. PySwip v0.2.10, 2020.

Designing a Theorem Prover for Reinforcement Learning and Neural Guidance

Jonathan Laurent[1] and André Platzer[1]

Carnegie Mellon University, Pittsburgh, United States
{jonathan.laurent,aplatzer}@cs.cmu.edu

Abstract

We discuss the design of Looprl, an experimental interactive theorem prover for loop invariant synthesis that has been optimized from the ground-up for a clean integration of theorem proving with reinforcement learning and neural guidance.

Context and motivation Augmenting tactic-based interactive theorem provers with neural guidance has been the focus of increased attention in recent years [7, 21, 2, 10, 6]. The dominant approach consists in using imitation learning on large corpora of formalized mathematics. However, despite recent efforts involving self-supervised pre-training [6] or data-augmentation [19], this approach is still limited by the amount of available human-produced training data.

Thus, a promising direction is to use reinforcement learning to train learning agents through sheer interactions with the theorem prover and without resorting to human proofs [8, 3]. Curriculum learning can be used to generate tasks of suitable difficulty for the learner [22]. Unfortunately, RL-based approaches tend to be computationally intensive and sample-inefficient, which raises scalability challenges.

We introduce Looprl, an interactive theorem prover for loop invariant synthesis for programs that is designed from the ground-up for effective neural guidance and which exposes an action space that allows efficient exploration by a reinforcement learning agent.

Loop invariant synthesis Given an imperative program with a loop and a final assertion that is to be proved, a loop invariant is a predicate that i) holds before the loop executes, ii) is preserved by the loop body and iii) implies the subsequent assertion or postcondition when assuming the negation of the loop guard. For example, in the program of Figure 1, a possible invariant that would enable us to prove the final assertion (Line 8) is $y \geq 0 \land x \geq 1 \land x \geq y$.

Loop invariant synthesis is an interesting benchmark for us to consider because it raises many of the same challenges as general theorem-proving (e.g. formal reasoning, need for conjecturing...) while being contained enough to allow for meaningful experiments on limited computing resources and interpretable failure modes. Moreover, it is a largely open problem of great relevance to the verification community, with many natural extensions (e.g. program repair, program synthesis...). Related work exists that uses deep reinforcement learning for invariant synthesis [15, 16]. In the aforementioned work, the agent is trained from scratch on every new problem, which typically takes hours. In comparison, our aim is to train an agent that generalizes across tasks and can therefore solve new problems quickly once it is trained.

The Looprl prover Here are some key features of Looprl:

- **Inducing search spaces with *strategies*:** At the outermost level, the neural network does not interact with a set of deterministic tactics. Rather, building on an idea from Selsam [14, 13], the user can define nondeterministic *strategies* that induce search spaces to be explored by the neural network (using a `Search` monad). This provides a flexible way

to leverage domain knowledge into defining reduced action spaces that are more amenable to the kind of semi-random exploration that is typical in reinforcement learning.

- **Tight integration of proof and synthesis via abductive reasoning:** For example, a typical *strategy* (in the sense defined above) for discovering loop invariants [4, 5] is to start considering the postcondition as an invariant and then try and prove it inductive. If the attempt fails, one can look for a missing assumption that would make it hold and suggest it as a new invariant candidate. (In reality, the default strategy of Looprl is more general and also integrates refinements of function symbols along with some form of forward reasoning.) This form of *abductive* reasoning is a key aspect of how humans find proofs [20] and it has built-in support in Looprl.
- **Tag-based proof guidance:** At its core, Looprl provides an `abduction` tactic that takes a formula as an input and returns either `valid` (if a proof is found) or otherwise a weighted list of suggestions for missing assumptions (or symbol refinements). This tactic is implemented using a rule-based rewriting system. Rewriting is guided by a cost function that is implicitly defined by tags on parts of the input formula. These tags are probabilistic and indicate how favorable it is to use a subformula in the proof, how they should be used (e.g. as a contradictory assumption, for eliminating variable x...) and with what level of certainty that prediction is made. Several factors make this architecture especially well-suited for neural guidance: i) the costly operation of evaluating the neural network only has to be performed once before search starts, ii) tagging the input formula can be done efficiently in a single pass using a Transformer encoder [18] or a Graph Neural Network and iii) the `abduction` tactic naturally leverages the uncertainty estimates given by the neural network.

Note that all these ideas are general and thus potentially applicable beyond the problem of invariant synthesis, which we are only considering here as an initial benchmark.

Learned agent Our agent is reminiscent of the AlphaZero algorithm [17], where a neural network (here, a choice of a Transformer [18, 12] or of a Graph Neural Network [11]) is used as a heuristic for Monte-Carlo Tree Search, which is iteratively improved as more experience becomes available. Training tasks are generated using a simple curriculum-learning scheme where random programs of increasing complexity are sampled. For each program, a random assertion is sampled too for which no easy counterexample can be found and whose validity the current network is uncertain about.

Project status and plans

- **The Looprl theorem prover:** We finished implementing the Looprl theorem prover and wrote a simple user interface (Figure 1) for testing purposes. Using this interface and providing manual guidance, we solved a large sample of problems from the invariant-synthesis track of the SyGUS 2017 competition [1].
- **AlphaZero.jl:** In the context of this project, we have released a novel open-source implementation [9] of Deepmind's AlphaZero algorithm [17] written in the Julia language. This implementation is consistently one to two orders of magnitude faster than competing Python implementations, while being equally simple and flexible.
- **By AITP 2021:** We plan to implement the AlphaZero-like agent mentioned earlier and evaluate it on the SyGUS 2017 benchmark also used in [15, 16].

References

[1] Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. Sygus-comp 2017: Results and analysis. *arXiv preprint arXiv:1711.11438*, 2017.

[2] Kshitij Bansal, Sarah Loos, Markus Rabe, Christian Szegedy, and Stewart Wilcox. Holist: An environment for machine learning of higher order logic theorem proving. In *International Conference on Machine Learning*, pages 454–463. PMLR, 2019.

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

[4] Isil Dillig, Thomas Dillig, Boyang Li, and Ken McMillan. Inductive invariant generation via abductive inference. *Acm Sigplan Notices*, 48(10):443–456, 2013.

[5] Mnacho Echenim, Nicolas Peltier, and Yanis Sellami. Ilinva: Using abduction to generate loop invariants. In *International Symposium on Frontiers of Combining Systems*, pages 77–93. Springer, 2019.

[6] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *arXiv preprint arXiv:2102.06203*, 2021.

[7] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. Gamepad: A learning environment for theorem proving. *arXiv preprint arXiv:1806.00608*, 2018.

[8] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. *arXiv preprint arXiv:1805.07563*, 2018.

[9] Jonathan Laurent. Alphazero.jl: A generic, simple and fast AlphaZero implementation. https://github.com/jonathan-laurent/AlphaZero.jl, 2021.

[10] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence Paulson. Isarstep: a benchmark for high-level mathematical reasoning. 2021.

[11] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2967–2974, 2020.

[12] Markus N Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. *arXiv preprint arXiv:2006.04757*, 2020.

[13] Daniel Selsam. The IMO grand challenge. http://aitp-conference.org/2020/slides/DS.pdf, 2020. Talk at AITP 2020.

[14] Daniel Selsam. The IMO grand challenge: A battle of ideas. https://dselsam.github.io/IMO-GC-battle-of-ideas/, 2020.

[15] Xujie Si, Hanjun Dai, Mukund Raghothaman, Mayur Naik, and Le Song. Learning loop invariants for program verification. In *Neural Information Processing Systems*, 2018.

[16] Xujie Si, Aaditya Naik, Hanjun Dai, Mayur Naik, and Le Song. Code2inv: a deep learning framework for program verification. In *International Conference on Computer Aided Verification*, pages 151–164. Springer, 2020.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[19] Mingzhe Wang and Jia Deng. Learning to prove theorems by learning to generate theorems. *arXiv preprint arXiv:2002.07019*, 2020.

[20] Yuhuai Wu, Markus Rabe, Wenda Li, Jimmy Ba, Roger Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. *arXiv preprint arXiv:2101.06223*,

2021.

[21] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In *International Conference on Machine Learning*, pages 6984–6994. PMLR, 2019.

[22] Zsolt Zombori, Adrián Csiszárik, Henryk Michalewski, Cezary Kaliszyk, and Josef Urban. Towards finding longer proofs. *arXiv preprint arXiv:1905.13100*, 2019.

A Looprl Screenshot

Figure 1: A screenshot of the Looprl user interface. Here, the user is manually adding a tag that indicates that the loop guard is likely irrelevant in proving the inductivity of the current invariant candidate.

(Auto)Complete this Proof: Decentralized Proof Generation via Smart Contracts

Jin Xing Lim[1], Barnabé Monnot[2], Georgios Piliouras[1], and Shaowei Lin[3]

[1] Singapore University of Design and Technology, {jinxing_lim@mymail., georgios@}sutd.edu.sg
[2] Ethereum Foundation, barnabe.monnot@ethereum.org
[3] shaowei@gmail.com

Automated theorem proving has come a long way in the last few decades. Multiple *automated theorem provers* (ATPs) have been developed, such as E [15] and Vampire [13] in the case of first-order logic (FOL). For higher-order logic, automated tools exist for *interactive theorem provers* (ITPs), such as Sledgehammer [11] (for Isabelle [18]), TacticToe [9] (for HOL4 [16]), CoqHammer [7] and Tactician [4] (both for Coq [3]). The effectiveness, however, of these automated tools is still not as desirable as one would hope. Paulson and Blanchette show in their experiments that Sledgehammer only managed to automate 52% of the proofs from seven selected Isabelle theories [12]. On the brighter side, combinations of different automated tools yield better results than the tools individually. In the experiments run by the developers of Proverbot9001 [14], the combination of CoqHammer and their tool is 32% more effective in automated proving than the tool itself. Furthermore, they concluded that CoqGym [20] and their approaches are complementary as both can complete proofs which the other cannot. Thus, this raises our driving challenge: *to design a system architecture that allows for the effective collaboration between multiple intelligent agents (humans and AI systems) towards producing formally correct proofs and programs.*

Our approach. We propose a blockchain-based protocol for proof and program synthesis through some ITP such as Coq. This approach allows both for the design of open problems via intents which have not been formally completed as well as for the storage of partial proofs. Using token-based mechanisms, we incentivize continued participation and completion of partial proofs. Furthermore, we encourage the development and application of generic proof tactics which are useful in completing numerous diverse theorems/programs.

Our architecture instantiates a top-down approach to proving theorems: provers publish partial proofs, formally correct up to specific unproven lemmas. To coordinate and distribute claims and partial proofs, the architecture features a programmable blockchain. Claims of first authorship are verified with time-stamped records on the chain. Smart contracts define complex incentive schemes to reward multiple collaborators, e.g., depending on how peers evaluate their respective contributions. Blockchain data is highly structured and available, ideal for AI agents to iterate over and attempt to solve unproven claims, possibly in collaboration with one another. Other than storing multiple copies of mathematical proofs and formally verified programs at different nodes, the decentralization critically allows for the emergence and persistence of *common knowledge* [8], which mirrors the process of academic publishing and public presentation of results.

Sketch of the system architecture

We assume the availability of a public, programmable blockchain, where smart contracts are deployed. We call *prover* an account (or address) on the blockchain participating in the protocol. A human or AI participant may operate several prover accounts, although for simplicity we consider the case where one entity corresponds to one prover only.

Data Layer

Provers add time-stamped *records* of *contributions* on the blockchain. The record is a simple data structure registered in the blockchain's state. Minimally, the record contains (a) the prover's address and (b) a reference to the contribution. In our protocol, the contribution is a Coq file containing additional metadata. Metadata attributes include the contribution type, among three distinct possibilities: (1) a conjecture (proposition/type with empty proof), (2) partial proof to some earlier conjecture (leaving sub-conjectures for other agents to complete), or (3) completed mathematical objects such as new definitions, propositions and tactics. The metadata also includes references to previous relevant contributions (e.g., "imports"). While records are stored on the blockchain, contributions are stored on a decentralized file storage infrastructure, such as IPFS [2], with the hash of the contents of the file used as the contribution's reference in its corresponding record.

Client Layer

Provers access records and contributions via the client layer. A client is an interface responsible for downloading records and contributions in a structured manner, e.g., by following the directed acyclic graph (DAG) of contributions built by imports declared in the metadata of a single contribution. While downloading contributions, clients perform validity checks that cannot be handled on the data layer, e.g., check that the Coq code is syntactically correct. Invalid contributions may either be flagged by the client or disregarded entirely. A client interface may be built, e.g., as a plug-in to popular Coq code editors, allowing provers to add structured metadata as well as publishing records and contributions to the blockchain. Additionally, the interface would present data derived from the incentive layer, introduced in the next section.

Incentive Layer

Value representation is handled natively by public, programmable blockchains, allowing the protocol to maintain for each prover a *balance* denominated in *tokens*. Our protocol uses incentives in a flexible manner: any prover can deploy some incentive mechanism as a smart contract, as long as the mechanism makes reference to contribution records. For instance, an organization can deploy a smart contract containing a reward for the proof of a theorem. The smart contract allows one prover only, e.g., a registered prover ("judge") belonging to the organization, to declare a previously published record as the winner of the prize, after which the balance of the winner is incremented with the prize tokens. The decision to offer the prize may be up to a vote by a quorum of provers in the organization, or provers who were certified by another set of provers as having made significant contributions in the past (such mechanisms "seed" an initial set who is responsible for co-opting new members, growing over time). The quorum could also split the reward via some allocation rule to incentivize the contribution of partial proofs.

To score contributions, different approaches could explore distinct trade-offs between human input and contextual scoring. For example, one approach could deem a contribution valuable whenever some index of its centrality within the contribution DAG is high (e.g., PageRank [5]). Another approach is to rank a contribution as more valuable if there are more provers declare that it is. For instance, Token-Curated Registries (TCRs) [10, 1] incentivize participants to vote and maintain a list according to some agreed upon criteria.

Comparison with related works

There are several prior projects such Qeditas [19], Mathcoin [17] and Proofgold [6] that discussed on how we could use blockchain mechanism to tie sources of formalized mathematics together in a decentralized way. To encourage participation, they discuss ideas such as rewarding creators with ownership rights and/or some form of currency. What distinguishes our idea from them is the proposal of having smart contracts to introduce incentive mechanisms. With appropriate incentive mechanism in place, we could gamify the process of formalizing mathematics and award contributors with virtual tokens for partial or completed results they have done. A particular importance is the fact that we can design complex incentive mechanisms that can incentivise multi-agents collaboration. The balance of these tokens can either be exchanged for real-life currency (e.g. US dollars) or be used to measure each mathematician's or computer scientist's contribution to formalized mathematics.

During AITP'21, we would be presenting a proof of concept to see how different formalized proofs of a sorting specification written in the Coq proof assistant can be generated from human-AI collaboration via a blockchain mechanism. We would like to gather feedback about our approach and discuss other techniques that may suit this framework.

References

[1] Aditya Asgaonkar and Bhaskar Krishnamachari. Token curated registries-a game theoretic approach. *arXiv preprint arXiv:1809.01756*, 2018.

[2] Juan Benet. Ipfs-content addressed, versioned, p2p file system. *arXiv preprint arXiv:1407.3561*, 2014.

[3] Yves Bertot. A short presentation of coq. In *International Conference on Theorem Proving in Higher Order Logics*, pages 12–16. Springer, 2008.

[4] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician. In *International Conference on Intelligent Computer Mathematics*, pages 271–277. Springer, 2020.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. 1998.

[6] Chad E. Brown. A theory supporting higher-order abstract syntax. Technical report, Tech. rep., Czech Technical University in Prague (Aug 2020), 2020.

[7] Łukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *Journal of automated reasoning*, 61(1-4):423–453, 2018.

[8] Ronald Fagin, Yoram Moses, Joseph Y Halpern, and Moshe Y Vardi. *Reasoning about knowledge*. MIT press, 2003.

[9] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Learning to reason with hol4 tactics. *arXiv preprint arXiv:1804.00595*, 2018.

[10] Mike Goldin. Token-curated registries 1.0. https://docs.google.com/document/d/1BWWC__-Kmso9b7yCI_R7ysoGFIT9D_sfjH3axQsmB6E/edit, 2017. Accessed: 2021-01-20.

[11] Jia Meng, Claire Quigley, and Lawrence C Paulson. Automation for interactive proof: First prototype. *Information and computation*, 204(10):1575–1596, 2006.

[12] Lawrence Paulson and Jasmin Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. 02 2015.

[13] Alexandre Riazanov and Andrei Voronkov. The design and implementation of vampire. *AI communications*, 15(2, 3):91–110, 2002.

[14] Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 1–10, 2020.

[15] Stephan Schulz. E–a brainiac theorem prover. *Ai Communications*, 15(2, 3):111–126, 2002.

[16] Konrad Slind and Michael Norrish. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, pages 28–32. Springer, 2008.

[17] Borching Su. Mathcoin: A blockchain proposal that helps verify mathematical theorems in public. *IACR Cryptol. ePrint Arch.*, 2018:271, 2018.

[18] Makarius Wenzel, Lawrence C Paulson, and Tobias Nipkow. The isabelle framework. In *International Conference on Theorem Proving in Higher Order Logics*, pages 33–38. Springer, 2008.

[19] B White. Qeditas: A formal library as a bitcoin spin-off (2016). *URL http://qeditas.org/docs/qeditas.pdf*, 2016.

[20] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. *arXiv preprint arXiv:1905.09381*, 2019.

Faster Smarter Proof by Induction in Isabelle/HOL with Definitional Quantifiers

Yutaka Nagashima[1,2]

[1] Department of Computer Science, University of Innsbruck, Austria
[2] Yale-NUS College, National University of Singapore, Singapore
yutaka@yale-nus.edu.sg

Abstract

Proof by induction plays a critical role in formal verification and mathematics at large. However, its automation remains as one of the long-standing challenges in Computer Science. To address this problem, we developed `sem_ind`. Given inductive problem, `sem_ind` recommends what arguments to pass to the `induct` tactic. To improve the accuracy of `sem_ind`, we introduced *definitional quantifiers*, a new kind of quantifiers that allow us to investigate not only the syntactic structures of inductive problems but also the definitions of relevant constants in a domain-agnostic style. Our evaluation shows that compared to its predecessor `sem_ind` improves the accuracy of recommendation from 20.1% to 38.2% for the most promising candidates within 5.0 seconds of timeout while decreasing the median value of execution time from 2.79 seconds to 1.06 seconds.

1 Proof by Induction in Isabelle/HOL

The automation of proof by induction is a long-standing challenge in Computer Science. To handle inductive problems, Isabelle [7] offers the `induct` tactics. When using the `induct` tactic, however, users have to manually specify its arguments by answering the following three questions:

- On which terms do they apply induction?
- Which variables do they pass to the `arbitrary` field for variable generalisations?
- Which induction rule do they pass to the `rule` field?

Unfortunately, answering these questions requires users to investigate problems at hand. To automate this process, we previously developed `smart_induct` [4] and `PSL` [6]. `PSL` is a domain-specific language, which allows users to describe proof search strategies. Based on such strategies, `PSL`'s interpreter tries to identify good arguments for the `induct` tactic by executing a possibly expensive proof search. The drawback of this approach is that `PSL` cannot make any recommendations at all if the interpreter fails to complete a proof search. `smart_induct` complements `PSL`'s limitation by suggesting promising arguments for the `induct` tactic without relying on a proof search but based on heuristics encoded in a language called `LiFtEr` [1]. Our previous evaluations, however, identifies two problems of `smart_induct`:

- `smart_induct` tends to be unreliable when variable generalisation is essential.
- `smart_induct` can be quite slow for some inductive problems.

2 Faster Smarter Induction with Definitional Quantifiers

To overcome these limitations, we developed `sem_ind`. Figure 1 presents the overall architecture of `sem_ind`: `sem_ind` firstly produces a small number of induction candidates, using the syntactic structure of problems as a hint. After filtering out candidates that do not even produce sub-goals, `sem_ind` ranks remaining candidates, using induction heuristics encoded in a domain-specific language called `SeLFiE`. Then, out of the five most promising candidates, `sem_ind` produces candidates including generalisation and ranks them using generalisation heuristics written in `SeLFiE`.

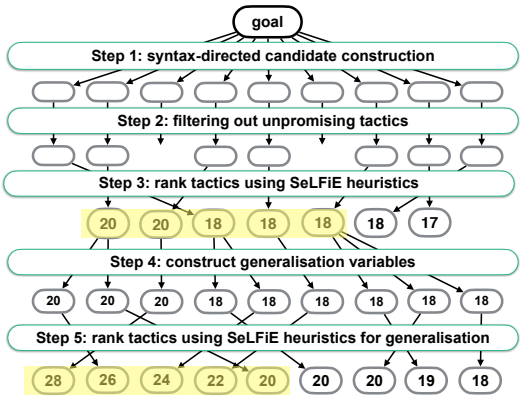


Figure 1: Overview of `sem_ind`.

Table 1 shows how often `sem_ind` produces recommendations within each timeout when applied to 1,095 inductive problems from 22 Isabelle theory files. The first row labelled as "new" shows the results of `sem_ind`, while the second row labelled as "old" shows those of `smart_induct`. This table makes it clear that `sem_ind` performs *faster* than `smart_induct`. This improvement is achieved mainly by the aforementioned architecture, which separates two problems: on what term we should apply induction, and which variables we should generalise while applying induction. This separation allows for the aggressive pruning of less promising candidates for each step, leading to a fewer number of candidates that `sem_ind` has to analyse using `SeLFiE` heuristics.

Table 2, on the other hand, shows how often the recommendations of each tool coincide with the choices of human engineers for the same problem set. For example, Table 2 shows 59.3% in the first row for "top 3". This means that for 59.3% problems the choices of human engineers appear among the three most promising candidates suggested by `sem_ind`. Thus, this table corroborates that `sem_ind` is *smarter* than `smart_induct`, producing more accurate suggestions. The main reason for this improved accuracy is its implementation language, `SeLFiE`. `SeLFiE` provides *definitional quantifiers*, $\exists_{def}$ and $\forall_{def}$, which allow us to encode heuristics that analyse relevant definitions in a domain-agnostic style. Conceptually, a definitional quantifier checks if certain properties hold for all or some of the clauses defining a given constant. For instance, $\exists_{def}(constant, heuristic, arguments)$, checks if there exists a clause defining *constant*, for which *heuristic* holds when applied to *arguments*.

3 Conclusion

We presented `sem_ind` and its implementation language `SeLFiE`. More comprehensive explanations are provided in our drafts [2, 3]. `sem_ind` is fully integrated into the Isabelle ecosystem and freely available at our GitHub repository [5].

tool	0.2s	0.5s	1.0s	5.0s
new	8.8%	24.7%	47.8%	86.8%
old	0.0%	3.5%	16.9%	70.2%

Table 1: Return Rates for Five Timeouts.

tool	top 1	top 3	top 5	top 10
new	38.2%	59.3%	64.5%	72.7%
old	20.1%	42.8%	48.5%	55.3%

Table 2: Coincidence Rates

Acknowledgments

This work has been supported by the following grants:

- the grant of Singapore NRF National Satellite of Excellence in Trustworthy Software Systems (NSoE-TSS),
- the European Regional Development Fund under the project AI & Reasoning (reg.no.CZ. 02.1.01/0.0/0.0/15_003/0000466), and
- NII under NII-Internship Program 2019-2nd call.

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. Faster smarter induction in Isabelle/HOL. *CoRR*, abs/2009.09215, 2020.

[3] Yutaka Nagashima. Selfie: Modular semantic reasoning for induction in isabelle/hol. *CoRR*, abs/2010.10296, 2020.

[4] Yutaka Nagashima. Smart induction for Isabelle/HOL (tool paper). In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*, pages 245–254. IEEE, 2020.

[5] Yutaka Nagashima et al. data61/PSL. https://github.com/data61/PSL, 2021.

[6] Yutaka Nagashima and Ramana Kumar. A proof strategy language and proof script generation for Isabelle/HOL. In *International Conference on Automated Deduction CADE 2017*, 2017.

[7] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - a proof assistant for higher-order logic*. Springer, 2002.

A Corpus of Spatial Reasoning Problems

Adam Pease, Paulo Santos, and Alexandre Rademaker

[1] Articulate Software, San Jose, CA, USA
apease@articulatesoftware.com
[2] Flinders University, Adelaide, South Australia
paulo.santos@flinders.edu.au
[3] IBM and EMAp/FGV, Rio de Janeiro, Brazil
alexrad@br.ibm.com

1 Introduction

Spatial reasoning is an important part of common-sense reasoning but has most often been examined in isolation from other areas of common sense knowledge representation. Spatial reasoning also has been a prominent area of research in linguistic semantics. In this work, we aim to create a corpus of common sense reasoning problems formalized in the context of a large and comprehensive theory of world knowledge, in the hopes that it will be more generally reusable on a wide variety of practical reasoning problems. An additional goal is to demonstrate the computational sufficiency of this work by employing automated reasoning to solve the problems. Finally, we also state the problems first in natural language, to provide a set of test cases for theories of linguistic semantics and computational linguistics, with a computational representation that can help to validate whether any method or process of interpreting language into a computational logic representation is correct or sufficient.

This is an ambitious set of goals, and it will not be possible to provide solutions for all the problems that could be formulated in natural language. In particular, we do not attempt to create an implementation, at least at this time, for a system that can translate all of the stated problems from language to logic. Many of the problems also appear to require a logic and reasoning system beyond first order logic. It is important however, to have challenges that are unsolved in order to motivate research.

We utilize the Suggested Upper Merged Ontology (SUMO)[5, 7][1], a comprehensive ontology of around 20,000 concepts and 80,000 hand-authored logical statements in a higher-order logic, that has an associated integrated development environment[9] integrated with leading theorem provers such as Eprover [10] Vampire [4] and LEO-II [1], and manually-created links[6] to the WordNet lexico-semantic database[3]. We described [9] elsewhere how to translate SUMO to the strictly first order language of TPTP [12], as well as TF0 [8] and THF[2].

2 Reasoning Problems

The current set of problems with solutions is available on GitHub[2]. Many of the problems are simple in representation and reasoning. For example

(26) *The road goes from MV to MP. Does the road go from MP to MV?* Yes.

We have the notion of a `BidirectionalTransitway` and Vampire easily solves the problem as formulated in SUO-KIF with SUMO terms in 16 steps, most of which are transformation

[1] https://www.ontologyportal.org
[2] https://github.com/ontologyportal/sumo/blob/master/tests/SpatialQs.txt

of the required axioms into conjunctive normal form. But an NLP system must interpret that 'go' means the road traverses a path, rather than moves, which in this case we have done by manually creating the SUMO-based formalization.

(4) *John is carrying a vase. There is a flower in the vase. Is John carrying a flower?* Yes

Problem 4 is solved in 26 steps and using four axioms from SUMO and the problem statement. In each case, we have common formalized terms to reuse from SUMO. This ensures that each problem has a degree of compatible and comparable semantics to anchor the semantics of terms. In some problems, the primary challenge is linguistic, as in

(37) *The painting went from the first to the second floor. Did the painting move?* Most likely.

where once it is decided that the painting actually moved as opposed to spanning two floors on an atrium wall, for example, the problem is easily formalized and solved in 15 steps by Vampire.

In other problems such as (28) below, although the statements can easily be formulated in SUO-KIF/SUMO (following the question, below), the challenge is performing higher-order reasoning with modals or temporal qualification of situations, using SUMO's `holdsDuring` construct. One can also reasonably have different pragmatic interpretations of the text depending upon whether one captures an implied legal prohibition (that one `holdsObligation` as a law-abiding driver not to use the road), or a practical one (maybe there's an actual gap in the bridge being fixed), and for what vehicles (they're tearing up the pavement but a motorcycle or bicycle could still get by if ridden by a permitted member of the road crew). That is an additional interesting area of study.

(28) *From Monday to Friday the Bay Bridge will be unusable from Yerba Buena Island to Oakland. Will you be able to use the road to go from Yerba Buena to Oakland between Monday and Friday?* No.

```
(=>
  (holdsDuring ?TIME
    (attribute ?T TransitwayClosed))
  (exists (?P)
    (holdsObligation ?P
      (holdsDuring ?TIME
        (not
          (exists (?TP)
            (and
              (instance ?TP Translocation)
              (located ?TP ?T)))))))
```

We hope that the range of different challenges for NLP, representation and reasoning, within a common ontology and tool set helps to unify and motivate some separate threads of research. For instance, the formalization of such spatial problems could aid the evidence analysis in digital forensics, and the combination of spatial reasoning capabilities could take Visual Question Answering Challenges from the current simple pattern recognition in images to the actual interpretation of scenes and the actions and actors depicted in them. Additional contributed problems (and solutions) are welcome. We expect that it should be possible to add some of these problems to the yearly CASC [11] competition as well.

We thank Dr. Annie Zaenen for contributing a number of the problems in the corpus.

References

[1] Christoph Benzmüller, Laurence Paulson, Frank Theiss, and A. Fietzke. (2008). *LEO-II - A Cooperative Automatic Theorem Prover for Higher-Order Logic. In Proceedings of the Fourth International Joint Conference on Automated Reasoning (IJCAR'08), LNAI volume*, 5195:162–170, 2008.

[2] Christoph Benzmüller and Adam Pease. Progress in automating higher-order ontology reasoning. In Boris Konev, Renate Schmidt, and Stephan Schulz, editors, *Workshop on Practical Aspects of Automated Reasoning (PAAR-2010)*. CEUR Workshop Proceedings, Edinburgh, UK, 2010.

[3] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[4] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *Proceedings of the 25th International Conference on Computer Aided Verification*, volume 8044 of *CAV 2013*, pages 1–35, New York, NY, USA, 2013. Springer-Verlag New York, Inc.

[5] Ian Niles and Adam Pease. Toward a Standard Upper Ontology. In Chris Welty and Barry Smith, editors, *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, pages 2–9, 2001.

[6] Ian Niles and Adam Pease. Linking Lexicons and Ontologies: Mapping WordNet to the Suggested Upper Merged Ontology. In *Proceedings of the IEEE International Conference on Information and Knowledge Engineering*, pages 412–416, 2003.

[7] Adam Pease. *Ontology: A Practical Guide*. Articulate Software Press, Angwin, CA, 2011.

[8] Adam Pease. Arithmetic and inference in a large theory. In *AI in Theorem Proving*, 2019.

[9] Adam Pease and Stephan Schulz. Knowledge Engineering for Large Ontologies with Sigma KEE 3.0. In *The International Joint Conference on Automated Reasoning*, 2014.

[10] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.

[11] Geoff Sutcliffe and Christian Suttner. The state of casc. *AI Commun.*, 19(1):35–48, January 2006.

[12] Steven Trac, Geoff Sutcliffe, and Adam Pease. Integration of the TPTPWorld into SigmaKEE. In *Proceedings of IJCAR '08 Workshop on Practical Aspects of Automated Reasoning (PAAR-2008)*. CEUR Workshop Proceedings, 2008.

Learning Equational Theorem Proving*

Jelle Piepenbrock[1,2], Tom Heskes[1], Mikoláš Janota[2], and Josef Urban[2]

[1] Radboud University, Nijmegen, The Netherlands
[2] Czech Technical University, Prague, Czech Republic

Abstract

We develop Stratified Shortest Solution Imitation Learning (3SIL) to learn equational theorem proving in a deep reinforcement learning (RL) setting. The self-trained models achieve state-of-the-art performance in proving problems generated by one of the top open conjectures in quasigroup theory, the Abelian Inner Mapping (AIM) Conjecture. To develop the methods, we first use two simpler arithmetic rewriting tasks that share tree-structured proof states and sparse rewards with the AIM problems. On these tasks, 3SIL is shown to significantly outperform several established RL and imitation learning methods. The final system is then evaluated in a standalone and cooperative mode on the AIM problems. The standalone 3SIL-trained system proves in 60 seconds more theorems (70.2%) than the complex, hand-engineered *Waldmeister* system (65.5%). In the cooperative mode, the final system is combined with the *Prover9* system, proving in 2 seconds what standalone *Prover9* proves in 60 seconds.

Automated theorem proving has been applied in the theory surrounding the Abelian Inner Mapping Conjecture, known as the AIM Conjecture [3]. This is one of the top open conjectures in quasigroup theory. Work on the conjecture has been going on for more than a decade. Automated theorem provers use hundreds of thousands of inference steps when run on problems from this theory. In this work, we train a machine learning model to guide proof decisions.

We use a dataset of theorems generated by this conjecture as a testbed for our machine learning methods [1]. The dataset comes with a simple prover called AIMLEAP that can take machine learning advice.[1] We use this system as a reinforcement learning environment. AIMLEAP keeps the state and carries out the cursor movements and tree rewrites.

The AIM conjecture concerns specific structures in *loop theory* [3]. A loop is a quasigroup with an identity element. A quasigroup is a generalization of a group that does not preserve associativity. Currently, work in this area is done using automated theorem provers such as *Prover9* [4, 3]. The *Prover9* theorem prover is especially suited to this approach because of its well-established *hints* mechanism [7]. The dataset is derived from this *Prover9* approach and contains around 3500 theorems that can be proven with the definitions and lemmas [1].

There are 177 possible actions in the environment. Three actions are cursor movements, where the cursor can be moved to an argument of the current position. The other actions all rewrite the current term at the cursor position with various axioms, definitions and lemmas that hold in the AIM context.

To develop a method that can solve equational theorem proving problems, we considered two simpler arithmetic tasks, which also have a tree-structured input and a sparse reward structure: Robinson arithmetic and polynomial arithmetic. In both cases, the task is to normalize a mathematical expression to one specific form. The learning environments incorporate two existing datasets. For the Robinson arithmetic normalization task, we use a dataset that was

*The results were supported by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902 and are part of the RICAIP project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

[1] https://github.com/ai4reason/aimleap

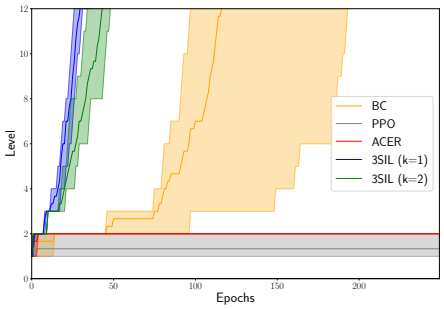


Figure 1: Comparison of methods on the Robinson arithmetic task. Y-axis denotes progress on a task curriculum, X-axis denotes the amount of training epochs.

constructed for reinforcement learning experiments in the interactive theorem prover HOL4 [2]. For the polynomial normalization task, we employ a dataset introduced for experiments in symbolic rewriting using recurrent language models [5].

Robinson arithmetic is a simple arithmetic theory. In the task, we are asking the agent to calculate the value of the expression. As an example, $S(S(0)) + S(0)$, representing $2 + 1$, needs to be rewritten to $S(S(S(0)))$. The setup for this RA normalization task is modeled after [2]. Based on the arithmetic tasks, we developed a method, called *stratified shortest solution imitation learning* (3SIL), which performed better than the baseline algorithms. In Figure 1, we show the relative performance of several methods on the Robinson arithmetic task. We compare standard reinforcement learning algorithms, such as PPO [6] and ACER [8] and behavioral cloning (BC), with our method 3SIL. On these tasks, our method outperforms the RL baselines, as shown in Figure 1. Both variants of our method advance more quickly through the curriculum than the baseline algorithms.

The method was then tested on the more difficult AIM theorem proving task. In Table 1, we show the performance of a model trained on the AIMLEAP task within the AIMLEAP environment. We compare with state-of-the-art theorem provers and observe that the model can outperform Waldmeister. In further experiments, we also observe that assisting Prover9 with the learned model can improve its performance. We let the model rewrite the starting state and add the rewritten states as lemmas to the Prover9 input. In this cooperative mode, the final system is combined with the *Prover9* system, proving in 2 seconds what standalone *Prover9* proves in 60 seconds. The results are shown in Table 2. This setup also outperforms E (shown in Table 1). In conclusion, we show that equational theorem proving in loop theory can be assisted by learned neural network models. In the future, we will explore whether we can automatically select the best previous proofs to learn from to accelerate the learning process.

Table 1: Theorem proving performance on the hold-out test set in fraction of problems solved. Means and standard deviations are the results of evaluations of 3 different models from 3 different training runs.

Method	Success Rate
E (60s)	0.802
Waldmeister (60s)	0.655
Prover9 (60s)	0.833
Model (1x)	0.586 ± 0.029
Model (60s)	0.702 ± 0.015

Table 2: Prover 9 theorem proving performance on the hold-out test set when injecting lemmas suggested by the learned model. *Prover9*'s performance increases when using the model's suggested lemmas.

Method	Success Rate
Prover9 (1s)	0.715
Prover9 (2s)	0.746
Prover9 (1s) + Model (1s)	0.841 ± 0.019

References

[1] Chad E Brown, Bartosz Piotrowski, and Josef Urban. Learning to advise an equational prover. *Artificial Intelligence and Theorem Proving*, 2020.

[2] Thibault Gauthier. Deep reinforcement learning in HOL4. *arXiv preprint arXiv:1910.11797*, 2019.

[3] Michael Kinyon, Robert Veroff, and Petr Vojtěchovský. Loops with abelian inner mapping groups: An application of automated deduction. In *Automated Reasoning and Mathematics*, pages 151–164. Springer, 2013.

[4] W. McCune. Prover9 and Mace. 2010.

[5] Bartosz Piotrowski, Josef Urban, Chad E Brown, and Cezary Kaliszyk. Can neural networks learn symbolic rewriting? *ICML Workshop on Learning and Reasoning with Graph-Structured Data*, 2019.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[7] Robert Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reason.*, 16(3):223–239, 1996.

[8] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *International Conference on Learning Representations*, 2016.

Deep Learning for Temporal Logics

Frederik Schmitt[1], Christopher Hahn[1], Jens U. Kreber[2], Markus N. Rabe[3], and Bernd Finkbeiner[1]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] Saarland University, Saarbrücken, Germany
[3] Google Research, Mountain View, CA, USA

Abstract

Temporal logics are a well established formal specification paradigm to specify the behavior of systems, and serve as inputs to industrial-strength verification tools. We report on current advances in applying deep learning to temporal logical reasoning tasks, showing that models can even solve instances where competitive classical algorithms timed out.

1 Introduction

The assumption that deep learning is not yet ready to tackle hard reasoning questions has been drawn into question. For example, Transformer models [25] perform surprisingly well on symbolic integration tasks [16], self-supervised training can lead to mathematical reasoning abilities [22], or large-enough language models learn basic arithmetic despite being trained on mostly natural language sources [21]. These success stories and uprising workshops and conferences on this topic (e.g., [5, 24, 26]), pose the question if challenging problems in the area of automated verification lend themselves to a direct learning approach as well. We report on current advances in applying deep learning to involved temporal logical reasoning tasks.

Many approaches in verification are based on temporal logics, a specification paradigm that is the basis for industrial hardware specification languages like the IEEE standard PSL [13]. For example, linear-time temporal logic (LTL) [20] can specify that some proposition P must hold at every point in time ($\Box P$) or that P must hold at some future point of time ($\Diamond P$). By combining these operators, one can specify that P must occur infinitely often ($\Box\Diamond P$). *LTL satisfiability* is the (PSPACE-complete) problem of computing a logical solution, i.e., a *trace*, which is sequence of propositions, that satisfies an LTL formula. In applications, solutions to LTL formulas can represent (counter-)examples for a specified system behavior. *LTL synthesis* is the (2EXPTIME-complete) problem of automatically constructing a circuit that satisfies an LTL specification for every input. This is an especially challenging and active research field including an annual tool competition (SYNTCOMP [14]).

Over the last decades, generations of advanced algorithms have been developed to solve temporal reasoning tasks automatically. We show that fully neural approaches are already competitive: By computing satisfying traces to LTL formulas, we show that Transformers generalize to the semantics of LTL (Section 2 and [11]) and based on those findings, we show that Transformers can be used to even synthesize fully functional circuits directly out of LTL specifications (Section 3 and [23]).

2 Transformers Generalize to the Semantics of Temporal Logics

For predicting a satisfying trace to an LTL formula, we generated training data in two different ways: randomly and as conjunctions of patterns typically encountered in practice [7]. We use `spot` [6], that implements a competitive classical algorithm, to generate solutions to formulas from these distribution and train a Transformer model to predict solutions directly. The

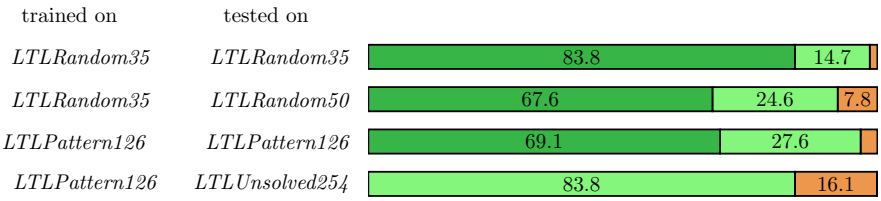


Figure 1: [11] Performance on different datasets, where the number refers to the size of the largest formula in the data set. The percentage of a dark green bar refers to the syntactic accuracy, the percentage of a light green bar to the semantic accuracy without the syntactic accuracy. Incorrect predictions are given in orange.

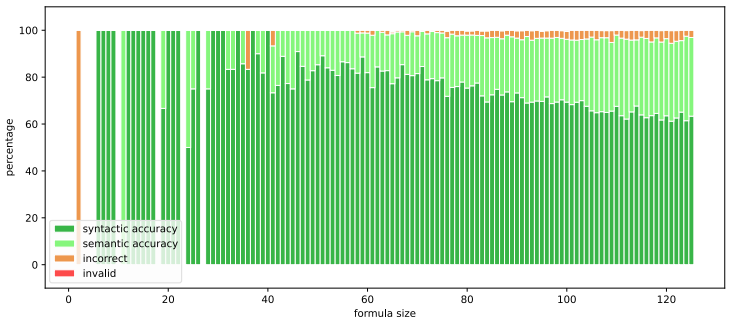


Figure 2: [11] Instances where the model agrees with the generator are displayed in dark green; deviations from the generators output but still correct predictions are displayed in light green; incorrect predictions in orange.

question, however, is whether Transformers learn to imitate the generator of the training data, rather than learn to solve the formulas according to the *semantics* of the logics. We, thus, differentiate between a syntactic accuracy, i.e., where the model predicts the same trace as the classical algorithm and the semantic accuracy, i.e., where the model deviates.

Figure 1 shows a subset of experiments conducted in [11] showing that, in fact, the latter holds true. In our experiments, we observed that Transformers predict correct solutions to 98.5% of the random formulas (line 1) and 96.8% of the pattern formulas (line 3) from a held-out test set. Figure 2 shows the performance of a model trained and tested on combinations of specification patterns *LTLPattern126* in more detail. When the formulas become larger, the gap between syntactic and semantic accuracy increases. We also observed that Transformers generalize to larger formulas than seen during training (line 2). Impressive enough, Transformers hold up pretty well and predict correct solutions in 83% of cases on a set of formulas that `spot` could not solve within 60 seconds (line 4). Finally, we performed an out-of-distribution test by predicting traces to held-out specification patterns from the literature ([8, 12, 19]). Trained on *LTLRandom126*, the model achieved an accuracy of 84.4% (62.2% syntactic accuracy) and trained on *LTLPattern126* it achieved 50.0% (11.3%).

3 Transformers Construct Circuits out of Temporal Logical Specifications

To predict a circuit that satisfies an LTL formula for every input, we utilized specification patterns mined from benchmarks of the annual reactive synthesis competition (SYNTCOMP) [14]. An example for such a pattern is a typical request-response property: $\square(\mathit{request} \rightarrow \diamond\, \mathit{grant})$.

$(assumptions)$
$(\square(\diamond \neg(r_m)))$
$\rightarrow$
$(guarantees:)$
$(\square((\neg(g_m)) \vee (\neg(g_0))))$
$(\square((r_0) \rightarrow (\diamond(g_0))))$
$(\square((r_m) \rightarrow (\bigcirc((\neg(g_0)) \mathcal{U} (g_m)))))$

```
aag 7 5 1 5 1
2
4
6
8
10
12 4
14
0
13
14
0
14 12 5
```

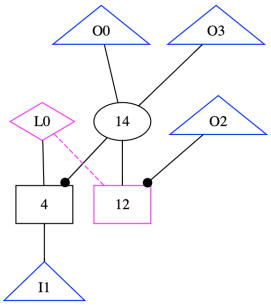


Figure 3: [23] The specification (left), the predicted AIGER circuit (middle) and the visualization of the circuit (right) for a prioritizing arbiter.

The formula states that at every point in time ($\square$) a request must be eventually ($\diamond$) followed by a grant. By combining specification patterns, we generated over 200 000 specifications including both realizable and unrealizable specifications. Using classical LTL synthesis tools [9, 18], we obtained a dataset consisting of formulas and their circuit implementations in AIGER [2]. We represented the specifications and circuits as sequences and trained hierarchical Transformers [17] on the circuit construction task. For example, when given the specification of a prioritizing arbiter that manages a shared resource, the model predicts a correct circuit given as an AIGER file, displayed in Figure 3. The specification consists of a combination of request-response and mutual exclusion patterns.

We ran several experiments [23], where the results of a subset can be found in Table 1. The `Testset` contains held-out data from the training distribution, i.e., combinations of mined patterns, where the model achieved an accuracy of 79.9%. We also tested on the challenging `SYNTCOMP` benchmarks, which contain practical specifications of systems, such as arbiters of the AMBA AHB bus [3, 4, 10] or robotic controllers [15]. For the instances that fit into the size restrictions of the model [23], the model achieved an accuracy of 66.8% (making this already a competitive approach). We stored specifications for which the synthesis tool timed out ($> 120s$) in the dataset `Timeouts`. The Transformer was able to solve 30.1% substantiating the strong generalization of this model. We also performed an out-of-distribution test by predicting circuits for a `Smart Home` benchmark [1] that has only recently been added to the SYNTCOMP competition 2021. Note that we have not mined patterns from this benchmark. We achieved an accuracy of 40.0% for the instances that fit into the size restrictions of the model. This means that, already today, direct machine learning approaches may be useful to augment classical algorithms in verification tasks.

Table 1: [23] Accuracy reported for 5 runs on `Testset`, `SYNTCOMP` benchmarks, `Timeouts`, and `Smart Home` benchmarks for different beam sizes, including the standard deviation. For the test data we show the syntactic accuracy in parenthesis.

Dataset	Beam Size 1	Beam Size 4	Beam Size 8	Beam Size 16
`Testset`	53.6(31.1) ± 2.4	70.4(39.0) ± 2.3	75.8(41.9) ± 2.1	79.9(44.5) ± 2.0
`SYNTCOMP`	51.9 ± 2.2	60.0 ± 1.5	63.6 ± 1.9	66.8 ± 1.2
`Timeouts`	11.7 ± 1.1	21.1 ± 0.9	25.9 ± 1.0	30.1 ± 1.2
`Smart Home`	22.9 ± 3.6	31.4 ± 7.1	44.8 ± 6.5	40.0 ± 6.5

References

[1] J.A.R.V.I.S. TSL/TLSF benchmark suite. https://github.com/SYNTCOMP/benchmarks/tree/master/tlsf/tsl_smart_home_jarvis, 2021.

[2] A. Biere. The AIGER and-inverter graph (AIG) format version 20071012. *FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr*, 69:4040, 2007.

[3] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1188–1193. IEEE, 2007.

[4] R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electron. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007.

[5] A. d'Avila Garcez, D. Silver, P. Hitzler, P. Földiák, K.-U. Kühnberger, L. C. Lamb, and L. de Raedt. Workshop series on neural-symbolic learning and reasoning. http://www.neural-symbolic.org.

[6] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, Oct. 2016.

[7] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In M. A. Ardis and J. M. Atlee, editors, *Proceedings of the Second Workshop on Formal Methods in Software Practice*, 1998.

[8] K. Etessami and G. J. Holzmann. Optimizing büchi automata. In *International Conference on Concurrency Theory*, pages 153–168. Springer, 2000.

[9] P. Faymonville, B. Finkbeiner, and L. Tentrup. Bosy: An experimentation framework for bounded synthesis. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, 2017.

[10] Y. Godhal, K. Chatterjee, and T. A. Henzinger. Synthesis of AMBA AHB from formal specification: a case study. *Int. J. Softw. Tools Technol. Transf.*, 2013.

[11] C. Hahn, F. Schmitt, J. U. Kreber, M. N. Rabe, and B. Finkbeiner. Teaching temporal logics to neural networks. *International Conference on Learning Representations, ICLR*, 2021.

[12] J. Holeček, T. Kratochvíla, V. Řehák, D. Šafránek, P. Šimeček, et al. Verification results in liberouter project, 2004.

[13] IEEE-Commission et al. Ieee standard for property specification language (psl). *IEEE Std 1850-2005*, 2005.

[14] S. Jacobs and G. A. Pérez. The reactive synthesis competition. www.syntcomp.org.

[15] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[16] G. Lample and F. Charton. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[17] W. Li, L. Yu, Y. Wu, and L. C. Paulson. Modelling high-level mathematical reasoning in mechanised declarative proofs. *CoRR*, abs/2006.09265, 2020.

[18] P. J. Meyer, S. Sickert, and M. Luttenberger. Strix: Explicit reactive synthesis strikes back! In *Computer Aided Verification - 30th International Conference, CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 578–586. Springer, 2018.

[19] R. Pelánek. Beem: Benchmarks for explicit model checkers. In *International SPIN Workshop on Model Checking of Software*, pages 263–267. Springer, 2007.

[20] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE

Computer Society, 1977.

[21] S. Polu and I. Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[22] M. N. Rabe, D. Lee, K. Bansal, and C. Szegedy. Mathematical reasoning via self-supervised skip-tree training. In *International Conference on Learning Representations, ICLR*, 2021.

[23] F. Schmitt, C. Hahn, M. N. Rabe, and B. Finkbeiner. Neural circuit synthesis from specification patterns. *arXiv preprint arXiv:2107.11864*, 2021.

[24] V. Thost, K. Talamadupula, V. Srikumar, C. Zhang, and J. Tenenbaum. Knowledge representation & reasoning meets machine learning. https://kr2ml.github.io/2020/, 2020.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[26] Y. Wu, K. Bansal, W. Li, M. Mitchell, D. McAllester, and J. Harrison. The role of mathematical reasoning in general artificial intelligence. https://mathai-iclr.github.io, 2021.

A Closer Look at Successful Clause Derivations Through the Lens of Recursive Neural Networks*

Martin Suda

Czech Technical University in Prague, Czech Republic

1 Motivation

Deepire [14,15] is an extension of the automatic theorem prover (ATP) Vampire [10] by machine-learned ENIGMA-style clause selection guidance [2, 6, 7, 11]. Its main distinguishing feature is the use of a recursive neural network (RvNN) to classify clauses based solely on their derivation history. This means that to decide whether a clause should be preferred in proof search, Deepire does not look at the logical content of the clause as a formula, but only at its ancestors in the derivation DAG and the inference rules that were applied to derive it.

Despite the simplicity of the approach (and its inherent inability to provide "the perfect guidance", even in principle), Deepire has substantially improved on plain Vampire's performance on (1) theory reasoning problems coming from SMT-LIB [1], see [14], and on (2) formal library export problems from the Mizar40 set [9], see [15]. On the latter benchmark, Deepire even improved on the impressive results of ENIGMA by Jakubův and Urban from 2019 [8].

Obviously, these successes required a certain amount of tuning. In particular, one needs to find the right balance between the capacity of the network and the time it takes to evaluate it. (On the tested benchmarks, Deepire worked the best with clause embedding dimension between 64 and 128, spending on average between 30 and 40 % of the prover runtime evaluating the network.) Additionally, it is also important to select a good mixture of the traditional heuristics for governing clause selection and the machine-learned advice. (Deepire pioneers the use of the layered clause selection scheme [4, 5, 16] for this, in which the traditional selection by clause's age and weight is preserved but alternately applied to only the clauses classified as positive by the network and to all the passive clauses; this alternation happens under a ratio, for which our experiments established an optimal value of 2:1.)

Ultimately, however, tuning notwithstanding, given how useful it can be for guiding the prover, a trained network represents an interesting artifact that, I believe, should be further analyzed. The aim of this work is therefore to conduct an analysis of the best neural models obtained in our previous work on Deepire [14,15] and to shed more light on the reasons behind the success of the strategies these models back up.

2 The Aims

Besides simply satisfying intellectual curiosity, our main aim with the proposed analysis is to look for *general theorem proving heuristics* that the training process implicitly discovered and that could be extracted, understood by a human and adapted for the design of better theorem proving strategies on other benchmarks. An example of the kind of a heuristic I have in mind could be the theory distance heuristic by Gleiss and Suda [5], whose computation proceeds

*Supported by the Czech Science Foundation project 20-06390Y and the project RICAIP no. 857306 under the EU-H2020 programme.

along a clause derivation in analogy to evaluation of Deepire's RvNN. However, theory distance was proposed *before* the Deepire experiments on SMT-LIB took place and it is at the moment not clear to what degree Deepire's network exploits the principle behind theory distance.

It can turn out, though, that no such heuristics are easily identifiable or even present in any form. That would mean that the knowledge extracted by the network pertains exclusively to the specific benchmark it was trained for. We might then hope to learn what were the properties of that benchmark that Deepire exploited to tackle it well. Note that simple memorization cannot fully explain the observed successes as there was always a significant jump between the number of problems solved by plain Vampire (whose solutions were used to train the first model) and the performance of Deepire using the first model, which suggests successful generalization.

3 The Techniques

In the experiment on Mizar [15], the network training procedure effectively compressed 800 MB (of disk space when zipped) worth of successful derivations into a 5 MB torch-script model file (consisting mostly of matrix and vector parameters). Five megabytes is definitely too much to be directly approached and analyzed manually. Therefore, I propose to use statistical techniques, be it ad hoc ones, tailored for the particular use case, or out-of-the-box solutions marketed under the label of explainable AI.

The "generalized age" perspective: Any clause selection heuristic that works as a function of clause's derivation history can be understood as generalizing the clause's *age*. (At least the way it is defined in Vampire, i.e., as the depth of the derivation DAG (only counting generating inferences and not reductions)). We can study to what degree is the evaluation function represented by the learned RvNN similar to the age function; for instance, asking:

- Is the evaluation function (most of the time) monotone along the derivations?[1]
- Does the network take into account the exact shapes of the trees[2] or is it (mostly) additive?

Since the intuition behind the age heuristic is that clauses with more complex derivation DAGs are less likely to contribute to a proof and since the training set for our network was, after all, finite, an interesting question also arises, namely whether the network allows for a positively classified clause of arbitrary large derivation. I will try to answer these questions in the talk.

Visualisations: A picture is worth a thousand words. Using the Graphviz library [3], I wrote a tool for visualising Vampire's derivations and labelling the nodes corresponding to clauses by the RvNN's classification judgments. Figure 1 show an example output. So far I was not able to glimpse any revealing pattern in these, but they seem to have a certain artistic value.

XAI: What I have just described, i.e., the "quest for opening a black box", seems to be a perfect case for the application of the techniques of explainable artificial intelligence. I am currently investigating whether there are methods and tools readily available to help explaining RvNNs or how to adapt the popular methods such as LIME [13] or SHAP [12] to analyze networks coming from the Deepire setting. As part of my talk, I am planning to give a review of the most relevant XIA methods and establish to what degree these methods, coming predominantly from the computer vision field, can be useful in our case.

[1] Note that the training examples all have a property that the parent of a positive clause is a positive clause.
[2] Although computed on a DAG, mathematically, the evaluation is a function of the unfolded tree.

References

[1] C. Barrett, P. Fontaine, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

[2] K. Chvalovský, J. Jakubuv, M. Suda, and J. Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, vol. 11716 of *LNCS*, pp. 197–215. Springer, 2019.

[3] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pp. 127–148. Springer-Verlag, 2003.

[4] B. Gleiss and M. Suda. Layered clause selection for saturation-based theorem proving. In *Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR), co-located with the (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, vol. 2752 of *CEUR Workshop Proceedings*, pp. 34–52. CEUR-WS.org, 2020.

[5] B. Gleiss and M. Suda. Layered clause selection for theory reasoning - (short paper). In *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part I*, vol. 12166 of *LNCS*, pp. 402–409. Springer, 2020.

[6] J. Jakubuv, K. Chvalovský, M. Olsák, B. Piotrowski, M. Suda, and J. Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, vol. 12167 of *LNCS*, pp. 448–463. Springer, 2020.

[7] J. Jakubuv and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, vol. 10383 of *LNCS*, pp. 292–302. Springer, 2017.

[8] J. Jakubuv and J. Urban. Hammering Mizar by learning clause guidance (short paper). In *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, vol. 141 of *LIPIcs*, pp. 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[9] C. Kaliszyk and J. Urban. Mizar 40 for mizar 40. *J. Autom. Reason.*, 55(3):245–256, 2015.

[10] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, vol. 8044 of *LNCS*, pp. 1–35. Springer, 2013.

[11] S. M. Loos, G. Irving, C. Szegedy, and C. Kaliszyk. Deep network guided proof search. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, vol. 46 of *EPiC Series in Computing*, pp. 85–105. EasyChair, 2017.

[12] S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4765–4774, 2017.

[13] M. T. Ribeiro, S. Singh, and C. Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144. ACM, 2016.

[14] M. Suda. Improving ENIGMA-style clause selection while learning from history. In *Proceedings of the 28th CADE*, 2021. To appear. See also https://arxiv.org/abs/2102.13564.

[15] M. Suda. Vampire with a brain is a good ITP hammer. *CoRR*, abs/2102.03529, 2021.

[16] T. Tammet. GKC: A reasoning system for large knowledge bases. In *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, vol. 11716 of *LNCS*, pp. 538–549. Springer, 2019.

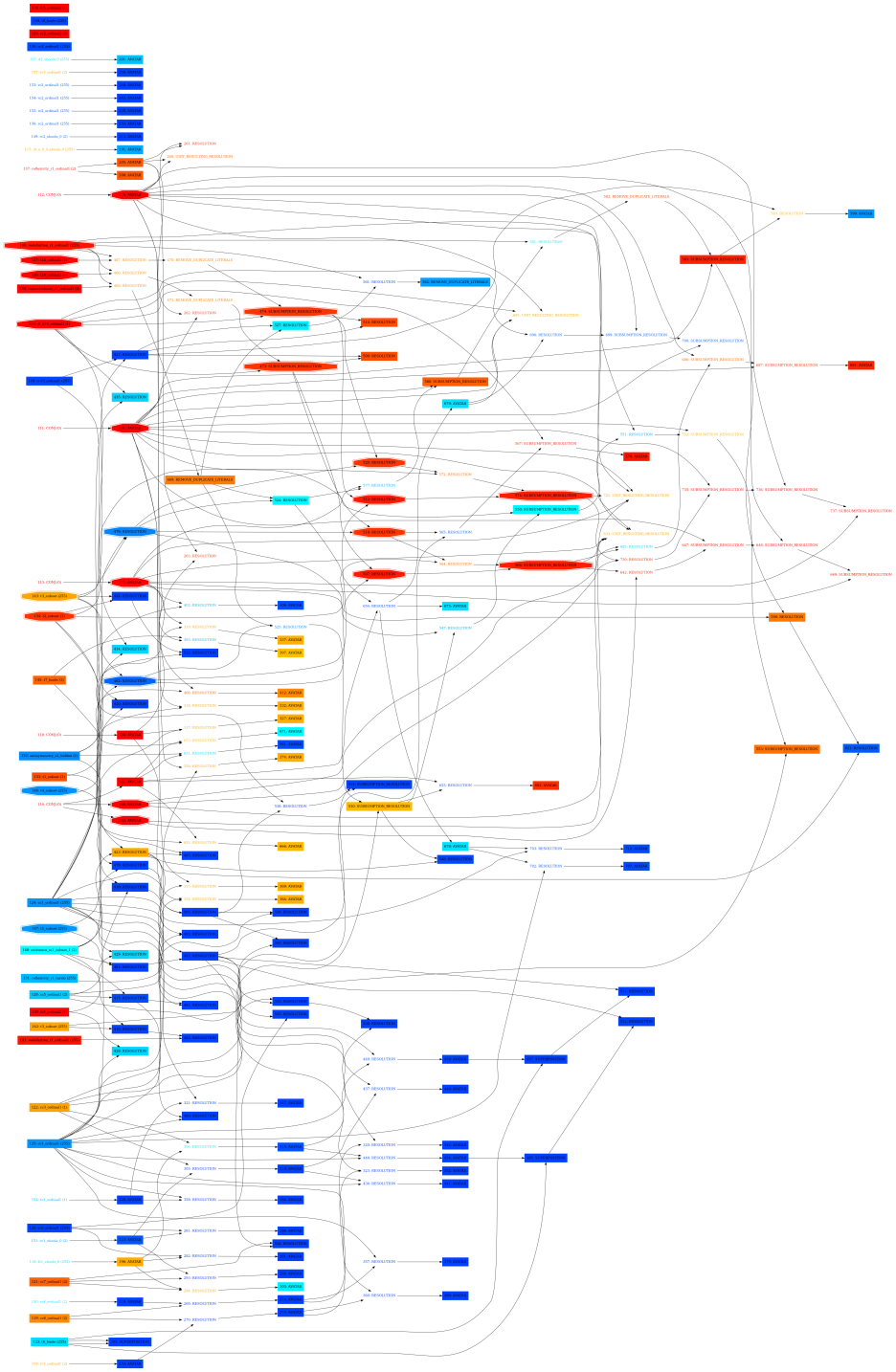


Figure 1: An example derivation from the Mizar benchmark (on the problem `t25_ordinal3`) evaluated by the network. Octagons mark clauses from the original proof, rectangles the remaining selected clauses, and labels without a box denote clauses that were never selected (the most common reason being immediate reductions). Hues of blue mark clauses classified as negative, and hues of orange and red those classified as positive. Since AVATAR was used in the proof, there is no final empty clause explicitly present (the ultimate contradiction was derived in the SAT solver). The derivation contains both false positives and false negatives.

Dreaming to Prove

Kristóf Szabó[1,2] and Zsolt Zombori[1,2]

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

Introduction *World models* represent the basic mechanisms of a system and can provide predictions about how transformations (actions) affect the state of the system. Such models have recently gained attention in Reinforcement Learning (RL) and in several domains model based learning systems performed similarly or better than highly tuned model free variants [1, 8, 12]. World models can increase sample efficiency since trajectories can be generated without interacting with the environment, and they can aid exploration by yielding a semantically meaningful latent structure that allows for identifying promising directions.

Our project raises the question whether such world models are achievable for theorem proving, i.e. whether state-of-the-art machine learning toolset is capable of capturing the underlying dynamics of an Automated Theorem Proving (ATP) system. A world model for an ATP system should know what moves are valid and when the proof search failed or succeeded.

An ATP system can often be framed as an RL environment: In each state of the prover, it has to select from a set of valid inferences (actions) that result in a new state, and a proving agent aims to select inferences that maximise its chance of finishing the proof. We select the leanCoP [6] connection tableau calculus for which we try to build a world model. leanCoP has a clean and compact notion of a *state* to which a limited number of inferences (*actions*) can be applied, making it a comfortable RL environment. Our two primary questions are: 1) what is length barrier within which the model can generate valid inference sequences and 2) to what extent the model can predict if an inference leads us closer to the end of the proof.

Most related to our work is [4], which train a latent embedding model, a state transition model in latent space, and a model that predicts the applicability of rewrite steps solely from the latent embedding. The trained models can predict rewrite step validity significantly better than simple baselines even after 10 rewrites in latent space. However, evaluation is only performed on valid rewrites, so we do not know how far the model can chain its own predictions. Furthermore, little is known about how much the state changes during the rewrites, while the inferences in leanCoP have a clear sense of directionality (there is no returning to the same state).

We turn leanCoP into an RL environment, as done in [2, 11, 10] and adapt the DreamerV2 [1] model-based RL algorithm which shows impressive results in modeling ATARI video games.

Dreamer Architecture The Dreamer system uses several neural network components to map observed states into a latent representation and then to perform actions in latent space. Given state space S and action space A, we train an *encoder* model $E : S \to \mathbb{R}^n$ which creates a latent vector. We also train a *decoder* $D : \mathbb{R}^n \to S$ which reconstructs the state from latent code. A *reward prediction model* $R : \mathbb{R}^n \to \mathbb{R}$ estimates the reward from latent code associated with the given state. Finally, a *transition* model $T : (\mathbb{R}^n, A, X) \to (\mathbb{R}^n, X)$ predicts the effect of an action on the latent code. T is implemented as a recurrent state-space model (RSSM), equipped with its own internal state $x \in X$, that is updated after each transition. All components are trained jointly, on sequences of (s_i, a_i, r_i, s_{i+1}) state, action, reward, successor tuples, using the following three objectives: 1) reconstruction of the state $D(E(s_i)) = s_i$, 2) reconstruction of the reward $R(E(s_i)) = r_i$ and reconstruction of the successor state $T(E(s_i), a_i, x_i) = E(s_{i+1})$.

Once the system is trained, Dreamer is capable of generating action and latent code sequences in its "head" without interacting with the real environment, hence the motivation for its name.

Dreamer for leanCoP: State Representation [4] only use training signal coming from reward reconstruction (predicting if a rewrite is valid), while [1] show that most of the representational power of Dreamer comes from state reconstruction. However, in the case of theorem proving, reconstructing the state is not as straightforward as for ATARI games, as the former has discrete proof objects, which requires some hand-crafted mapping before processing by neural models. We consider three methods for extracting features from leanCoP states: 1) manual features developed in [3] and successfully used in many systems, e.x. [2, 11], referred to as *ENIGMA features* and 2) using the graph neural network developed in [5] specifically for logic formulae, referred to as *GNN* and 3) *transformer* language models that process text directly.

ENIGMA features rely on consistent naming of concepts, while the GNN exploits structural similarities of terms, so they are somewhat complimentary. Transformers are recently gaining attention in theorem proving (e.g. [9, 7]). Hence, both the input of the encoder E and the output of decoder D can be either ENIGMA features, a graph or plain text. We intend to experiment with all combinations to see what results in the best latent space structure.

Dreamer for leanCoP: Action Selection Dreamer assumes either a fixed size discrete action space or a real n-dimensional space, neither of which matches the action space of a theorem prover, since the valid actions are state dependent. However, in leanCoP, the set of literals contained in the input clauses constitute a fixed size superset of valid actions, so we can let our model select from this superset. All actions are encoded using either ENIGMA features, graph embedding or plain text and we train an action encoder $A_e : A \to \mathbb{R}^n$. The transition model concatenates latent state and action codes before predicting the successor state.

Our current implementation uses the same graph network for states and actions, which contains nodes for each clause derived either from the tableau or from the actions. In the case of state embedding, we collapse every clause while in the case of the action embedding we extract only the relevant part of the graph for each action.

Note that different axioms yield different possible actions, hence the action space can vary across problems. For this reason, our current architecture does not support model building for heterogeneous problem corpora; it is instead suitable for modeling problems within a single theory. We argue that this is the setup in which a world model makes most sense.

Dreamer for leanCoP: Sample Selection Dreamer maintains a buffer of previously explored episodes (proof attempts) which it samples from in each training step. In order to make the length of samples uniform, each sample is a fixed length slice of an episode. However, the length of proof attempts can vary greatly across problems, so we introduce a length balancing mechanism. We maintain a set of buckets $b_1 \dots b_k$ where b_j holds slices of length l in the range $2^{j-1} < l \leq 2^j$. During replay, we generate 2^{M-j} samples from bucket b_j where 2^{M-j} is the batch size. This ensures that samples from episodes of different length contribute equally. To make the sampled data more balanced, 20% of the samples are chosen with a positive reward sum to compensate for the fact that most proof attempts fail and yield no positive reward.

Dreamer for leanCoP: Rewards and Losses Assigning rewards to theorem prover actions is a well known challenge. A specialty of latent space reasoning is that we do not know what inferences are valid in a particular state, so the model has to learn that as well. To aid this, we

give high negative reward for invalid moves. Our reward function R for action a performed in state s is:

$$R(s,a) = \begin{cases} 1 & \text{if a proof was found} \\ -0.2 & \text{if there are no more valid moves} \\ -1 & \text{if } a \text{ is an invalid move} \\ -0.5 & \text{if a step limit is reached} \end{cases}$$

During training, we have the convenience that we know which actions are valid. We exploit this by adding an extra loss term to the world model, which is the negative log probability of choosing a valid step. This helps to make convergence faster.

Current Status We created an RL environment that encapsulates leanCoP and adapted the Dreamer codebase to the particularities of the environment. We implemented ENIGMA feature extraction as well as the graph model, but we have not started working on the transformer model yet.

We are running first experiments, tuning hyperparameters and evaluating the consistency of the latent representation. We find that the model struggles with the sparsity of the rewards, even when training slices are balanced. While the model quickly finds ways to avoid illegal moves and failure states by infinite derivations, it cannot yet make good use of the positive signal coming from successful proof attempts.

Conclusion Our project explores the possibility of building a world model for an automated theorem prover that captures its internal dynamics. We adapted the DreamerV2 architecture to the leanCoP connection tableau calculus and started running first experiments. The promise of such world models is to yield a semantically meaningful latent structure, in which one can identify promising directions, leading to better exploration and more targeted proof search.

Acknowledgments This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), the Hungarian National Excellence Grant 2018-1.2.1-NKP-00008 and by the Hungarian Ministry of Innovation and Technology NRDI Office within the framework of the Artificial Intelligence National Laboratory Program.

References

[1] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

[2] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *NeurIPS*, pages 8836–8847, 2018.

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *Proc. of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, pages 3084–3090. AAAI Press, 2015.

[4] Dennis Lee, Christian Szegedy, Markus Rabe, Sarah Loos, and Kshitij Bansal. Mathematical reasoning in latent space. In *International Conference on Learning Representations*, 2020.

[5] Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on*

Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.

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

[7] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[8] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019.

[9] Josef Urban and Jan Jakubuv. First neural conjecturing datasets and experiments. In Christoph Benzmüller and Bruce R. Miller, editors, *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2020.

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

[11] Zsolt Zombori, Josef Urban, and Chad E. Brown. Prolog technology reinforcement learning prover. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 489–507, Cham, 2020. Springer International Publishing.

[12] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.

Retrieval-Augmented Proof Step Synthesis

May 2021
Christian Szegedy, Markus Rabe, and Henryk Michalewski

Google Research, Mountain View, CA, USA
(szegedy,mrabe,henrykm)@google.com

Abstract

Automated theorem proving is relying increasingly on sophisticated language modelling approaches for synthesizing proof steps (tactic applications, rewrite rules). However, one of the most significant difficulties of proof search is finding the correct premises to be used. This raises the problem of combining premise selection with language modeling. There are two obvious avenues towards this goal: synthesizing the full theorem text to be utilized as a premise, or using a separate premise selection model that is used as an extra component to be used when referencing theorems. In this paper, we suggest a new solution based on language modelling that allows premise selection to become an organic component of the deep learning model and is not trained in separation. We compare this approach to theorem proving using a combination of pretrained premise selection and tactic synthesis on the HOList dataset.

1 Introduction

Premise selection [9] is a central problem of theorem proving in large theories. Realistic benchmarks of large-scale automated theorem proving are based on corpora of human-formalized mathematics and can include theories with hundreds of thousands of theorems to be utilized [10]. This suggests an evaluation setup in which the proof of each theorem is allowed to utilize only premises that were available for the original author of the corpus. Typically, the theorems are sorted in some topological order of dependence and only theorems preceding the current theorem are allowed during proving. In order to avoid information leaks, the machine learning models for premise selection have to be retrained incrementally for each theorem to be proved. This is a realistic scenario for machine learning models that can be updated very quickly, but has posed a challenge for deep-learning-based approaches. For this purpose, premise selection systems using deep learning have been evaluated by a two-phases methodology, in which the performance is measured on a held-out set of the theorems to be proved, but is trained on all possible premises, first. While this approach is compatible with most modern deep-learning-based setups, it has a potential for information leak as the premise selection for theorems might be based on information of theorems proved layer in the database. This does not model the real-life constraints in a conservative manner. Most current deep-learning-based theorem proving systems are evaluated based on the same questionable assumptions. While some results [2] on FlySpeck suggest that the effect of training on future theorems is not too critical, this methodology also reinforces the practice of developing systems that are not easily used in an incremental setup and does not measure this important aspect of the system faithfully.

2 Related Work

Theorem proving in large theories and premise selection was pioneered by [9, 10] and later in [5] for first order theorem proving. DeepMath [1] proposed a deep-learning-based approach

for premise selection in a similar setup for the Mizar [6] corpus, however their methodology suffers from the same issue of training on the proof of future theorems. Later, TacTicToe [3] suggested premise selection for higher-order-logic theorem proving. HOList [7] was suggested to combine HOL-based theorem proving with graph-neural-network-based premise-selection. However, deep-learning-based language modeling has shown surprising effectiveness for this purpose [12] and recently, GPT-f [8] has demonstrated the usefulness of large language models for proof-automation for Lean.

3 Incremental Proof Step Synthesis

Here, we present an incremental proof step synthesis approach that relies heavily and integrates seamlessly with the state-of-the-art neural architectures: especially with transformer networks [11] designed for language modelling. While language modelling has been increasingly and successfully used for synthesizing proof steps, it is typically used in a setup in which the transformer model is given enough training steps to memorize the statements to be used. This way, the network can produce either the full theorem text or a reference by naming the theorem. However, as we will demonstrate, this approach results in theorem proving performance that lags behind systems that were trained for premise selection directly via contrastive training [1]. In this talk, we present an approach that augments transformers with a retrieval based model similarly to [4]. Our approach differs from pure language modeling in that our approach allows for looking up theorems immediately after they are proved: the embeddings of theorems are stored in a database that is consulted by the transformer model using a side-attention mechanism into this dynamically database and the keys of the embeddings are updated using the standard backpropagation mechanism of that attention later and the theorem names can be extracted from the value associated with those premises. The advantage of this approach is that it integrates directly with the transformer architecture and the lookup is trained, incrementally using standard attention layers which includes a large number of negative premises and therefore alleviates the need for hard negative mining. Still, the inference mechanism utilizes standard autoregressive decoding and the final result can consult any premises that are appropriate in the given context. This is different from previous approaches [7] in which the premises were preselected and the decoder did not have full control of the synthesized proof step. We present experiments with a system that integrates this memory lookup into the transformer architecture and trained in end-to-end manner and verify that it is competitive with those approaches that utilize a separate premise selection model trained explicitly for this purpose. This paves the way towards simpler systems that allow knowledge utilization conditioned on large knowledge bases in incremental fashion and requiring less training steps.

References

[1] Alexander A Alemi, François Chollet, Geoffrey Irving, Niklas Eén, Christian Szegedy, and Josef Urban. Deepmath-deep sequence models for premise selection. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 2235–2243, 2016.

[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. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 2019.

[3] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 125–143. EasyChair, 2017.

[4] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In Hal Daumé III and Aarti Singh, editors, *International Conference on Machine Learning*, pages 3929–3938. PMLR, 2020.

[5] Cezary Kaliszyk and Josef Urban. Mizar 40 for mizar 40. *Journal of Automated Reasoning*, 55(3):245–256, 2015.

[6] The Mizar Mathematical Library. Accessed: 2018/01/18.

[7] Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, and Christian Szegedy. Graph representations for higher-order logic and theorem proving. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020.

[8] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving, 2020.

[9] Josef Urban. MPTP–motivation, implementation, first experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004.

[10] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. Malarea sg1-machine learner for automated reasoning with semantic guidance. In Baumgartner Peter Dowek Gilles Armando, Alessandro, editor, *International Joint Conference on Automated Reasoning*, pages 441–456. Springer, 2008.

[11] 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*, pages 5998–6008, 2017.

[12] Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef Urban. Exploration of neural machine translation in autoformalization of mathematics in Mizar. In Jasmin Blanchette and Cătălin Hriţcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 85–98, 2020.

NaturalProofs: Mathematics meets Natural Language

Sean Welleck[12], Jiacheng Liu[1], Ronan Le Bras[2],
Hannaneh Hajishirzi[12], Yejin Choi[12], and Kyunghyun Cho[3]

[1] University of Washington
[2] Allen Institute for Artificial Intelligence
[3] New York University

1 Introduction.

Solving the problem of understanding and creating mathematics using natural mathematical language – the mixture of symbolic and natural language used by humans – is a path towards developing agents capable of reasoning. The mixture of symbolic and natural text in informal mathematics, along with the existence of a formal counterpart, offers a unique setting for studying reasoning that complements research involving natural language alone or purely within a formal system. Moreover, systems that operate on informal mathematical text have applications in education and scientific discovery [2, 5, 9], while bridging informal and formal mathematics can be a key driver of progress in automated reasoning [10].

This talk will discuss NaturalProofs, a multi-domain corpus of mathematical statements and their proofs, written in natural mathematical language. NaturalProofs consists of 32k theorem statements and proofs, 14k definitions, and 2k other types of pages (e.g. axioms, corollaries) derived from three domains: *broad-coverage* data from ProofWiki,[1] an online compendium of mathematical proofs written by a community of contributors; *deep-coverage* data from the Stacks project,[2] a collaborative web-based textbook of algebraic geometry; and *low-resource, real-world* data from mathematics textbooks.[3] NaturalProofs unifies these sources in a common schema and is made publicly available as a resource to drive progress on tasks involving informal mathematics, complementing existing work in this direction (e.g. [4, 11, 7]).

We use NaturalProofs for *mathematical reference retrieval*, an analogue of premise selection [1, 4]: given a theorem $\mathbf{x}$, retrieve the set of references $\mathbf{y} = \{\mathbf{r}_1, \ldots, \mathbf{r}_{|\mathbf{y}|}\}$ (theorems, lemmas, definitions) that occur in its proof. As a bridge towards generative tasks, we consider *mathematical reference generation*: given a theorem $\mathbf{x}$ generate the *sequence* of references in its proof, $\mathbf{y} = (\mathbf{r}_1, \ldots, \mathbf{r}_{|\mathbf{y}|})$, which requires determining the order and number of references.

In addition to standard *in-distribution* evaluation, we evaluate *out-of-distribution*, zero-shot generalization to textbooks. We design an evaluation protocol that tests a system's ability to retrieve references for novel theorems in each setting, and benchmark methods based on large-scale neural sequence models [3, 6], including a strong *joint retrieval* method and a *sequential* variant for reference generation. The multiple informal domains, evaluation protocol, joint retrieval model, and reference generation task differ from previous work on ProofWiki [4] and formal [1, 8] premise selection.

We find that the neural methods are effective for in-domain retrieval compared to classical techniques, yet out-of-distribution generalization, leveraging symbolic mathematical content, and fully recovering a proof's references remain as fundamental challenges. The NaturalProofs data, code, and pretrained models are made publicly available.[4]

[1] https://proofwiki.org/
[2] https://stacks.math.columbia.edu/
[3] *Introduction to Real Analysis* by William F. Trench and *Elementary Number Theory* by William Stein.
[4] https://github.com/wellecks/naturalproofs.

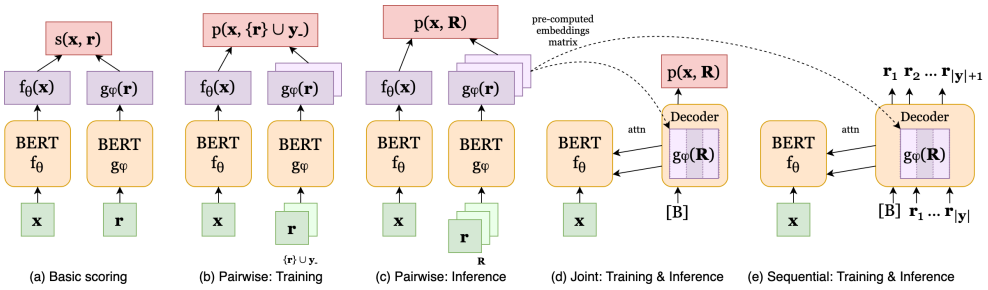


Figure 1: The pairwise, joint, and sequential methods.

1.1 Retrieval and generation methods.

As benchmark methods for our tasks, we introduce *pairwise* and *joint* retrieval methods, and a *sequential* method trained for reference generation. Figure 1 illustrates the methods.

Pairwise model. The pairwise model scores a reference $\mathbf{r}$ against a theorem $\mathbf{x}$ using two instances of BERT [3], $s_\theta(\mathbf{x}, \mathbf{r}) = f_{\theta_1}^{\text{thm}}(\mathbf{x})^\top g_{\theta_2}^{\text{ref}}(\mathbf{r})$, as illustrated in Figure 1 (a). The pairwise model is trained to contrast each positive reference with a set of negative references,

$$\mathcal{L}(\mathbf{x}, \mathbf{r}, \mathbf{y}_-) = -\log \frac{\exp(s_\theta(\mathbf{x}, \mathbf{r}))}{\exp(s_\theta(\mathbf{x}, \mathbf{r})) + \sum_{\mathbf{r}_- \in \mathbf{y}_-} \exp(s_\theta(\mathbf{x}, \mathbf{r}_-))}, \tag{1}$$

where $\mathbf{r}$ is a reference that occurs in the proof of $\mathbf{x}$, and $\mathbf{y}_-$ is a set of in-batch negatives [6] (Figure 1 (b)). This benchmark represents methods such as the dense passage retriever [6].

Joint model. The joint model scores all references in a single pass, $p_\theta(\cdot|\mathbf{x}) = \text{softmax}\left(\mathbf{R} f_\theta(\mathbf{x})\right)$, where $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$ is a reference embedding matrix and $f_\theta(\mathbf{x}) \in \mathbb{R}^d$ is a neural theorem encoder (Figure 1 (d)). This model has the advantage of computing the loss denominator in Equation 1 over *all* references rather than a subset of negatives. However, it must learn implicit representations of each reference without observing reference contents, thus we populate its embedding matrix using the pairwise reference encoder's embeddings, $\mathbf{R} = \left[g^{\text{ref}}(\mathbf{r}_1); \ldots; g^{\text{ref}}(\mathbf{r}_{|\mathcal{R}|})\right]$.

Retrieval evaluation. Given an input theorem $\mathbf{x}$, every reference is scored to induce a ranking $\mathbf{r}^{(1)}, \ldots, \mathbf{r}^{(|R|)}$ (Figure 1 (c,d)). The ranked references are compared against the ground-truth references from the proof of $\mathbf{x}$ using standard retrieval metrics, as well as whether *all* ground-truth references were ranked in the top-k.

Sequential model. We use an autoregressive model, $p_\theta(\mathbf{r}_1, \ldots, \mathbf{r}_{|\mathbf{y}|}|\mathbf{x}) = \prod_{t=1}^{|\mathbf{y}|+1} p_\theta(\mathbf{r}_t|\mathbf{r}_{<t}, \mathbf{x})$, where $\mathbf{r}_{|\mathbf{y}|+1}$ is a special $\langle\text{eos}\rangle$ token denoting the end of the reference sequence (Figure 1 (e)). The autoregressive model is trained to maximize the log-likelihood of ground-truth reference sequences. Unlike the preceding retrieval models, this model predicts the order and total number of references and can predict multiple occurrences of each reference.

For reference generation, beam search is used to generate a reference sequence $\hat{\mathbf{y}} = (\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_{|\hat{\mathbf{y}}|}\, \langle\text{eos}\rangle)$. For retrieval, we populate a ranked list using generations $\{\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_{|\hat{\mathbf{y}}|}\}$ followed by references ordered according to the first step's probabilities, $p_\theta(\mathbf{r}_1|\mathbf{x})$.

		mAP	R@10	Full@10
PWiki	**TF-IDF**	6.19	10.27	4.14
	BERT pair	16.82	23.73	7.31
	BERT joint	**36.75**	**42.45**	**20.35**
Stacks	**TF-IDF**	13.64	25.46	18.94
	BERT pair	20.93	37.43	30.03
	BERT joint	**28.32**	**39.10**	**31.96**

Table 1: *In-domain* test performance on mathematical reference retrieval, measured with mean average precision (mAP), recall (R@10) and full retrieval (Full@10) at 10.

		mAP	R@10	Full@10
RA	**TF-IDF**	**15.79**	**34.65**	**27.54**
	BERT pair	13.24	24.01	19.16
	BERT joint	11.24	20.97	16.77
NT	**TF-IDF**	**16.42**	39.62	30.00
	BERT pair	15.12	**41.51**	**35.00**
	BERT joint	15.85	41.51	35.00

Table 2: *Zero-shot* retrieval performance on out-of-domain Real Analysis (**RA**) and Number Theory (**NT**) textbooks. We show results for BERT models trained on ProofWiki.

Source	**ProofWiki**
Theorem	**Category of Monoids is Category**
	Let Mon be the category of monoids.
	Then Mon is a metacategory.
Rank	**Retrieved Reference (Joint Model)**
1	*Metacategory*
2	*Monoid*
3	Identity Morphism
4	*Identity Mapping is Right Identity*
5	*Identity Mapping is Left Identity*
6	Associative
7	Identity (Abstract Algebra)/Two-Sided Identity
8	*Composition of Mappings is Associative*
9	Composition of Morphisms
10	Semigroup

Table 3: Example top-10 retrievals. Italicized references are in the ground-truth proof.

	Edit(↓)	**BLEU**(↑)	**EM**(↑)	**F1**(↑)
**-set*	58.51	7.18	18.09	97.04
**-multiset*	58.09	16.68	19.23	100.0
**-halfseq*	58.84	25.88	0.00	56.86
Joint	93.03	0.00	0.09	25.30
Sequential	84.30	5.48	3.78	25.61

Table 4: *Reference generation* performance on ProofWiki. We show oracle benchmarks for correctly predicting the first half of the sequence (**-halfseq*), the full multiset (**-multiset*) set (**-set)* with random order. Metrics are computed on reference ids.

1.2 Main Results.

We overview key results here and provide further results and analysis in the talk. Table 1 shows *in-domain* retrieval performance, meaning that each model was trained and evaluated on the same domain. The BERT models substantially outperform the TF-IDF baseline, with the joint model showing the best performance. Table 3 shows example retrievals from the joint model.

We find substantial room for future improvement on out-of-domain generalization and reference generation. As seen in Table 2, the BERT models trained on ProofWiki show worse or similar retrieval performance as TF-IDF on the Real Analysis and Number Theory textbooks, which we also found was the case for models trained on Stacks, or both ProofWiki and Stacks.

On reference generation (Table 4), the sequential model improves over using the top-5 predictions from the retrieval model, yet falls behind oracle benchmarks that only predict the correct set (**-set*) or half of the correct sequence (**-halfseq*), leaving much room for improvement.

1.3 Looking forward.

Overall, our results show both promising immediate use of neural models for in-domain retrieval, and open challenges for the future. In the final part of the talk we discuss future work based on using or extending NaturalProofs, as well as NLP techniques that may be of interest. We hope to promote discussion about which tasks serve as meaningful proxies, the difficulty of evaluation, and bridging informal and formal reasoning.

References

[1] Alexander A Alemi, François Chollet, Niklas Een, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - Deep sequence models for premise selection. In *Advances in Neural Information Processing Systems*, pages 2243–2251, 2016.

[2] Nathan C. Carter and Kenneth G. Monks. Lurch: a word processor that can grade students' proofs. In Christoph Lange, David Aspinall, Jacques Carette, James H. Davenport, Andrea Kohlhase, Michael Kohlhase, Paul Libbrecht, Pedro Quaresma, Florian Rabe, Petr Sojka, Iain Whiteside, and Wolfgang Windsteiger, editors, *Joint Proceedings of the MathUI, OpenMath, PLMMS and ThEdu Workshops and Work in Progress at CICM, Bath, UK*, volume 1010 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[4] Deborah Ferreira and André Freitas. Natural language premise selection: Finding supporting statements for mathematical text. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 2175–2182, Marseille, France, May 2020. European Language Resources Association.

[5] Dongyeop Kang, Andrew Head, Risham Sidhu, Kyle Lo, Daniel S. Weld, and Marti A. Hearst. Document-level definition detection in scholarly documents: Existing models, error analyses, and future directions, 2020.

[6] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[7] Aditya Ohri and Tanya Schmah. Machine Translation of Mathematical Text. *IEEE Access*, 2021.

[8] Bartosz Piotrowski and J. Urban. Stateful premise selection by recurrent neural networks. In *LPAR*, 2020.

[9] Yiannos Stathopoulos, Angeliki Koutsoukou-Argyraki, and Lawrence Paulson. Developing a concept-oriented search engine for isabelle based on natural language: Technical challenges. 12 2020.

[10] Christian Szegedy, editor. *A Promising Path Towards Autoformalization and General Artificial Intelligence*, 2020.

[11] Qingxiang Wang, Chad Brown, Cezary Kaliszyk, and Josef Urban. Exploration of neural machine translation in autoformalization of mathematics in Mizar. In *CPP 2020 - Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, co-located with POPL 2020*, 2020.

Latent Action Space for Efficient Planning in Theorem Proving

Minchao Wu*, Yuhuai Wu*

August 24, 2021

Abstract

One of the most critical challenges in applying machine learning techniques to automated reasoning is the need to work with an enormous action space. Not only does it make the exploration difficult, but also it is very time-consuming to generate at inference time. In this work, we introduce latent action space with a world model to speed up the efficiency of action generation, with the potential of alleviating the exploration problem, as well as improving sample efficiency using the world model.

1 Introduction

One of the major challenges in theorem proving [14, 2, 7, 12, 10, 11, 6] is the need to deal with an enormous action space. In the most general setting, the action space for a theorem proving agent consists of a sequence of strings, representing a tactic application along with theorem parameters, or an intermediate proposition, or a new definition, lemma, theorem statements. This approach also has been adopted in recent works using transformer-based models [12, 6], because of its generality (e.g., the capability of generating new terms). However, due to the nature of autoregressive generation for such actions, even doing one action generation requires a quiet amount of time, not to mention if one wants to search multiple steps ahead.

In this work, we propose to learn such a high-level representation, by embedding the raw action space into a latent action space.

2 Method

In order to embed the action into a continuous latent space, we first introduce an action encoder and an action decoder.

- Encoder$_{action}$: action space $\rightarrow$ latent action space : $\alpha \sim q(\alpha|a)$.
- Decoder$_{action}$: latent action space $\rightarrow$ action space : $\hat{a} \sim p(\hat{a}|\alpha)$.

We can train the action encoder and decoder using the reconstruction objective:

$$\mathcal{L}_{rec} = D(p(a)||p(\hat{a}|a)), \tag{1}$$

where $p(\hat{a}) = \sum_\alpha p(\hat{a}|\alpha)q(\alpha|a)$ and D is a metric on the distribution of actions. The most natural choice is the KL divergence (cross entropy loss) between the original action distribution (label) and the decoded action distribution.

*Equal contribution. MW is at the Australian National University and YW is at the University of Toronto. Correspondence to Minchao.Wu@anu.edu.au and ywu@cs.toronto.edu.

However, we cannot directly work with latent action space, because the environment only accepts the raw action space as input. Therefore, an "environment" in the latent space is necessary for this purpose. This then naturally leads us into the model-based RL techniques [13, 3, 4, 8, 5].

We hence introduce the following components. Firstly, we introduce a state encoder, that encodes the proof state x_t into a latent state space $\mathcal{Z}$ represented by a continuous vector, as well as its counterpart, a state decoder – that decodes the latent state back to the proof state.

- State encoder: $z_t \sim q(z_t|z_{t-1}, \alpha_t, x_t)$
- State decoder: $\hat{x}_t \sim p(\hat{x}_t|z_t)$

Next, given the latent state space, we introduce the latent transition operator. The transition operator will sample the next latent state given the latent state and action at the current time step. Namely, we use a neural network model to learn the internal dynamics of the theorem proving engine, performing the deduction step of theorem proving.

- Latent transition operator: $\hat{z}_t \sim p(\hat{z}_t|z_{t-1}, \alpha_{t-1})$

To train the state encoder and decoder, we also use a reconstruction objective as the case of latent actions. To train the transition operator, we use a forward prediction loss – the cross-entropy loss between the ground truth latent state and the predicted latent state. Furthermore, to add more semantic groundings for the latent action space, we also use the forward prediction loss to train the action and state encoder. To summarize, the total loss objective is written below:

$$\mathcal{L} = \mathcal{L}_{rec}(a) + \mathcal{L}_{rec}(x) + \mathcal{L}_{forward}.$$

Given a latent transition operator, one can perform efficient planning in the latent space – looking ahead by unrolling the state dynamics for a number of steps. Unlike generating a full sequence of tokens at each step, the latent action allows a one-shot generation, immensely shortening the planning time. There are many possibilities in terms of integrating the transition operator with various kinds of search algorithms, such as best-first search, MCTS, etc. – a vast space for exploration.

3 Experiments

We start by learning the state and action encoding, as well as the dynamics of the INequality Theorem proving benchmark (INT) [15]. We generate 40000 proof trajectories using INT using a cardinality of an axiom combination $K = 3$ and a length of a proof $L = 7$. The data set then contains 149009 distinct transitions which are split into training and test data sets with a 80:20 ratio.

We use a character-level transformer to learn the latent representations of both state and action, and use an MLP to learn the internal dynamics (*i.e.* the transition operator) of INT. The transformer uses 256 embedding dimensions,

Table 1: Performance on the test set. $\texttt{BLEU}_{rec-action}$ (resp. $\texttt{BLEU}_{rec-state}$) denotes the BLEU score of the reconstructed actions. $\texttt{BLEU}_{trans}$ denotes the BLEU score of the predicted states by applying the transition operator once. QED accuracy is the percentage of correctly predicted QEDs by applying the transition operator once.

Methods	$\texttt{BLEU}_{rec-action}$	$\texttt{BLEU}_{rec-state}$	$\texttt{BLUE}_{trans}$	QED accuracy (%)
$\mathcal{L}_{CE}$	96.98	94.12	88.23	94.20
$\mathcal{L}_{MSE}$	73.87	69.38	60.18	0

8 attention heads and 1024 hidden dimensions for position-wise feed-forward layers. We also use dropout with rate 0.1, label smoothing with coefficient 0.1, and a maximum 128 tokens for both training and evaluation examples. The MLP is a residual block with two hidden layers of dimensions 1024 and 512. We use the Adam optimizer [9] for training.

We experiment with two different forward losses for training the transition operator. $\mathcal{L}_{CE}$ denotes the cross-entropy loss between the ground truth target state and the decoded predicted latent state. $\mathcal{L}_{MSE}$ denotes the mean squared error between the encoded ground truth of target state and the predicted latent state. We implement our algorithms in JAX [1] and run both experiments for 100k training steps using a single NVIDIA Tesla V100 GPU and 8 cores of an Intel(R) Xeon(R) CPU @ 2.20GHz.

Table 1 shows the quality of the transition prediction and the reconstruction of states and actions when evaluated on the test set. When calculating the BLEU scores for transition predictions, we separate out those whose references are a single "QED" token (which indicates the end of the proof) to make sure that BLEU scores reflect the quality of prediction properly[1]. We add an additional metric called QED accuracy which is the percentage of exact matches of the QED token. Figure 1 shows the quality of transition prediction when the state dynamics is unrolled for a number of steps using the learned transition operator. It can be seen that the transition operator trained using $\mathcal{L}_{CE}$ outperforms the one trained using $\mathcal{L}_{MSE}$ by a large margin, and that the latter lacks the ability to correctly predict QEDs.

4 Discussion

There has been an early investigation on latent space for mathematical reasoning [10], which shows promising results of neural networks for predicting the latent state several steps ahead, in the HOList system with an ad-hoc action

[1]For example, if we have references: ["QED","QED","QED","QED"] with predictions: ["*to* ((((*b* ∗ *a*) + (*a* ∗ *b*)) ∗ (*a* ∗ (*a* + *b*))) ∗ (*c* ∗ 1)) = ((((*a* + *b*) ∗ *a*) ∗ (*a* + *b*)) ∗ (*c* ∗ 1))","QED","QED","QED"]), the BLEU score of this corpus is only 0.79, which does not reflect the quality of prediction properly.

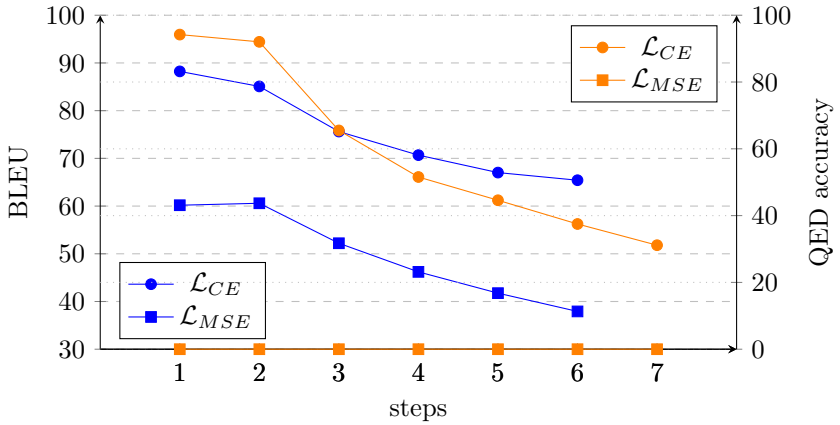


Figure 1: Quality of transition operator with respect to the number of steps unrolled. Given a state s, we look ahead n steps by recursively applying the transition operator to s and the subsequent ground truth actions corresponding to s. The further we unroll, the more difficult it becomes for the transition operator to correctly predict the target states. Note the different scale on right for QED accuracy. Step 7 has a QED accuracy instead of a BLEU score because all target states at step 7 are QEDs.

space. Greatly inspired by it, we propose to build a full-fledged latent space system for the most general action space, to improve mathematical reasoning in planning efficiency. In the meantime, the world model potentially can speed up the interaction time with the environment, and also improve the sample efficiency. Furthermore, we believe if the latent space is semantically grounded, exploration in the latent action space can also provide big gains over exploring with a sequence of long tokens. We hope our work provides a meaningful direction to future machine learning models for theorem proving.

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[2] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving - learning to select a good heuristic. *J. Autom. Reason.*, 53(2):141–172, 2014.

[3] David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018.

[4] Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[5] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193, 2020.

[6] Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. Proof artifact co-training for theorem proving with language models. *CoRR*, abs/2102.06203, 2021.

[7] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. Deepmath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike von 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.

[8] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 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.

[10] Dennis Lee, Christian Szegedy, Markus N. Rabe, Sarah M. Loos, and Kshitij Bansal. Mathematical reasoning in latent space. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[11] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence Paulson. Isarstep: a benchmark for high-level mathematical reasoning. 2021.

[12] Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020.

[13] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Bruce W. Porter and Raymond J. Mooney, editors, *Machine Learning, Proceedings of the Seventh International Conference on Machine Learning, Austin, Texas, USA, June 21-23, 1990*, pages 216–224. Morgan Kaufmann, 1990.

[14] Josef Urban, Jirí Vyskocil, and Petr Stepánek. Malecop machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings*, volume 6793 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2011.

[15] Yuhuai Wu, Albert Jiang, Jimmy Ba, and Roger B. Grosse. INT: an inequality benchmark for evaluating generalization in theorem proving. *CoRR*, abs/2007.02924, 2020.

Decision Trees for Tactic Prediction in Coq *

Liao Zhang[1,3], Lasse Blaauwbroek[1,2], Bartosz Piotrowski[1,4], Cezary Kaliszyk[3,4], and Josef Urban[1]

[1] Czech Technical University, Prague, Czech Republic
[2] Radboud University, Nijmegen, The Netherlands
[3] University of Innsbruck, Austria
[4] University of Warsaw, Poland

1 Introduction

We present ongoing work of comparing several different decision tree learning methods on predicting tactics for proof states in Coq [5] by grasping knowledge from preceding human-written proofs. The work is based on Tactician[1], a plugin enabling proof automation for Coq. The authors use Tactician to extract features from the abstract syntax trees of proof states in the form of term walks of length one and two. We apply Tactician to record all the proof states $(253,538)$ and respective tactics in the Coq standard library and randomly choose 1% $(2,530)$ as the evaluation data. Three algorithms are implemented: random forests and gradient boosted trees working on binary learning with negative examples and multilabel regression. Finally, we compare the results of the decision tree models and the Tactician's original method, k-nearest neighbors (k-NN).

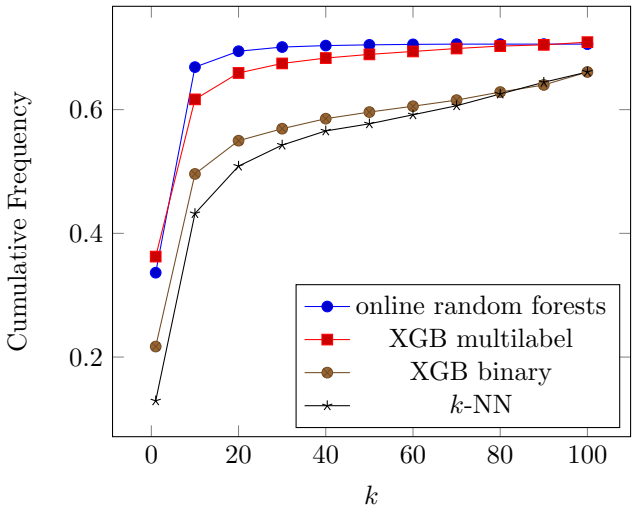
2 Tactic Prediction Models

After establishing the dataset of state features and tactics as outlined above, we can apply machine learning algorithms for tactic prediction. The naive k-NN simply computes the similarity between proof state features by the Jaccard index [4] to find likely tactics.

Random forests, a popular machine learning method, provide predictions by voting from multitude decision trees. Typically, random forests will not update the structure after training. Since grasping local knowledge is essential for Tactician, we take the inspiration from [3] and implement an online version here [6] capable of incremental optimization. When a new training instance is passed to a leaf of the decision tree, we split the leaf and create a new decision node with two successive leaves. Under a certain size limitation, a new tree grows in the forests with probability $\frac{1}{n}$ where n is the current tree number. To make predictions for a given example, it should be passed to the respective leaves (denoting a tactic to be predicted) of all the trees by the decision rules, and the tactics are ranked by their occurring frequency in the pointed leaves.

Gradient boosted trees apply boosting techniques, appropriate combinations of weak base learners, to construct a powerful learner, and XGBoost[2] is such a popular library. On the binary regression setting, XGBoost receives (f,t) where f denotes the features of the proof state, and t represents the hash of the corresponding tactic in the Coq library. The input is labeled 0 or 1 according to the tactic being advantageous or not for the proof state. A tactic for a proof state gets a positive label if it is exactly the one applied to the state in the library. In contrast,

*This work was supported by the ERC grant no. 714034 *SMART*, by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466), and by the Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project *POSTMAN* no. LL1902.



negative tactics are the elements in the tactic space and differ from the positive instance. We apply k-NN to predict tactics for a state in the library by learning from the preceding states and arbitrarily select a subset from the best-100 predictions as the negative instances. While predicting tactics for a certain state, we first preselect the top-100 k-NN predictions. Then, we use the XGBoost model to obtain scores for each tactic and return tactics by the order of the corresponding scores. The input to XGBoost on multilabel regression is simply f with the respective positive tactic of the state as the label, which builds an individual regressor for each label. Every label's regressor performs binary regression on two kinds of data: the proof states with the same label (tactic) applied and the others. For a given example, each tree model calculates a score, and we predict the tactics related to the models in score order.

3 Experimental Evaluation

The online random forests learn from all the previous states and dynamically update the structure during the prediction. Since the introduced XGBoost algorithms are offline, we build an individual model for each state while performing experiments. To speed up the XGBoost evaluation, we merely provide the previous $1,000$ proof states encountered before the current proof state for training.

Figure 3 depicts the accuracy of the presented machine learning methods. The k-NN always has the lowest accuracy rates. In the beginning, the XGBoost's multilabel regression performs best, while the random forests surpass others later. The binary regression setting is much worse than the other decision tree approaches for all the k. This indicates that we need better semantic characterizations of tactics instead of the naive hash.

References

[1] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician. In *International Conference on Intelligent Computer Mathematics*, pages 271–277. Springer, 2020.

[2] 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*, pages 785–794, 2016.

[3] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[4] Paul Jaccard. The distribution of the flora in the alpine zone. 1. *New phytologist*, 11(2):37–50, 1912.

[5] The Coq Development Team. The Coq proof assistant, version 8.11.0, Oct 2019.

[6] Liao Zhang, Lasse Blaauwbroek, Bartosz Piotrowski, Prokop Černỳ, Cezary Kaliszyk, and Josef Urban. Online machine learning techniques for coq: A comparison. *arXiv preprint arXiv:2104.05207*, 2021.