AITP 2018

The Third Conference on Artificial Intelligence and Theorem Proving

Abstracts of the talks

March 25 – 30, 2018, Aussois, France

Preface

This volume contains the abstracts of the talks presented at AITP 2018: The Third Conference on Artificial Intelligence and Theorem Proving held on March 25–30, 2018 in Aussois, France.

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 2018 consists of several sessions discussing connections between modern AI, ATP, ITP and (formal) mathematics. The sessions are discussion oriented and based on 10 invited talks and 21 contributed talks.

We would like to thank the Aussois CNRS conference center for hosting AITP 2018. Many thanks also to Andrei Voronkov and his EasyChair for their support with paper reviewing and proceedings creation. The conference was partly funded from the European Research Council (ERC) under the EU-H2020 projects SMART (no. 714034) and AI4REASON (no. 649043), and the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 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!

March 25, 2018
Pittsburgh
Innsbruck
Stuttgart
Prague

Thomas C. Hales
Cezary Kaliszyk
Stephan Schulz
Josef Urban

Table of Contents

Axiomatizing consciousness, with applications 6
Henk Barendregt and Antonino Raffone

Some Reflections on a Computer-aided Theory Exploration Study in Category Theory (Extended Abstract) 7
Christoph Benzmüller and Dana Scott

Project Proposal: Proof Guidance by Compression 9
Lasse Blaauwbroek

Hammer for Coq: Automation for Dependent Type Theory 12
Łukasz Czajka and Cezary Kaliszyk

Computational Exploration of String Theory 13
Michael Douglas

Revisiting SAD 15
Steffen Frerix and Peter Koepke

Safe Reinforcement Learning via Formal Methods 17
Nathan Fulton and André Platzer

Reasoning and Consciousness — Teaching a Theorem Prover to let its Mind Wander 19
Ulrich Furbach and Claudia Schon

TacticToe: Learning to Prove with Tactics 22
Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar and Michael Norrish

First Experiments with Watchlist Guidance on Mizar 23
Zarathustra Goertzel, Jan Jakubuv and Josef Urban

Guiding SMT Solvers with Monte Carlo Tree Search and Neural Networks 26
Stéphane Graham-Lengrand and Michael Färber

LET'S MAKE SET THEORY GREAT AGAIN! 30
John Harrison

Automation by Analogy, in Coq 31
Alasdair Hill and Ekaterina Komendantskaya

Enhancing ENIGMA Given Clause Guidance 33
Jan Jakubuv and Josef Urban

Towards Machine Learning for Quantification 36
Mikolas Janota

Project Proposal: Reinforcement Learning for leanCoP 37
Cezary Kaliszyk, Henryk Michalewski, Piotr Milos, Mirek Olsak and Josef Urban

Mizar in Isabelle for Formal Abstracts 40
Cezary Kaliszyk and Karol Pak

Mechanizing Principia Logico-Metaphysica in Functional Type Theory ... 42
Daniel Kirchner, Christoph Benzmüller and Edward N. Zalta

Toward AI for Lean, via metaprogramming 45
Robert Lewis

Cumulative Effects in Learning 47
Erik Martin-Dorel and Sergei Soloviev

Machine comprehension of math problem text 49
Takuya Matsuzaki and Noriko H. Arai

If mathematical proof is a game, what are the states and moves? 50
David McAllester

Towards logics for neural conceptors 51
Till Mossakowski and Razvan Diaconescu

Designing Games of Theorem Proving 53
Yutaka Nagashima

Who cares about Euclidean geometry? 56
Mirek Olšák

ATP-guidance for Learning Premise Selection 57
Bartosz Piotrowski and Josef Urban

Dynamic Strategy Priority: Empower the strong and abandon the weak. (Extended Abstract) 60
Michael Rawson and Giles Reger

Implementation of Lambda-Free Higher-Order Superposition 62
Petar Vukmirovic

Disambiguating ProofWiki into Mizar: First Steps 63
Jiri Vyskocil and Josef Urban

Building an Auto-Formalization Infrastructure through Text-Mining on Mathematical Literature: Project Description 66
Qingxiang Wang

Program Committee

David Aspinall	The University of Edinburgh
Jasmin C. Blanchette	Vrije Universiteit Amsterdam
Joseph Corneli	Department of Computing
Cameron Freer	Remine
Ulrich Furbach	University of Koblenz
Thomas Hales	University of Pittsburgh
Sean Holden	University of Cambridge
Geoffrey Irving	Google
Moa Johansson	Chalmers University of Technology
Cezary Kaliszyk	University of Innsbruck
Michael Kohlhase	Computer Science, FAU Erlangen-Nürnberg
Ramana Kumar	Datat61, CSIRO; and UNSW
Jens Otten	University of Oslo
Stephan Schulz	DHBW Stuttgart
Dawn Song	University of California, Berkeley
Martin Suda	Vienna University of Technology
Geoff Sutcliffe	University of Miami
Charles Sutton	The University of Edinburgh
Christian Szegedy	Google
Josef Urban	Czech Technical University in Prague

Additional Reviewers

Betzendahl, Jonas
Brown, Chad
Färber, Michael
Müller, Dennis
Schäfer, Frederick

Axiomatizing consciousness, with applications

Henk Barendregt

Radboud University Nijmegen

Abstract. Consciousness is defined as a stream of configurations that consist of three components: object, state, and action. The configurations change in discrete moments in time, while their three components influence each other recurrently, depending on the situation in the world. Mindfulness enables modifying the stream of configurations by taking states as objects of consciousness, on which an influence can be exerted. Several mental mechanisms can be understood by the axiomatic theory. 1. Memory, learning and deconditioning. 2. Mental suffering, including clinical phenomena. 3. The Church-Turing thesis on human computability. Applications of the understanding of mental suffering can be found in clinical practice and meditation. The validity of these practices are increasingly investigated in cognitive (neuro)psychology.

Some Reflections on a Computer-aided Theory Exploration Study in Category Theory

Christoph Benzmüller[1] and Dana S. Scott[2]

[1] University of Luxembourg, Luxembourg & Freie Universität Berlin, Berlin, Germany
c.benzmueller@gmail.com
[2] Visiting Scholar at University of California, Berkeley, USA
dana.scott@cs.cmu.edu

We present some reflections on the use of automated theorem proving and model finding technology in the context of a recent theory exploration study in category theory [1, 2].

In our stepwise development of mutually equivalent axioms sets for category theory we started out with a generalised notion of monoids. More precisely, the first axiom system in our study was obtained by generalizing the standard axioms for a monoid to a partial composition operation. In subsequent development steps we simplified this initial axioms set until we reached the axioms as proposed by Scott [15] in the 1970s. We then compared this axioms set with an alternative proposal by Freyd and Scedrov [11]. In the course of this comparison we revealed a technical flaw for the axiom set of Freyd and Scedrov: either all operations, e.g. morphism composition, are total or their axiom system is inconsistent. The repair for this problem is quite straightforward and it essentially corresponds to the set of axioms proposed by Scott.

Our experiments were enabled by a semantical embedding of free logic [14] in classical higher-order logic (HOL), which we implemented in the proof assistant system Isabelle/HOL [12]. Free logic was utilised to support an adequate handling of partiality and undefinedness in the modeling of morphism composition, and the domain and codomain operators. Our experiments were substantially supported by automated reasoning technology, in particular, by the model finder Nitpick [7] and by various automated theorem provers (CVC4 [10], E [13], Leo-II [3], Satallax [8], SPASS [6], Z3 [9], etc.) integrated with Isabelle/HOL via Sledgehammer [5].

In our presentation at AITP 2018 we particularly want to reflect on the role these systems played in our experiments. This is of practical and also of epistemological relevance, since these systems, as we will evidence, can indeed substantially foster the gain of new knowledge. We will therefore highlight relevant points in our stepwise development in which these systems, in particular, the model finder Nitpick, supported the gain of intuition by providing countermodels to still slightly flawed axioms or definitions. And the theorem provers supported the detection of the constricted inconsistency, in addition to the important, albeit more traditional, role they played in confirming equivalences between different axioms sets as soon as we arrived at their correct formulations.

Despite our reassuring overall teamwork experience, which involved a domain expert (Scott), a theorem proving expert (Benzmüller) and the Isabelle/HOL framework, we also collected several critical remarks pointing to a range of improvement opportunities. Some of these improvement opportunities are of technical nature, others may include theoretical aspects. For example, Nitpick should be improved by devising and implementing better readable and eventually more domain specific representations of models and countermodels. In our experiments such conversions were in fact laboriously handled by hand by Benzmüller and the results were then communicated by email to Scott. In some cases calls of external theorem provers via Sledgehammer resulted in technical error messages, which may demotivate non-expert users, and when the theorem provers succeeded, then their proofs could most of the time not be converted into informative Isar style proofs. The constricted inconsistency result, for example, had to be reconstructed by hand to obtain an human-friendly Isar style proof (see [4] for a similar experience in a different context).

Hence, our successful experiments, in which automated reasoning tools integrated in Isabelle/HOL have demonstrated their capabilities beyond mere proof verification, still required a close interaction between three players: a domain expert, a theorem proving expert and the Isabelle/HOL proof assistant. The challenge in fact still is to get the second player completely out of the loop, without requiring the first player to adopt a nearly identical level of technical expertise in a resource-intensive, laborious manner.

References

[1] C. Benzmüller and D. Scott. Automating free logic in Isabelle/HOL. In G.-M. Greuel, T. Koch, P. Paule, and A. Sommese, editors, *Mathematical Software – ICMS 2016*, volume 9725 of *LNCS*, pages 43–50, Berlin, Germany, 2016. Springer.

[2] C. Benzmüller and D. Scott. Axiomatizing category theory in free logic. *CoRR*, abs/1609.01493, 2016. This work is currently submitted for journal publication.

[3] C. Benzmüller, N. Sultana, L. C. Paulson, and F. Theiss. The higher-order prover Leo-II. *Journal of Automated Reasoning*, 55(4):389–404, 2015.

[4] C. Benzmüller and B. Woltzenlogel Paleo. The inconsistency in Gödel's ontological argument: A success story for AI in metaphysics. In *IJCAI 2016*, volume 1-3, pages 936–942. AAAI Press, 2016.

[5] J. C. Blanchette, S. Böhme, and L. C. Paulson. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 51(1):109–128, 2013.

[6] J. C. Blanchette, A. Popescu, D. Wand, and C. Weidenbach. More SPASS with Isabelle - Superposition with Hard Sorts and Configurable Simplification. In *ITP*, volume 7406 of *LNCS*, pages 345–360. Springer, 2012.

[7] J.C. Blanchette and T. Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In *ITP 2010*, number 6172 in LNCS, pages 131–146. Springer, 2010.

[8] C. E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning – IJCAR*, volume 7364 of *LNCS*, pages 111–117. Springer, 2012.

[9] L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[10] M. Deters, A. Reynolds, T. King, C. W. Barrett, and C. Tinelli. A tour of CVC4: how it works, and how to use it. In *FMCAD*, page 7. IEEE, 2014.

[11] P. Freyd and A. Scedrov. *Categories, Allegories.* North Holland, 1990.

[12] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic.* Number 2283 in LNCS. Springer, 2002.

[13] S. Schulz. System description: E 1.8. In *LPAR-19*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[14] D. Scott. Existence and description in formal logic. In R. Schoenman, editor, *Bertrand Russell: Philosopher of the Century*, pages 181–200. George Allen & Unwin, London, 1967. (Reprinted with additions in: Philosophical Application of Free Logic, edited by K. Lambert. Oxford Universitry Press, 1991, pp. 28 - 48).

[15] D. Scott. Identity and existence in intuitionistic logic. In M. Fourman, C. Mulvey, and D. Scott, editors, *Applications of Sheaves: Proceedings of the Research Symposium on Applications of Sheaf Theory to Logic, Algebra, and Analysis, Durham, July 9–21, 1977*, volume 752 of *Lecture Notes in Mathematics*, pages 660–696. Springer, 1979.

Project Proposal: Proof Guidance by Compression

Lasse Blaauwbroek*

Czech Institute for Informatics, Robotics and Cybernetics,
Czech Republic
lasse@blaauwbroek.eu

Abstract

We propose a method for guiding the proof search of theorem provers based on compression. Given a pre-existing corpus of proof states and corresponding helpful proof steps, we propose compression as a means to find the state that is closest to the current proof state, thereby giving us a likely next proof step. We compare two states by treating them as strings, and then compressing their concatenations. If the states are very similar, we may expect the compressed string to be much smaller due to information sharing between the states. This can then be used to create a distance metric between proof states.

Normalized Compression Distance Let C be a reference compression algorithm such that $C(s)$ denotes the compressed version of s. As first done by Cilibrasi and Vitnyi [1] we will be utilizing this compressor to build a similarity metric. The intuitive idea behind this is that if two string s and t are similar, the compressor can use the information in s to reduce the compression size of t. Hence, we would expect $|C(st)|$ to be much smaller than $|C(s)| + |C(t)|$, and when s and t are equal we expect $|C(st)|$ to approach $|C(s)| + b$ where b is some small constant used to encode the duplicate information. On the other hand, when s and t share no information, $|C(st)|$ is expected to be equal or larger than $|C(s)| + |C(t)|$.

We now define a distance function, called the *Normalized Compression Distance* (NCD), which is formed by normalizing $|C(st)|$ such that longer strings are not penalized.

$$\mathrm{NCD}_C(s,t) = \frac{|C(st)| - \min(|C(s)|, |C(t)|)}{\max(|C(s)|, |C(t)|)}$$

Let us for a moment consider the perfect compressor K, of which the compression size is called the Kolmogorov complexity. It is known that NCD_K satisfies all the standard laws of a distance metric [4]. More than that, it can be shown that NCD_K is in some sense the "universal" metric because it simultaneously captures all other (computable) metrics. Using this metric would therefore subsume all other metrics, effectively solving all problems in Artificial Intelligence. Unfortunately, it is well-known that the Kolmogorov complexity is an undecidable function, thereby crushing our dream of perfect AI. We can, however, still hope to create an approximation to this metric by using existing compression algorithms.

Resolution Selection In this proposal, we endeavour to apply the NCD to mathematical formulas. Our hope is that a good compression scheme will automatically find similar structures within formulas, which would make for a good predictor of similar formulas. However, the fact that formulas are usually represented by relatively short strings is a practical problem here. Most compression algorithms are optimized for large bodies of text, and often have a substantial constant overhead (i.e. to store dictionaries). Therefore, these algorithms perform rather poor on short strings (often inflating the size of the string instead of deflating it).

*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)

In order to combat this problem, we propose not to compare individual formulas with each other, but rather to compare entire proof states that represent partial proofs of theorem provers. These states should generally be much larger and therefore better suited for compression. Although this idea should in principle be widely applicable, we have chosen the leanCoP theorem prover [3] for our initial experiments. This prover tries to derive a contradiction by applying extension and reduction inferences to a connection tableau. In order to apply an inference rule on a leaf of the tableau, the prover must choose from a list of possible clauses to perform a rule with. To make this choice, one can try to look at the current branch of the tableaux. One option is to try and find a previously encountered proof situation that is similar to the current situation and for which the best next inference is known. We propose to find this similar proof state using the NCD. For this, a string is created that is representative of the proof state, by simply concatenating the literals on the current proof branch together with the list of possible clauses to perform resolution with.

Prediction by Partial Matching Selecting a good compression algorithm for the task described above is not easy. We wish the algorithm to have a small constant size overhead, and one that compresses good on small texts. This means that algorithms based on dictionaries are not a good fit. The state of the art in compression is based on *Prediction by Partial Matching* (PPM) [2]. In short, the idea is to process the input as a stream. While processing, a predictive model is built that tries to guess what the next character(s) in the stream are. Whenever the guess is correct, these characters do not have to be included in the output stream. For decoding one can then build the same predictive model to obtain the missing characters. This approach has the advantage that the model does not need to be stored in the output stream, making the constant overhead effectively zero. The predictive models of general purpose compressors are generally geared towards (human) text. We are looking into optimizing the model for formulas by making use of the tree-structure found in formulas.

Efficiency The task of finding the most similar string to a given string s out of a large pool of candidates can be very computationally expensive because the compressor needs to be invoked for every candidate in the pool. We need a more efficient method of finding the most similar string in the pool. It can be shown that under reasonable assumptions, the NCD of an imperfect compressor approximately admits the laws of a metric [1]. However, the NCD does not give us a vector space to work with to speed up this algorithm.

We propose a method that imposes a graph on the set of strings in the pool, such that strings that are similar are close neighbors of each other in the graph. The idea is to approximate an n-dimensional vector space. The n neighbors of a string s are chosen such that they are as close as possible to s while being as far as possible from each other. This ensures that different neighbors are as orthogonal to each other as possible. We define this graph as follows. Let S be the set of strings in our pool. The notation S_n, denotes the set containing all subsets of S of size n.

$$S_n = \{X \subseteq S \mid |X| = n\}$$

Now, let $s \in S$. We define the set of successors of s as follows.

$$\mathrm{out}(s) = \arg\max_{X \in S_n} \frac{\sum\limits_{t,u \in X} NCD(t,u)}{\sum\limits_{t \in X} NCD(s,t)}$$

Equipped with this graph, we propose a hill-climbing algorithm to find the most similar string in the pool. We simply start with a random node in the graph, and find the neighbor that provides the most improvement in similarity. This process is repeated until a local optimum is reached.

References

[1] Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *CoRR*, cs.CV/0312044, 2003.

[2] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. Communications*, 32(4):396–402, 1984.

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Certified connection tableaux proofs for HOL Light and TPTP. In Xavier Leroy and Alwen Tiu, editors, *Proc. of the 4th Conference on Certified Programs and Proofs (CPP'15)*, pages 59–66. ACM, 2015.

[4] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004.

Hammer for Coq: Automation for Dependent Type Theory

Łukasz Czajka and Cezary Kaliszyk

University of Innsbruck

Abstract. We present an architecture of a full hammer for dependent type theory together with its implementation for the Coq proof assistant. A key component of the hammer is a proposed translation from the Calculus of Inductive Constructions, with certain extensions introduced by Coq, to untyped first-order logic. The translation is "sufficiently" sound and complete to be of practical use for automated theorem provers. We also introduce a proof reconstruction mechanism based on an eauto-type algorithm combined with limited rewriting, congruence closure and some forward reasoning. The algorithm is able to re-prove in the Coq logic most of the theorems established by the ATPs. Together with machine-learning based selection of relevant premises this constitutes a full hammer system. The performance of the overall procedure is evaluated in a bootstrapping scenario emulating the development of the Coq standard library. For each theorem in the library only the previous theorems and proofs can be used. We show that 40.8% of the theorems can be proved in a push-button mode in about 40 seconds of real time on a 8-CPU system.

Computational Exploration of String Theory

Michael R. Douglas

keywords: computational mathematics, string theory, theoretical physics

Abstract

Superstring theory is a physical framework that unifies quantum mechanics and general relativity in ten-dimensional space-time. By assuming that the extra six dimensions are a small compact manifold, one can derive theories of physics in four dimensions that can reproduce all the experiments and observations to date, as well as predicting new physics such as supersymmetry and dark matter.

The detailed predictions of the theory depend on what we assume for the extra dimensions – their topology and metric, and the existence and location of a variety of extra fields and physical objects such as "branes." Given a set of these choices one can derive a potential function, and each local minimum of one of these potential functions is referred to as a "string vacuum." Given a choice of vacuum, much is known about how to compute detailed physical predictions such as the spectrum of particles and their masses, data which can be encoded in a relatively compact structure called an effective field theory (EFT).

The choices and minima which determine a vacuum can be classified so that the problem of enumerating vacua becomes combinatorial. This classification and the computations involve a great deal of mathematics, mostly group theory and algebraic geometry, but also Riemannian geometry and many other fields. More specifically the required mathematics includes the theories of Calabi-Yau manifolds, toric geometry and polytopes, resolution of singularities, sheaf cohomology, Hodge theory, moduli spaces, automorphic functions and forms, and many numerical techniques.

The number of string vacua is extremely large, both because of the number of combinatorial choices involved, and on physical grounds. Certain physical problems – most importantly the cosmological constant problem – can only be solved if the number of vacua is greater than roughly 10^{120}. There are mathematical arguments that the total number of vacua is finite, with estimates ranging from 10^{500} to $10^{250,000}$. The full structure is not just a set of this number of vacua, each with an associated EFT, but is actually a weighted graph whose nodes are the vacua. This graph describes quantum tunneling processes which connect the vacua, which occur in cosmological dynamics and generate a Markov process on the set of vacua. This structure is known as the string landscape.

Physicists have been using computational methods to study the string landscape since its beginnings. A celebrated example developed in the 90's and still of central importance is the

Kreuzer-Skarke database of reflexive polytopes, which determines the set of six-dimensional Calabi-Yau manifolds which can be realized as toric hypersurfaces. As time goes on, more and more sophisticated computational techniques are being employed. Computational algebra systems such as GAP, Macaulay 2 and Sage are regularly used, and some of this work has been done in collaboration with computational algebraic geometers. A new trend is to use machine learning and data science techniques – this was the subject of a recent meeting **String Theory and Data Science** at Northeastern University. Repositories for the code and data generated by this research are now in the process of being created.

In this talk we give an introduction to this area for computer scientists, sketching the ideas and some of the basic questions we hope to answer. We will also present some ideas about what a really satisfactory platform for this research and indeed for any repository of mathematical knowledge would look like. In a nutshell, it should be a distributed and collaborative knowledge repository, in some ways like Wikipedia, but in which the basic units of knowledge are not expressed in natural language but instead in formal languages. This raises many difficult problems of how to maintain formal consistency in the face of distributed and asynchronous updates at all levels, from the foundational definitions on up. We will identify some of these problems and hope to stimulate discussion of them.

Revisiting SAD

Steffen Frerix and Peter Koepke
University of Bonn, Germany
Email: `s6stfrer@uni-bonn.de` and `koepke@math.uni-bonn.de`

2nd December 2017

The *System for Automated Deduction* (SAD) by Andrei Paskevich et.al. (see [3] and `http://nevidal.org/sad.en.html`) combines natural language input with first-order proof checking. Mathematical texts are expressed in the controlled mathematical language ForTheL, parsed into a first-order based internal format, and checked for logical correctness by a "reasoner" together with some standard automated theorem prover. The following excerpt of a document accepted by SAD demonstrates that ForTheL/SAD can come close to standard mathematical language and argumentation.

Theorem The set of prime numbers is infinite.

Proof Let A be a finite set of prime numbers. Take a function p and a number r such that p lists A in r steps. $\operatorname{ran} p \subseteq \mathbb{N}^+$. $\prod_{i=1}^{r} p_i \neq 0$. Take $n = \prod_{i=1}^{r} p_i + 1$. n is nontrivial. Take a prime divisor q of n.

Let us show that q is not an element of A. Assume the contrary. Take i such that ($1 \leq i \leq r$ and $q = p_i$). p_i divides $\prod_{i=1}^{r} p_i$ (by MultProd). Then q divides 1 (by DivMin). Contradiction. qed.

Hence A is not the set of prime numbers. □

The typesetting is done by LaTeX; standard ForTheL texts allow *patterns* like `\Product{p}{1}{r}` that SAD reads as a ternary term with arguments $p, 1, r$, whereas a custom-designed LaTeX-macro outputs a pretty-printed $\prod_{i=1}^{r} p_i$.

Some "natural mathematics" features of SAD are present in the example: anonymous variables like "set of prime numbers", soft type specifications in adjectives like "prime" or "finite", user-specified linguistic and symbolic patterns like "x divides y", proof methods like induction, case splits or contradiction. SAD employs a light logical background system with efficient and mathematically well-motivated handling of premises, definitions and local theses. Moreover, an SAD verification includes ontological checking [6], resembling typechecking for typed languages, which, e.g., allows a correct treatment of partial functions.

The current SAD-system is an admirable prototype based on the doctoral project of Andrei Paskevich [3]. It was tuned to check some impressive miniatures (e.g. [5]), but in general it has severe limitations. Longer texts, like a common foundation file for sets, functions, numbers etc. cannot be checked efficiently. Therefore basic notions for sets and functions have to be reimplemented in every example text; there is only minimal and inefficient built-in support for sets and none for functions. While the input language allows formalizations close to natural language, the parser does not check for grammatical correctness and also accepts completely ungrammatical texts. Furthermore, the range of allowed variables and constants is limited, the ForTheL formats do not interact well with LaTeX.

Reimplementing basic notions each time without strong correctness checks makes the system insecure. Inconsistencies and unintended interpretations can easily be introduced accidentally. This can be especially problematic if one wishes to combine existing ForTheL texts.

SAD has not been developed further for ten years. The impressive features and examples, however, motivate us to go into the system again, identify and remedy some weaknesses, and explore possibilities for further development. In his MA project, the first author has analysed the code and studied the internal behaviour. Already simple modifications of the thesis handling and of internal reasoning methods allow the checking of much longer texts. It appears possible that a strengthened SAD is able to verify hierarchies of texts rather than isolated examples.

To write interdependent mathematical texts necessitates a common foundation. In SAD, the built-in support for sets, or rather Fregian classes, is unintuive and allows the verification of undesirable statements like “There exists a set equal to { set x | x $\notin$ x }”. It is not sufficient to write a common foundational, set-theory inspired ForTheL-text on sets, functions, numbers etc., since ForTheL does not support schemas of axioms or theorems. The ubiquity of sets and functions in mathematics really demands a special implementation, on which we are working.

The basis of SAD is first order logic. The ForTheL language requires soft, “linguistic” typings or sortings like “Let f be a function”; quantified variables have to be typed. Types are interpreted as unary predicates in the background logic. The numerous recurrences of such predicates unfortunately clutter the input of the backend ATP. Therefore the verification process should make use of some sorted logic. There have been ATP developments towards sorted first order logic: the TPTP language has added the input forms TFF0 [4] and TFF1 [1] which are now supported by powerful ATPs. We are currently working on a suitable logical setting and mechanisms that may safely and fruitfully replace unary types by sorts. The implementation of sets and functions in particular should greatly benefit from sorting.

In ForTheL formalizations, one chooses certain notions as undefined base notions depending on the level of abstraction one is working on. While this can produce elegant and impressive example texts, it makes the question of how to import or export theorems and definitions between documents difficult. A theory of relations between ForTheL texts is necessary to provide the means to build hierarchical libraries.

The naturality of ForTheL is a strong point of SAD that we want to further improve on. We are working on some desirable language extensions and LaTeX-compatible symbolic extensions. We shall also examine whether proper natural language parsers from computer linguistics should replace the ad hoc parsing used by SAD or be used as a preprocessor. This work is guided by our experiences from the Naproche project [2].

In our talk we shall survey the classical and improved SAD system, including examples, and we shall discuss some theoretical aspects. Our experiments make it conceivable that textbook mathematics like an introduction to number systems can be formulated and feasably be checked by an improved SAD. The missing resemblance between formalized mathematics and real mathematics seems to be a reason why the general mathematical community is reluctant to integrate proof assistants into their work [7]. We view our research as a contribution to the question whether formal mathematics can be made more acceptable by using ForTheL-like natural mathematical languages and SAD-like reasoning.

References

[1] Blanchette, J. C., and Paskevich, A. TFF1: The TPTP typed first-order form with rank-1 polymorphism. In *International Conference on Automated Deduction* (2013), Springer, pp. 414–420.

[2] Cramer, M., Koepke, P., Kühlwein, D., and Schröder, B. The Naproche system. *Intelligent Computer Mathematics, Springer LNCS, ISBN* (2009), 978–3.

[3] Paskevych, A. *Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques.* PhD thesis, Université Paris 12, 2007.

[4] Sutcliffe, G., Schulz, S., Claessen, K., and Baumgartner, P. The TPTP typed first-order form with arithmetic. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning* (2012), Springer, pp. 406–419.

[5] Verchinine, K., Lyaletski, A., and Paskevich, A. System for Automated Deduction (SAD): a tool for proof verification. *Automated Deduction–CADE-21* (2007), 398–403.

[6] Verchinine, K., Lyaletski, A., Paskevich, A., and Anisimov, A. On correctness of mathematical texts from a logical and practical point of view. In *International Conference on Intelligent Computer Mathematics* (2008), Springer, pp. 583–598.

[7] Wiedijk, F. The QED manifesto revisited. *Studies in Logic, Grammar and Rhetoric 10*, 23 (2007), 121–133.

Talk on Safe Reinforcement Learning via Formal Methods

Nathan Fulton and André Platzer

Carnegie Mellon University, Pittsburgh, PA, U.S.A.
{nathanfu, aplatzer}@cs.cmu.edu

Abstract

This Contributed Talk will present our recently published [3] work on applying verification technologies to safe reinforcement learning and recent extensions to this work. We show how to use formally verified hybrid systems models to precisely sandbox reinforcement learning algorithms in robotics contexts. We prove that our approach preserves safety guarantees, and demonstrate that we retain the empirical performance benefits provided by reinforcement learning. We also explore various points in the design space for these *justified speculative controllers* in a simple model of adaptive cruise control for autonomous cars. Finally, we discuss how to use verification results even when model inaccuracies are detected at run time.

The work presented in [3] and discussed in this talk provides a general approach toward provably safe learning that is amenable to extension via different learning algorithms, approaches toward formal verification, and methods for achieving safe reinforcement learning.

1 Introduction and Background

Cyber-physical systems (CPSs) are difficult to get right, which is why formal verification provides rigorous ways of establishing the safety of controllers with respect to a physical model of the system under control. KeYmaera X and other hybrid systems verification tools provide a way of obtaining safety results for cyber-physical systems [2].

Difficulties with formally verified controllers arise whenever there are discrepancies between the verified models and the real implementation. Such discrepancies between model and reality are inevitable in physical systems operating in open environments [4].

Reinforcement learning (RL) [7] provides ways of learning controllers that tend to perform well without the need for a perfect model – or even any model at all. Most approaches toward reinforcement learning provide no guarantee about the safety of the learned controller or about the safety of actions taken during learning. Absence of safety guarantees become a crippling problem when reinforcement learning is applied to safety-critical CPSs. Unfortunately, testing alone is an intractable approach toward system verification and validation.

This talk will present a technique, called *Justified Speculative Control* (JSC) [3], for transferring formal verification results to controllers obtained via reinforcement learning. We also discuss some experiments which demonstrate that formal verification results can be used to help guide a system back into modeled portions of state-space.

2 Summary of Results

Our approach toward safe learning combines hybrid systems verification, runtime monitoring, and reinforcement learning. Justified Speculative Control (JSC) extends model-based safety theorems about hybrid systems to policies obtained through reinforcement learning.

The approach begins with a hybrid system specified in Differential Dynamic Logic [5, 6] and verified in KeYmaera X [2]. The verified system has the general shape

$$init \rightarrow [\{ctrl; plant\}^*]safe$$

where *init* describes a set of initial conditions, *ctrl* is a (typically non-deterministic) discrete program describing all possible control actions, *plant* is a system of ordinary differential equations describing the physical behavior of the system, and *safe* is a description of the safe states for the system. The entire formula states that if the system starts in the set *init*, then after any arbitrary number of control actions followed by physical movement, the system will always remain within the set *safe*.

Given such a verified model, we generate runtime monitors that monitor both the controller (*CM*) and the entire model (*MM*) for deviation from the verified model. These monitors are generated using the ModelPlex algorithm implemented as a tactic in KeYmaera X [1, 4].

The JSC algorithm takes a generic reinforcement learning algorithm $\mathbb{A}$ and constrains this algorithm using our verified monitors. As long as the model monitor returns *True*, $\mathbb{A}$ may only optimize over actions for which the controller monitor *CM* returns *True*. Otherwise, when a model violation is detected, $\mathbb{A}$ is justified in optimizing over the entire state space. This talk will present some basic results about the JSC algorithm, discussed at greater length in [3]:

- When environments are accurately modeled (i.e., when the hybrid systems model accurately characterizes observed sensor inputs), formal verification results for the model from which the controller and model monitors are derived transfer to the learning process. Verification results also transfer to extracted policies.
- When model violations are detected, transforming boolean-valued model monitors *MM* to real-valued monitors provides an experimentally promising approach toward incorporating verification results into safe reinforcement learning. Intuitively, these real-valued monitors drive the learning process back into modeled state-space.

References

[1] Nathan Fulton, Stefan Mitsch, Brandon Bohrer, and Andrè Platzer. Bellerophon: Tactical theorem proving for hybrid systems. In *Interactive Theorem Proving 2017*, 2017.

[2] Nathan Fulton, Stefan Mitsch, Jan-David Quesel, Marcus Völp, and André Platzer. KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In *Conference on Automated Deduction*, 2015.

[3] Nathan Fulton and André Platzer. Safe Reinforcement Learning via Formal Methods: Toward Safe Control Through Proof and Learning. In *The Thirty Second AAAI Conference on Artificial Intelligence*, 2018.

[4] Stefan Mitsch and André Platzer. ModelPlex: Verified runtime validation of verified cyber-physical system models. *Form. Methods Syst. Des.*, 49(1):33–74, 2016. Special issue of selected papers from RV'14.

[5] André Platzer. The complete proof theory of hybrid systems. In *LICS*, pages 541–550. IEEE, 2012.

[6] André Platzer. Logics of dynamical systems. In *LICS*, pages 13–24. IEEE, 2012.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998. A Bradford Book.

Reasoning and Consciousness Teaching a Theorem Prover to let its Mind Wander

Ulrich Furbach
Universität Koblenz-Landau
uli@uni-koblenz.de

Claudia Schon*
Universität Koblenz-Landau
schon@uni-koblenz.de

Automated Reasoning in commonsense contexts is challenged by the need of taking into account very large knowledge sources. The authors addressed this aspect in various previous papers on the topic of natural language query answering ([3, 4]) where an engineering approach is taken, putting together what is available and what looks helpful. In this contribution we are discussing instead of this ad hoc methods the use of a framework from psychology.

During an ongoing series of workshops around the topic *Bridging the Gap between Human and Automated Reasoning* ([6]) we are aiming at learning from human reasoning on how to reason efficiently even when large and different knowledge sources are necessary. This ability of humans seems to be strongly related to consciousness. The problem of consciousness and in particular the question whether an AI system can be conscious is discussed in depth in philosophy, neuroscience or psychology. There is one very well accepted theory about human consciousness, namely the *Global Workspace Theory* introduced by Bernhard J. Baars ([1]). Baars' motivation for his theory are the following observation in human reasoning and behavior: The human brain 'suffers' from a kind of limited capacity, e.g. immediate memory, selectivity of attention and the observation that we rarely can do two demanding actions at a time. The situation, however, is different, when we observe the brain directly: a huge neural network, a lot of layers and connections and various parts that are specialized to different tasks, e.g. recognition of visual information, controlling body functions or language recognition and generation. All this is highly parallel and most of the time unconscious. One, at least for our community astonishing capability is, the use of the long-term memory. We do not know its size, but as Baars is pointing out, if we pay attention to 10,000 different pictures during several days, we can recognize each of them without attempting to memorize them. Memory search performed by the brain is highly efficient and it looks like we have a huge domain of knowledge, that is unconscious and we get access to it by consciousness. And this is exactly what Baars' Global Workspace Theory is aiming to model – Consciousness is the key to reason efficiently and goal oriented! Following the attempt by Baars and his co-workers, the question whether AI systems can be conscious has to be rewritten: Can we allow ourselves to design AI systems without consciousness? In the following we sketch the Global Workspace Theory (GWT) and we try to relate it to an automated reasoning context. We furthermore present an approach on how to let the theorem prover Hyper [2] 'let its mind wander', which is inspired by GWT.

Global Workspace Theory GWT is usually explained by describing it as a *Working Theatre of Consciousness*. We can think about the brain a theatre consisting of a stage, an attentional spotlight shining at the stage, actors which represent the contents, an audience and some people behind the scene. Lets look at the parts in more detail and relate it to automated reasoning aspects.

The stage. The working memory consist of verbal and imagined items. Most parts of the working memory are in the dark, but there are a few active items, usually the short time memory. – We consider the clause set which is the input to a reasoning system together with the derived clauses to be the stage.

The spotlight of attention. This bright spotlight helps in guiding and navigating through the working memory. Humans can shift it at will, by imagining things or events. This is the part of the clause set which is currently processed by the theorem prover.

The actors correspond to the application of inference rules on the set of clauses currently processed by the theorem provers. The result of the actors' actions correspond to new formulae derived by an inference step.

*Work supported by DFG SCHO 1789/1-1 'CORG'.

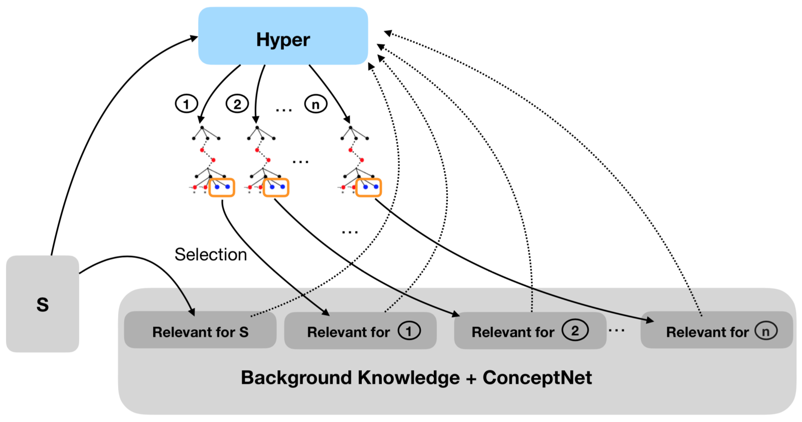


Figure 1: Hyper's mind wandering.

Context behind the scene. Behind the scenes, the director coordinates the show and stage designers and make-up artists prepare the next scenes. – We consider the reasoner and its control as a director.

The audience. According to Baars, the audience represents the vast collection of specialized knowledge. It can be considered as a kind of long-term memory and consists of specialized properties, which are unconscious. Navigation through this part of the knowledge is done mostly unconscious. – This is the background knowledge, like Cyc, Yago or knowledge graphs.

Let Hyper's mind wander Proof tasks in the area of commonsense reasoning usually are different from classical proof tasks. Due to incomplete background knowledge, one cannot expect a theorem prover to find a proof for a certain problem. In this area, the task is rather to start from an initial problem description and to perform as many inferences as possible in order to approach a goal. This is why computed models are of special interest in this area. The Hyper theorem prover is a hypertableau based theorem prover which is able to construct a model for satisfiable clause sets.

The Working Theatre of Consciousness metaphor can be used to give Hyper the ability to let its mind wander, similar to the way humans move their attention. Fig. 1 provides an overview of the interplay between Hyper, background knowledge and selection mechanisms that is used to accomplish this goal. Starting with an initial clause set S, background knowledge appropriate for this clause set is selected (for example by SInE) and fed into the theorem prover (indicated by the dashed line) together with the clause set. Hyper constructs a model for this input which is indicated by number 1 in Fig. 1. The open branches of this tableau correspond to new knowledge derived by Hyper. We use this new knowledge to select new clauses from the large background knowledge. In order to allow for some creative variation, we include ConceptNet [5] into this step by looking up some entities which are related to some of the predicate symbols in this set and selecting background knowledge for these entities as well. In addition to that, we bridge vocabularies as described in [4]. This new background knowledge is then fed into Hyper together with the freshly inferred knowledge. Hyper constructs a second model (number 2 in Fig. 1), which again contains open branches. The new knowledge in these open branches is again used to select appropriate background knowledge. This process is repeated as long as desired. During the whole process, Hyper is provided with different sets of clauses. Each of them representing the current focus of Hyper's mind. The process of searching for new background knowledge can be seen as a shift of his mind.

This proposal presents first ideas about incorporating aspects from consciousness research into automated reasoning. We are currently setting up the infrastructure for extensive experiments. Results will be available until the workshop.

References

[1] B. J. Baars. In the Theatre of Consciousness. Global Workspace Theory, A Rigorous Scientific Theory of Consciousness. *Journal of Consciousness Studies*, 4(4):292–309, 1997.

[2] M. Bender, B. Pelzer, and C. Schon. System description: E-KRHyper 1.4 - extensions for unique names and description logic. In M. P. Bonacina, editor, *CADE-24*, LNCS, 2013.

[3] U. Furbach, I. Glöckner, and B. Pelzer. An application of automated reasoning in natural language question answering. *AI Commun.*, 23(2-3):241–265, 2010.

[4] U. Furbach and C. Schon. Commonsense reasoning meets theorem proving. In *MATES*, volume 9872 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2016.

[5] H. Liu and P. Singh. ConceptNet — a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, Oct. 2004.

[6] C. Schon and U. Furbach, editors. *Proceedings of the Workshop on Bridging the Gap between Human and Automated Reasoning co-located with 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York, USA, July 9, 2016*, volume 1651 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

TacticToe: Learning to Prove with Tactics

Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish

[1] University of Innsbruck
[2] Czech Technical University
[3] DeepMind
[4] Data61

Abstract. The talk will discuss the tactical prover TacticToe implemented on top of the HOL4 interactive theorem prover. TacticToe learns from human proofs which mathematical method is useful in a particular proof situation. This knowledge is then used in a Monte Carlo tree search algorithm that explores the most promising tactic-level proof paths. On a single CPU, with a time limit of 60 seconds, TacticToe proves 66.4% of 7164 theorems in HOL4's standard library whereas Eprover solves 34.5%. The success rate rises to 69.0% by combining the results of TacticToe and Eprover.

First Experiments with Watchlist Guidance on Mizar[*]

Zarathustra Goertzel, Jan Jakubův, and Josef Urban

Czech Technical University, Prague,

Using Watchlists for Guiding ATPs on Mizar

The E automated theorem prover [8] has recently gained (thanks to Stephan Schulz) the ability to guide the selection of the given clauses by using the *watchlist* mechanism (also called *hints* [10] in Prover9 [5]). This mechanism allows E to construct watchlists from lemmas participating in the proofs of related problems to guide clause selection. The talk will discuss several experiments with using this mechanism over the Mizar40/MPTP dataset [9]. Watchlists have proved essential in the AIM project [4] done with Prover9 for obtaining very long and advanced proofs. Each inferred clause C is checked for subsumption of the watchlist clauses. If C subsumes a watchlist clause W, then C is given higher priority, and W is removed from the watchlist. One way to think of the watchlist of high-frequency clauses is as a toolkit of mathematical tricks.

We experimented with mining watchlist clauses from 24702 proofs found by E on a benchmark of 57897 Mizar40 [2] problems.[1] The proofs were found by an evolutionarily optimized [1] ensemble of 32 E strategies (our *baseline*), each run for 5 s. Each strategy is specified as a frequency-weighted combination of parametrized *clause evaluation functions* (CEF) combined with a selection of inference rules. Below we show a simplified example strategy specifying the term ordering *KBO*, and combining (with weights 2 and 4) two CEFs made up of weight functions *Clauseweight* and *FIFOWeight* and priority functions *DeferSOS* and *PreferProcessed*.

```
-tKBO -H(2*Clauseweight(DeferSoS,20,9999,4),4*FIFOWeight(PreferProcessed))
```

Watchlist Selection

In AIM, using all previous proof lemmas as a watchlist is often a method that works. Problems in large ITP libraries such as Mizar/MML [6] however differ much more than the AIM problems, making it more likely for unrelated watchlist lemmas to mislead the proof search. We have experimented with the following methods for watchlist creation:

1. Initially, all 100,000+ clauses were used. This slows E down to 6 given clauses per second.
2. Watchlists were constructed on a per Mizar article basis. The size ranges from 0 to 4000.
3. We also order proof clauses by frequency and test watchlist sizes using one watchlist for a strategy or on an article basis.
4. Last, k-NN learning is used to suggest useful clauses based on symbol and term-based features [3], that is symbols, walks of length 2 on formula trees and common subterms (with variables and skolem symbols equalized).

Using Watchlist in E Strategies

Watchlist subsumption defines a particular priority function [8] called `PreferWatchlist`, assigning weights to clauses. We test several ways how to use this priority function:

1. E prover has a default strategy evolved (genetically [7]) for watchlist use. (*EVO*)

[*]Supported by the ERC Consolidator grant no. 649043 AI4REASON.

[1]Precisely, we have used the small (*bushy*, re-proving) versions, but without ATP minimization.

2. Add a CEF based on `PreferWatchlist` with high frequency to the 32 strategies. (uwl_n)
   `-H(n*6*Defaultweight(PreferWatchlist),2*Clauseweight(DeferSoS,20,9999,4),...)`
3. Instead of Defaultweight in uwl_1, use the CEFs used in EVO. (uwl_evo)
4. Replace all priority functions in a strategy with `PreferWatchlist`. (pref)
   `-H(2*Clauseweight(PreferWatchlist,20,9999,4),4*FIFOWeight(PreferWatchlist))`
5. Modify E to always prefer watchlist clauses and default to the given strategy. (uwl)
6. For all above strategies we add a "no-remove" option to keep the subsumed watchlist clause in the watchlist, thus allowing its repeated subsumption with different clauses.

Watchlist Performance

Strategy	baseline	EVO	uwl_evo	pref	uwl
02	14223	17419	16485	**17822**	14762
08	14498	16790	16286	15472	**15089**
09	11917	13758	13327	**13852**	12382
26	12504	**16478**	14807	14006	13056
28	12803	14580	14290	13115	**12069**
total	21122	21948	22147	22477	21617

Size	pref02	pref28
10	3275	2410
100	3275	2279
256	3287	2211
512	3283	2180
1000	3248	2211
10000	2912	2212

Table 1: Left: results on the Mizar40 dataset (57897 problems) using per-article watchlists. The 5 greedily best strategies (in bold) cover 22725 (7.6% more) problems (we call this *Greedy1*). Right: Tests of the watchlist size influence (ordered by frequency) on a random sample of 10000 problems using the "no-remove" option. Pref28 uses per-article watchlists, and pref02 uses one common watchlist.

For testing, we use five greedily best E strategies covering 80% (21122) of the 24702 proofs. Table 1 shows the first results. The watchlists are always constructed from the proofs found by the baseline strategy. The 5 *pref* strategies prove together 1503 problems that the corresponding 5 baseline strategies don't, and 514 problems on top of the 24702 found by all 32 baseline strategies. The good performance of single watchlist strategies may however be also due to memorization of the baseline proofs. For a fairer comparison of the individual strategies, we create a random test set of 2000 problems and only construct watchlists from the proofs found by baseline strategies on the remaining training set of 55897 problems, see Table 2.

baseline	pref_baseline	uwl	pref	greedy1	greedy2
726	733	728	745	732	755
gain:	0.9%	0.2%	2.6%	0.8%	4%

Table 2: Test performance of the 5 strategies and their gain over the baseline. *Pref_baseline* is pref run with empty watchlist. *Greedy1* is the 5-cover found in Table 1, and *greedy2* is the new greedy 5-cover.

The individual performance becomes more realistic: *greedy1* falls from 7.6% to under 1%. *Pref* is 2.6% better than the baseline, and 1.6% better than itself without any watchlist. The *Greedy2* cover includes *pref02*, *pref_baseline09*, *baseline26*, *EVO26*, and *baseline28* strategy. Surprisingly, *pref_baseline* performs better than the *baseline*. This means that the weight functions in CEFs often perform better without the priority function input.

Altogether, the watchlist feature helps E prove at least 4% more of the Mizar40 problems than the baseline ensemble. Inspection of some of the watchlist-based proofs shows that some are completely new, extending the set of 32524 ATP proofs found with high time limits in [2]. An example is `BCIALG_4:44`[2] where a nontrivial 30-line Mizar proof is obtained by E using 23 axioms, 200 proof steps and 7067 given clause loops. Experiments suggest the watchlist can both guide and distract E, so a schedule may include both watchlist and no-watchlist runs. Consecutive runs also seem feasible. Frequency sorted smaller watchlists appear to significantly help. More work has to be done on the best way to learn from prior proofs with the watchlist.

[2]`http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/bcialg_4.html#T44`

References

[1] 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, January 16-17, 2017*, pages 43–52. ACM, 2017.

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

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *IJCAI'15*, pages 3084–3090. AAAI Press, 2015.

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

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

[6] The Mizar Mathematical Library. http://mizar.org/.

[7] Simon Schäfer and Stephan Schulz. Breeding theorem proving heuristics with genetic algorithms. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 263–274. EasyChair, 2015.

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

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

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

Guiding SMT Solvers with Monte Carlo Tree Search and Neural Networks

Stéphane Graham-Lengrand[1], Michael Färber[2]

[1] CNRS - École Polytechnique, 91120 Palaiseau, France
graham-lengrand@lix.polytechnique.fr
[2] Universität Innsbruck, 6020 Innsbruck, Austria
michael.faerber@gedenkt.at

Abstract

Monte Carlo Tree Search (MCTS) is a technique to guide search in a large decision space by taking random samples and evaluating their outcome. Frequently, MCTS is employed together with reward heuristics learnt by neural networks. The talk will propose a guidance mechanism for SMT solvers based on a combination of MCTS and neural networks.

Machine learning methods gain importance in automated reasoning. A particularly strong trend are neural networks, having produced state-of-the-art results for premise selection [WTWD17, ISA$^+$16]. Outside of automated reasoning, neural networks have been combined with Monte Carlo Tree Search, treating problems as diverse as finding good strategies to play the game of Go [SHM$^+$16] and planning of chemical syntheses [SKTW17, SPW17]. In automated reasoning, Monte Carlo Tree Search (MCTS) has been applied to first-order automated theorem proving, using hand-crafted heuristics instead of neural networks [FKU17]. We propose a combination of Monte Carlo Tree Search and neural networks to guide the search performed by an SMT solver.

We are exploring the idea of such guidance in the *Psyche* platform [GL13], which offers a modular architecture for theorem proving. It implements an adaptation, to automated reasoning in general and to SMT solving in particular, of the LCF architecture [Mil79, GMW79].

LCF is mostly used in Interactive Theorem Proving and is particualrly widely implemented in the proof assistants of the HOL family, such as the HOL system [HOL], Isabelle [Isa], etc. The LCF architecture allows theorem proving strategies to be programmable, while guaranteeing the correctness of any claim that a formula is provable. The architecture's *kernel* component offers an API whose primitives implement basic reasoning inferences, and strategies can be programmed on top of the kernel via the API. The claims of provability are then necessarily *correct-by-construction*, assuming the correctness of the kernel, but regardless of any potential defects in the design or in the implementation of strategies (or in the user's input, for the case of Interactive Theorem Proving).

Psyche embraces this paradigm and, in the case of SMT solving, relates to a position paper by de Moura and Passmore, entitled "The strategy challenge in SMT solving" [dMP13], which promoted the programmability of strategies as compositions of basic reasoning tasks, explicitly referring to the LCF paradigm. This approach opens up the possibilities of extensively experimenting with various strategies, whether they be handcrafted or machine-learned, while never jeopardising the correctness of the solver's output. Guiding the search by techniques such as MCTS and neural networks can be envisaged more easily in provers whose architecture implements this approach. *Psyche*'s architecture does so at a rather fine-grained level, separating the code that implements reasoning inferences from the code that implements search strategies. More precisely, the CDSAT branch of *Psyche* [CDS] implements the *Conflict-Driven Satisfiability* framework [BGS17, BGLS18], which lifts from Boolean logic to generic theory combination the conflict-driven clause learning (CDCL) algorithm used in pure SAT-solving:

Given an input SMT problem, the search space explored by *Psyche*/CDSAT consists of *states* that describe specifications for a desired model of the input problem. The *moves* or *actions* that can be made from such a state consist of assigning a value to a term or a literal, thereby specifying the model further, until the existence or non-existence of a model satisfying those specifications is manifest. In the former case, the input problem is concluded to be SAT. The latter case represents a *conflict*, which is analysed so that a lemma can be learnt explaining the reason for the conflict. Some of the assignments are reverted so that another area of the search space can be explored, taking into account the learnt lemmas. If and when these lemmas conclude that no model will be found in the entire search space, the problem is concluded to be UNSAT. *Psyche*/CDSAT is modular in the collection of agents that contribute background knowledge about different theories such as propositional logic and linear arithmetic. These agents offer for each state a range of possible moves, e.g. assigning a truth value to a literal or a rational value to a rational variable, and apply theory-specific inference rules in order to compute consequences of such assignments and detect conflicts.

We propose to apply MCTS guidance for applying moves, which requires a transition probability heuristic for the moves available from a state, and a reward heuristic for states. The former quickly orients the search towards the next states to look at, while the latter, possibly more costly to compute but called less often, contributes to maintaining and updating reward scores for states. These scores are then used to determine whether the search should explore more deeply an area of the search space or whether it should jump to another area. A specificity of satisfiabilty solving, when expressed as a tree-search problem, is that there are two kinds of conclusions, namely SAT and UNSAT, which may impact what the MCTS heuristics try to achieve, particularly with respect to the exploitation/exploration balance of an MCTS search.

We are investigating the use, for transition probabilities, of existing theory-agnostic heuristics for choosing assignments, usually based on the *activity score* of terms and literals. Those that have often participated to recent conflicts have a high activity [MMZ$^+$01] and will be picked with higher probability. This encourages exploitation, triggering the use of recently used lemmas and possibly combining them into a proof of UNSAT.

We propose on the other hand to use, for the reward heuristic, an estimation of proximity between the state to be evaluated and a SAT state / model. This estimation lends itself to being learnt by a neural network, trained on previously completed runs. We propose to trigger this evaluation for conflict states, which comprise a trail of assignments, the lemma it generates, and previous lemmas present at the time of conflict. All of these need to be embedded to a feature vector that is tractable by a neural network, using similar methods as [WTWD17] or [JU17]. Training data for the neural network can be generated by feeding actual SAT states to it, labelling them with maximal reward, as well as feeding it conflict states, labelling them e.g. with the Hamming distance between the conflict state and the actual SAT state.

One of the reasons why we believe that *Psyche* lends itself to this approach is that the basic inferences and the search space are well-identified. Moreover, the prover's states are persistent data-structures, inherited from the functional programming nature of the LCF approach, which should simplify the recording of the states' rewards and allow quick state switches during exploration.

At the moment, two components of the proposed approach have been integrated to *Psyche*/CDSAT, which is written in OCaml. First, the OCaml code for MCTS, which was originally developed for connection tableaux [FKU17], but which is sufficiently modular to be applicable to other tree search problems. Second, the OCaml bindings for TensorFlow, which can train and apply a neural net directly in *Psyche*. What is left to do before evaluating the approach with benchmarks is to encode the feature extraction and organise the training on a suitable set of examples.

References

[BGLS18] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. Proofs in conflict-driven theory combination. In June Andronick and Amy Felty, editors, *Proc. of the 7th Int. Conf. on Certified Programs and Proofs (CPP'18)*. ACM Press, January 2018.

[BGS17] Maria Paola Bonacina, Stéphane Graham-Lengrand, and Natarajan Shankar. Satisfiability modulo theories and assignments. In de Moura [dM17], pages 42–59.

[CDS] The CDSAT system. Available at https://github.com/disteph/cdsat.

[dM17] Leonardo de Moura, editor. *CADE-26*, volume 10395 of *LNCS*. Springer, 2017.

[dMP13] Leonardo Mendonça de Moura and Grant Olney Passmore. The strategy challenge in SMT solving. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788, pages 15–44, 2013.

[FKU17] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte Carlo tableau proof search. In de Moura [dM17], pages 563–579.

[GL13] Stéphane Graham-Lengrand. Psyche: a proof-search engine based on sequent calculus with an LCF-style architecture. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Proc. of the 22nd Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux'13)*, volume 8123 of *LNCS*, pages 149–156. Springer-Verlag, September 2013.

[GMW79] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation*, volume 78. 1979.

[HOL] The HOL system.

[Isa] The Isabelle theorem prover.

[ISA$^+$16] 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, *NIPS*, pages 2235–2243, 2016.

[JU17] 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, *CICM*, volume 10383 of *LNCS*, pages 292–302. Springer, 2017.

[Mil79] Robin Milner. LCF: A way of doing proofs with a machine. In Jirí Becvár, editor, *Proc. of the the 8th Int. Symp. on Mathematical Foundations of Computer Science*, volume 74 of *LNCS*, pages 146–159. Springer-Verlag, 1979.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, pages 530–535, New York, NY, USA, 2001.

[SHM+16] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[SKTW17] Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *CoRR*, abs/1701.01329, 2017.

[SPW17] Marwin H. S. Segler, Mike Preuss, and Mark P. Waller. Learning to plan chemical syntheses. *CoRR*, abs/1708.04202, 2017.

[WTWD17] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2783–2793, 2017.

LET'S MAKE SET THEORY GREAT AGAIN!

John Harrison

Amazon

Abstract. Although set theory has long been regarded as the "standard" foundation for mathematics, it's somewhat underrepresented in current activities in the formalization of mathematics. I'll be discussing how this situation came about and then suggest that it's time to take another look at set theory as a foundation. I'll talk a bit about how set theory, particularly with a few natural conventions, can give quite a clean and direct formalization of much of elementary mathematics, in particular aspects that are difficult or ugly in type theory like subtypes and notions of "undefined". On the other hand I'll also discuss some of the possible problems. The link to AITP topics is tenuous but real: I believe that set theoretic foundations can have a decluttering effect that makes the correspondence with everyday mathematics more direct and hence more susceptible to learning.

Automation by Analogy, in Coq

Alasdair Hill[1]*and Ekaterina Komendantskaya[2]

[1] Heriot-Watt University Edinburgh, UK
ath7@hw.ac.uk
[2] Heriot-Watt University Edinburgh, UK
ek19@hw.ac.uk

Abstract

Automation of popular interactive theorem provers like Coq has become a hot topic, due to the growing number and size of verification projects such provers accommodate. Several automation tools for Coq have been suggested, from advanced tactics like *Crush*, to SMT-solving solutions like *SMT-Coq* and *Hammer for Coq*. In this talk, we will present a complementary set of automation methods for Coq, based on discovery of proof similarities and common proof patterns in the code.

Extended Abstract

The problem of automating (or aiding) Coq proof construction is as old as Coq itself. *Ltac* tactic language is by a wide margin the most popular Coq automation tool to date. Over the years it hosted a range of impressive extensions like e.g. SSReflect [6] and Crush [3]. Neither of these extensions is "AI based", i.e. neither uses automated reasoning or machine learning. Seeing the succes of Isabelle/HOL in AI based automation [2], it is not unreasonable to predict that incorportaion of some kinds of AI based tools in Coq may aid to further automate some aspects of proof development. The main two questions are: (1) what kinds of AI tools? and (2) which aspects of proof development? Very often, the answer to question (1) determines the answer to question (2).

For example, one answer to question (1) is to encorporate powerful SMT solvers into Coq, thus aiming for automation of Coq proofs that correspond to the first-order theory of the underlying SMT solver. For the phase of translation of proofs from the language of the SMT-solver back to Coq, two engineering solutions are possible. Hammers for Coq [4] approach suggests to use the "Hammer" methods [2] also employed in Isabelle and HOL, i.e. to reconstruct the Ltac tactics from SMT proof traces. SMT-Coq [5] approach uses the small scale reflection to *reflect* the proofs generated by the SMT-solver back into Coq's language.

In this talk, we will propose an alternative answer to the questions (1) and (2). We propose to use a method of statistical pattern-recognition to detect structural similarities among Coq proofs and definitions. It has been implemented in ML4PG (*Machine-Learning for Proof General*) [10, 8]. ML4PG performs a structural analysis of all Coq objects in the given libraries, and discovers their mutual dependencies and similarities. Based on the discovered patterns, it outputs small sets (clusters) of similar proofs.

If a theorem of interest belongs to a certain cluster, other lemmas and theorems in that cluster are deemed to be structurally similar to it, and we can try to reconstruct an Ltac proof script for a new theorem by analogy with the Ltac scripts of similar proofs in the cluster. Unlike the SMT-based tools, this method will not be restricted to first-order fragment of Coq proofs, and it will work similarly for SSReflect or plain Coq proofs. But this method will be limited by

*A.Hill is funded by an EPSRC DTA grant.

the power of the **analogical argument**. For example, a new theorem may not be provable by analogy with any other existing theorem, or the analogy may run deeper than any Ltac tactic combination we may generate. The recent preliminary results [11] showed a big variation in success of the analogical method, depending on the libraries, ranging from 94% in HoTT Path library [1], and dropping to 36% in the standard SSReflect library.

In this talk, we will give a detailed experimental study of the power and limitations of the analogical proof reconstruction in Coq setting. We will show four new prototype tools that explore the analogies arising from structural similarities of proofs in four different ways, some involving heuristics such as the automata generation of SEPIA [7]. We compare the performance of these four new analogical methods on SSReflect, CompCert [12], and CoqHoTT libraries.

A similar study of proof automation by analogy has been done in ACL2 [9].

References

[1] S. Awodey, T. Coquand, V. Voevodsky, et al. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013. https://github.com/HoTT/HoTT/wiki.

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

[3] Adam Chlipala. *Certified Programming with Dependent Types*. MIT Press, 2011.

[4] Lukasz Czajka and Cezary Kaliszyk. Goal translation for a hammer for coq (extended abstract). In *Proceedings First International Workshop on Hammers for Type Theories, HaTT@IJCAR 2016, Coimbra, Portugal, July 1, 2016.*, volume 210 of *EPTCS*, pages 13–20, 2016.

[5] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark W. Barrett. Smtcoq: A plug-in for integrating SMT solvers into coq. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017.

[6] G. Gonthier and A. Mahboubi. An introduction to small scale reflection. *Journal of Formalized Reasoning*, 3(2):95–152, 2010.

[7] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: search for proofs using inferred automata. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 246–255. Springer, 2015.

[8] J. Heras and E. Komendantskaya. Recycling Proof Patterns in Coq: Case Studies. *Journal Mathematics in Computer Science*, 2014.

[9] Jónathan Heras, Ekaterina Komendantskaya, Moa Johansson, and Ewen Maclean. Proof-pattern recognition and lemma discovery in ACL2. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 389–406. Springer, 2013.

[10] E. Komendantskaya et al. Machine Learning for Proof General: interfacing interfaces. *Electronic Proceedings in Theoretical Computer Science*, 118:15–41, 2013.

[11] Ekaterina Komendantskaya and Jónathan Heras. Proof mining with dependent types. In *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 303–318. Springer, 2017.

[12] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

Enhancing ENIGMA Given Clause Guidance [*]

Jan Jakubův and Josef Urban

Czech Technical University in Prague, Prague, Czech Republic

1 ENIGMA: Efficient Learning of Given Clause Guidance

State-of-the-art saturation-based automated theorem provers (ATPs) for first-order logic (FOL), such as E [9], are today's most advanced tools for general reasoning across a variety of mathematical and scientific domains. Many ATPs employ the *given clause algorithm*, translating the input FOL problem $T \cup \{\neg C\}$ into a refutationally equivalent set of clauses. The search for a contradiction is performed maintaining sets of *processed* (P) and *unprocessed* (U) clauses. The algorithm repeatedly selects a *given clause* g from U, extends U with all clauses inferred with g and P, and moves g to P. This process continues until a contradiction is found, U becomes empty, or a resource limit is reached. The search space of this loop grows quickly and it is a well-known fact that the selection of the right given clause is crucial for success.

ENIGMA [5] is an *efficient* learning-based method for guiding clause selection in saturation-based ATPs. ENIGMA is based on a simple but fast *logistic regression* algorithm effectively implemented by the LIBLINEAR open source library [4]. In order to employ logistic regression, first-order clauses need to be translated to fixed-length numeric *feature vectors*. ENIGMA uses (top-down-)oriented term-tree walks of length 3 as *features*. For example, a unit clause "$P(f(a,b))$" contains only features "(P,f,a)" and "(P,f,b)" (see [5, Sec. 3.2] for details). Features are numbered and a clause C is translated to the feature vector Φ_C whose i-th member counts the number of occurrences of the i-th feature in clause C. In practice, we also count top-level literal symbols (positive or negative) and we unify variables and Skolem symbols

In order to train an ENIGMA *predictor*, all the given clauses from a set of previous successful proof searches are collected. The given clauses used in the proofs are classified as positive and the remaining given clauses as negative. The clauses are turned into feature vectors and a LIBLINEAR classifier is trained based on this classification, classifying a clause as *useful* or *un-useful* for the proof search. The predictor is used to guide next proof searches in combination with other E heuristics. Thanks to the fast feature extraction mechanism and the fast (linear) evaluation of the features in a particular learned model, there is no slowdown of the given clause loop. In fact, ENIGMA is faster than some of the more advanced hand-programmed E evaluation heuristics [6]. The training speed allows fast MaLARea-style [12] iterative loop between ATP proving and re-learning [5, Sec. 5.1].

2 Enhancements

The talk will present several ENIGMA improvements and experiments. First, ENIGMA was previously tested only on the CASC 2016 AIM benchmark [11] which contains only about 10 different symbols. Using a dense encoding yields feature vectors (and corresponding ENIGMA models) of size 10^3. Such exhaustive enumeration of feature vectors gets too big with larger symbol signatures. It however turns out that the term-tree walks features are relatively sparse: symbols are typically applied only to a small number of other symbols. Hence we switch to a sparse encoding, using only the features which actually appear in the training data. This

[*]Supported by the ERC Consolidator grant no. 649043 *AI4REASON*.

AIM	train accuracy		10-fold cross-val		MZR	train accuracy		10-fold cross-val	
data	noconj	conj	noconj	conj	data	noconj	conj	noconj	conj
simple	84.7	84.6	84.6	84.5	*simple*	92.2	95.0	90.8	93.9
50-50	76.3	78.0	76.3	77.8	*50-50*	89.2	91.9	88.8	91.5

Table 1: ENIGMA training and 10-fold cross-validation accuracies (MZR and AIM) (Sec. 3).

significantly reduces the feature vector sizes while preserving the predicting accuracy. This alone allows us to test ENIGMA on ITP-based benchmarks. An even larger step in this direction is the use of fast dimension-reduction methods. So far we have efficiently built SVD into E and are planning to extend this to state-of-the-art fast SVD-based methods that have recently shown very good performance [8, 2] compared to (slower) neural embeddings.

Second, for AIM we initially did not consider conjectures to be a part of the model. Hence the same clauses were being recommended in every possible AIM proof search. This can work (similarly to the *hint* method [13]) on benchmarks of very related problems (such as AIM), but hardly on large ITP-based formal libraries, which are much more heterogeneous. To overcome this, we embed the conjecture features in the feature vectors. For a clause C, instead of using the vector Φ_C of length n, we use a vector (Φ_C, Φ_G) of length $2n$ where Φ_G contains the features of the conjecture G. For a training clause C, G corresponds to the conjecture of the proof search where C was selected as a given clause. When classifying a clause C during the proof search, G corresponds to the conjecture currently being proved.

3 Experimental Evaluation

Standard ENIGMA predictors were previously evaluated with a single E strategy on the CASC 2016 AIM benchmark [11]. This was extended to 11 good E strategies [7] and we additionally evaluate on the CASC 2012 MZR benchmark [10] based on the Mizar MPTP2078 [1] dataset. Both of these benchmarks provide around 1000 training problems. The MZR problems contain altogether around 500 different symbols (compared to 10 in AIM), which are used consistently across different problems. This is crucial for the current symbol-based ENIGMA.[1]

The original (non-conjecture - *noconj*) and conjecture-enhanced (*conj*) predictor accuracies are presented in Table 1 both for *simple* (unbalanced) data and for the *50-50* data with the positive examples boosted to 50%. The 11 E prover strategies are used to generate the training data and to build 11 different predictors. We measure the *training accuracy* where the predictor is tested on the training data, and the standard *10-fold cross-validation* accuracy, where the data are divided into 10 subsets (*folds*) with 9 folds used for training and one fold left aside for evaluation. The results are averaged across the 11 E strategies.

The differences between the training and 10-fold cross-validation accuracies are minimal. This shows that the relatively simple learner is *not overfitting*. As expected, adding the conjecture features helps on the MZR data and less on AIM. This is likely because the AIM problems have more similar conjectures. The training data contain hundreds of thousands clauses (around 110000 on average) and the training times are in seconds (from 5 to 20 on Intel 2.30GHz). The feature vector sizes vary from 60 to 130 on AIM and from 2300 to 22000 on MZR. A proper ATP evaluation on MZR is however still future work. It seems that the learned classifiers still underperform on positive examples. A promising approach is *adaptive boosting* when we iteratively build a predictor and boost misclassified positive samples.

[1]However, this is already less crucial with the SVD-based embeddings, which generalize over the features [3].

References

[1] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.

[2] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[3] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *JASIS*, 41(6):391–407, 1990.

[4] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[5] Jan Jakubův and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In *CICM*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[6] Jan Jakubuv and Josef Urban. Extending E prover with similarity based clause selection strategies. In Michael Kohlhase, Moa Johansson, Bruce R. Miller, Leonardo de Moura, and Frank Wm. Tompa, editors, *Intelligent Computer Mathematics - 9th International Conference, CICM 2016, Bialystok, Poland, July 25-29, 2016, Proceedings*, volume 9791 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2016.

[7] 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, January 16-17, 2017*, pages 43–52. ACM, 2017.

[8] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. Representing sentences as low-rank subspaces. *CoRR*, abs/1704.05358, 2017.

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

[10] G. Sutcliffe. The 6th IJCAR Automated Theorem Proving System Competition - CASC-J6. *AI Communications*, 26(2):211–223, 2013.

[11] Geoff Sutcliffe. The 8th IJCAR automated theorem proving system competition - CASC-J8. *AI Commun.*, 29(5):607–619, 2016.

[12] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.

[13] Robert Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *Journal of Automated Reasoning*, 16(3):223–239, 1996.

Towards Machine Learning for Quantification

Mikoláš Janota

INESC-ID, Lisboa

Abstract. The talk will introduce and discuss QBF (Quantified Boolean Formulas) solving and some machine learning methods for solving QBF problems. We will also discuss their extensions to the EPR class and finite model finding.

Project Description: Reinforcement Learning for leanCoP

Cezary Kaliszyk, Henryk Michalewski, Piotr Miłoś, Mirek Olšák, and Josef Urban

1 Introduction: Guiding leanCoP

The talk will describe our initial work on setting up reinforcement learning (RL) experiments for guiding the leanCoP prover. leanCoP [OB03] is an automated theorem prover (ATP) implementing connected tableau search with iterative deepening, written very economically in Prolog by Otten. Given the very compact implementation, leanCoP's performance is surprisingly high, regularly outperforming much larger ATPs such as Metis and even Prover9 in the CASC competition and in particular on problems coming from large formal libraries [KUV15a].

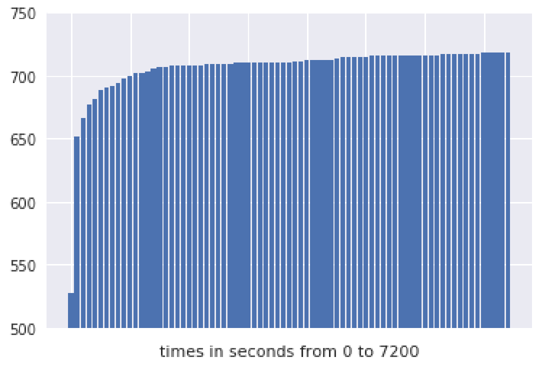
This has already led to several experiments with using machine learning (ML) methods for guiding leanCoP's proof search. In particular, MaLeCoP [UVv11] and FEMaLeCoP [KU15a] are guiding the choice of the extension clause/step by a naive Bayes classifier trained on a large number of pairs of previous proof states and proof steps. The states (current path) and steps (clauses) are characterized using custom term-based features [KUV15b]. First experiments have also been done with Monte Carlo guidance [FKU17] of leanCoP. Here we continue this work in several ways:

- Use of state-of-the-art ML methods. We want to use more advanced ML methods such as gradient boosted trees (e.g. XGBoost [CG16]) and deep neural networks.
- We export leanCoP as a Python interface usable with more advanced ML/RL methods and tools.
- We do first experiments with the more advanced ML methods using this interface.

2 Exporting leanCoP as a Python Interface

We start with leanCoP's OCaml reimplementation [KUV15a], extended by the feature extraction and advising mechanisms used in FEMaLeCoP. Part of the OCaml code is exported in C, and compiled as a shared library loadable from Python. The exported functions include `cop_start(filename)` and `cop_action(length, array)`. `cop_start` takes a problem, clausifies it, initializes the prover, and returns the initial state with a list of possible actions (clauses). `cop_action` takes a re-ordering (ranking) of the actions (typically provided by the trained classifier), applies the best to the current proof state, and returns the next proof state and its possible actions. The states and possible clauses can be printed in different formats, either as strings, or as lists of standard symbol and term-based features, or as lists of integers encoding the prefix notation introduced in [KCS17] and used by several neural approaches. If a proof is found, it can be printed as a sequence of positive and negative examples. The positive examples are pairs of `(proof_state,clause)` that are steps in the final proof. Negative examples are obtained as alternatives `(proof_state,clause)` to the positive examples, where `clause` was applicable to `proof_state`, but this step did not participate in the final proof.

The communication overhead introduced by the API is small, as witnessed by the inference speed of about 50,000 inferences per second on selected MPTP problems. The API is very simple and one of our plans is to connect it to the OpenAI environment [BCP+16]. This would allow further experiments with various RL and supervised methods.



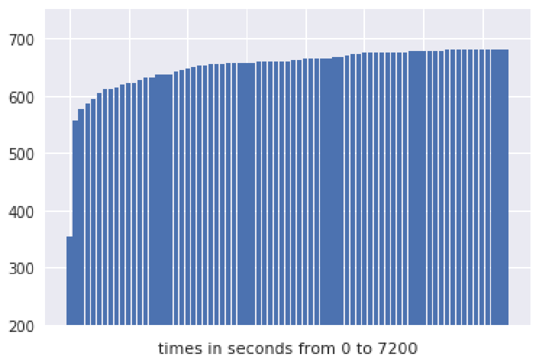


Figure 1: Naive Bayes-guided PyCoP progress over time (left) and XGBoost-guided PyCoP progress over time (right)

3 First Experiments

We experiment with the large Mizar40 [KU15c] dataset of ATP-provable (and ATP-minimized) 31250 MPTP [Urb06] problems.[1] We also use a smaller dataset of the 2078 MPTP2078 [AHK+14] Mizar bushy problems.[2] Unguided Python-based leanCoP (PyCoP) solves 11402 of the 31250 Mizar40 problems with a 10 s time limit.

In the first experiment we test the accuracy of our tree-RNN classifier[3] on the positive/negative examples extracted (in the prefix representation) from the 11402 proofs found by unguided PyCoP. The results of the experiment are available online.[4] After 15 epochs, the training accuracy reaches 97.3% and the validation accuracy is about 87.1%.

We also experiment with XGBoost using the standard feature-based representation of the data. XGBoost is a high-performance implementation of a tree-based classifier which typically performs well on large sparse datasets like ours with millions of features. Figure 1 compares on the MPTP2078 benchmark the performance of XGBoost-guided PyCoP with PyCoP guided by the naive Bayes classifier developed in [KU15b]. The naive Bayes version so far outperforms the XGBoost version, however the XGBoost guidance is approximately 30 times slower, making about 30-times less steps in the same time. We can also see that XGBoost can still prove new theorems with higher time limits. With appropriate ensembling, XGBoost-guided PyCoP proves 777 MPTP2078 theorems, which seems to be the current record in the 2-hour timeout heavy-weight leanCoP category (PyCoP guided by naive Bayes proves 720 theorems).

Finally, we have done first tests of a simple (Monte Carlo) unsupervised reinforcement learning setting using the full Mizar40 dataset and our tree-RNN classifier. So far we only allow 50 PyCoP steps guided by the (Monte-Carlo modified) RNN evaluation, which can get only very simple proofs in the iterative deepening approach used by leanCoP. After a single pass and learning from the feedback, the system solves about 5% of the problems compared to 2.5-3% when unguided.

[1] https://github.com/JUrban/deepmath
[2] https://github.com/JUrban/MPTP2078
[3] https://github.com/mirefek/HolStep-Tree
[4] http://atrey.karlin.mff.cuni.cz/~mirecek/fm/pycop_supervised.log

References

[AHK$^+$14] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.

[BCP$^+$16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *CoRR*, abs/1606.01540, 2016.

[CG16] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[FKU17] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte Carlo tableau proof search. In Leonardo de Moura, editor, *26th International Conference on Automated Deduction (CADE 2017)*, volume 10395 of *LNCS*, pages 563–579. Springer, 2017.

[KCS17] Cezary Kaliszyk, François Chollet, and Christian Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. *CoRR*, abs/1703.00426, 2017.

[KU15a] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 88–96. Springer, 2015.

[KU15b] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *20th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2015)*, volume 9450 of *LNCS*, pages 88–96. Springer, 2015.

[KU15c] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.

[KUV15a] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Certified connection tableaux proofs for HOL Light and TPTP. In Xavier Leroy and Alwen Tiu, editors, *Proc. of the 4th Conference on Certified Programs and Proofs (CPP'15)*, pages 59–66. ACM, 2015.

[KUV15b] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *IJCAI'15*, pages 3084–3090. AAAI Press, 2015.

[Lai15] Matthew Lai. Giraffe: Using deep reinforcement learning to play chess. *CoRR*, abs/1509.01549, 2015.

[OB03] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003.

[RGB10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[Urb06] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.

[UVv11] Josef Urban, Jiří Vyskočil, and Petr Štěpánek. MaLeCoP: Machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *TABLEAUX*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.

Mizar in Isabelle for Formal Abstracts*

Cezary Kaliszyk[1] and Karol Pąk[2]

[1] Department of Computer Science, Universität Innsbruck, Austria
`cezary.kaliszyk@uibk.ac.at`
[2] Institute of Informatics, University of Białystok, Poland
`pakkarol@uwb.edu.pl`

Abstract

One of the main goals of the Mizar project has been to create a formal system that would be attractive for mathematicians. Various developed features have therefore became an inspiration for extensions and improvements in other systems. At the same time, the architecture of Mizar has not been flexible enough to accommodate the solutions developed in other systems. In this talk we present a combination of the Isabelle – a modern logical framework – with a Mizar object logic and argue that it can serve as an attractive environment for formal mathematics. Indeed, the Mizar foundations are a variant of set theory, which is familiar for mathematicians. Its proof style does correspond to natural proofs. And the type system was designed to correspond to how mathematicians classify objects. We will finally discuss the various mechanisms that allow for grater usability for mathematical statements and proofs.

The Mizar project [1] from its beginning aimed to make a system for human readable formalization of mathematics. Many aspects distinguish it from other proof assistants. Its proof style imitates informal mathematical proofs. Its type system tries to express how mathematicians use and categorize mathematical objects. Combining these two features provide a more intuitive environment for formalized mathematics than other systems [8]. Furthermore, formalized Mizar results have been gathered in the *Mizar Mathematical Library* (MML). Its focus is on mathematical results, which makes it complementary to the libraries of various other proof assistants, including results that have not been formalized in other systems, such as the theory of lattices, topological manifolds, and random access Turing machines.

In this talk we will present a combination of a modern logical framework – Isabelle – with the foundations and the proof style of Mizar and discuss the usability of the resulting proof environment for the development of formal mathematical abstracts. Our Mizar emulator [7] created in the Isabelle logical framework provides selected constructs from the Mizar language [5]. We imitate the type system including intersection types and structures [6], as well as higher-order concepts, such as set comprehensions and schemes [4].

We will argue that the environment is more convenient for stating and formalizing formal mathematical statements. It is possible to naturally state mathematical object classifications, for example: `X is n-dimensional topological-manifold` is a compact Mizar-like statement that is clear to a mathematician, but stating it in many systems would require unnatural constructions. We show that it will be possible import and cross-verify the whole MML with all its mathematical results. We will present various manually re-formalized Mizar theorems including results from set theory, algebra and random access Turing machines. We will also discuss the Mizar level of human readability.

We discuss our foundations of the Mizar system as an object logic in the Isabelle logical framework, especially the number of needed constants and axioms. Then we focus on faithful

*The paper has been financed by the resources of the Polish National Science Centre granted by decision no DEC-2015/19/D/ST6/01473

imitation the Mizar definitional mechanisms. We show adequate mechanisms for each kind of these definition including the Mizar structures that allows multiple inhabitance. Finally we show also an experimental mechanism that provides selected Mizar type information in justifications of proof steps.

Various extensions of other proof assistants that imitate the Mizar language have been proposed. Examples include the Mizar Mode for HOL [11] and the Isar language for Isabelle [10]. These are however limited to a few rules the Jaśkowski natural deduction style [3], and omit other crucial parts of mathematical text, such as Mizar definitions of more complex objects that require Mizar-style justification of correctness. Similarly, there have been a number of attempts to translate the Mizar logic to various other formalisms. Urban [9] exported the MML to the TPTP first-order language, and with Brown this was extended to higher-order logic [2]. Such approaches try to preserve the semantics of Mizar, however do not preserve any of the user commands or notations. In consequence such translations significantly reduce proof readability, so important for formal proof abstracts.

[1] G. Bancerek, C. Byliński, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban. Mizar: State-of-the-art and Beyond. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015.

[2] C. E. Brown and J. Urban. Extracting higher-order goals from the Mizar Mathematical Library. In M. Kohlhase, M. Johansson, B. R. Miller, L. de Moura, and F. W. Tompa, editors, *Proc. 9th International Conference on Intelligent Computer Mathematics (CICM 2016)*, volume 9791 of *LNCS*, pages 99–114. Springer, 2016.

[3] S. Jaśkowski. On the rules of suppositions. *Studia Logica*, 1, 1934.

[4] C. Kaliszyk and K. Pąk. Isabelle formalization of set theoretic structures and set comprehensions. In J. Blamer, T. Kutsia, and D. Simos, editors, *7th International Conference on Mathematical Aspects of Computer and Information Sciences, MACIS 2017*, volume 10693 of *LNCS*. Springer, 2017.

[5] C. Kaliszyk and K. Pąk. Presentation and manipulation of Mizar properties in an Isabelle object logic. In H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *LNCS*, pages 193–207. Springer, 2017.

[6] C. Kaliszyk and K. Pąk. Progress in the independent certification of Mizar Mathematical Library in Isabelle. In M. Ganzha, L. A. Maciaszek, and M. Paprzycki, editors, *Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017*, pages 227–236, 2017.

[7] C. Kaliszyk, K. Pąk, and J. Urban. Towards a Mizar environment for Isabelle: Foundations and language. In J. Avigad and A. Chlipala, editors, *Proc. 5th Conference on Certified Programs and Proofs (CPP 2016)*, pages 58–65. ACM, 2016.

[8] A. Trybulec, A. Korniłowicz, A. Naumowicz, and K. T. Kuperberg. Formal mathematics for mathematicians - special issue. *J. Autom. Reasoning*, 50(2):119–121, 2013.

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

[10] M. Wenzel. Isar - A generic interpretative approach to readable formal proof documents. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, volume 1690 of *LNCS*, pages 167–184. Springer, 1999.

[11] F. Wiedijk. A synthesis of the procedural and declarative styles of interactive theorem proving. *Logical Methods in Computer Science*, 8(1), 2012.

Mechanizing *Principia Logico-Metaphysica* in Functional Type Theory

Daniel Kirchner[1], Christoph Benzmüller[12], and Edward N. Zalta[3]

[1] Freie Universität Berlin
[2] University of Luxembourg
[3] Stanford University

Principia Logico-Metaphysica (PLM) [14] aims at a foundational logical theory for metaphysics, mathematics and the sciences. It contains a canonical presentation of Abstract Object Theory (AOT) [15, 16], which distinguishes between abstract and ordinary objects, in the tradition of the work of Mally [7]. The theory systematizes two fundamental kinds of predication: classical exemplification for ordinary and abstract objects, and *encoding* for abstract objects. The latter is a new kind of predication that provides AOT with expressive power beyond that of quantified second-order modal logic, and this enables elegant formalizations of various metaphysical objects, including the objects presupposed by mathematics and the sciences. More generally, the system offers a universal logical theory that may have a greater capability of accurately representing the contents of human thought than other foundational systems.

Independently, the use of *shallow semantical embeddings* (SSEs) of complex logical systems in classical higher-order logic (HOL) has shown great potential as a metalogical approach towards universal logical reasoning [1]. The SSE approach aims to unify logical reasoning by using HOL as a universal metalogic. Only the distinctive primitives of a target logic are represented in the metalogic using their semantical definitions (hence the *shallow* embedding), while the rest of the target system is captured by the existing infrastructure of HOL. For example, quantified modal logic can be encoded by representing propositions as sets of possible worlds and by representing the connectives, quantifiers, and modal operators as operations on those sets. This way the world-dependency of Kripke-style semantics can be elegantly represented in HOL. Utilizing the powerful options to handle and hide such definitions that are offered in modern proof assistants such as Isabelle/HOL [10], a human-friendly mechanization of even most challenging target logics, including the AOT, can thus be obtained.

AOT and the SSE approach are rather orthogonal. They have very different motivations and come with fundamentally different foundational assumptions. AOT uses a *hyperintensional second-order modal logic*, grounded on a *relational type theory*, as its foundation. It is in the tradition of Russell and Whitehead's *Principia Mathematica* [11, 8], which takes the notion of *relation* as primitive and defines the notion of *function* in terms of relations. The metalogic HOL in the SSE approach, by contrast, is fully extensional, and defined on top of a functional type theory in the tradition of the work of Frege [6] and Church [4]. It takes the notion of (fully extensional) *function* as primitive and defines the notion of *relation* in terms of functions. These fundamentally different and, to some extent, antagonistic roots in turn impose different requirements on the corresponding frameworks, in particular, with regard to the comprehension principles that assert the existence of relations and functions. Devising a mapping between the two formalisms has, unsurprisingly, been identified as a non-trivial, practical challenge by Oppenheimer and Zalta [12].

The work reported here tackles this challenge. Further details can be found in Kirchner's M.A. thesis [9], where the SSE approach is utilized to mechanize and analyze AOT in HOL. Kirchner constructed a shallow semantical embedding of the second-order modal fragment of AOT in HOL, and this embedding was subsequently represented in the proof assistant system

Isabelle/HOL. The proof assistant system enabled us to conduct experiments in the spirit of a *computational metaphysics*, with fruitful results that have helped to advance the ideas of AOT.

The inspiration for Kirchner's embedding comes from the model for AOT proposed by Peter Aczel.[1] Kirchner also benefited from Benzmüller's initial attempts to embed AOT in Isabelle/HOL. An important goal of the research was to avoid *artifactual theorems*, i.e., theorems that (a) are derivable on the basis of special facts about the Aczel model that was used to embed AOT in Isabelle/HOL, but (b) aren't theorems of AOT. In previous applications of the SSE approach, this issue didn't arise. For example, in the context of the analysis of Gödel's modal ontological argument for the existence of God (cf. [2]), extensive results about the Kripke models were available *a priori.* But AOT is, in part, a body of theorems, and so care has been taken not to derive artifactual theorems about the Aczel model that are not theorems of AOT itself.

This explains why the embedding of AOT in Isabelle/HOL involves several layers of abstraction. In the Aczel model of AOT that serves as a starting point, abstract objects are modeled as sets of properties, where properties are themselves modeled as sets of urelements. Once the axioms of AOT are derived from the shallow semantic embedding of AOT in HOL, a controlled and suitably constricted logic layer is defined: by reconstructing the inference principles of AOT in the system that derives the axioms of AOT, only the theorems of AOT become derivable. By utilizing Isabelle/HOL's sophisticated support tools for interactive and automated proof development at this highest level of the embedding, it became straightforward to map the pen and paper proofs of PLM into corresponding, intuitive, and user-friendly proofs in Isabelle/HOL. In nearly all cases this mapping is roughly one-to-one, and in several cases the computer proofs are even shorter. In other words, the *de Bruijn factor* [13] of this work is close to 1. In addition, the layered construction of the embedding has enabled a detailed, experimental analysis in Isabelle/HOL of the underlying Aczel model and the semantical properties of AOT.

As an unexpected, but key result of this study, we discovered that if the classical logic for λ-expressions and definite descriptions is adjoined to AOT's specially-formulated comprehension principle for relations without taking any special precautions, a known paradox ('the Clark-Boolos paradox' [5, 3]) that had been successfully put to rest is reintroduced.[2] Since the complex terms add significant expressive and analytic power to AOT, and play a role in many of its more interesting theorems, the re-emergence of the known paradox has become a new paradox that has to be addressed. In the ongoing attempts to find an elegant formulation of AOT that avoids the new paradox, the computational representation in Isabelle/HOL now provides a testing infrastructure and serves as an invaluable aid for analyzing various conjectures and hypothetical solutions to the problem. This illustrates the very idea of *computational metaphysics*: humans and machines team up and split the tedious work in proportion to their cognitive and computational strengths and competencies. And as intended, the results we achieved reconfirm the practical relevance of the SSE approach to universal logical reasoning.

[1]An earlier model for the theory was proposed by Dana Scott. His model is equivalent to a special case of an Aczel model with only one *special urelement.*

[2]The Clark-Boolos paradox of *encoding* arises if there are properties of the form $[\lambda \exists F(xF \& \neg Fx]$ (*encoding a property that is not exemplified*). If such properties were to exist, one could, by object comprehension, generate an abstract that encodes such a property. And such an object would exemplify this property if and only if it does not. To address the paradox, object theory disallows encoding subformulas in the matrix of λ-expressions.
This paradox gets reintroduced, however, if care isn't taken to disallow λ-expressions with definite descriptions from β-Conversion. Object theory considers the term $[\lambda x\, G\iota z\psi]$ as well-formed, even if ψ contains encoding subformulas, since these latter are not subformulas of the full matrix. So by choosing G to be a property that is universally true (e.g., $[\lambda y\, \forall p(p \rightarrow p)]$) and ψ as $z = x \wedge \exists F\, (xF \wedge \neg Fx)$, the result is a property that, by β-Conversion, is extensionally equivalent to the property that led to the Clark-Boolos paradox. The restriction on β-Conversion, which had been a part of earlier versions of object theory, was originally omitted from PLM, though as we now know, at the peril of the system.

References

[1] Christoph Benzmüller. Universal Reasoning, Rational Argumentation and Human-Machine Interaction. *CoRR*, abs/1703.09620, 2017.

[2] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 93–98. IOS Press, 2014.

[3] George Boolos. The Consistency of Freges Foundations of Arithmetic. In J. Thomson, editor, *On Being and Saying*. Cambridge, MA: MIT Press, spring 2017 edition, 1987.

[4] Alonzo Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[5] Romane Clark. Not Every Object of Thought has Being: A Paradox in Naive Predication Theory. *Noûs*, 12(2):181–188, 1978.

[6] Gottlob Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Verlag von Louis Nebert, Halle, 1879.

[7] Alexander Hieke and Gerhard Zecha. Ernst Mally. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[8] Andrew David Irvine. Principia Mathematica. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[9] Daniel Kirchner. Representation and Partial Automation of the Principia Logico-Metaphysica in Isabelle/HOL. *Archive of Formal Proofs*, September 2017. http://isa-afp.org/entries/PLM.html, Formal proof development.

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

[11] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume 3. Cambridge University Press, Cambridge, 2 edition, 1913.

[12] Paul E. Oppenheimer and Edward N. Zalta. Relations Versus Functions at the Foundations of Logic: Type-Theoretic Considerations. *Journal of Logic and Computation*, 21(2):351–374, 2011.

[13] Freek Wiedijk. The de Bruijn factor. http://www.cs.ru.nl/~freek/factor/.

[14] Edward N. Zalta. Principia Logico-Metaphysica. http://mally.stanford.edu/principia.pdf. [Draft/Excerpt; accessed: April 01, 2017].

[15] Edward N. Zalta. *Abstract Objects: An Introduction to Axiomatic Metaphysics*. Synthese Library. Springer, 1983.

[16] Edward N. Zalta. *Intensional Logic and the Metaphysics of Intentionality*. A Bradford book. MIT Press, 1988.

Toward AI for Lean, via metaprogramming

Robert Y. Lewis[1,2]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
[2] Vrije Universiteit, Amsterdam, NL
rob.y.lewis@gmail.com

Lean is a proof assistant being developed at Microsoft Research [3]. The system has been designed from the beginning to support strong automation. It aims to eventually straddle the line between an interactive theorem prover with powerful automation, and an automated theorem prover with a verified code base and interactive mode. A distinguishing feature of Lean is its *metaprogramming* framework [4]. This framework is designed to allow users to write their own complex tactics and programs with access to internal system tools, without needing to touch a line of source code; these tactics and programs exist as part of and in-line with theory developments.

There has been much recent interest in using AI-based methods for guiding proofs in ITP systems, and the metaprogramming framework provides a natural way to investigate such methods in Lean. We have taken preliminary steps toward implementing a relevance filter for heuristic lemma selection, as described in [2]. The framework also allows connections to external programs. Using an established connection between Lean and Mathematica [5], it is possible to apply Mathematica's black-box machine learning tools for similar purposes. This link is bi-directional, and from within Mathematica, we can use these tools to investigate the Lean library as a repository of mathematics.

1 Metaprogramming in Lean

Lean is based on the Calculus of Inductive Constructions (CIC) [1]. Dependent type theory is a convenient language for formalizing mathematical definitions, theorems, and proofs, but it is also a practical and effective programming language. Alongside its trusted kernel, Lean implements a virtual machine for evaluating Lean expressions. The VM uses many optimizations to allow for efficient computation: among others, it replaces terms of type `nat` with machine integers and terms of type `array α n` with a native implementation of arrays, allowing destructive updates when the reference counter for the array is 1 [4]. Declarations in Lean tagged with the keyword `meta` are invisible to the kernel and used only for VM computations. For such declarations, termination checking for recursive calls can be relaxed without making the non-meta language inconsistent.

A number of meta constants, including `expr`, `tactic_state`, and `environment`, are defined to reflect the underlying implementations of these objects. When the VM evaluates terms that involve these constants, it associates them with the actual underlying implementations. A tactic can then be thought of as a term `t : tactic_state → tactic_state`; more generally, a tactic producing data has type `tactic_state → α × tactic_state`. To invoke such a tactic on a particular goal, Lean constructs the corresponding tactic state and evaluates `t` applied to this data.

2 A relevance filter, and beyond

Using this framework, we can write metaprograms that fold over a Lean environment – containing the types and values of all declarations to a point – and produce data. We have followed [2, Section 4], adjusting for differences between Coq and Lean expressions, to implement simple k-nearest-neighbors and sparse naive Bayes classifiers to associate types with constants that are likely to appear in their proofs. Since the code is implemented entirely in Lean, it is relatively easy to extend the feature and label sets.

This work is preliminary and is intended to be part of a future Lean hammer, but can already be used for informative purposes. Our implementation extends the Lean VM to allow computation with native floats. While the filter is relatively quick, it does not compare favorably to the more native implementation of the CoqHammer project. It is an interesting question whether, and how, we could avoid modifying the core VM, and how we can more efficiently manage big-data computations in the metaprogramming framework.

3 Linking Lean and Mathematica

The metaprogramming framework provides an interface from Lean to command-line and file IO. This allows users to write tactics that communicate with external programs. In [5] and [6] we describe a bi-directional interface for exchanging information between Lean and Mathematica. Through this interface, Lean tactics can query Mathematica for information about Lean expressions – for example, to request the factored form of a polynomial – and Mathematica can query Lean's automation and proof library. The translation process is generic and can be extended as part of theory development.

Mathematica includes a suite of machine learning tools that are designed to work for many applications without configuration [7]. These tools could be used to implement a more powerful relevance filter than the simple one described above, without the overhead of designing customized algorithms. This filter would be accessible from Lean using the link described. These tools are also integrated with Mathematica's functions for classifying and displaying data. Following work on computing with arXiv papers, we plan to develop techniques to explore and visualize the Lean library from within Mathematica.

References

[1] T. Coquand and C. Paulin. Inductively defined types. In *COLOG-88 (Tallinn, 1988)*, volume 417 of *Lec. Notes in Comp. Sci.*, pages 50–66. Springer, Berlin, 1990.

[2] L. Czajka and C. Kaliszyk. Hammer for Coq, 2017.

[3] L. de Moura, S. Kong, J. Avigad, F. van Doorn, and J. von Raumer. The Lean theorem prover. *http://leanprover.github.io/files/system.pdf*, 2014.

[4] G. Ebner, S. Ullrich, J. Roesch, J. Avigad, and L. de Moura. A metaprogramming framework for formal verification. *Proceedings of the ACM on Programming Languages*, 1(ICFP):34, 2017.

[5] R. Y. Lewis. An extensible ad hoc interface between Lean and Mathematica. In *Proof eXchange for Theorem Proving*, 2017.

[6] R. Y. Lewis and M. Wu. A bi-directional extensible ad hoc interface between Lean and Mathematica. *http://www.andrew.cmu.edu/user/rlewis1/leanmm/lean_mm_cpp.pdf*, 2017.

[7] S. Wolfram. *An Elementary Introduction to the Wolfram Language.* Wolfram Media, Incorporated, 2015.

Cumulative Effects in Learning[*]

Érik Martin-Dorel and Sergei Soloviev

IRIT, Université de Toulouse, CNRS, Toulouse, France
erik.martin-dorel@irit.fr
sergei.soloviev@irit.fr

Extended Abstract

In our previous works we considered the value of information in games (usually in extensive form or repeated) with respect to winning. It is clear that if the purpose is to win, the complete knowledge of the opponent behavior is not necessary but some form of learning can be used as a rule in order to develop a better strategy adapted to the behavior of an opponent.

We studied some situations where a quantitative answer is possible: (*i*) focusing on 2-player Boolean games and relying on probabilistic methods [MDS18][1] and (*ii*) focusing on the construction of winning strategies (using hypothesis testing, which can be viewed as a form of learning) [DSS12].

An interesting observation was that often the probability of existence of a winning strategy grows very fast with respect to the quantity of information obtained by a player. For example in the case of Boolean games, the probability of guaranteed win for the first player grows as fast as 2^{2^s}, where s is the number of bits of information known on the second player's strategies.

In some sense, the case of primitive recursive strategies is more interesting because the recursive function cannot be identified by any finite number of its values, however in the class of games that we considered, one player could win using a universal function for primitive recursive functions and finite number of moves of its opponent. In other words, "complete" learning would require an infinite amount of information, but finite information was sufficient for winning.

In our present work we study certain game-theoretic models where an interplay between winning and learning can create interesting cumulative effects.

Observation. *Small changes in probabilistic parameters may have huge effects in cases of*

- *iterated events (queues);*
- *phase transitions;*
- *chain reactions;*
- *a positive feedback.*

For example, consider a large population where all members play simultaneously a game against some opponent that we may call Ignorance (for example, each plays some Boolean game with a randomly selected formula of some class). Ignorance may always choose an optimal strategy at his side, he has an unlimited computational power, however he does not know the choices of the members of the population. Some of them still may win (they may even have a

[*]This work was partly supported by the FAGames project of LabEx CIMI.

[1]A preliminary version of our article is available online as an IRIT research report, cf. the following URL: https://www.irit.fr/publis/ACADIE/IRIT-RR-2017-01-FR.pdf

universal winning strategy in their personal game). Assume now that those who win pass to the next level and at this level, may know in advance one bit of the strategies chosen by Ignorance against themselves, and moreover, may help others, for example by telling their friends (defined by some relation) one bit of the strategies that Ignorance plays against them. This will increase their probability to win. The win will then let them pass to the next level, and so on.

We think that this approach may be interesting in the models of collective learning. In particular, it could be applied to the modeling of distributed theorem proving, in order to gain indications on how to coordinate efforts and better organize feedback. Also, we may try to apply such an approach to model paradigm shifts.

The starting point of the reflection is the following model that we considered in [MDS18]: two players A and B control a certain number of Boolean variables (A controls k bits $(a_1, a_2, \ldots, a_k) = a$ and B controls $n-k$ bits $(b_1, b_2, \ldots, b_{n-k}) = b$). They play simultaneously a given game represented by a Boolean function $F : \mathbf{2}^k \times \mathbf{2}^{n-k} \to \mathbf{2}$. Player A wins if $F(a,b) = \texttt{true}$, otherwise player B wins. The game itself (function F) is assumed to be randomly chosen among the whole class of Boolean functions with n variables, and we specifically considered the case of Boolean functions generated by a Bernoulli scheme on Boolean vectors with a parameter $0 < p < 1$. We gave quantitative results regarding the probability that there exists a winning strategy for player A in this model (assuming player A knows $s \geq 0$ bits of information among the strategy $(b_1, b_2, \ldots, b_{n-k})$ of player B). These results were obtained symbolically and formally proved in the Coq proof assistant.[2]

We would like to discuss possible generalizations of this model (e.g., regarding the choice of the probability distribution, regarding the number of players, or regarding the kind of the strategies. . .) that could fit our proposed approach to model cumulative effects in learning, with the ultimate goal to obtain quantitative results that are amenable to formal proof.

Our approach strongly relies on game-theoretic notions but can also be related to the ACRE learning problem (viz. adversarial classifier reverse engineering [LM05]), which can itself be viewed as an instance of supervised machine learning.

Beyond the interest on modeling learning scenarios, we believe that the combination of formal verification techniques and openly discussed (probabilistic) models can move explainable artificial intelligence forward, in the same way as Belle explains the need to combine logical and statistical AI in his recent survey [Bel17].

References

[Bel17] Vaishak Belle. Logic meets probability: Towards explainable AI systems for uncertain worlds. In Carles Sierra, editor, *IJCAI 2017*, pages 5116–5120. ijcai.org, 2017. doi:10.24963/ijcai.2017/733.

[DSS12] Evgeny Dantsin, Jan-Georg Smaus, and Sergei Soloviev. Algorithms in Games Evolving in Time: Winning Strategies Based on Testing. In *Isabelle Users Workshop – ITP 2012*, 2012. 18 pages.

[LM05] Daniel Lowd and Christopher Meek. Adversarial Learning. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, pages 641–647, New York, NY, USA, 2005. ACM. doi:10.1145/1081870.1081950.

[MDS18] Erik Martin-Dorel and Sergei Soloviev. A Formal Study of Boolean Games with Random Formulas as Payoff Functions. In Herman Geuvers, Silvia Ghilezan, and Jelena Ivetic, editors, *Post-Proc. of TYPES 2016, Novi Sad, 23/05/2016-26/05/2016*, LIPIcs. Schloss Dagstuhl Leibniz-Zentrum fur Informatik, 2018. To appear.

[2]The accompanying formal proofs are available online at https://sourcesup.renater.fr/coq-bool-games/

Machine comprehension of math problem text

Takuya Matsuzaki[1] and Noriko H. Arai[2]

[1] Nagoya University, Aichi, Japan
matuzaki@nuee.nagoya-u.ac.jp
[2] National Institute for Informatics, Tokyo, Japan
arai@nii.ac.jp

Abstract

In a joint work with many people, we have developed a computer system that solves pre-university level math problems written in natural language. The system is comprised of two parts. One is a language processing pipeline, which translates a math problem into a logical formula. The other is a computer algebra system that derives an answer from the translated problem. In the talk, I will mainly talk about the former part. The main obstacle in the translation from a natural language into a logical language is the flexibility of the natural language, which enables us to convey complex meaning in a concise expression but makes the sentences highly ambiguous for a machine. I will explain how we combat with it using both logical and statistical means. These two means work complementarily. First, a statistical model provides a mechanism for selecting a plausible interpretation of a genuinely ambiguous sentence although such a sentence is rare in a carefully written math problem. Second, a statistical model works as a 'mending tape,' which seals a crack in a logical model of language. Third, a statistical model can be used as an (ad-hoc) heuristic function during the search for the most plausible interpretation.

If mathematical proof is a game, what are the states and moves?

David McAllester

Toyota Technological Institute at Chicago

Abstract. Alpha Zero has achieved dramatic success in Go, Chess and Shogi using general deep RL and self-play. Alpha Zero applies a single learning architecture to different states and moves. Intuitively, mathematical proof is also a game. But what are the states and moves of proof? This talk will argue that the states should be taken to be tableau-style contexts. We propose a notion of context consisting of variable declarations (let X be foo-space), assumptions (suppose Phi), goal declarations (to show Phi ...), or focus declarations (consider e). Tableau-style proof has always been motivated by its naturality — its similarity to human argumentation. Alpha Zero invokes enormous computation in evaluating states and proposing moves. This talk will also discuss how deep RL might be applied to evaluating states and proposing moves in tableau-style argumentation.

Towards logics for neural conceptors

Till Mossakowski[1] and Razvan Diaconescu[2]

[1] Otto-von-Guericke-Universität Magdeburg, Germany
till@iws.cs.ovgu.de

[2] Simion Stoilow Institute of Mathematics of the Romanian Academy, Bucharest, Romania
razvansdiaconescu@gmail.com

Abstract

Conceptors are an approach to neuro-symbolic integration based on recurrent neural networks. We develop a logic for neural conceptors that turns out to be fuzzy. Also, proof support and theorem proving is discussed.

Neural networks have been successfully used for learning tasks [10], but they exhibit the problem that the way they compute their output generally cannot be interpreted or explained at a higher conceptual level [11]. The field of neuro-symbolic integration [1] addresses this problem by combining neural networks with logical methods. However, most approaches in the field (like e.g. logic tensor networks [5]) are localist, that is, predicates or other symbolic items are represented in small sub-networks. This contrasts with the distributed representation of knowledge in (deep learning) neural networks, which seems to be much more flexible and powerful.

Jaeger's conceptors [8, 9] provide such a distributed representation while simultaneously providing logical operators and concept hierarchies that foster explainability. The basic idea is to to take a recurrent neural network and not use it for learning through back-propagation, but rather feed it with input signals, leading to a state space that can be captured as a certain ellipsoid using a conceptor matrix. Conceptor matrices are positive semidefinite matrices with singular values (which represent the lengths of the ellipsoid axes) ranging in [0,1].

In [9], Jaeger introduces and studies algebra of conceptors, providing the logical operations "and", "or" and "not" (which however satisfy only part of the laws of Boolean algebra) as well as a scaling operation called aperture adaption, and an interpolation operation. A crucial advantage of conceptors over ordinary neural networks is that using the algebra of conceptors, training examples can easily be added to conceptors, without the need of re-training with the whole sample. Moreover, the Löwner ordering on conceptor matrices expresses a concept hierarchy. For reasoning about conceptors, two logics are introduced, an extrinsic and an intrinsic one. Both logics are based on the conceptor algebra operations. The extrinsic logic provides a first-order logic with atomic formulas based on the Löwner ordering. This leads to two levels of Boolean operations: one within conceptor terms, and one within the first-order logic. The intrinsic operation avoids this duplication by only working on conceptor terms and comparing them with a fixed conceptor. In [9], Jaeger formalises both logics as institutions [6], which are an abstract formalisation of the notion of logical system. Moreover, he states the open problem of developing a proof calculus and theorem proving support for these logics.

We here argue that both of these logics are not completely adequate for reasoning about conceptors, because they both can ultimately speak only about the Löwner ordering, i.e. crisp statements that can be either true or false. We propose that a more promising approach is to view conceptors as a kind of fuzzy sets. Indeed, their Boolean operators satisfy the (appropriate generalisation of) T-norm and T-conorm laws, and form a (generalised) De Morgan Triplet [12, 7]. This is remarkable, because conceptors have not been introduced as a neuro-fuzzy approach (and note that neuro-fuzzy approaches generally are localist in the above sense, while conceptors provide a global distributed representation of knowledge).

We argue that an appropriate conceptor logic should not have crisp but fuzzy statements as its atomic constituents:

- classification of an N-dimensional signal vector z by a $N \times N$ conceptor matrix C, yielding the fuzzy truth value $z^T Cz/N$ (which can be seen as fuzzy set membership),
- a "fuzzy subset" relation $C_1 \leq C_2$ between conceptors.

Atomic formulas use conceptor terms formed with the same operations as in Jaeger's logics. On this basis, we develop a many-valued institution [3] for conceptors. A (fuzzy) first-order logic on top of that can be obtained using general methods of defining fuzzy connectives and quantifiers.

Reasoning in such a logic can be done using the algebraic and order-theoretic laws of conceptors. Theorem proving support can be obtained by a translation to first-order or higher-order logic (giving approximation or exact capturing of the real numbers), as well as by using SMT solving over the real numbers.

Fuzzy reasoning can be based on either crisp or graded consequence relations (see [4]). The latter can capture fuzzy reasoning in a more fine-grained way. Future work will consider the development of a graded proof calculus that directly captures graded consequence.

References

[1] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902, 2017.

[2] R. Diaconescu. *Institution-independent Model Theory*. Birkhuser, 2008.

[3] Razvan Diaconescu. Institutional semantics for many-valued logics. *Fuzzy Sets and Systems*, 218:32–52, 2013.

[4] Razvan Diaconescu. Graded consequence: an institution theoretic study. *Soft Comput.*, 18(7):1247–1267, 2014.

[5] Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1596–1602. ijcai.org, 2017.

[6] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39:95–146, 1992. Predecessor in: LNCS 164, 221–256, 1984.

[7] M.M. Gupta and J. Qi. Theory of T-norms and fuzzy inference methods. *Fuzzy Sets and Systems*, 40:431–450, 1991.

[8] Herbert Jaeger. Conceptors: an easy introduction. *CoRR*, abs/1406.2671, 2014.

[9] Herbert Jaeger. Controlling recurrent neural networks by conceptors. *CoRR*, abs/1403.3369, 2014.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] E. Smith and S. Kosslyn. *Cognitive psychology: Mind and brain*. Upper Saddle River, NJ: Prentice-Hall Inc., 2006.

[12] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Designing Games of Theorem Proving

Yutaka Nagashima

CIIRC, CTU, Prague / The University of Innsbruck, Innsbruck
first_name.last_name@cvut.cz

Abstract

"Theorem proving is similar to the game of Go. So, we can probably improve our provers using deep learning, like DeepMind built the super-human computer Go program, AlphaGo [1]." Such optimism has been observed among participants of AITP2017. But is theorem proving really similar to Go? In this paper, we first identify the similarities and differences between them and then propose a system in which various provers keep competing against each other and changing themselves until they prove conjectures provided by users.

1 The Game of Go and Theorem Proving

Formally defined rules [similarity]. Both the games of Go and theorem proving have algorithms to evaluate the results. In the game of Go, one can judge the result of each game when it is over by counting the stones and spaces for each player, and no ambiguity is left in deciding the result of a game. In theorem proving, when one finds a proof, others can systematically check if the alleged proof is a valid proof or not.

Expressive power of the system [difference]. Even though both systems are based on a set of simple rules, the expressive power of these systems differ. Depending on the underlying logics, a theorem proving task can involve advanced concepts such as abstraction, universal quantification, existential quantification, and polymorphism, which Go scores cannot express natively. This is especially true for more expressive logics such as classical higher-order logics or variants of calculus of constructions, where stronger proof automation is needed.

Amount of available training data [difference]. Some theorem proving researchers boast that they have large proof corpora. For example, the Isabelle community has the Archive of Formal Proofs (AFPs) [2], consisting of more than 1.5 millions of lines of code and 100 thousands lemmas. Unfortunately, even though these proof corpora are large for the small community of theorem provers, they are small compared to the data deployed in other domains.

Preference towards small data [difference]. The size of the community is not the only reason of small data available in the theorem proving community: logicians and mathematicians have developed expressive logics to describe general ideas in a concise manner. Combined with the trade-off between proof automation and expressive power of underlying logic, this is doubly unfortunate: the more expressive logic we use, the less proof automation we have, but the more expressive the logic is, the less training data we can expect, which makes it hard to improve the proof automation for expressive logics using machine learning techniques.

Self-playability [similarity/difference]. One might suspect that large data are not necessary to develop a powerful proof automation tool using machine learning. After all, DeepMind has made AlphaGo Zero [3] stronger than any previous versions of AlphaGo via self-play without using data from human games. Unfortunately, even though both Go and theorem proving are based on clearly defined rules, theorem proving is not a two-player game by default. In the rest of this paper, we propose an approach to introducing self-playability to theorem proving.

2 The Design of Self-playable Games of Theorem Proving

One straightforward design of self-playable games of theorem proving is as follows: (1) prepare a set of proof obligations from existing proof corpora, (2) let two competing provers try to prove these proof obligations, (3) count how many obligations each prover discharges, (4) consider the prover that solves more obligations as the winner, and the other one as the loser. We can use this naive approach as a part of reinforcement learning or evolutionary computation to optimize our provers for proof obligations that have already been proved. However, this approach is probably not powerful enough to improve provers for conjectures that are significantly different from the theorems in the training data

For example, let us assume that we enhance our prover, say `P`, via self-play using 100 theorems and their proofs in the AFPs. Since we already know how to prove these theorems, we can improve `P`, so that `P` can prove all of the 100 theorems within a reasonable time-out. However, when we try to improve `P` to discharge a new conjecture, say Goldbach's conjecture, we will find ourselves at a loss of training data: Currently, there is no known proofs of Goldbach's conjecture or auxiliary lemmas that are verified to be useful to prove Goldbach's conjecture.

If we add Goldbach's conjecture to the above dataset, the improvement via self-play would saturate after producing a prover that can discharge the 100 theorems from the AFP but not Goldbach's conjecture: since the gap between the theorems from the AFPs and Goldbach's conjecture is too large, minor mutations to `P`'s variants cannot produce a useful observable difference in the result of the game. What we need here is a mechanism to produce conjectures that we can reasonably expect to be useful to prove our target conjecture (Goldbach's conjecture in this example) but not too difficult for our current prover `P`.

Therefore, we propose to *treat conjecturing and proof search as one problem.* Of course, we cannot be 100% sure which conjecture is useful to train our prover for Goldbach's conjecture, since nobody has proved it yet. But if we consider a heuristic proof search as the exploration of an and-or tree, we can estimate how important each node in the tree is from the search heuristics of the prover. Furthermore, given a long time-out, we can expect that the prover can discharge some of emerging subgoals, even if the prover cannot discharge the root-node, which corresponds to the target conjecture (Goldbach's conjecture, in this example).

Our idea is to *use these proved subgoals to judge the competence of other versions of prover* `P`. First, we produce two versions of our prover `P` by mutation. Let us call them `Pa` and `Pb`, respectively. Using the approach explained above, we let `Pa` produce a dataset `Da` and let `Pb` produce `Db`. Now, we let `Pb` try to prove the theorems in `Da`, and let `Pa` try to prove the theorems in `Db`. When `Pa` and `Pb` run out of time, we sum up the estimated values of theorems proved by each prover (`Pa`, for example). Note that it was the opponent prover (`Pb` in this example) that has decided the value of each theorem in each dataset (`Db` in this case) when finding proofs of these subgoals for the first time. The prover that has gained more value is the winner of the game, and the other is the loser. Then, we keep running this game by mutating the winner until we produce a prover that can discharge the target conjecture. Since this process generates new conjectures tagged with their estimated values from the target conjecture in each iteration, we expect this approach continues producing conjectures useful to prove the target conjecture.

We are still in the early stage of the design. We might generalize this idea to n-player games to avoid over-fitting. For the moment, we prefer the design of the game to be irrelevant to any of underlying logics, ML algorithms for search heuristics, and mutation algorithms.

Acknowledgement

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

References

[1] David Silver, *et. al.* Mastering the game of Go with deep neural networks and tree search Nature 529, January 2016

[2] Gerwin Klein, Tobias Nipkow, and Larry Paulson. The Archive of Formal Proofs.

[3] David Silver, *et. al.* Mastering the game of Go without human knowledge Nature 550, October 2017

Who cares about Euclidean geometry?

Mirek Olšák

Charles University

Abstract. We will discuss several reasons why Euclidean geometry could be a good topic for an AITP project.

ATP Guidance for Learning Premise Selection

Bartosz Piotrowski and Josef Urban

Czech Technical Univeristy in Prague, Prague, Czech Republic

Premise Selection as a Machine Learning Problem

The most efficient methods for premise selection are based on *data-driven/machine-learning* approaches. Such methods work as follows. Let T be a set of theorems with their proofs. Let C be a set of conjectures without proofs, each associated with a set of available premises that can be used to prove them. Having this, we want to learn a (statistical) model from T, which for each conjecture in $c \in C$ will rank its available premises according to their relevance for producing an ATP proof of c. Two different machine learning settings can be used for this task:

1. *multilabel classification*: we treat premises used in the proofs as opaque labels and we create a model capable of labeling conjectures based on their features,
2. *binary classification*: here the aim of the learning model is to recognize pairwise-relevance of the *conjecture-premise* pairs, i.e. to decide what is the chance of a premise being relevant for proving the conjecture based on the features of both the conjecture and the premise.

Most of the machine learning methods for premise selection have so far used the first setting [6, 5, 3]. This includes fast and robust machine learning algorithms such as *naive Bayes* and *K nearest neighbors* (k-NN) capable of multilabel classification with many examples and labels. This is needed for large formal libraries with many facts and proofs. There are however several reasons why the second approach may be better:

1. Generality: in binary classification it is easier to estimate the relevance of *conjecture-premise* pairs where the premise was so far unseen (i.e., not in the training data).
2. State-of-the-art ML algorithms are often capable of learning subtle aspects of complicated problems based on the features. The multilabel approach trades the rich feature representation of the premise for its opaque label.
3. Many state-of-the-art ML algorithms are binary classifiers or they struggle when performing multilabel classification for a large number of labels.

Recently, substantial work [2] has been done within the second setting. In particular, applying deep learning to premise selection has improved state of the art in the field.

Premise Selection in Binary Setting with Multiple Proofs

The availibility of multiple ATP proofs makes premise selection different from conventional machine learning applications. This may be important especially in the binary classification setting. The ML algorithms for recognizing pairwise relevance of *conjecture-premise* pairs require good-quality data consisting of two (preferably balanced) classes of positive and negative examples. But there is no conventional way how to construct such data. For every true conjecture there are infinitely many proofs, based on many different sets of premises.

Consider the following frequent situation: a conjecture c can be ATP-proved with two sets of axioms: $\{p_1, p_2\}$ and $\{p_3, p_4, p_5\}$. Learning only from one of the examples as positives and presenting the other as negative *conjecture-premise* pairs may considerably distort the learned notion of a *useful premise*. This differs from our previous research done in the multilabel setting [7], where using only one proof typically helped. In the multilabel setting negative data are typically not used by fast ML algorithms such as naive Bayes and k-NN. They just aggregate different positive examples into the final ranking. The talk will discuss our experiments that attempt to properly take into account multiple ATP proofs while training binary-classification ML algorithms.

The following observations should be taken into account when designing the ML experiments. The number of ATP proofs may be high and it is intractable to initially generate them all. Also, some of the proofs are easier to find than other proofs by a particular ATP. And some of the premises appear more often in the proofs of a given conjecture. This leads to a MaLARea-style [9] feedback loop between the learner and an ATP, where we gradually learn which *conjecture-premise* pairs are *relevant* for this particular ATP and which are not.

First Experiments Combining Binary Classification and ATP Feedback

We use the E prover as the underlying ATP and the XGBoost [4] system to learn pairwise-relevance of *conjecture-premise* pairs. XGBoost implements a tree-based, gradient boosting ML algorithm, which has proved efficient in many ML competitions. As a benchmark we use the MPTP2078 [1] subset of MML. Our feedback loop between E [8] and XGBoost is as follows:

1. We split the ATP-provable (60s CPU limit) theorems into a training set and a test set. Our initial data consists of theorems with their proofs. Initially, there is only one proof per theorem. These are the positive examples in our initial training set. Negative examples are constructed by choosing similar (in terms of feature similarity) examples not present in the positive class. We train an initial classifier C_0 on such training data and set $i = 0$.
2. We randomly choose a subset S of 10% of the theorems from the training data. For each $c \in S$ we ask C_i to rank the premises available for c.
3. E is run on the theorems in S with initial segments of the 1, 2, ..., 512 top-ranked premises.
4. The positive part of the training data set is updated with all pairs (c, p) where $c \in S$ and p was a premise used in at least one proof we already know for c. The negative part is updated with all pairs (c, p) where $c \in S$ and p was not used in any proof of c, and p was ranked by C_i at least at position $2 \times N_c$, where N_c is the number of all premises used in all known proofs for c. For each newly introduced training example, we randomly drop an old example from training set.
5. We train classifier C_{i+1} on the such updated training set.
6. We measure the ATP performance of C_{i+1}'s predictions on the test set.
7. We go back to step (2) and repeat the whole procedure for $i = i + 1$ as long as new proofs are found or the performance of the trained model improves on the test set.

For a comparison, we also simultaneously train another XGBoost model C_{const} with the same parameters but on the constant training set we created in the step 1. After each cycle we compare the ATP-performance of the two models. Already the model trained in the first cycle outperforms the model trained in the standard way and the difference increases up to 20 – 25th cycle, see the table below, and the plot at http://bartoszpiotrowski.pl/plot-aitp18.png.

In addition to several related ML/ATP MaLARea-style feedback loops, related work includes [2]. There, large improvement are obtained with *negative mining* done however without interaction with an ATP. We believe that our setting has several advantages:

1. The machine learner learns on multiple proofs.
2. The frequency of premises appearing in these proofs has influence on the learning.
3. Evolving the data set throughout the training may help prevent overfitting.
4. Randomly dropping data might help to get rid of contradictory examples and gradually evolve the classifier to optimally reflect pairwise relevance for a given ATP.

Cycle of training (i)	0	5	10	15	20	25
Model trained with ATP-negative-mining (C_i)	44%	67%	70%	72%	73%	73%
Model trained on a constant dataset (C_{const})	44%	50%	51%	53%	53%	53%

References

[1] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.

[2] Alex A. Alemi, Francois Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban, editors. *DeepMath - Deep Sequence Models for Premise Selection*, 2016.

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

[4] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. 2016.

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

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

[7] Daniel Kuehlwein and Josef Urban. Learning from multiple proofs: First experiments. In Pascal Fontaine, Renate A. Schmidt, and Stephan Schulz, editors, *PAAR-2012*, volume 21 of *EPiC Series*, pages 82–94. EasyChair, 2013.

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

[9] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.

Dynamic strategy priority: empower the strong and abandon the weak

Michael Rawson and Giles Reger

University of Manchester, Manchester, UK

Many modern automated theorem provers (e.g. CVC4[1] [1], E [8], iProver [2], Vampire [3]) rely on *portfolio modes* [7] utilising from tens to hundreds of distinct strategies, of which only a few might solve a hard problem quickly. Typically, a portfolio of strategies has a pre-defined order that the prover will execute the strategies in, running each until a strategy succeeds. Portfolios are important as, in practice, there is no best strategy i.e. it is uncommon that two hard problems are efficiently solvable by the same strategy. However, portfolio execution is not without problems: selecting the optimal ordering and time allocation is hard in general [6], and produces overly-rigid, brittle engineering when applied to specific domains, such as those found in TPTP [9]. Moreover, for any particular problem, some lengthy strategies that are successful on other problems are doomed to failure from the outset, but are left to run unchecked by the prover, wasting time that could be spent on more productive strategies.

In this work we first demonstrate correlation between trends in dynamic properties of proof search, and the success or failure of a strategy. We then utilise this to implement strategy scheduling, prioritising those strategies most likely to succeed. This approach differs from previous work [4,5,8] which attempts to predict successful strategies *a priori* from static features of the input problem; instead we tip running strategies for success based on run-time features and use this information to make scheduling decisions. Initial experiments on Vampire produce a performant neural-network that achieves classification accuracy of 81% (±2%).

Obtaining and filtering Vampire execution data. Modifying Vampire to log execution data for different strategies obtained from its primary portfolio mode[2] is straightforward, but there are choices to be made along the way. First, which data *sources* are interesting? As a proof of concept, we focus mostly on the numerical data (e.g. the number of generated clauses) that is immediately available in the prover environment, but there is scope here for non-numeric and/or derived data sources that could provide greater insight into the proof state. Data was logged at a fixed interval of resolution steps — unfortunately, this does not necessarily correspond to a fixed amount of time. Addressing this is left as future work. Overall, this methodology produces extremely voluminous, irregular data. We chose to apply a rolling time average to normalise/reduce trace size, then allow the neural net to do its own feature selection.

Identifying and predicting successful strategies. Being able to predict which strategies are going to succeed, and which will fail (or exceed the time limit) at first seems unlikely. However, it is known that the "slowly-growing search space" maxim, which states that strategies which minimise the number of derived clauses over time are more likely to succeed, is an effective heuristic for finding good strategies in saturation-based theorem proving [6]. Since the data we use includes the number of derived clauses, among many other features, it appears more plausible that a machine-learnt approach might work at least as well as the slow-growth heuristic

[1]Whilst SMT solvers are less reliant on heuristic strategies than saturation-based techniques, they still typically employ various strategies, for example for quantifier instantiation.

[2]CASC-mode, a portfolio designed for the CASC competition [10].

alone. We engineer a prediction algorithm that attempts to partition traces into "succeeding" and "failing" classes using a simple neural network. Conveniently, these methods do not usually produce a binary output, but instead some $f(\mathbf{X}) \in [0, 1]$ which might be seen as the "level of confidence" in success of the trace $\mathbf{X}$. This success score can be used to apply an ordering to executing strategies, allowing "smart" dynamic scheduling of strategies.

Smart scheduling for Vampire We show that this abstract predictor can be used in a concrete implementation for the Vampire prover. In the modified prover, it is used to run several strategies from Vampire's portfolio in a time-slicing scheduler: at each slice, the most promising strategies are run. The eventual aim is to improve Vampire's overall performance, if not in the number of total theorems proved (this will likely not change in this experiment — if the entire strategy schedule runs and fails, it doesn't matter which way it is ordered), but in the time taken to prove problems. Current benchmark results indicate a 10% average improvement in time, in exchange for losing some problems in the benchmark.

For this demonstration we focus on maximising the accuracy and efficiency of the predictor (a perfect predictor would schedule the best strategy first, every time!), rather than tweaking performance of the scheduler. As well as improvements to our prediction techniques, further research might include designing scheduling algorithms which keep predictions as up-to-date as possible, maximise processor utilisation, minimise memory usage/swapping, reduce context-switching overhead, or even minimise calls to the predictor, and observing the effect on prover performance.

References

[1] Morgan Deters, Andrew Reynolds, Tim King, Clark W. Barrett, and Cesare Tinelli. A tour of CVC4: how it works, and how to use it. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*, page 7, 2014.

[2] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning, (IJCAR 2008)*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

[3] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In *International Conference on Computer Aided Verification*, pages 1–35. Springer, 2013.

[4] Daniel Kühlwein, Stephan Schulz, and Josef Urban. E-MaLeS 1.1. In *International Conference on Automated Deduction*, pages 407–413. Springer, 2013.

[5] William McCune and Larry Wos. Otter — the CADE-13 competition incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997.

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

[7] Giles Reger, Martin Suda, and Andrei Voronkov. The challenges of evaluating a new feature in Vampire. In *Vampire Workshop*, pages 70–74, 2014.

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

[9] G. Sutcliffe. The TPTP problem library and associated infrastructure, from CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[10] Geoff Sutcliffe and Christian Suttner. The state of CASC. *AI Communications*, 19(1):35–48, 2006.

Implementation of Lambda-Free Higher-Order Superposition

Petar Vukmirovic

Vrije Universiteit Amsterdam

Abstract. In the last decades, first-order logic has emerged as a standard language for describing a large number of mathematical theories. Many first-order automatic theorem provers have been developed. Higher-order logic enables one to describe more theories and to describe some theories more succinctly, but higher-order provers are much less mature than their first-order counterparts. In this work, we extend the state-of-the-art first-order prover E to lambda-free higher-order logic, resulting in the new prover hoE. We devise generalizations of E's indexing data structures, as well as algorithms like matching and unification. Generalizations we give exhibit exactly the same behavior and time complexity as the original E on first-order problems. Furthermore, experimentation showed that on lambda-free higher-order problems, hoE is 20% faster on average than E with the traditional encoding of higher-order terms.

Disambiguating ProofWiki into Mizar: First Steps

Jiří Vyskočil*
Czech Technical University in Prague

Josef Urban*
Czech Technical University in Prague

1 Autoformalization and Pr∞fWiki

The talk will describe progress in the project of automatically formalizing informal mathematics by using statistical parsing methods and large-theory automated reasoning. In the first part of the talk we will summarize the overall autoformalization approach and its performance on the *ambiguated* Flyspeck [3] and Mizar [2] corpora as recently described in [5, 6]. The second part will present our initial adventures in the land of strictly informal mathematics: trying to align and learn the translation between the informal Pr∞fWiki and the formal Mizar corpora.

The Pr∞fWiki[1] project was started in 2008, aiming to provide an online platform for explaining and editing mathematical knowledge, particularly proofs. Each Pr∞fWiki page contains one (proved) fact or definition. The pages use wiki formatting and MathJax[2] rendering of TEX. Concepts, facts and proofs are presented in great detail[3] and they link to the facts and concepts that they directly depend on. The language is informal but quite regular: our early exploration [7] has shown that the top 100 generalized proof sentences cover about 50% of the corpus.

Pr∞fWiki has no formal syntax or semantics and there is no formal checker/verifier. However, in the recent work of Bancerek done within the AI4REASON project, Pr∞fWiki has been extended by 500 new definitions/theorems that are linked to their formal counterparts in the Mizar library (MML). Our present goal is to develop methods for parsing and translating the Pr∞fWiki-style informal texts into formal Mizar-style parse trees. The formal parse trees can then be typechecked [6], translated into first-order logic [10], and verified with large-theory ATP methods for Mizar [4], providing semantic feedback to the statistical parsing methods.

2 Lexical Analyzer for Informal Mathematics and Pr∞fWiki

For the earlier autoformalization experiments we have used a simple lexical analyzer based on whitespace-separated tokens. This was sufficient, because the ambiguous texts were produced by our informalization of the formal corpora using spaces as token separators.

Pr∞fWiki and many informal math resources however typically combine TEX commands with text without obvious token separators. Compared to ordinary English, which usually has simple word separation, mathematicians often compose operators and arguments together, and they use different separators in different situations. Sometimes they omit the operators altogether (e.g. multiplication). This prevents us from using standard lexical analyzers developed by the Natural Language Processing (NLP) community. In the talk, we will explain how we have modified our CYK-based parser so that it can parse several possible tokenizations of all words in one pass. Our experiments show that this does not significantly affect the efficiency of parsing.[4]

3 From Pr∞fWiki to Mizar

Consider the following short example[5][6] of the Pr∞fWiki–Mizar alignment:

*Funded by the ERC project no. 649043 – AI4REASON.

[1]`https://proofwiki.org`

[2]`https://www.mathjax.org/`

[3]In particular, the proof style is very reminiscent of the Jaśkowski-style natural deduction used by Mizar.

[4]The slowdown due to the modification measured on the informalized MML is 9%.

[5]`https://proofwiki.org/wiki/Singleton_is_Chain`

[6]`http://www.mizar.org/version/current/html/orders_2.html#T8`

PW code	`Let $\left(S, \preceq\right)$ be an ordered set. Let $x \in S$. Then $\left\ x\right\$ is a chain of $\left(S, \preceq\right)$.`
PW display	Let $(S, \preceq)$ be an ordered set. Let $x \in S$. Then $\{x\}$ is a chain of $(S, \preceq)$.
Mizar	`for A being non empty reflexive RelStr for a being Element of A holds {a} is Chain of A`
Mizar parse	`(Bool for (Varlist (Set (Var A))) being (Type (@ListOfAdjectives (Adjective ($#~nv2_struct_0 non (Attribute ($#nv2_struct_0 empty)))) (Adjective (Attribute ($#nv3_orders_2 reflexive)))) ($#nl1_orders_2 RelStr)) (Bool for (Varlist (Set (Var a))) being (Type (@ListOfAdjectives) ($#nm1_struct_0 Element) of (Set (Var A))) holds (Bool (Set ($#nk6_domain_1 {) (Set (Var a)) ($#nk6_domain_1 })) is (Type (@ListOfAdjectives) ($#nm2_orders_2 Chain) of (Set (Var A))))))`

Parsing ProofWiki into a Mizar-like parse tree requires transformations of varied difficulty:

1. The ProofWiki `chain` can map directly to the Mizar-style subtree `($#nm2_orders_2 chain)`, possibly additionally aligning `chain` with `Chain` as synonyms.
2. The ProofWiki TEX text `"\left\{ {x}\right\}"` needs to be mapped to the Mizar-style subtree `(Set ($#nk6_domain_1 {) (Set (Var x)) ($#nk6_domain_1_part_1 }))`.
3. `"ordered set"` needs to be mapped to Mizar `"non empty reflexive RelStr"`.
4. `"Let...Then..."` needs to be mapped to Mizar as `"for...holds..."`. Etc.

(1) is just a new grammar rule that can be learned from the treebank. The other examples however require more complex tree transformations that cannot be expressed as simple grammar rules. As a general approach, we have introduced a grammar extension [7] that allows evaluation of arbitrary Lisp-like programs at nonterminal positions. This allows us to use our existing PCFG infrastructure for CYK-like parsing with powerful ways of modifying already parsed parts of the input. Following is an example of a subtree (and code) that performs the mapping (2):

```
(Set  ("PW_TeX_Singleton@@@
        (lambda (L LSB LB X RB R RSB)
               (list (gtree '$#nk6_domain_1 '{) X (gtree '$#nk6_domain_1_part_1 '})))"
      "\left" "\{" "{" (Set (Var "x")) "}" "\right" "\}" ))
```

When the whole subtree is parsed, the Lisp-like code in the nonterminal *`"PW_TeX_Singleton@@@..."`* (in italics) is executed using its children/subtrees as arguments, resulting in the tree in (2).
We hope that in many cases, such Lisp-like programs will be semi-automatically learnable by the following bootstrapping procedure:

1. The parser run on the corpus of ProofWiki texts will identify the parts of input that cannot be parsed yet. This can be done by using a special low-probability nonterminal `"UNKNOWN"` that propagates through most of the grammar rules, thus marking the failed fragments.
2. The failed fragments will be aligned with the corresponding Mizar subtrees.
3. This yields a corpus of ProofWiki - Mizar pairs where the parsing fails so far.
4. This corpus can be mined for common frequent patterns by symbolic learning methods such as genetic programming and inductive logic programming. Such methods can gradually create a corpus of more and more advanced Lisp-like functions that build on each other (also during the learning phase).
5. Some frequently occurring pairs will probably need transformations that are beyond the means of the learning methods (at least initially). Having the statistics of such failed pairs will however still be useful for focusing the human annotators on the bottlenecks.
6. As in the Flyspeck and Mizar experiments, the most probable parses will be subjected to typechecking and large-theory ATP, using the whole Mizar library as a background knowledge. We will try to use the (parsed) steps in the ProofWiki proofs as additional lemmas useful for structuring the ATP work into smaller parts.

[7]Compared to some existing frameworks [8, 9], our extension is lightweight and its integration of more advanced probabilistic models (compared to [1]) has already been tested at large on the Flyspeck project [5].

References

[1] Krasimir Angelov and Peter Ljunglöf. Fast statistical parsing with parallel multiple context-free grammars. In Gosse Bouma and Yannick Parmentier, editors, *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2014, April 26-30, 2014, Gothenburg, Sweden*, pages 368–376. The Association for Computer Linguistics, 2014.

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

[3] T. Hales, M. Adams, G. Bauer, D. Tat Dang, J. Harrison, T. Le Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. Tat Nguyen, T. Quang Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. Hoai Thi Ta, T. N. Tran, D. Thi Trieu, J. Urban, K. Khac Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

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

[5] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Automating formalization by statistical and semantic parsing of mathematics. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2017.

[6] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. System description: Statistical parsing of informalized mizar formulas. In *SYNASC 2017*, 2017. To appear. `https://www.ciirc.cvut.cz/~urbanjo3/synasc17.pdf`.

[7] Cezary Kaliszyk, Josef Urban, Jiří Vyskočil, and Herman Geuvers. Developing corpus-based translation methods between informal and formal mathematics: Project description. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *LNCS*, pages 435–439. Springer, 2014.

[8] Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

[9] Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. On multiple context-free grammars. *Theor. Comput. Sci.*, 88(2):191–229, 1991.

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

Building an Auto-Formalization Infrastructure through Text-Mining on Mathematical Literature: Project Description

Qingxiang, Wang[1][2]

[1] University of Innsbruck
[2] Czech Technical University in Prague
shawn.wangqingxiang@gmail.com

Formally checking the correctness of research-level mathematical proofs would inevitably involve the formidable task of formalizing mathematics knowledge in existing literature. Current formalization corpora however, despite many man-years of manual effort, barely capture the majority of mathematical knowledge beyond undergraduate mathematics curriculum. The abundance of raw mathematical literature freely available on the internet encourages a machine learning approach to formalization. Deep learning [1], being the most effective form of machine learning at present, and having shown spectacular results in planning [2] and translation tasks [3], should be helpful to advance research in theorem proving. However, despite initial applications of deep learning in theorem proving [4, 5, 6, 7], we have not seen any application that can directly leverage the abundance of natural language proofs.

Previous works trying to bridge the gap between informal and formal mathematics have gradually drawn the conclusion that machine learning approach should be used [8]. Various algorithms [9, 10] have been experimented with and training data sets have been extracted from Flyspeck [11] and Mizar [12, 13] throughout the last three years. In this research project, we will propose a set of deep learning enabled tools that gradually transform informal mathematics into formal mathematics.

The transformation will consist of the following three main phases.

1. **Syntactical Analysis Phase**. After proper preprocessing, the first component will conduct several phases of syntactic analysis to extract structural information from each of the mathematical statements. In addition to texts, we will attempt to extract the logical information embedded in formulas and eventually perhaps also in diagrams. As proofs may omit details at advanced level, cross-document analysis between elementary materials and advanced materials can be adopted to uncover details that are implicit in sketchier proofs. The end result of the first component will be a more refined, more explicit and more controlled natural language (or pseudo programming language) proof that mimics the proof presentation format as proposed by Lamport [14].

2. **Mapping Phase**. In the second phase we will first attempt to obtain a crude mapping by attaching formalized proofs to their corresponding informal proofs that have been syntactically analyzed in the first phase. The structural similarity between the informal and formal proofs will then be further exploited to obtain a more refined mapping. We will then experiment with various methods so as to further refine this mapping until token-level details can be put to correspondence. By the end of this refinement process we will be able to obtain a collection of semantically-annotated and formally-correct proofs that have complete information on informal-to-formal correspondence.

3. **Generalization phase**. The joining of the bridge will be the design of a deep reinforcement learning architecture that can generalize the formalization correspondence from the

above subset of semantically-annotated formally-correct proofs to all the other proofs that have no formalized counterparts. The neural networks in this architecture may be affected by overfitting when there is not enough training data, but it is notable that if a formalization generated from the architecture is formally correct, then this formalization can be used as training data to further train the neural networks. This positive feedback loop can continue provided enough mathematical literature is fed into the architecture.

Our auto-formalization infrastructure will be able to generate formalized proofs for human-proved theorems that have never been formally verified. We believe that the main focus of deep learning applications to theorem proving should be in auto-formalization instead of proof automation. Such a formalization infrastructure, if fully developed, can significantly increase the amount of formalized mathematics and help to ensure the quality of research works done by contemporary mathematicians.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning, 2017.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[4] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Eén, François Chollet, and Josef Urban. 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*, pages 2235–2243, 2016.

[5] Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary 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*, pages 85–105, 2017.

[6] Daniel Whalen. Holophrasm: a neural automated theorem prover for higher-order logic, 2016.

[7] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding, 2017.

[8] Cezary Kaliszyk, Josef Urban, Jiří Vyskoçil, and Herman Geuvers. Developing corpus-based translation methods between informal and formal mathematics. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of *LNCS*, pages 435–439. Springer Verlag, 2014.

[9] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora (rough diamond). In Christian Urban and Xingyuan Zhang, editors, *Proc. 6h Conference on Interactive Theorem Proving (ITP'15)*, volume 9236 of *LNCS*, pages 227–233. Springer-Verlag, 2015.

[10] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Automating formalization by statistical and semantic parsing of mathematics. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *8th International Conference on Interactive Theorem Proving (ITP 2017)*, volume 10499 of *Lecture Notes in Computer Science*, pages 12–27. Springer, 2017.

[11] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean Mclaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller. A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, 5, 2017.

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

[13] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. System description: Statistical parsing of informalized Mizar formulas. Submitted, available at `http://grid01.ciirc.cvut.cz/~mptp/synasc17sd.pdf`.

[14] Leslie Lamport. How to write a proof. *American Mathematical Monthly*, 102(7):600–608, 1995.