AITP 2019

Fourth Conference on
Artificial Intelligence and Theorem Proving

Abstracts of the Talks

April 7–12, 2019, Obergurgl, Austria

Preface

This volume contains the abstracts of the talks presented at AITP 2019: Fourth Conference on Artificial Intelligence and Theorem Proving held on April 7–12, 2019 in Obergurgl, Austria.

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 2019 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 27 contributed talks.

We would like to thank the University of Innsbruck conference center in Obergurgl for hosting AITP 2019. Many thanks also to Andrei Voronkov and his EasyChair for their support with paper reviewing and proceedings creation. The conference was partly funded from the European Research Council (ERC) under the EU-H2020 projects SMART (no. 714034) and AI4REASON (no. 649043), and the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15003/0000466 and the European Regional Development Fund. Finally, we are grateful to all the speakers, participants and PC members for their interest in discussing and pushing forward these exciting topics!

April 2019

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

Program Committee

Jasmin C. Blanchette	INRIA Nancy
Ulrich Furbach	University of Koblenz
Thomas C. Hales	University of Pittsburgh
Sean Holden	University of Cambridge
Mikoláš Janota	University of Lisbon
Moa Johansson	Chalmers University
Cezary Kaliszyk	University of Innsbruck
Michael Kohlhase	FAU Erlangen-Nürnberg
Konstantin Korovin	The University of Manchester
Ramana Kumar	DeepMind
Claudio Sacerdoti Coen	University of Bologna
David McAllester	Toyota Technological Institute at Chicago
Adam Pease	Infosys
Stephan Schulz	DHBW Stuttgart
Geoff Sutcliffe	University of Miami
Christian Szegedy	Google Research
Josef Urban	Czech Technical University in Prague

Additional Reviewers

Jonas Betzendahl	Chad Brown
Chad Brown	Dennis Müller
Qingxiang Wang	Karol Pąk
Robert Y. Lewis	Petar Vukmirović
Jan Jakubuv	Sarah Winkler
Ralph Bottesch	Burak Ekici
Robert Veroff	Thibault Gauthier

Table of Contents

Experiments With Connection Method Provers

Wolfgang Bibel[1] and Jens Otten[2]

[1] Darmstadt University of Technology, bibel@gmx.net
[2] University of Oslo, jeotten@ifi.uio.no

In this presentation we are focussing on Automated Theorem Proving (ATP). ATP traditionally has been part of AI, a discipline which is in the headlines these days as never before. It is therefore a natural question whether or not ATP might take advantage of the current exorbitant efforts to advance AI R&D. We begin with presenting arguments which favor an ongoing substantial role for ATP within AI. The arguments engage a perspective of natural intelligence and its structure as evolved over mankind's thousands of years long history.

In recent years, AI often is identified just with learning systems, not only in the public but also within IT. In fact, learning systems have achieved such impressive results that the mainstream in the research community currently expects the creation of an artificial intelligence to be achieveable more or less exclusively by way of learning.

In the early days of AI it was said that it would be much easier to build a system simulating the work of a math professor than the intelligence of a child. After the recent breakthroughs in learning this judgment is no more tenable since the child's learning could now be simulated to some extent by some appropriate learning system. The math professor, in addition to his or her abilities learned as a child, invests a lot into his or her competence to reason logically in a correct and scientific way on the basis of knowledge acquired in a disciplined manner. The simulation of a math professor hence will require a combination of techniques including learning as well as knowledge representation and reasoning (KR) in whatever form, not least in that of ATP. Thus, ATP will remain of central importance for AI also in the forseeable future.

A more detailed exposition of this analysis of the development of the human behavior in everyday circumstances as well as in professional achievements leads to new lines of research into the details of the composition forming the human mind and in consequence the human behavior. Such insights influence the way ATP is realized and integrated in the simulation of mathematical research. One of the goals of the remaining presentation consists in pointing to some of the directions of research deriving from these insights.

Thus encouraged to continue research in a *suitably modified ATP*, we present the basics of the Connection Method (CM), its implementation through a family of theorem provers, results of experiments with these and challenges for future research in the directions just indicated. The history of ATP has led to at least three different proof methods: Resolution, Tableaux, and CM. Among these the CM features the following advantages.

It has been shown [1] that (an extended version of) the CM can linearly simulate Resolution. Thus, the performance of any resolution prover could as well be realized on the basis of the CM. Also, in contrast to Tableaux, the CM performs an extremely compressed proof search [5]. So in terms of performance, the CM in principle is at

least as powerful as its main competitors. In addition, it offers the following striking advantages.

The CM is unique in that it performs the proof search in terms of the structural features of the very formula which is to be proved (illustrated in Fig. 1) rather than destroying this original structure by adding instances of subformulae (in case of resolution or instance-based methods) to or decomposing (in case of tableaux) the original formula. This *formula-orientedness* minimizes the data to be manipulated in the proof search and thus, in principle, offers a comparatively optimal mechanism in comparison with the competing proof methods (just compare the manipulated data in a proof with your favorite method with that in Fig. 1 for the same formula). In other words, the CM offers a potentially higher *performance* than its competitors. Due to a strict *goal-orientedness* during the *connection-driven* proof search, both suggested by the formula-orientedness, current CM systems do exhibit a comparatively strong performance.

Apart from performance the CM proof search is covering many logics in a completely uniform way, rendering *uniformity* as the second striking advantage. There is a third, yet hardly exploited major advantage of the CM over the competitive methods. Namely, the formula-orientedness enables the proof process to take a *global* view over the object of analysis, an aspect which still offers a great potential for being tapped and exploited in future research.

The talk introduces the CM in an illustrative way referring the reader to the literature such as [2, 3, 1, 5] for any details. In this abstract we just show a connection proof displayed in Fig. 1 for a simple first-order formula F_1, consisting of a spanning set of two connections along with a substitution unifying the connected literals.

$$\exists a(Pa \vee \neg\exists x\, Qx) \rightarrow \exists y\, Py \vee \forall b\, \neg Qfb \qquad \text{with} \quad \sigma = \{x_1 \backslash fb, y_1 \backslash a\}$$

Fig. 1. The connection proof for F_1.

We then briefly discuss connection calculi, the adequate tools for proof search in the CM, and their underlying principles leading to the goal-orientedness mentioned above. A basic connection calculus for first-order logic (fol) involving backtracking was developed and implemented in high-level PROLOG [12, 18]. The program, called leanCoP, although comprising only a few PROLOG clauses, shows a performance comparable to competitive theorem provers consisting of hundreds of thousands of lines of code [18, 11]. Due to its high-level code its correctness, in contrast to the large systems, can be and has been verified which we regard as an important issue for proof systems [19]. leanCoP operates on formulas in clause form, not on fol formulas in their original form as illustrated in Fig. 1. A variant of leanCoP has been realized as an OCaml version [9].

A connection calculus for skolemized, but non-clausal fol formulas has also been developed and implemented in the style of leanCoP as a system called nanoCoP [13, 16] showing again a comparatively impressive performance. nanoCoP performs the proof search directly on the structure of the original formula, no translation steps to any clause form are necessary. It combines the advantages of more natural non-clausal calculi (eg. sequent calculi), with the goal-oriented efficiency of a connection-driven proof search.

Hence, it is also significantly easier, to translate the resulting non-clausal proofs back into a human-readable form.

The CM's unique uniformity bears its fruits in porting the proof-search technology developed in one logic into other logics. This way the fol technologies built into leanCoP and nanoCoP have been ported to many other logics resulting in a uniform set of systems such as ileanCoP and nanoCoP-i for intuitionistic fol, and MleanCoP and nanoCoP-M for modal fol, thus forming a large family of uniformly designed and powerful provers [11, 14, 15, 17]. The CM is the unique and unrivalled proof-method in ATP which features such a wide variety of systems, many of which are outperforming any of its competitors.

At the outset of this abstract we have argued for a *suitably modified ATP*. In the presentation we will discuss a few elements in the modifications needed for an ATP that takes into account the progress in the pertinent scientific disciplines. First of all, ATP needs to base its research on the best possible proof-method available. As our research, briefly outlined above, has demonstrated in various ways the CM offers the best possible basis for the development of powerful systems. So, the time has come for the community to take a major step towards a change in the method of choice.

This step involves many different aspects. One of those is a greater flexibility in terms of the underlying logic in particular applications. As we demonstrated the CM supports this flexibility in the best possible way. Another important aspect refers to the form of the problems to be proved. While it has become standard in ATP to simplify matters by using clause form, more advanced techniques have exposed the considerable disadvantages going along with these simplifications (see eg. [20]). As we have seen, the CM again supports the avoidance of these disadvantages.

In the latter respect, we have experimentally compared problems stated in non-clausal form with their transformed versions in clausal form with respect to runtime and resulting proof size. As one of the results it turns out that the length of the returned proofs, in terms of the number of connections, is – on average – significantly shorter in the case of the non-clausal CM of nanoCoP compared to the clause form CM of leanCoP. Furthermore, nanoCoP does not only prove problems not proved by leanCoP, but also a few hundred problems not proved by one of the fastest provers available today.

Among the modifications towards a modern ATP certainly is the integration of learning techniques into ATP systems, as initiated by members of our research group already in the 1980's in the context of our proof system SETHEO [6, 7]. Although the integration of learning into ATP by now has become a rather active area of research [21, 8], it is far from clear how exactly learning techniques could best support the work in ATP. It is clear that the way of their integration may strongly differ from one kind of problems to another.

In order to illustrate this difficulty, we discuss a problem due to Łukasiewicz [10], $\forall xyzu\, Pi(i(ixy,z),i(izx,iux)) \ \wedge \ \forall vw\,(Pv \wedge Pivw \rightarrow Pw) \ \rightarrow \ \forall abc\, Pi(iab,i(ibc,iac))$. While Łukasiewicz was able to find a proof for it by hand, current provers need tens of thousands or even millions of search steps to discover a proof. Also, the effect of applying a standard learning system to the prover E for learning a better strategy leads to limited improvement only [4]. What kind of artificial intelligence is needed to improve our systems to a point where they are able to find the proof of this or other problems in a more direct way similar to that taken by Łukasiewicz?

References

1. Bibel, W., Eder, E.: Methods and calculi for deduction. In: Gabbay, D.M., Hogger, C.J., Robinson, J.A. (eds.) Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 1, chap. 3, pp. 71–193. Oxford University Press, Oxford (1993)
2. Bibel, W.: Automated Theorem Proving. Vieweg Verlag, Braunschweig, second edn. (1987), first edition 1982.
3. Bibel, W.: Deduction: Automated Logic. Academic Press, London (1993)
4. Bibel, W.: A vision for Automated Deduction rooted in the connection method. In: Schmidt, R., Nalon, C. (eds.) The 26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2017). LNAI, vol. 10501, pp. 3–21. Springer (2017), dOI: 10.1007/978-3-319-66902-1_1
5. Bibel, W., Otten, J.: From Schütte's formal systems to modern automated deduction. In: Kahle, R., Rathjen, M. (eds.) The Legacy of Kurt Schütte. Springer (to appear)
6. Ertel, W., Schumann, J.M., Suttner, C.B.: Learning heuristics for a theorem prover using back propagation. In: Retti, J., Leidlmair, K. (eds.) 5. Österreichische Artificial-Intelligence-Tagung: Igls/Tirol, 28.-31. März. pp. 87–95. Springer, London (1989)
7. Goller, C.: A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems. Akademische Verlagsgesellschaft AKA, Berlin (1999)
8. Kaliszyk, C., Urban, J.: FEMaLeCoP: Fairly efficient machine learning connection prover. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2015). Lecture Notes in Computer Science, vol. 9450, pp. 88–96. Springer, Berlin, Heidelberg (2015)
9. Kaliszyk, C., Urban, J., Vyskočil, J.: Certified connection tableaux proofs for HOL Light and TPTP. In: CPP '15 Proceedings of the 2015 Conference on Certified Programs and Proofs. pp. 59–66. ACM (2015)
10. Łukasiewicz, J.: The shortest axiom of the implicational calculus of propositions. Proceedings of the Royal Irish Academy. Section A: Mathematical and Physical Sciences 52, 25–33 (1948), https://www.jstor.org/stable/20488489
11. Otten, J.: *leanCoP* 2.0 and *ileanCoP* 1.2 – High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 283–291. Springer, Heidelberg (2008)
12. Otten, J.: Restricting backtracking in connection calculi. AI Communications 23, 159–182 (2010)
13. Otten, J.: A non-clausal connection calculus. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNAI, vol. 6793, pp. 226–241. Springer, Heidelberg (2011)
14. Otten, J.: Implementing connection calculi for first-order modal logics. In: Ternovska, E., Korovin, K., Schulz, S. (eds.) 9th International Workshop on the Implementation of Logics (IWIL 2012). Merida, Venezuela (2012)
15. Otten, J.: *MleanCoP*: A connection prover for first-order modal logic. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) Automated Reasoning, 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014, Proceedings. LNCS, vol. 8562, pp. 269–276. Springer, Heidelberg (2014), ISBN 978-3-319-08586-9
16. Otten, J.: *nanoCoP*: A non-clausal connection prover. In: Olivetti, N., Tiwari, A. (eds.) International Joint Conference on Automated Reasoning, IJCAR 2016. LNAI, vol. 9706, pp. 302–312. Springer, Heidelberg (2016)
17. Otten, J.: Non-clausal connection calculi for non-classical logics. In: Schmidt, R., Nalon, C. (eds.) 26th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 209–227. LNAI 10501, Springer, Cham, Switzerland (2017)

18. Otten, J., Bibel, W.: *leanCoP*: Lean connection-based theorem proving. Journal of Symbolic Computation 36, 139–161 (2003)
19. Otten, J., Bibel, W.: Advances in connection-based automated theorem proving. In: Bowen, J., Hinchey, M., Olderog, E.R. (eds.) Provably Correct Systems, pp. 211–241. Springer, London (2016)
20. Urban, J., Vyskočil, J.: Theorem Proving in Large Formal Mathematics as an Emerging AI Field. In: Automated Reasoning and Mathematics – Essays in Memory of William W. McCune, Maria Paola Bonacina and Mark E. Stickel (eds.), Lecture Notes in Computer Science, vol. 7788, pp. 240–257. Springer (2013)
21. Urban, J., Vyskočil, J., Štěpánek, P.: MaLeCoP Machine learning connection prover. In: Brünnler, K., Metcalfe, G. (eds.) 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6793, pp. 263–277. Springer, Berlin (2011)

Rigorous Explanations for Machine Learning Models*

Extended Abstract

Joao Marques-Silva

Faculty of Science, University of Lisbon, Portugal
jpms@ciencias.ulisboa.pt

The recent successes of Machine Learning (ML) motivate an ever growing range of applications. In some settings, e.g. in safety critical applications, one is often expected to explain the predictions made by ML models. For example, this is the case when such predictions are to be assessed by a human decision maker, or used for later diagnosis in the case of failure. Some ML models are naturally amenable to interpretation. This is the case with logic models, including decision trees, lists and sets. In such cases, the models represent the explanations explicitly, and so the goal is to synthesize models such that the resulting explanations are as succinct as possible [8,16,3,12,7]. However, in many settings the most successful ML models are *not* naturally interpretable and, from the perspective of a human decision maker, operate as black-boxes. Concrete examples include (Deep) Neural Networks ((D)NNs), Support Vector Machines (SVMs), Bayesian Network Classifiers (BNCs), model ensembles, among many others. Approaches for explaining non-interpretable ML models are most often heuristic [13,4,11,14,10,17,1,2,9,5], in that explanations are computed by only exploiting information that is *local* to a given instance. Alternatively, some recent works focused on devising rigorous approaches for computing explanations. One such example is a compilation-based approach for BNCs [15]. This recent work also established a natural relationship between explanations and prime implicants of the classification function, concretely prime implicant explanations. A different approach [6] is to bypass the need for compilation, and relate explanations with abduction. The special setting of ML predictions enables relating abduction with the computation of prime implicants. Furthermore, and instead of exploiting compilation, this approach develops dedicated algorithms for computing explanations. More importantly, these two works [15,6] enable the computation of explanations which hold *globally*, in clear contrast with existing approaches for computing local explanations. More formally, given some logic-based representation $\mathcal{M}$ of a target ML model, a concrete instance $\mathcal{I}$, and a prediction π for that instance, a (global) explanation $\mathcal{E} \subseteq \mathcal{I}$ is such that $\mathcal{E} \models (\mathcal{M} \rightarrow \pi)$. This problem formulation enables the computation of both cardinality-minimal and subset-minimal explanations, provided a oracle (i.e. a reasoner) for the decision problem $(\mathcal{M} \rightarrow \pi)$ exists. This talk provides an overview of these recent approaches for computing global explanations. The current focus is on explaining NNs models, but the approach can conceptually be applied to any other setting where the ML model accepts a logic-based

* Work supported by FCT grants ABSOLV (PTDC/CCI-COM/28986/2017) and FaultLocker (PTDC/CCI-COM/29300/2017). Joint work with Alexey Ignatiev and Nina Narodytska.

representation. Moreover, the talk summarizes existing experimental results and highlights ongoing research work.

References

1. Adebayo, J., Gilmer, J., Muelly, M., Goodfellow, I.J., Hardt, M., Kim, B.: Sanity checks for saliency maps. In: NeurIPS. pp. 9525–9536 (2018)
2. Alvarez-Melis, D., Jaakkola, T.S.: Towards robust interpretability with self-explaining neural networks. In: NeurIPS. pp. 7786–7795 (2018)
3. Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., Rudin, C.: Learning certifiably optimal rule lists for categorical data. Journal of Machine Learning Research **18**, 234:1–234:78 (2018)
4. Frosst, N., Hinton, G.E.: Distilling a neural network into a soft decision tree. In: CExAIIA (2017)
5. Ibrahim, M., Louie, M., Modarres, C., Paisley, J.: Global explanations of neural networks. In: AIES (2019)
6. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: AAAI (2019)
7. Ignatiev, A., Pereira, F., Narodytska, N., Marques-Silva, J.: A SAT-based approach to learn explainable decision sets. In: IJCAR. pp. 627–645 (2018)
8. Lakkaraju, H., Bach, S.H., Leskovec, J.: Interpretable decision sets: A joint framework for description and prediction. In: KDD. pp. 1675–1684 (2016)
9. Lakkaraju, H., Kamar, E., Caruana, R., Leskovec, J.: Faithful and customizable explanations of black box models. In: AIES (2019)
10. Li, O., Liu, H., Chen, C., Rudin, C.: Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In: AAAI. pp. 3530–3537 (2018)
11. Montavon, G., Samek, W., Müller, K.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73**, 1–15 (2018)
12. Narodytska, N., Ignatiev, A., Pereira, F., Marques-Silva, J.: Learning optimal decision trees with SAT. In: IJCAI. pp. 1362–1368 (2018)
13. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": Explaining the predictions of any classifier. In: KDD. pp. 1135–1144 (2016)
14. Ribeiro, M.T., Singh, S., Guestrin, C.: Anchors: High-precision model-agnostic explanations. In: AAAI (2018)
15. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: IJCAI. pp. 5103–5111 (2018)
16. Wang, T., Rudin, C., Doshi-Velez, F., Liu, Y., Klampfl, E., MacNeille, P.: A bayesian framework for learning rule sets for interpretable classification. Journal of Machine Learning Research **18**, 70:1–70:37 (2017)
17. Wu, M., Hughes, M.C., Parbhoo, S., Zazzi, M., Roth, V., Doshi-Velez, F.: Beyond sparsity: Tree regularization of deep models for interpretability. In: AAAI. pp. 1670–1678 (2018)

On semiformal natural language theorems and proofs

Arnold Neumaier
Fakultät für Mathematik, Universität Wien
Oskar-Morgenstern-Platz 1, A-1090 Wien, Austria
email: Arnold.Neumaier@univie.ac.at
WWW: http://www.mat.univie.ac.at/~neum

March 8, 2019

Abstract. My lecture reports on progress in our FMathL project
https://www.mat.univie.ac.at/~neum/FMathL.html

for working automatically with mathematics written in a natural langugage and represented in the CONCISE system for semantic modeling.

Mathematical discourse is representable in abstract form in terms of a context logic, and in terms of semantic data structures as a record in CONCISE, typed with our adaptation VMathL of the MathNat grammar by Muhammad Humayoun. This record can be created automatically from a text in a specified controlled natural language (English with highly restricted grammar).

The interpretation of ordinary mathematical text written in Latex is more difficult, but we made partial progress. We collected a large multilanguage library of mathematical terms. We recognize at present a well-defined small set of typical phrases, handling English, German, and French declination of terms and conjugation of verbs reasonably correct way. The internal formal semantic structure created does not yet match that of VMathL but still lacks a dedicated formal rearrangement step from a language-oriented structure to the content-oriented structure of VMathL.

We are presently experimenting with a semiautomatic way of checking small, detailed proofs represented in VMathL using the theorem prover Vampire.

Is Meta-learning for Theorem-Proving One of the Keys to Artificial General Intelligence?

Ben Goertzel

Within the class of cognitive architectures that leverage uncertain logical inference as a key tool for creating and evaluating knowledge and hypotheses, it appears a handful of hard technical issues stand between the state of the art and AGI at the human level or beyond. One of these is neural-symbolic integration, i.e. efficiently and usefully translating between the representations of deep neural nets and other sub-symbolic pattern recognizers and logical formalisms. Another is meta-learning for inference, i.e. inference guidance via recognizing patterns across previously done inferences, and via inductive, abductive and deductive inference based on these patterns. I will describe some current work being done in the latter direction, within the OpenCog cognitive architecture, using a hypergraph pattern miner integrated with a probabilistic logic engine to recognize simple patterns among commonsense inferences. I will also present some ideas about potential synergies between work on inference meta-learning for commonsense reasoning with an AGI goal, and work on meta-learning for reasoning in a more conventional automated-mathematics context.

Deep learning: Challenges in learning and generalization

Tomas Mikolov

Artificial neural networks became very popular in the research community during the last decade. In particular, recurrent networks are widely known for their excellent performance on sequential tasks that involve language. However, their ability to learn and generalize is often confused with their memorization capability. Recurrent networks are often used as black boxes today, with little insight into what they can actually learn. In this talk, I will try to explain some of the current limitations of deep learning, and show some basic problems where the common neural architectures fail to properly learn. I will finish the talk with a discussion about neural networks augmented with a memory, and will describe ideas on how we may overcome reliance on supervised learning to improve generalization performance.

SCHEMES IN LEAN

KEVIN BUZZARD, CHRIS HUGHES, KENNY LAU, AND RAMON FERNÁNDEZ MIR

Abstract. We talk about the issues which arose when formalising Grothendieck's notion of a *scheme*, and the construction of affine schemes, in the Lean theorem prover.

1. Introduction

In my (KB) talk, I will talk about how and why we formalised a mathematical object called a *scheme* in the Lean theorem prover. I will provide context and background in the talk; in this extended abstract we stick mostly to the issues which arose in the formalisation process.

A scheme is a mathematical object whose definition and basic properties are usually taught at MSc or early PhD level in a typical mathematics department. The basic idea behind the definition is simple, and we start by describing it informally.

A *ring* is a mathematical structure where one can perform addition, subtraction and multiplication, and various natural axioms are satisfied. The integers $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ are an example of a ring. Note that all rings in this abstract are commutative, meaning that one of the natural axioms mentioned above is $a \times b = b \times a$.

Informally speaking, given a geometric object X like $\mathbb{R}^3$ or a circle, the set of real-valued functions defined on X is a ring; one can define addition and multiplication of functions "pointwise" – if f and g are functions $X \to \mathbb{R}$, one can set $(f + g)(x) = f(x) + g(x)$ and so on. If X has some extra structure (like a topology, or the structure of a manifold) then one can also consider spaces of continuous functions, or differentiable functions on X, and these are also rings. Analysis of these rings (perhaps done using algebraic methods) may be able to shed light on the geometric structure of X.

In the 1960s Grothendieck turned this situation on its head. Given as input a ring R, Grothendieck was able to define a geometric object $X := \mathrm{Spec}\,(R)$ and give a definition of "function" on X such that the ring of functions on X was, in a natural way, isomorphic to R. Such an object X is called an *affine scheme*, and a *scheme* is a geometric object which "locally looks like an affine scheme", that it, it is the union of open subsets each of which is isomorphic to an affine scheme. The devil, however, is in the details, and it is these details which the authors formalised.

The history of the project is as follows. The first author (KB) is a mathematician who wanted to learn about theorem provers, and was surprised to learn early on that even though these programs had been around for decades, what seemed to be being formalised in the main was theorems (sometimes with very intricate and long proofs) about elementary objects. He hence embarked on a project to formalise some elementary theorems about more complex mathematical objects, as a way of learning how to use a theorem prover. When the scale of the task became apparent, the second (CH) and third (KL) authors (both first year mathematics undergraduates at the time, who had been attending KB's "Xena project" formal verification club at Imperial College London) joined the project, writing ring theory libraries and proving theorems necessary for the definition of a scheme, and the proof that affine schemes were schemes. The resulting code base was in parts extremely amateurish, perhaps unsurprisingly, given that none of the authors had ever worked with formal proof verification systems before. The fourth author (RM) is a joint mathematics and computer science MSc student at Imperial College who is rewriting the code base from scratch, supervised by KB who had now learnt from the many mistakes he made in the original code base.

The first author was supported in part by EPSRC grant EP/L025485/1.

The original code is at https://github.com/kbuzzard/lean-stacks-project and the refactoring is at https://github.com/ramonfmir/lean-scheme.

2. Mathematical details.

The formal notion of "geometric object" referred to above which Grothendieck uses is a *locally-ringed space*. A locally-ringed space is a topological space equipped with a sheaf of rings such that all the stalks of the sheaf are local. Each of the technical terms in this definition can be taken apart further of course – for example a sheaf of rings is a presheaf of rings satisfying a further axiom, and a presheaf of rings is a contravariant functor from the category of open sets of the space to the category of rings.

To define a scheme in Lean one first has to define the notion of a locally ringed space. The main part of the work here, given the state of Lean's maths library when we embarked upon this project, was to define the following notions: a direct limit of rings (and the universal property of this limit), presheaves and sheaves, and the construction of a locally ringed space structure on an open subset of a locally ringed space. None of these constructions presented any particular difficulty.

More serious hurdles appeared when constructing schemes. A general scheme is a union of affine schemes, and the main work in the project was to construct the topological space $\mathrm{Spec}\,(R)$ associated to a ring R, and to show that it was a scheme. The main problem was in defining the presheaf of rings, and showing it was a sheaf, so we finish this abstract with a summary of the problems we faced.

There were two major hurdles to defining the presheaf of rings on $\mathrm{Spec}\,(R)$. The first involved making a robust API for localisation of rings. The second was the realisation that mathematicians are extremely good at identifying two isomorphic objects as "equal", especially in situations where the isomorphism is "canonical" a weasel word which apparently has no formal definition. In the middle of a mathematical proof, a ring might suddenly be replaced with a canonically isomorphic ring without warning. Our initial approach to this was rather ugly, however it got much cleaner after we learnt of Neil Strickland's predicate which classifies maps between rings which are isomorphic to localisations, and this predicate played a key role in the refactoring, greatly simplifying some arguments.

Showing the presheaf was a sheaf was a two step process. The construction we formalised was the argument in Johan de Jong's stacks project. The first step involves showing that the presheaf satisfies the sheaf axiom on basic open sets; this boiled down to a combinatorial lemma in ring theory. Originally it was proved for localisations; in the refactoring it will be proved using Strickland's predicate (this is the only part of the refactoring yet to be completed). A more formal argument then shows that the sheaf property propagates to the entire space. In formalising this argument we learnt a lot about equality in type theory, for example the notion that if two open subsets U and V of a space with a sheaf were equal, then the values the sheaf takes on U and V should not be regarded as equal (equality of types) but instead as isomorphic. This key realisation, one of many realisations we learnt at the Zulip Lean chat, turned several arguments from battles in dependent type theory to trivialities.

The first three authors would like to thank Mario Carneiro and the other regulars in the Lean Zulip chat room, for teaching them how to use the software.

Email address: k.buzzard@imperial.ac.uk

Department of Mathematics, Imperial College London

Email address: christopher.hughes17@imperial.ac.uk

Department of Mathematics, Imperial College London

Email address: kin.lau17@imperial.ac.uk

Department of Mathematics, Imperial College London

Email address: ramon.fernandez-i-mir15@imperial.ac.uk

Department of Mathematics, Imperial College London

HOList: an Open-Source, AI-Oriented Environment for Tactic-Based Theorem Proving

Kshitij Bansal, Christian Szegedy

Given the fundamental nature of mathematics and its foundational aspects for almost all scientific disciplines, the capability for high level formal mathematical reasoning is both an important practical task as well as one of the most challenging case studies in AI.
Higher-order logic is a prime contender for (automatic) formalizing of any kind of mathematical content.

In this talk, we give the general outline of a new open-source environment named HOList that is designed for machine learning and AI researchers to interact with tactic-based higher-order proof assistants. The goal of our system is to encapsulate and abstract away the technical details of dealing with the communication and proof search within proof assistants that were designed for human use. Our philosophy was to create an open-source, easy to install and use, modular environment that allows AI agents to interact with such logic proof assistants. The simple, stable APIs and the modular nature of HOList allows for researchers to share code at various levels of abstraction. Our current implementation can only interface with the HOL Light proof assistant, but we rely on a very thin, stateless, language agnostic RPC-based ProofAssistantService as a communication medium between the proof-search system and the assistant, which could be implemented with low overhead for other proof assistants as well. That would allow the rest of HOList to be trained to act as a theorem proving agent for other similar proof assistants as well. A more significant effort went into instrumenting HOL Light with tactic execution tracing mechanisms allowing for imitation learning in order to bootstrap reinforcement learning systems.

Our system comes with a simple theorem prover, named DeepHOL, that utilizes deep neural networks for predicting the most efficient tactic invocations. This includes selecting the correct tactic parameters from a large corpus of theorem parameters and works as a premise selection mechanism. Again, we have a thin API between the proof search graph and the AI model. In essence the search graph can act as an environment of AI agents. HOList's architecture is aimed to make it simple to integrate with other machine learning models.

We also publish a large corpus of human proof logs derived from the core and complex libraries of HOL Light. Furthermore we transformed this data into TF-Examples which is a file format for storing data which can be easily and efficiently read by TensorFlow models. In order to allow for transparent benchmarking, this data was split into well defined training, validation, and testing sets to evaluate the performance of various machine learning models trained with imitation

learning. The training splits are done at the proof level, so performance on each set can also be evaluated by number theorems proved using the prover with the learned models. We have also imported the whole formal proof of the Kepler conjecture for further benchmarking purposes.

We share all the training data and theorem databases which allows for benchmarking various AI algorithms in the final theorem proving context. Our goal is to allow AI researchers to focus on the machine learning algorithmic aspects of theorem proving without needing to understand the technical details of the underlying logic or proof assistant. To further lower the complexity of setup, we provide docker images for both the proof assistant and the proof-search part of the system that interact with each other via RPC calls. We have also taken performance considerations into account by addressing the slow startup time of HOL Light by creating mechanisms for “cheating in” theorems and definitions quickly. However, this methodology runs into the risk of programming errors leading to inconsistent theories as not all theorem objects are fully checked by the trusted kernel. To prevent this, we created proof verifiers that can fully check the correctness of proof logs by running through the whole theory together with their new proofs through the trusted kernel as a final verification step.

By creating a simple, efficient system for higher order theorem proving, we lower the barriers of entry for AI researchers to study this important area of application. In addition, by providing stable well defined APIs and benchmarks, we hope to foster collaboration between research teams and to allow for more transparent and comparable evaluation of various approaches in this domain.

Large Scale Deep Learning for Theorem Proving in HOList: First Results and Future Directions

Sarah Loos

Theorem proving in large theories comes with unique challenges compared to other tasks on which reinforcement learning has been applied successfully: unlimited action space, sparse reward, and quickly growing knowledge base. Here, I present our approaches to deal with these difficulties and our first practical results on the HOList benchmarks. Our particular baseline solution is named DeepHOL and builds upon the HOList infrastructure and APIs. The action space is unlimited in our context, as some tactics may take an arbitrarily long list of theorems for tactic parameters. Also, newly proved theorems are added to the knowledge base, increasing the complexity of further possible actions. In our baseline approach, we assume that each formula is given as a sequence of a finite number of tokens and these tokens are known beforehand. Our tokens correspond to the tokenization produced by HOL Light, but we communicate formulas in a simple S-expression format to make it easy to process and interpret them. Although DeepHOL ignores the tree structure and relies on sequence-based models, we expect more sophisticated machine learning models to exploit this structure for further improvements. A further simplification is that we assume a relatively small, fixed set of possible tactic applications. However, these simplifications (finite and fixed set of tokens and actions) are not assumed by the HOList system in general.

We present a detailed description of our SearchGraph architecture and how DeepHOL interacts with it. The nodes of the SearchGraph are goals/subgoals, and edges track tactic applications and resulting subgoals. Any node of the SearchGraph can be expanded and further tried to be proved. Also, the SearchGraph automatically merges identical goals, which prevents unsound cyclic proof attempts and other inefficient cyclic behavior. DeepHOL performs proof search in a breadth-first manner, but omits expanding those subgoals that have no chance to contribute to closing the main goal.

Our system relies on a two-tower, two-headed policy network that combines a standard classification model with a ranking model. The classifier head predicts the tactic to be applied, while the ranking head is for ranking premises for their usefulness as arguments passed to the tactic application. The two towers of the network are trained for encoding the goal and premises; these encodings are further processed by a ranking network taking them as inputs. The tactic prediction head only uses the encoding produced by the goal tower. Both encoding towers utilize the WaveNet architecture, which is a residual network with dilated convolutions. While the application of the tactic prediction head is straightforward, we cache the premise encodings in order to make the evaluation ranking model faster: we need to process all the preceding

premises in the database, which can have over ten thousand statements in the multivariate complex analysis corpus.

The reward is very sparse because it takes several minutes to find proofs, and a lot of time is spent in the harder theorems while not learning from unsuccessful proof search traces. So, we adopt a slow-feedback strategy that is highly distributed: we use two thousand workers running the proof search mining for new training examples, while the policy network is trained on a single GPU. In order to decrease the latency of training on newly found examples, we maintain several pools of examples: old, fresh and imitation and we trained on some predefined mixtures of those examples. In order to create the training data for the policy network, we prune the successful proof searches by keeping only those proof-search nodes that were essential for closing the goal. Furthermore, we prune the parameter lists of the tactic applications by keeping only those parameters that are necessary to end up with the same subgoals. In addition, hyperparameters of the proof search (branching factor, theorem list length and maximum unsuccessful tactic applications per node) are randomized to increase the variety of produced proofs. All high ranking theorems that were pruned away for some tactic in a successful proof are stored as hard negatives for their respective goals and are used more frequently as negative examples in the contrastive loss of the premise ranking model.

After comparing the results of our large scale reinforcement learning pipeline with the model trained by imitation learning, we present several ways that our HOList and DeepHOL infrastructure could be utilized for new research.

MATHEMATICAL DEFINITIONS, FORMALLY SPEAKING

THOMAS HALES

This talk will give an introduction to my current project, which aims to write all the theorem statements and definitions of mathematics in a computer-readable form. By "computer-readable", we mean much more than TeX or formulas in a computer algebra system. We mean that the math is expressed in terms of the rules of logic and foundations of mathematics. This project is expected eventually to encompass all branches of mathematics.

Some of the goals of the project are to

- develop formalized data that will be useful for AITP projects,
- offer a service for mathematicians to search and contribute formal definitions and statements of theorems, and
- bring the benefits of formalized mathematics to a broader community of mathematicians.

Formalizing Mathematics-In Praxis: First Experiences with Isabelle/HOL

Angeliki Koutsoukou-Argyraki

This talk is an overview of my first year of Isabelle/HOL, working within the ALEXANDRIA project at the University of Cambridge, as a pure mathematician with no prior formalization experience. I give a summary of our work so far and I comment on some of the early difficulties I encountered, as my goal is to point out some suggestions that could make Isabelle more practical for current and future users as well as to share with new users several technical and conceptual observations that might prove to be helpful in their early learning stages.

Hint Selection and Prioritization

Robert Veroff[1]*, Josef Urban[2]*, and Michael Kinyon[3]†

[1] University of New Mexico, Albuquerque, New Mexico, U.S.A.
[2] Czech Technical University, Prague, Czech Republic
[3] University of Denver, Denver, Colorado, U.S.A.

Abstract

The method of proof sketches has been an effective strategy for proving challenging theorems—including open questions—with theorem provers such as Prover9[6]. In studies with multiple conjectures, however, a very large number of hint clauses can cause significant distractions, resulting in a failed search. We present refinements to the original proof sketches method[10], including new methods for selecting and prioritizing hints to help minimize these distractions.

1 Introduction

Our search for proofs of difficult theorems with the automated theorem prover Prover9 [6] generally involves sequences of runs that rely heavily on the use of hints [9] and on the method of proof sketches [10]. Under the hints strategy, a generated clause that *matches* (subsumes) a user-supplied hint clause is given a high priority in the proof search. A *proof sketch* for a theorem T is a sequence of clauses *sufficient* to prove T. Proof sketches are a natural and potentially effective source of hints in a search for a proof.

In [10], we consider how the generation and use of proof sketches, together with the sophisticated strategies and procedures supported by an automated reasoning program such as Prover9, can be used to find proofs of challenging theorems, including open questions. The general approach is to find proofs with additional assumptions and then to systematically eliminate these assumptions from the input set, using all previous proofs as hints. It also can be effective to include as proof sketches proofs of related theorems in the same area of study.

Much of our work is in areas of mathematics that can be organized into *theory hierarchies* defined by a base theory and the repeated addition of independent axioms. Theory hierarchies provide a natural source for extra assumptions that have been especially effective in our work. For a specific example, a lattice theory hierarchy is described in [7]. It begins with axioms for lattice theory, is extended with properties such as compatibility ($(x \vee y)' = x' \wedge y'$) and modularity ($x \vee (y \wedge (x \vee z)) = (x \vee y) \wedge (x \vee z)$) and ends with axioms for Boolean algebra.

2 Hint Selection and Prioritization

In a study involving several related theorems the number of accumulated hints can get very large. Having too many hints that are provable but not useful for finding a proof of the current theorem can cause a significant distraction to a search. In this case, careful management of hints becomes especially important.

*Partially supported by the *AI4REASON* ERC Consolidator grant 649043, the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/ 0000466 and the European Regional Development Fund.

†Partially supported by Simons Foundation Collaboration Grant 359872.

One way to deal with the problem of too many hints is to limit the number of proofs that contribute hints, for example, by tightening the definition of "related" theorem. Another way that has been especially effective is to prioritize hints and to have the search prefer hint matchers of higher priority hints. The intention is that hints that are more likely to be helpful in the current search are given a higher priority.

Our methods for prioritizing hints include ordering proofs, ordering clauses within a proof, and ordering clauses between proofs.

Ordering proofs: Prefer hints from most recent proofs. For example, in a sequence of proofs resulting from the systematic elimination of extra assumptions, the most recent proofs are likely to be the most relevant to a sought after proof.

Ordering clauses within a proof: Prefer hints with higher clause numbers. The reasoning is that the higher numbered clauses in a proof are closer to completing a derivation of the target clause (generally the empty clause). This is especially significant when we are trying to eliminate an extra assumption from a previous proof (as opposed to proving a different goal).

A variation of this method prefers hints that are closer to the target clause by inference distance. Hints from several proof sketches can easily be combined by merging the clauses by their inference distances from the empty clause in their respective proofs. That is, all of the clauses that are n steps away in their respective proofs are preferred over any proof clauses that are $m > n$ steps away.

Ordering clauses between proofs: Prefer proof clauses that appear in more proofs in the current study. Furthermore, we can use this frequency count criterion to limit the number of hints by including only hints that appear in some minimum number of proofs.

These methods and examples of their application will be presented in some detail.

3 Some Results

Support for hint prioritization has been added to Prover9, and the new methods have been applied to numerous problems. For example, much of our most recent work has been on a large, long-term project in loop theory [5]. The *AIM* project (for Abelian inner mapping groups) includes numerous open questions of mathematical interest. These are difficult problems; many having proofs that are several tens of thousands of steps long. It has become increasingly difficult to make progress in this study.

The new methods have led to new results and a growing library of relevant hints. For example, before the inclusion of hint prioritization into Prover9, we had 549 proofs of "mathematically interesting" properties in 117 output files.[1] Together the files contribute 167K distinct hint clauses for later runs (47K when we eliminate those that appear in only 1 output file). As of November 2018, we have 641 proofs in 149 files contributing 2.3 million distinct hints (90K when we eliminate the hints that appear in at most 2 output files). This continues to be a work in progress.

4 Related and Future Work

ENIGMA [3, 4] is a learning-based method used to influence given clause selection in E Prover [8]. This works by (i) learning to classify given clauses from previous successful searches as "useful" or "un-useful", and (ii) applying the classifier in a new search to the generated

[1] Many of the output files contain proofs of multiple goals.

clauses. Analogously, an ENIGMA classifier could be applied to candidate hint clauses before a new search is initiated.

ProofWatch [1] and ENIGMAWatch [2] are learning-based methods for dynamically modifying the given selection priorities of hint-matching clauses during an E Prover search. In particular, the progress made toward completing each of several supplied proofs is monitored and used to focus on the more completed proofs in ProofWatch. In ENIGMAWatch, the proof-completion vectors from ProofWatch are additionally used as features characterizing the proof state and added to the features used for training ENIGMA classifiers. We have started experiments with replaying the Prover9 hint-guided proofs in E equipped with these methods.

References

[1] Z. Goertzel, J. Jakubuv, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018.

[2] Z. Goertzel, J. Jakubuv, and J. Urban. ProofWatch Meets ENIGMA: First Experiments. G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 15–22. EasyChair, 2018.

[3] J. Jakubuv and J. Urban. ENIGMA: efficient learning-based inference guiding machine. 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 *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[4] J. Jakubuv and J. Urban. Enhancing ENIGMA given clause guidance. In F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

[5] M. Kinyon, R. Veroff and P. Vojtěchovský. Loops with abelian inner mapping groups: an application of automated deduction. In M. P. Bonacina and M. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune, volume 7788 of Lecture Notes in Artificial Intelligence*, pages 151–164. Springer, 2013.

[6] W. McCune. Prover9, version 2009-11A. www.cs.unm.edu/~mccune/prover9/

[7] M. Rose and K. Wilkinson. Application of model search to lattice theory. Tech. Report ANL/MCS-P900-0801, Argonne National Laboratory, Argonne, IL, 2001.

[8] S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer, 2013.

[9] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: case studies. J. Automated Reasoning **16**:223–239, 1996.

[10] R. Veroff. Solving open questions and other challenge problems using proof sketches. J. Automated Reasoning **27**:157–174, 2001.

Towards A New Type of Prover: On the Benefits of Discovering Sequences of "Related" Proofs

David M. Cerna[1,2]

[1] Research Institute for Symbolic Computation, Johannes Kepler University Linz, Austria
[2] Institute for Formal Models and Verification, Johannes Kepler University Linz, Austria
david.cerna@risc.jku.at

Abstract

This extended abstract has been written with two goals in mind: 1) to motivate the need for a prover which derives sequences of proofs rather than a single proof, and 2) to present this problem to an audience of experts who can critique this approach and who may be interested in the further development of this project. A *uniform sequence prover* can address issues arising in the area of inductive theorem proving, in particular automated discovery of induction invariants by instance analysis. We illustrate this approach using a novel tree grammar based method introduced by S. Eberhard and S. Hetzl (implemented as *Viper* within the GAPT system). This method necessitates that first-order theorem prover produces sequences of instance proofs which are "related". This notion of relatedness has so far remained extra-logical and any precision has come from empirical analysis of numerous examples.

While automated theorem proving in the presence of induction is in general undecidable (concerning both the validity and unsatisfiability problems), there are fragments of first-order logic extended by inductive definitions which have a semi-decidable unsatisfiability problem [6]. Unfortunately, these fragments are mathematically quite weak. Even the most powerful theorem provers, based on the techniques outlined in [6][1], such as the superposition prover of V. Aravantinos *et al.* [1], fail to find invariants for even the simplest mathematically interesting problems; for an example see the analysis of the *Eventually Constant Assertion* [5]. Even statements as simple as $x + (x + x) = (x + x) + x$ are problematic for loop discovery methods.

Given the apparent limitations of loop discovery methods, S. Eberhard & S. Hetzl [7] took a different approach to the invariant discovery problem by extracting information from instance proofs. Consider your run-of-the-mill theorem prover such as SPASS, prover9, etc. When such a theorem prover finds a proof, this proof will be cut-free[2]. *Herbrand's Theorem* (also known as the Mid-Sequent Theorem) [9] tells us that we can find and extract a set of quantifier-free instances of the proven statement which together provide a proof of validity.

Without going into too much detail consider a valid statement of first-order logic extended by inductive definitions $\forall x P(x)$ provable from a set of formula Δ where x is a variable over the natural numbers[3]. If $\forall x P(x)$ is provable from Δ using a single induction, then $P(\alpha)$ ought to be provable from Δ without induction (α being a natural number). Thus, using a theorem prover we can get an instance proof Π_α from which we can extract a *Herbrand sequent* (mid-sequent) denoted by $\mathcal{H}_\alpha$. In [7], they produce a special type of tree grammar (based on Herbrand's Theorem) from $\Pi_0, \cdots, \Pi_\alpha$ which induces a quantifier-free second-order unification problem. Solving this problem results is the discovery of an invariant.

Unlike loop discovery methods which have an exceedingly hard time with $x + (x + x) = (x + x) + x$, the tree grammar approach is able to find the invariant $(x + x) + y = x + (x + y)$

[1] What we will refer to as *loop discovery methods*.
[2] For those unfamiliar with the concept of *cut*, this essentially means *analytic*.
[3] In [7] one is not limited to numerals.

and show that $x + (x + x) = (x + x) + x$ is provable by induction from the axioms of addition and successor [8]. While this is a significant step toward more general methods of automated inductive theorem proving, solving the second-order unification problem requires the discovery of uniform constructs within the instance proofs $\Pi_0, \cdots, \Pi_\alpha$. Consider, instead of $x+(x+x) = (x+x)+x$ the statement $x * (x * x) = (x * x) * x$. Note that multiplication is typically defined in terms of addition, and thus it is not entirely clear if the invariant ought to be multiplication based or an addition based, or a mix of both. Each of these possibilities can be abstracted from a particular sequence of instance proofs. A theorem prover may instead of producing instance proofs from one of these sequences unfruitfully zig-zag between various sequences. This makes solving the second-order unification problem intractable.

However, this unfortunate behavior does not mean that the prover is incapable of handling harder problems. Consider the following sentence:

$$F = \forall n(\forall x(E(g(x), n) \vee L(x, n) \ \wedge \forall x(E(x, n) \vee L(x, n)) \ \wedge \hat{Q}(n))$$

where $\hat{Q}$ is defined as follows:

$$\begin{array}{rcl} \hat{Q}(0) & \Rightarrow & \neg L(a, 0) \ \wedge \ \forall x(\neg E(x, 0) \vee \neg E(g(x), 0)) \\ \forall n(\hat{Q}(s(n)) & \Rightarrow & \forall x(\neg E(x, s(n)) \vee \neg E(g(x), s(n))) \ \forall x(\neg L(x, s(n)) \vee E(x, n) \vee L(x, n)) \\ & & \wedge \ \forall x(\neg L(g(x), s(n)) \vee E(g(x), n) \vee L(x, n)) \ \wedge \hat{Q}(n)) \end{array}$$

This sentence is unsatisfiable for all values of n, though it is not completely obvious how one can prove this by resolution. To provide a more intuitive understanding of what the sentence states, one ought to consider it as an invariant of the Infinitary Pigeonhole Principle. Encoded in the predicates E and L is a total function f from the natural numbers (encoded by g and a) to a finite set of natural numbers (encoded by s and 0). The sentence F is the negation of the statement that there exists a value α in the domain of f such that $f(\alpha) = f(g(\alpha))$. Viper found the following invariant after roughly 5 hours of search.

$$(F\{n \leftarrow x\} \rightarrow (E(0, g(a)) \vee E(0, a) \vee \hat{Q}(0))) \wedge \neg(\hat{Q}(s(x)) \wedge \hat{Q}(x) \wedge F\{n \leftarrow s(x)\})$$

Unlike $x * (x * x) = (x * x) * x$, it is not the difficulty of sifting through the various instance proofs which makes the problem difficult because there is (modulo structural variation) only one way to prove each instance, and furthermore each instance is relatively straight forward to prove. Thus, a significant portion of the 5 hours was spend on finding the invariant.

What this example highlights is how difficult constructing the invariant is even when we are provided with a uniform sequence of instance proofs. Instead of producing an arbitrary sequence of instance proofs $\Pi_0, \cdots, \Pi_{s(\alpha)}$, the prover ought to relate $\Pi_{s(\alpha)}$, to the proofs $\Pi_0, \cdots, \Pi_\alpha$. These observations leave a few open questions:

- What does it mean for two proofs Π_1 and Π_2 to be "related"?
- How does one compute whether or not two proofs are "related"?
- Is "related" enough to guarantee discovery of the uniform structures hidden within the instance proofs?

The first question is currently under investigation and is limited to the class of primitive recursive sentences F (see above) inhabits [2]. In [2], we consider a notion of relatedness based on a intrinsic clausal representation of the instances of a primitive recursive formula definition. The clauses of this intrinsic representations can be related through anti-unification techniques [4, 3].

Thus, we consider two clauses arising from the intrinsic clausal representation of two different instance formula related if they have the same anti-unifier. This is a great simplification of the idea and for more details please see [2].

Concerning the other two questions, these are open research question which have so far been approach empirically. Our conjecture is that modern artificial intelligence techniques, especially those arising from the sub-field of machine learning, can be of benefit to computation of "relatedness", i.e. how to choose which clauses which have the same anti-unifier ought to be used in the instance proofs. How one would precisely integrate such methods is still an open question and begs further investigation.

References

[1] Vincent Aravantinos, Mnacho Echenim, and Nicolas Peltier. A resolution calculus for first-order schemata. *Fundamenta Informaticae*, pages 101–133, 2013.

[2] David M. Cerna. On the complexity of unsatisfiable primitive recursively defined $\sum_1$-sentences. Technical report, RISC, 2019.

[3] David M. Cerna and Temur Kutsia. Higher-Order Equational Pattern Anti-Unification. In Helene Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction (FSCD 2018)*, volume 108 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Peer Reviewed.

[4] David M. Cerna and Temur Kutsia. Idempotent Anti-unification. Technical report, RISC, January 2018. In Review.

[5] David M. Cerna and Alexander Leitsch. Schematic cut elimination and the ordered pigeonhole principle. In *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, pages 241–256, 2016. Peer Reviewed.

[6] Hubert Comon. Inductionless induction. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 913–962. Elsevier and MIT Press, 2001.

[7] Sebastian Eberhard and Stefan Hetzl. Inductive theorem proving based on tree grammars. *Ann. Pure Appl. Logic*, 166(6):665–700, 2015.

[8] Gabriel Ebner and Stefan Hetzl. Tree grammars for induction on inductive data types modulo equational theories. Technical report, Technical University of Vienna, 2018. https://gebner.org/pdfs/2018-01-29_indmodth.pdf.

[9] Gaisi Takeuti. *Proof Theory*, volume 81 of *Studies in logic and the foundations of mathematics*. American Elsevier Pub., 1975.

Arithmetic and Inference in a Large Theory

Adam Pease

Infosys, Foothill Research Center, Palo Alto, CA, USA
adam.pease@infosys.com

1 Abstract

Imagine a voice-enabled household robot that can pick up objects and transfer them from room to room. The owner might say "Pick up my red cart and put it in the garage." but the robot might hear either "Pick up my red car and put it in the garage." or the original sentence. A reasoning system might disambiguate the utterance or correct it by reasoning that a typical car weights 1.5 tons and that's 3000 pounds and the carrying capacity of the robot is 100 pounds and the alternate text must be what was said. A system must construct an answer to a question that has never been asked before ("Can the robot carry a car?") perform some simple computation involving unit conversions, to understand that 3000lbs >100lbs and possibly explain its answer if required ("Why didn't you move my car like I asked you to?")

General purpose first-order theorem provers historically haven't done proofs with arithmetic. A relatively new language that addresses sound arithmetic calculation in a first order logic is called Typed First-order Form (TFF) [7]. It is implemented in several of the best modern, first-order provers, including Vampire [2].

To have useful and non-trivial reasoning about, for example, a robot's capabilities, or to do question answering, we need not only a language capable of arithmetic calculation (as well as first order logical reasoning) but also a non-trivial body of axioms that has the information about the real world needed to form answers to such questions. For that reason, we need to use the Suggested Upper Merged Ontology (SUMO) [3, 4], which is a comprehensive and diverse set of logical statements about the world. At approximately 20,000 concepts and 80,000 logical statements (as well as including the large factbase of YAGO [1] and other such resources), it is large enough to answer interesting questions about a wide range of topics. Writing a translator from SUMO's native formalization into TFF should open up many new opportunities for doing practical automated reasoning involving numbers and arithmetic.

In earlier work, we described [5] how to translate SUMO to the strictly first order language of TPTP [6]. SUMO has an extensive type structure and all relations have type restrictions on their arguments. Translation to TPTP involved (among many other steps) implementing a sorted (typed) logic axiomatically in TPTP by altering all implications in SUMO to contain type restrictions on any variables that appear. Note also that all the strictly higher-order content in SUMO is lost in translation to first-order, whether TPTP or TFF.

Like TPTP, TFF forms are valid Prolog syntax (although obviously not the same semantics!) TFF has five disjoint *sorts*: integers, real numbers, rational numbers, booleans and everything else. These are respectively called $int, $real, $rat, $o and $i in TFF syntax. Each variable that is used in a logical statement must be declared to be one of these sorts, or by default it will be assumed to be type $i.

TFF has built in to the language the basic arithmetic functions and arithmetic comparison operators. Each function and operator is *polymorphic* - it is actually a set of three different operators that can handle integers, rationals and reals. Equality is also defined for $o and $i.

TFF's creators have planned to include the ability to define subtypes (subsorts) but this is not yet defined in specification or implemented in any prover. An additional issue is that since

all types are disjoint, and SUMO allows multiple inheritance, there is a mismatch between the two type systems. So we have to continue to implement much of SUMO's sort system axiomatically in TFF as in TPTP, but have a special treatment of integers, rationals and reals that does use the TFF type system, so we can use its arithmetic and comparison operators.

We also need to handle SUMO's subsorts of `Integer`, `RealNumber` and `RationalNumber`. This entails adding the constraints that specify these types to any axiom that uses them. For example, `PositiveInteger` has a constraint that it's simply a TFF $int that must be greater than 0.

Because $int and $real are disjoint sorts in TFF, but `Integer` is a subtype of `RealNumber` in SUMO we have to commit to one or the other when there is ambiguity, as in the case of a number appearing without a decimal.

Lastly, all of these types can interact, requiring constraint propagation within an axiom. For example, if we have addition between an `Integer` and a variable that is otherwise constrained only to `Number`, the `Number` will have to be constrained to an `Integer`. An example translation with such an issue is shown in Listing 1.

In the talk for this paper, I present examples of the required transformations and discussion and examples about inferences with the resulting TFF theory translated from SUMO.

```
(=>                                          ! [QUAKE : $i, VALUE : $real] :
  (measure ?QUAKE                              (instance(QUAKE,Object) =>
    (MeasureFn ?VALUE RichterMagnitude))         (measure(QUAKE,
  (instance ?VALUE PositiveRealNumber))             MeasureFn(VALUE,RichterMagnitude)) =>
                                                 $greater(VALUE,0)))
```

Listing 1: SUO-KIF to TFF with sorts as conditions, built-in TFF types and comparison operator

References

[1] Gerard de Melo, Fabian Suchanek, and Adam Pease. Integrating YAGO into the Suggested Upper Merged Ontology. *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, 2008.

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

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

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

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

[6] Geoff Sutcliffe. TPTP, TSTP, CASC, etc. In *Proceedings of the Second International Conference on Computer Science: Theory and Applications*, CSR'07, pages 6–22, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP Typed First-order Form with Arithmetic. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2012)*, pages 406–419, 2012.

Towards Machine Learning Induction

Yutaka Nagashima[1,2]

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

Abstract

Induction lies at the heart of mathematics and computer science. However, automated theorem proving of inductive problems is still limited in its power. In this abstract, we first summarize our progress in automating inductive theorem proving for Isabelle/HOL. Then, we present `MeLoId`, our approach to suggesting promising applications of induction without completing a proof search.

1 PSL and Goal-Oriented Conjecturing for Isabelle/HOL

Previously, we developed `PSL` [4] for Isabelle/HOL [6] and its extension to conjecturing mechanism [5] as initial steps towards the development of a smart proof search in Isabelle [2]. With `PSL` one can write the following strategy for induction:

```
strategy DInd = Thens [Dynamic (Induct), Auto, IsSolved]
```

`PSL`'s `Dynamic` keyword creates variations of the `induct` method by specifying different combinations of promising arguments found in the proof goal and its background proof context. Then, `DInd` combines these induction methods with the general purpose proof method, `auto`, and `is_solved`, which checks if there is any proof goal left after applying `auto`. `PSL` keeps applying the combination of a specialization of `induct` method and `auto`, until either `auto` discharges all remaining sub-goals or `DInd` runs out of the variations of `induct` methods.

Sometimes it is necessary for human-engineers to come up with auxiliary lemmas, from which they can derive the original goal. To automate this process, we developed a new atomic strategy, `Conjecture`, as an extension to `PSL`. Given a proof goal, `Conjecture` first produces various conjectures that might be useful to discharge the original proof goal, then inserts these conjectures as the premise of the original goal. Thus, for each conjecture, `PSL` produces two sub-goals: the first sub-goal states that the conjecture implies the original goal, and the second sub-goal states that the conjecture indeed holds. With `Conjecture` integrated into `PSL`, one can write the following strategy:

```
strategy CDInd = Thens [Conjecture, Fastforce, Quickcheck, DInd]
```

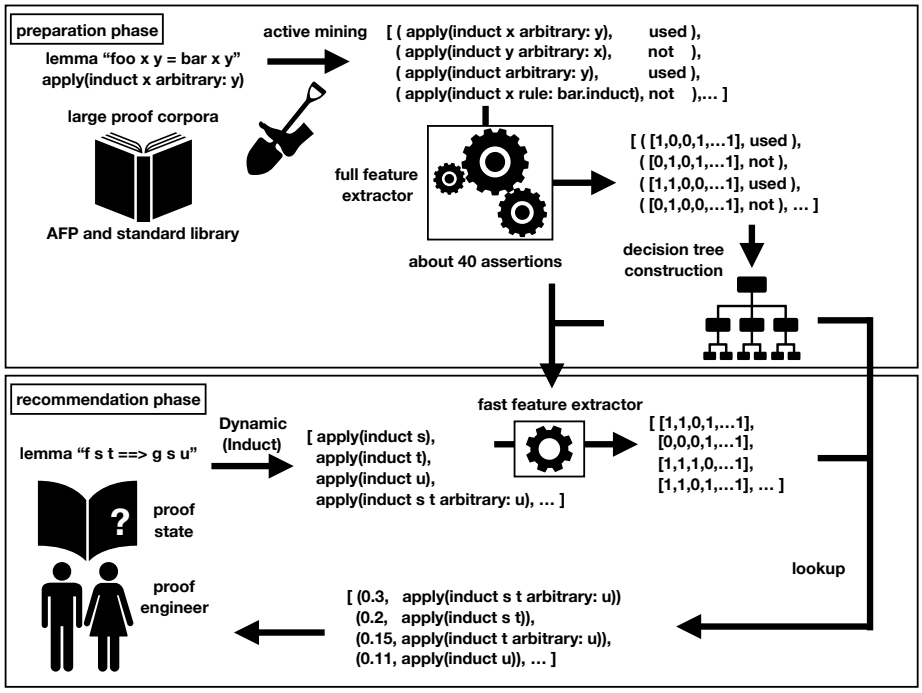
The sequential application of `Fastforce` prunes conjectures that are not strong enough to prove the original goal, whereas the application of `Quickcheck` attempts to prune conjectures that are equivalent to `False`. This way, we can narrow the search space by focusing on promising conjectures; however, when proof goals require many applications of inductions and multiple conjecturing steps, the search space blows up rapidly due to the various `induct` methods produced by the `Dynamic` keyword. Since the `induct` method usually preserves the provability of proof goal, even when the `induct` method has arguments that are inappropriate to discharge the proof goal, counter-example finders, such as `Quickcheck`, cannot discard them. To address this problem, we are developing `MeLoId` to suggest how to apply induction without completing a proof.

2 MeLoId: Machine Learning Induction

The figure below illustrates the overall architecture of MeLoId. Similarly to PaMpeR [3], which suggests promising proof methods for a given proof goal and its underlying context, MeLoId tries to learn how to apply induction effectively using human-written proof corpora as training data. In the preparation phase, MeLoId collects invocations of the induct method appearing in the proof corpora and converts each of them into a simpler format, a vector of booleans using an assertion-based feature extractor. Then, MeLoId constructs a regression tree [1], which describes not only which variations of the induct method are promising but also which assertions are useful to make such recommendations in the recommendation phase.

The mechanism of MeLoId differs from that of PaMpeR in multiple ways. First of all, MeLoId analyzes proof corpora via what we call *active mining*: MeLoId first creates various induct methods with distinct combinations of arguments, applies each of them to the goal, and compares their results. Secondly, the input to MeLoId's assertions are the triples of a goal with its context, the arguments to the induct method, and the sub-goal appearing after applying induct, whereas PaMpeR's assertions consider only the first two as input. MeLoId takes the emerging sub-goals into considerations: Since the application of the induct method alone is not time-consuming, we expect that it is desirable to improve the accuracy of recommendation using the emerging sub-goals even at the cost of the extra time spent by the induct method. Third, MeLoId assertions tend to analyze the structures of the triples, while PaMpeR's assertions tend to focus on the names of constants and types appearing in the proof goal at hand.

We have implemented the active mining mechanism and around 40 assertions. Our preliminary experiment suggests that the feature extractor successfully distills the essence of some undesirable combinations of arguments of induct. However, more comprehensive evaluation and further engineering efforts remain as our future work.



3 Acknowledgments

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

References

[1] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[2] Yutaka Nagashima. Towards smart proof search for isabelle, 2017.

[3] Yutaka Nagashima and Yilun He. PaMpeR: Proof method recommendation system for Isabelle/HOL. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 362–372, New York, NY, USA, 2018. ACM.

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

[5] Yutaka Nagashima and Julian Parsert. Goal-oriented conjecturing for Isabelle/HOL. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics*, pages 225–231, Cham, 2018. Springer International Publishing.

[6] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - a proof assistant for higher-order logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

Using Machine Learning to Minimize User Intervention in Theorem Proving based Dynamic Fault Tree Analysis

Yassmeen Elderhalli, Osman Hasan, and Sofiène Tahar

Concordia University, Montreal, Quebec, Canada
{y_elderh,o_hasan,tahar@ece.concordia.ca}

Dynamic fault trees (DFTs) have become one of the commonly used modeling techniques that capture the dynamic failure behavior of systems. Recently, DFTs have been formalized in higher-order logic (HOL), which allows performing DFT analysis within the sound core of a HOL theorem prover. However, due to the interactive nature of HOL theorem proving, the proof process involves significant user guidance. In this paper, we propose to use machine learning techniques to facilitate automating the proof of the subgoals. The machine learning can use the existing proofs of these goals as well as the verification steps being performed at runtime to come up with reasoning to verify the remaining subgoals. This kind of support from machine learning can lead to the creation of a tool for DFT analysis that requires minimum user intervention in the formal DFT analysis and thus can facilitate the industry to benefit from a sound DFT analysis approach.

Dynamic Fault Trees A dynamic fault tree (DFT) is a graphical representation of the sources of system failure in a tree format [12]. The system modeling starts with a top event that represents an undesired failure event, then the sources that lead to the occurrence of this top event are modeled using fault tree gates. The importance of DFTs lies in the fact that they can capture the failure dependencies among system components using DFT gates, which is suitable for analyzing real-world systems specially the safety-critical ones.

Formal DFT Analysis Giving the safety-critical nature of the applications of DFT analysis, we have recently provided the formalization of DFT in HOL4 [7], which allows conducting the DFT analysis in a sound theorem prover (TP) [4, 3] based on the algebraic approach [9]. However, due to the interactive nature of HOL TP, using our formalization for DFT analysis is limited to users with a considerable experience in HOL TP. Our formalization is based mainly on verifying the probabilistic behavior of DFT gates and utilizing the probabilistic principle of inclusion and exclusion (PIE) to express the probability of the top event of a given DFT. The number of subgoals to verify a given DFT depends on the number of subevents to be included in the PIE. For example, if the PIE is used with 7 subevents, it is required to verify 127 different subgoals. We have verified several intermediate lemmas that facilitate the proof process of DFT case studies. Moreover, we have identified certain patterns in the proof process of these lemmas that enable extending them to cover larger case studies. However, so far there is no automation involved in this process, as the generated subgoals are different since they represent the different combinations of intersection of the subevents of the PIE.

Using Machine Learning in Formal DFT Analysis In this project, we propose to use machine learning to facilitate automating the proofs of DFT analysis. These proofs deal mainly with iterated Lebesgue integrals and their measurability. Although these proofs are complex, there are some common patterns that can be utilized in the automation process. This automation enables using this formalization by other users that are not experts in TP. Ultimately, we plan to develop a tool that would provides the user with an interface for conducting the analysis without getting involved into the details of the proofs.

Related Work Machine learning (ML) is used with automated TP to select the heuristic for proof search based on features of the conjecture to be proved [2]. Recently, ML has been proposed to be used with HOL TP. For example, in [8], a dataset is created for the proof steps based on the multivariate theory and the proof of the Kepler conjecture [6] in HOL Light TP [1]. This dataset and similar ones can be utilized in classifying useful steps in proving a certain conjecture.

Proposed Methodology We propose to create a similar dataset for the proof steps of the measure, Lebesgue integral and probability theories [10, 11] of HOL4 TP, since these theories form the basis of our proofs for DFTs, as depicted in Figure 1. In this dataset, we will also include our formalization for DFT, particularly the intermediate lemmas that have certain patterns in their proof steps. This dataset will be divided into two subsets; a training and a testing set. Based on the created set, we plan to use ML in the premise selection of the proper verified theorems that can be helpful in verifying a given conjecture. This is basically a classification problem of whether a certain theorem is helpful or not for proving the current conjecture. Therefore, including

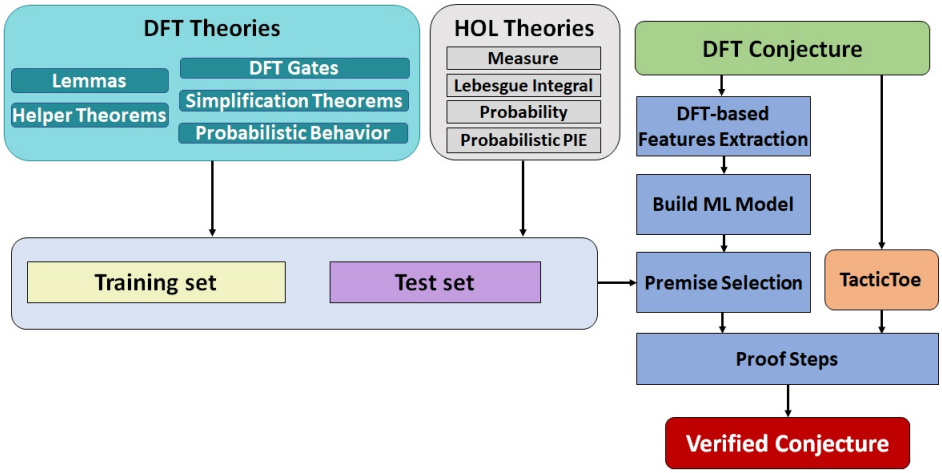


Figure 1: Proposed Methodology

our intermediate lemmas of DFT analysis in the training set is of a great importance, as the patterns that we identified can facilitate determining the right steps and theorems to verify similar conjectures. This task requires creating an ML model, such as convolutional neural networks [13], in order to classify the theorems. We plan to consider features related to DFT gates that can further help in the classification process, as the proofs that we developed differ depending on the DFT gate. In addition, we will make use of the TacticToe approach implemented in [5] that automates the selection of the proper tactics to prove a conjecture in HOL4.

After successfully implementing the above-mentioned steps, we plan to enhance the learning process by enabling learning on the fly, i.e., continuously enabling learning while conducting new proofs. This step will positively impact the classification results as the training set will be continuously improved with each new proven conjecture. We believe that implementing this automation for formal DFT analysis will allow end users that are unfamiliar with TP to benefit from our DFT formalization to provide sound analysis.

References

[1] HOL-light Theorem Prover, http://www.cl.cam.ac.uk/ jrh13/hol-light//, 2018.

[2] J. Bridge. *Machine Learning and Automated Theorem Proving*. PhD thesis, University of Cambridge, UK, 2010.

[3] Y. Elderhalli, W. Ahmad, O. Hasan, and S. Tahar. Formal Probabilistic Analysis of Dynamic Fault Trees in HOL4. Tech. rep., Concordia University, Canada. https://arxiv.org/abs/1807.11576, 2018.

[4] Y. Elderhalli, O. Hasan, W. Ahmad, and S. Tahar. Formal Dynamic Fault Trees Analysis Using an Integration of Theorem Proving and Model Checking. In *NASA Formal Methods*, LNCS 10811, pages 139–156. Springer, 2018.

[5] T. Gauthier, C. Kaliszyk, and J Urban. TacticToe: Learning to reason with HOL4 Tactics. In *Logic for Programming, Artificial Intelligence and Reasoning*, volume 46, pages 125–143, 2017.

[6] T. C. Hales, J. Harrison, S. McLaughlin, T. Nipkow, S. Obua, and R. Zumkeller. A Revision of the Proof of the Kepler Conjecture. *Discrete & Computational Geometry*, 44(1):1–34, 2010.

[7] HOL4. https://hol-theorem-prover.org/, 2018.

[8] C. Kaliszyk, F. Chollet, and C. Szegedy. Holstep: A machine learning dataset for higher-order logic theorem proving. *arXiv preprint arXiv:1703.00426*, 2017.

[9] G. Merle. *Algebraic Modelling of Dynamic Fault Trees, Contribution to Qualitative and Quantitative Analysis*. PhD thesis, ENS, France, 2010.

[10] T. Mhamdi, O. Hasan, and S. Tahar. On the Formalization of the Lebesgue Integration Theory in HOL. In *Interactive Theorem Proving*, LNCS 6172, pages 387–402. Springer, 2010.

[11] T. Mhamdi, O. Hasan, and S. Tahar. Formalization of Entropy Measures in HOL. In *Interactive Theorem Proving*, LNCS 6898, pages 233–248. Springer, 2011.

[12] E. Ruijters and M. Stoelinga. Fault Tree Analysis: A Survey of the State-of-the-art in Modeling, Analysis and Tools. *Computer Science Review*, 15-16:29 – 62, 2015.

[13] J. Schmidhuber. Deep learning in Neural Networks: An Overview. *Neural networks*, 61:85–117, 2015.

Maedmax at School: Learning Selection in Equational Reasoning

Sarah Winkler*

University of Innsbruck
sarah.winkler@uibk.ac.at

Abstract

MædMax is an equational theorem prover based on maximal ordered completion. Like in many automated deduction tools, the selection of equations and inequalities constitutes a critical choice point in the search for a proof. Here we describe the use of random forests to guide selection in two ways: (a) to learn which equations are useful, and (b) to learn a measure for proof progress, which in turn triggers the selection of additional equations.

1 Introduction

The tool MædMax performs equational reasoning by implementing *maximal ordered completion* [8]. As in the given-clause algorithm, selection of facts to process next is a crucial choice point. Here we outline two experiments exploiting machine learning techniques to improve MædMax' selection heuristic. First, random forests were used to learn a measure for the usefulness of equations and inequalities. Second, an estimate of proof progress was learned. Before giving details about these experiments we summarize the main control loop of MædMax.

Maximal completion maintains a pool of equations and inequalities $\mathcal{E}$, split into active and passive items $\mathcal{E}_A$ and $\mathcal{E}_P$. A reduction order giving rise to a terminating rewrite system $\mathcal{R}$ is determined from $\mathcal{E}_A$ by employing a maxSMT call (for instance, by orienting as many equations as possible). If thereby a goal becomes joinable or the system gets ground confluent the procedure succeeds. Otherwise the extended critical pairs of $\mathcal{R}$ and $\mathcal{E}_A$ are added to $\mathcal{E}_P$, a small subset of $\mathcal{E}_P$ is selected into $\mathcal{E}_A$, and the procedure gets reiterated. In MædMax, this selection was so far guided by a straightforward size-age ratio. In addition, a heuristic is used to estimate proof progress turned out to be useful: if progress is assumed to be small, additional old nodes are fed into the active set. For further details the reader is referred to [8].

2 Selection of Equations

We instrumented MædMax to keep track of selections. When a proof is found, it outputs for all selected items a feature vector and a classification as positive or negative, depending on whether it contributed to the proof or not. This vector comprises hand-crafted properties of both the current proof state and the equation itself along with features describing the term structure. For an equation e and a current set of active (in)equalities $\mathcal{E}_A$, the former include the size of $\mathcal{E}_A$, the iteration count, the size, size balance, and age of e, properties of e related to linearity and orientability, plus the number of matches and critical pairs of e on $\mathcal{E}_A$. To capture the term structure pq-grams [1] get computed (where $p = 1$ and $q = 2$). Function symbols were renamed uniformly according to their arity, and we counted occurrences for all 12-grams up to arity 3 (105 features per term).

*Supported by Austrian Science Fund project T789.

In a first experiment, MædMax was run in random selection mode, recording selections as described above. This set was balanced and classified with a random forest of maximum depth 14 and 100 trees, using scikit-learn.[1] Using 5-fold cross-validation, this resulted in a precision of 0.86 and a recall of 0.94 (especially the latter is relevant, since it gives the ratio of useful facts that get positively classified). Term and hand-crafted features contributed 60% and 40% of importance, respectively. Among the latter, fact and state size, number of matches and critical pairs turned out to be most relevant.[2]

We continued this experiment using a reinforcement loop to emulate the way a human might optimize a selection heuristic: we used the obtained classifier as a filter, picking randomly selected facts only if they are positively classified with probability > 0.4. After adding the new proofs' selections to the data set, the classification was repeated and the procedure reiterated. In that way, after four iterations 433 problems get solved within 60s, opposed to 206 beforehand, applying classification to 352000 selections.

Finally, when combining the previously used size-age ratio with the obtained classifier, MædMax solves 613 instead of 606 problems within 60s, with the maximal number of equalities dropping from over 440 to 220 and the maximal number of goals from 1800 to 800. The time spent on selection rises by 1-2% of the overall proof time.

3 Estimating Proof Progress

In MædMax an estimate of the proof progress is used to select additional old facts. The heuristic used so far simply checks whether the cost of the maxSMT call remained unchanged for some iterations. To gain data on proof progress, we implemented a proof track mode: Taking a TSTP proof P as additional input, the tool keeps track of its progress with respect to P, by recording in every iteration features of the prover's current state along with the ratio of facts in P that are already present in $\mathcal{E}_A$ and/or $\mathcal{E}_P$. As features we used 10 properties including iteration count, the size of $\mathcal{E}$, memory used, number of SMT checks, cost of the last maxSMT check, the numbers of facts in $\mathcal{E}_A$ reducible by the last rewrite system $\mathcal{R}$, and critical pairs between $\mathcal{R}$ and $\mathcal{E}_A$.

We ran the tool in proof track mode on all proofs obtained with E and Vampire. This resulted in about 20000 data records of MædMax iterations. We computed the differences of consecutive iterations to learn about changes in the proof state. Random forest classification with 100 trees of a maximum depth 10 resulted in a cross-validated precision and recall of both 0.72. Despite this moderate accuracy, incorporating a decision tree based on the most influential features into MædMax increased the number of solved problems by about 1.5%.

4 Related Work

Although the first efforts in this direction date more than 20 years back, learning from previous proof experience in ATPs is still a field offering many challenges [7]. Here we focus on work about guiding fact selection in ATPs by learning from earlier proofs.

Fuchs [3] employs learning heuristics in the CoDe system to select the clauses. To that end, preference is given to *focus facts* which contributed to earlier proofs, as well as their descendants. He also already aggregated similarity based on some syntactic features to assess new clauses.

[1] See scikit-learn.org.

[2] All experiments use the 897 unsatisfiable TPTP 6.4.0 UEQ problems and were run on Starexec with a timeout of 60s, see http://cl-informatik.uibk.ac.at/users/swinkler/maedmax_at_school for details.

Specifically for the case of purely equational theorem proving, Denzinger and Schulz evaluated two learning-based heuristics to improve selection of equations in the Discount system [2]. First, equations were compiled into patterns to abstract from the signature, and their usefulness recorded for later proof attempts. Second, the recorded equation patterns were arranged in a tree from which based on similarity a measure for the usefulness of all terms could be derived. However, similarity was based on a single string representation.

This approach was carried over for the superposition prover E [6]. Clauses (also abstracted to signature-independent patterns) were recorded with the number of proofs they participates in and their distance to the proof. Problems were classified, by some simple features, and given an input problem the relevant pattern set to guide selection was retrieved based on similarity. In this approach only exactly matching patterns were taken into account.

More recently, Jakubův and Urban proposed feature-based classification of clauses to improve the selection heuristics in saturation-based theorem proving [4]. As features the occurrence counts of *term walks* are used. Term walks resemble *pq*-grams, though the former do not abstract from the signature. The classification model is built with LIBLINEAR. An evaluation on E showed a large increase of performance.

Also a line of experiments with the tableau prover leanCoP investigated guidance of inference algorithms by machine learning techniques; it turned out to significantly improve the relevance heuristics. For instance, in [5] the selection of a clause for the tableau extension is guided by a naive Bayes classification based on features of the current proof state. To this end, the proof state is characterized by the frequency of terms on the active path.

5 Conclusion

In summary, the conducted experiments helped to improve the heuristics of MædMax such that about 2.5% additional problems can be solved, and delivered insights about relevant features. In the future, we plan more thorough reinforcement learning experiments to obtain further data, and will investigate alternative features such as term walks [4].

References

[1] N. Augsten, M. Böhlen, and J. Gamper. The *pq*-gram distance between ordered labeled trees. *ACM Transactions on Database Systems*, 35(1):1–36, 2010.

[2] J. Denzinger and S. Schulz. Automatic acquisition of search control knowledge from multiple proof attempts. *IC*, 162:59–79, 2000.

[3] M. Fuchs. Experiments in the heuristic use of past proof experience. In *Proc. 13th CADE*, volume 1104 of *LNCS*, pages 523–537, 1996.

[4] J. Jakubův and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In *Proc. CICM 2017*, volume 10383 of *LNCS*, pages 292–302, 2017.

[5] C. Kaliszyk and J. Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *Proc. LPAR 2015*, volume 9450 of *LNCS*, pages 88–96, 2015.

[6] S. Schulz. Learning search control knowledge for equational theorem proving. In *Proc. KI 2001*, volume 2174 of *LNCS*, pages 320–334, 2001.

[7] S. Schulz. We know (nearly) nothing! But can we learn? In *Proc. ARCADE 2017*, volume 51 of *EPiC*, pages 29–32, 2017.

[8] S. Winkler and G. Moser. MædMax: A maximal ordered completion tool. In *Proc. 9th IJCAR*, volume 10900 of *LNCS*, pages 472–480, 2018.

Project Proposal: Creating a Database of Definitions From Large Mathematical Corpora

Luis Berlioz

University of Pittsburgh
lab232@pitt.edu

We propose a method to gather large amounts of mathematical definitions from mathematical documents available online. Recent work indicates that well known text classification algorithms [2, 3] can have excellent accuracy at determining when a certain paragraph is in fact a definition [6]. These algorithms are trained on large math corpora available online like the arXiv website. The LaTeX source code of these documents is first converted into a more structured format like XML or HTML with the software package LaTeXML [10]. The content of the resulting files is then tokenized and fed into a word embedding algorithm like GloVe [12]. This has been implemented already and is available in [5].

As training data for the classifier, we use the passages of certain articles that are labeled as definitions by the author by placing them in certain LaTeX macro environments. These macros are normally defined in the preamble of the document using the `\newtheorem` macro. LaTeXML deals with the user defined macros and tags the corresponding text in the output. We have performed small experiments which show great promise. And these were confirmed with the results shown on the website https://corpora.mathweb.org/classify_paragraph.

The classifier takes the text of each paragraph of an article and outputs an estimate of the probability of it being a definition. Alternatively, a sliding window method can be used to obtain passages that produce a high probability. This method has the advantage of finding the definitions that are not expressed in precisely one paragraph, nevertheless it implies evaluating the classifier on a larger number of passages. In this situation, we consider the *fasttext* method in [8] which has a slightly lower accuracy but evaluates a passage much faster than any method previously considered.

Next, we plan to organize the definitions in an ordered tree structure where the nodes of the tree are definitions and the order represents the dependence between the nodes. In each definition we will identify the *definiendum* (i.e., the term being defined) by adapting a named entity recognition algorithm described in [13]. Moreover, by applying well established methods like [11, 4] to detect common phrases we can identify concepts with name spanning multiple words. We can also deal with the polysemy and synonymy [14, 7] which is very common in mathematical jargon by performing disambiguation on the cases polysemy and marking or merging the nodes that show synonymy.

We plan to produce a data set that would be useful in the formalization of mathematical theories, by giving a rough survey of the mathematical landscape. As another example, a database of virtually all the definitions in mathematics can be used to create user interfaces that allows authors to produce semiformal [9] versions of their work. This user interface would let authors browse all the alternative definitions of a given term, allowing them to reuse and improve on previous entries. We also plan to make all data freely available as part of the Formal Abstracts Project [1], in the hope of getting feedback from the interested community to improve and shape future iterations of this work.

References

[1] Formal abstracts. https://formalabstracts.github.io/, 2019.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[3] Tao Chen, Ruifeng Xu, Yulan He, and Xuan Wang. Improving sentiment analysis via sentence type classification using bilstm-crf and cnn. *Expert Systems with Applications*, 72:221–230, 2017.

[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[5] Deyan Ginev. arxmliv:08.2018 dataset, an html5 conversion of arxiv.org, 2018. SIGMathLing – Special Interest Group on Math Linguistics.

[6] Deyan Ginev. A web demo for scientific paragraph classification. https://github.com/dginev/web-scipara-demo, 2018.

[7] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[8] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, 2017.

[9] Christoph Lange. *Enabling collaboration on semiformal mathematical knowledge by semantic web integration*, volume 11. IOS Press, 2011.

[10] Bruce Miller. Latexml: A latex to xml converter. url: http://dlmf. nist. gov. *LaTeXML/(visited on 03/12/2013)*, 2013.

[11] Richard C Murphy. Phrase detection and the associative memory neural network. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pages 2599–2603. IEEE, 2003.

[12] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[13] Horacio Saggion. Identifying definitions in text collections for question answering. In *LREC*, 2004.

[14] Yifan Sun, Nikhil Rao, and Weicong Ding. A simple approach to learn polysemous word embeddings. *arXiv preprint arXiv:1707.01793*, 2017.

Teaching the Structure of First-order Formulas to Neural Networks

Julian Parsert[1]*, Stephanie Autherith[1], and Cezary Kaliszyk[1]*

University of Innsbruck, Innsbruck, Austria
julian.parsert@gmail.com
stephanie.autherith@student.uibk.ac.at
cezary.kaliszyk@uibk.ac.at

Abstract

Logical reasoning as performed by human mathematicians involves an understanding of terms and formulas as well as transformations on them. In this paper we consider a number of syntactic and semantic properties of logical expressions. Based on these we extract and generate data sets. We develop models that encode these formulas in a continuous vector space while preserving the aforementioned properties. We train, evaluate and compare multiple models on the extracted data sets. Furthermore, we show that these models generalize to properties they have not explicitly been trained on.

Many previous examples can be found where artificial intelligence technology was applied to (interactive) theorem proving problems. While Färber [2] use simple machine learning algorithms for proof search in theorem proving, Loos et al. [6] use a deep learning approach. Also other tasks such as tactic and premise selection have been improved using different types of artificial intelligence [3, 7, 8, 4]. All of these examples and many more apply their machine learning to specific problems and extract features, engineer data etc. that precisely describes the problem at hand. We propose a learned encoding and embedding of (first-order) formulas that can later be used by more complex as well as naive models alike. Clearly, encodings of formulas need to carry syntactic and semantic information about the original formula. In addition, one would like such encodings to be relation-preserving. Ideally, the encoding of two "related" formulas will carry that relation as well. As an example, when applying these encodings to premise selection, one could imagine that useful premises would have a vector representation which are close in distance to the conjecture in question. Similarly, one could imagine application to clause selection for theorem proving, etc.

Learning Framework We propose a deep learning based encoding. Following the results from [1], we use CNNs and LSTMs based architectures. Our encoding networks are trained on char level embeddings as shown in Figure 1. This learning framework essentially consists of two main parts, the encoding network (which we are mainly interested in) and a set of classifiers. The models are trained by propagating the loss that is obtained from the classifiers back to the encoding network. Once the training phase is done, we discard the classifiers and use the encodings.

Properties The properties which are recognized in the classifiers are extracted beforehand. For now the considered properties are the subformula relation, modus ponens, term-formula distinction, well-formedness, unifiability, and alpha-equivalence. It is worth noting that there are two iterations of the subformula classification, one multilabel classification with one input

*This work is supported by the European Research Council (ERC) grant no 714034 SMART.

and one binary classification with two inputs. These formulas or pairs of formulas are fed to the learning framework where each of the formulas is first encoded by the encoding network. Then, the encodings are used as input to different types of classifiers which, as mentioned above, propagate the loss back to the encoding. The properties where chosen by considering the application of the encoding to (interactive) theorem proving. The two main focuses were 1) the structure of first-order formulas, and 2) useful properties for theorem proving. For the former we chose the properties well-formedness and subformula, whereas for the latter unifiability and modus ponens is important. The properties such as term-formula classification and alpha equivalence form an important part in both. Syntacitc and structural properties of first order logic nowadays form an important part in premise selection[5] whereas unifiability is an important property of resolution in theorem proving. We leave it up to future work to consider the minimality or the addition of these or additional properties.

Encoding Models We consider different models for our encoding network. But they can be split into a group of CNN based models and a group of LSTM based models as shown in Figure 2. All models first go through and embedding layer. After that, we have either a set of convolution/pooling layers or a set of LSTM layers depending on the model. On top of the model specific layers we put a final fully connected layer. We did not mention this explicitly yet, however we consider two types of encoding networks. Encoding networks that is functions of the form $\mathbb{N}^n \to \mathbb{R}^n$ and embedding networks, which correspond to $\mathbb{N}^n \to \mathbb{R}^m$ where $m \leq n$. The latter of which, is achieved through appending a projection layer to the encoding. Hence, we get a lower dimensional continuous representation of formulas.

Results The training data and evaluation data is split 9:1 before the training phase. The evaluation seems to confirm the results achieved in [1] where the CNNs based models outperform the LSTM based models. The best CNN based models are the ones with a fully connected layer following the convolution/pooling layers. Of the seven properties that we considered, the CNN based networks achieved anywhere between 80% and 100%. The 100% results came from the classification of terms/formulas and alpha-equivalence. Meanwhile the LSTM based models performed similarly in 4 out of the 7 considered properties. However, they perform considerably less when being tasked with classifying modus ponens, well-formedness, and subformulas. When trying to recognize a modus ponens inference step, the best LSTMs only reach an accuracy of 61%, while the best CNNs reach up to 99%. We also used the encodings and embeddings of formulas to train simpler models such as SVMs. Here, SVMs were able to recognize whether or not a term contained a variable with an accuracy of 90%. Doing a nearest neighbor analysis it also seems that the concept of variables are learned by the network.

In the future we aim for two things, adding additional properties as well as incorporating these encodings in actual theorem proving problems.

References

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

[2] Michael Färber and Chad E. Brown. Internal guidance for satallax. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra,*

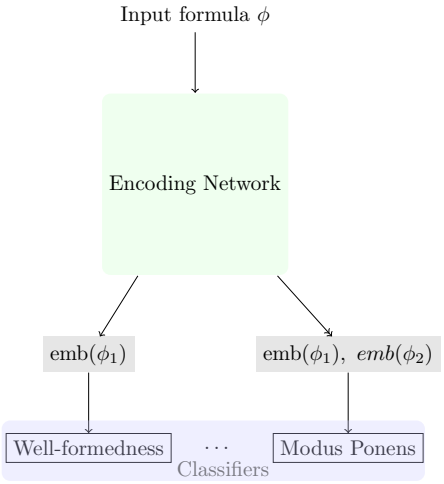


Figure 1: This graph shows the training framework we developed. The bottom area contains the classifiers that get one or more continuous representations of formulas emb(ϕ) as input. The encoding networks are described subsequently (cf. Figure 2).

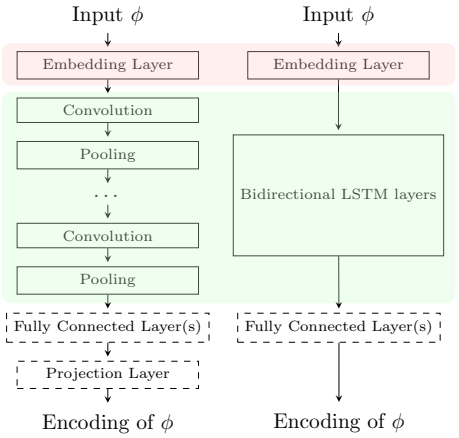


Figure 2: The encoding models we considered with the layers that the input passes through. On the left we show the CNN based models, while on the right the LSTM based models are presented. The dashed boxes describe layers that are not present in each model of that type.

Portugal, June 27 - July 2, 2016, Proceedings, volume 9706 of *Lecture Notes in Computer Science*, pages 349–361. Springer, 2016.

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

[4] Andrzej Stanislaw Kucik and Konstantin Korovin. Premise selection with neural networks and distributed representation of features. *CoRR*, abs/1807.10268, 2018.

[5] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. Mash: Machine learning for sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving*, pages 35–50, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[6] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David Sands, editors, *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017.

[7] Yutaka Nagashima and Yilun He. PaMpeR: proof method recommendation system for Isabelle/HOL. In Marianne Huchard, Christian Kästner, and Gordon Fraser, editors, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 362–372. ACM, 2018.

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

Beyond Surface Event Representation

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

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

Abstract

The Knowledge Representation community has developed sufficient know-how to represent basic action events effectively for a wide variety of applications. The next step beyond this, we argue, is the representation of symbolic events, especially the underlying motivations and the event valuations. This task is vitally connected to reasoning about real world situations in a global discourse centered around identity politics. We discuss some of the concrete issues of representing "collective memories" as targets for symbolic allusion and sketch an event corpus suitable for representation testing.

1 Introduction

On November 10, 2018, Angela Merkel and Emmanuel Macron visited a railroad museum and signed the guestbook [16]. The event sounds banal, but it was important. And this significance hails not from the actions that these leaders took, but from prior significant events that the actions alluded to.

Specifically, Merkel and Macron were visiting the museum of the so-called Compiègne wagon, the locale for signing the armistice between Germany and France, which ended World War I in 1918. As the place where the armistice signing occurred at, the wagon became a cultural memorabilium, though its meaning was dependent on the point of view. In France, where the wagon was an allusion to the hard-won victory, it was glorified by receiving its own museum. In the anti-Republican rhetoric of 1930s Germany, especially of the rising National Socialist Workers' Party (NSDAP) and its leader, Adolf Hitler, the wagon alluded to a "humiliating peace".

Therefore, when the German occupying army returned to France, the cart was dragged from its museum and made the place for the French armistice delegation in turn to sign their surrender papers [14]. Thus converted in its valence, the cart was hitched to a train bound for Berlin, a trophy of the German victory now.

When seeing the visit to the Compiègne wagon museum in this light, the absence of news becomes the message, the tourist-like visit of the leaders of France and Germany a political symbol for normalized Franco-German relations.

2 Capturing the Allusions dignifying Events

Since the 1970s, the Knowledge Representation community has sussed out how to capture events in terms of actions, scripts, plans and goals [11] and developed flexible representations [6] for dealing with the fluctuating levels of detail [2] real world events provide. Doing so has supported applications ranging from indexing and search to event comparison [7], from case-based reasoning [9] to modus-operandi applications [17] and abductive simulations of the "what-if" type [13] [3].

*Corresponding email: robert.kahlert@gmail.com

But as Schank argued [12], for some events, there is meaning for the actors in terms of stories alluded to; meaning that is a key aspect of the individual event. Such meaning cashes out as the motivation for undertaking the event in the first place; as reasons for remembering or forgetting an event; or as valence judgments of the event as "appropriate", "insulting", "encouraging", "demeaning", or similar.

Capturing the motivations and the valence of symbolic actions, we argue, is the next frontier for event knowledge representation. Developing that representation would allow the construction of a new suite of comparable applications, dealing with indexing and retrieval of documents mentioning symbolic events, as well as case-based comparison and modus-operandi reasoning, but this time both at the level of the symbolic message intended with the events as well as the surface description.

The key problem, representationally, is to model the target of the allusions, what Cultural Studies term the "collective memory" or "cultural remembrance" [1], i.e. collections of narratives that imbue events with meaning for social groups. Drawing out the connections from the public events that make the allusions to the elements of "collective memories" referenced is vital for appreciating why these events occurred. It also makes it possible to track their influence in the public discourse.

After all, symbolic communications are interpretation offers [10, 27–34] only. Different "collective memories" structure the same events into incommensurate narratives. In an age of identity politics, the opposition voters, such as the French voters of Marine Le Pen and the German voters of the "New Right" parties, will not receive the message of the Compiègne wagon visit as intended by the advisors of Merkel and Macron who planned the visit.

3 Challenges of Modeling Historical Allusions

We are under no illusion as to the difficulties of the modeling task at hand. Unlike scripts, which describe habitual events, allusions are exemplar-based and therefore require appropriate scoping of applicable relations: which aspects of the exemplar are carried over, which ones are reshaped or even suppressed.

Structurally, allusion is a two-way mapping task, with reduction and alignment problems, raising issues similar to entity-matching in database merging.

Taken individually, the commemorated events do conform to the basic actor-role models of standard knowledge representation practice. However, in the process of becoming adopted into the "collective memory", events are often simplified or even distorted to the point of historical falsehood.[1]

In addition, some forms of group-narratives depend on trans-personal actors and entities—"the Nordic Race", "international Capitalism" and its twin "international Communism", etc.—without clear referents in present-day discourse. Such actors are themselves gross simplifications, or even personifications of social processes, and usually viewed disadvantageously by the group's members. Reasoning about the behaviors of such actors may require counter-factual reasoning, a challenging problem in most knowledge representation systems.

Furthermore, this means that, taken together, the entirety of the "collective memory" of any particular group may be contradictory if pressed hard enough. This argues for care when unifying the elemental events into a single knowledge graph.

[1] This holds for many "collective memories"; for example, for modern biology, consider the discrepancy between what proto-geneticist Mendel is credited with and the actual motivations underpinning his (explicitly anti-Darwinian) research; cf. [15].

4 Some Implementation Considerations

As an implementation choice, we are developing our representation in ResearchCYC.[2] Taking a cue from Davidson [6], we reify the allusion directly as a relation between events; notice that we are not reifying an interpretation event! Within the ResearchCYC ontology, an allusion is best captured as an `AspatialInformationStore`, a broad collection that includes proverbs and cliches. For modeling event valences, we have a base set of eighty emotions available in ResearchCYC to extend as needed.

As far as a contents base is concerned, modern mass media report almost daily on symbolic actions for our representation investigation. Such occurrences range from the economic (yellow-jacket fuel protests in Paris) to the political (the assassination of critics of despotic regimes) to the athletic (the kneeling of the US football players of African-American descent) to the cultural (establishing of the Alma Roseé exhibit on the Women Orchestra of Auschwitz in Vienna).

We counter presentist biases with historical cases of symbolic communication, such as the reign of Tudor monarch Henry VII [4], who underwent enormous efforts to reconcile his nobles symbolically after a gruesome civil war. These efforts included naming his first-born son Arthur, as a promise of a regency as peaceful, just and prosperous as the Arthurian past was then imagined to have been.[3]

[2] This is KB 7168 from May 2018 [5]. In addition to the rich ontology, we leverage the conveniences of CYC's HOL extensions, while expecting that most of the theory will readily down-compile to FOPL and thus be usable on a variety of FO theorem proving systems; cf. [8].

[3] Prince Arthur's death at fifteen allowed his brother to become king as Henry VIII.

References

[1] Aleida Assmann and Dietrich Harth, editors. *Mnemosyne: Formen und Funktion der kulturellen Erinnerung*. Fischer, Frankfurt am Main, 1991.

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

[3] A. Clarkson, S. Kaisler, Doug Lenat, and C. Oreski. *Strategic Automatic Discovery System (STRADS)*. Springer Verlag, 1991.

[4] Sean Cunningham. *Prince Arthur: The Tudor King Who Never Was*. Amberley, The Hill, Stroud, Gloucestershire, GL5 4EP, 1st edition, 2016.

[5] Cycorp. Researchcyc, December 2018.

[6] Donald Davidson. *Essays on Actions and Events*. Oxford University Press, 2001.

[7] Chris Deaton, Blake Shephard, Charles Klein, Corinne Mayans, Brett Summers, Antoine Brusseau, Michael Witbrock, and Doug Lenat. The comprehensive terrorism knowledge base in cyc. In *Proceedings of the 2005 International Conference in Intelligence Analysis*, 2005.

[8] Deepak Ramachandran, Pace Reagan, and Keith Goolsbey. First-orderized ResearchCyc: Expressivity and efficiency in a common-sense ontology. *AAAI*, 2005.

[9] Christopher K. Riesbeck and Roger Schank. *Inside Case-based Reasoning*. Artificial Intelligence Series. Psychology Press, 1st edition, 1989.

[10] Jörg Rüpke. *Pantheon: Geschichte der antiken Religion*. Historische Bibliothek der Gerda Henkel Stiftung. Beck, 1st edition, 2016.

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

[12] Roger Schank. *Tell Me a Story: Narrative and Intelligence*. Northwestern University Press, 1995.

[13] Blake Shepard, Cynthia Matuszek, C Bruce Fraser, William Wechtenhiser, David Crabbe, Zelal Güngördü, John Jantos, Todd Hughes, Larry Lefkowitz, Michael J Witbrock, et al. A knowledge-based approach to network security: applying cyc in the domain of network risk assessment. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1563. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[14] William L. Shirer. *The Rise and Fall of the Third Reich*. RosettaBooks LLC, New York, 1st edition, 2011 (= 1961).

[15] John Waller. *Fabulous Science: Fact and Fiction in the history of Scientific Discovery*. Oxford University Press, 1st edition, 2002.

[16] Kim Willsher. Trump misses cemetery visit as macron and merkel vow unity. *The Guardian*, (10), November 2018.

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

Curriculum Learning and Theorem Proving

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

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

1 Introduction

We propose a proof guidance approach for ATP systems based on reinforcement learning (RL). Unlike previous approaches, we specifically target long proofs, i.e., our aim is to be able to find proofs that are hundreds or even thousands of steps long.

2 Learning Setup

Theorem proving can be seen as a special kind of reinforcement learning environment where states are proof states and actions are inference rules that are applicable in a given state. What makes this setup particularly challenging is that proof trajectories can be long with extremely sparse rewards: only once a proof has been found.

We propose to use curriculum learning on the length of proofs to counterbalance sparse rewards. We assume that we have a few problems for which proof trajectories are already available. We start exploration from the end of these trajectories, where rewards are reasonably close, then we gradually move backwards towards the beginning of the proof. This approach has already been successfully applied to Atari games in [5] and in many other reinforcement learning experiments.

Another peculiarity of theorem proving as a reinforcement learning environment is that the action space is discrete, but potentially infinite. Depending on the proof system, the set of actions can grow indefinitely during the proof (such as selecting the given clause in saturation-style provers), while in other systems the action space is fixed for a given problem (which is the case in many tableau calculi). However, even in the latter scenario, actions can be different from problem to problem.

In this current work we focus on providing guidance for LeanCoP/FCoP [4, 7, 2, 3] theorem prover that is based on the connection tableau. In FCoP, the action space is roughly correlated with the size of the problem set, extended with axioms to handle equality. While this can be large for large problems, only a few actions are available in a particular state. We describe in Section 3 how we approach this action space.

A well known challenge of combining theorem provers with learning systems is embedding discrete proof objects (states and actions) into continuous Euclidean spaces that learners can work on. We use the hand engineered feature representation that is provided by FCoP [3], mostly based on triples occurring in the proof tree, first used in [1]. In the longer run, we believe that finding better representations can yield significant improvement.

We focus on Proximal Policy Optimization (PPO) [6], a successful reinforcement learning technique, which is a variant of the actor-critic framework. We jointly train a value and a policy model that evaluate proof states and actions, respectively. In PPO, the policy is forced to stay close to its previous version at each update, which results in a more stable training.

3 Handling the Action Space

As described earlier, we are dealing with a discrete action space that can be large and that may not be known in advance. This does not fit into the traditional RL setups where either 1) the action space is small and fixed and a policy can output a probability vector, or 2) the action space is continuous and the policy can directly parameterize actions.

We have briefly experimented with using a fixed action space, which can be used for narrow problem types, such as addition/multiplication problems in Robinson arithmetic. Here the policy can directly compute a probability for all actions, irrespective of whether the action is a valid move.

However, most of our work focuses on a more robust setup, that we called the *action selector model*. The policy receives the state, as well as all available actions as input and returns a probability vector over the available actions. This model can easily generalize to yet unseen actions, since actions appear as inputs and the model just selects one of them. It can also handle large action spaces, provided not too many actions are available at once. Also note that the policy can make decisions based on what actions are available. Another advantage of this approach is that we exploit the features of actions, not only those of the space. Instead of having to learn "Given such state features, you should make the this action", we now can learn "If state and action features are related in a particular way, then this is the action to be selected". The fact that all proof objects are embedded into the same feature space, makes this approach particularly comfortable. This model is very similar to the action selection in [3].

Implementing the action selector model poses technical difficulties, since different number of actions are available at each step. However, we can exploit the fact that at any proof state only a few actions are available, which can be upper bounded by a small constant C. The policy always assumes C action inputs and the unused slots are filled with zeros (empty actions).

4 Experiments

In our first set of experiments, we wanted to show the feasibility of our approach on problems that require long proofs, but that have a rather simple structure. These experiments target arithmetic formulae in Robinson arithmetic. We use successor notation, which yields very long expressions. Trying other representations is future work. For these problems, we upper-bounded the number of available actions by 22, though typically it was not more than 5-6.

Our learners are implemented in Python and the FCoP theorem prover is accessed through its Python interface PyCoP. We experimented with decision trees and neural networks and the latter provided better results. Our networks are simple multi-layer perceptrons, using 4 layers of 512 neurons, with ReLU nonlinearity.

4.1 Generalizing from Long Proofs

We trained on the proof of $7 \times 2 = 14$ (51 steps) and successfully generalized to formulae $A \times B = C$, $A + B = C$ with arbitrarily large numbers. E.x. the proof of $29 \times 29 = 841$ (2641 steps) was found in 13.5 sec. It is instructive to see what this time was spent on: FCoP proof steps (3 sec), guidance neural network evaluation (9 sec), feature extraction (1.5 sec). Training took around 10^5 steps.

4.2 Generalizing from Short Proofs

We were interested to see how large the training problem needs to be in order to generalize. With a little tuning, we managed to obtain the very same results of Subsection 4.1 when trained on $1 \times 1 = 1$ (9 steps). This opens up the possibility of fully unsupervised learning: the 9 step long proof of this simple problem can be found with various brute force methods, which then can be used to learn general addition and multiplication. Training took 2×10^5 steps.

4.3 Generalizing to Complex Formulae

Next we experimented with more complex formulae of the shape $F = N$, where N is a number and F contains a random sequence of operators and operands. We generated a set S of 66 problems with 3 operators and each operand bounded to be in $[0, 2]$. After training on $1 \times 1 = 1$, the system could prove most of S and we continued training on them. By the end of this two step training process, we generated new random sequences with larger numbers and more operators, all of which the system could solve. Training took around 2×10^6 steps. Some examples of test problems:

- $((8 + 5) \times 8) \times 5 = 520$: 16856 steps, 95.7 sec
- $((7 \times 9) + 3) \times 8 = 528$: 13937 steps, 78.9 sec

5 Conclusion

We built a reinforcement learning system using PPO that learns to provide guidance for the FCoP connection prover. The system learns to prove equations involving addition and multiplication in Robinson arithmetic from extremely little supervision and can generalize to complex formulae. In the future, we hope to be able to extend these results to more complex problem domains.

References

[1] Jan Jakubuv 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.

[2] 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.

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

[4] Cezary Kaliszyk, Josef Urban, and Jiři Vyskočil. Certified connection tableaux proofs for hol light and tptp. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15, pages 59–66, New York, NY, USA, 2015. ACM.

[5] Tim Salimans and Richard Chen. https://blog.openai.com/learning-montezumas-revenge-from-a-single-demonstration/.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[7] Josef Urban, Jirí Vyskocil, and Petr Stepánek. MaLeCoP: Machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings*, volume 6793 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2011.

Can Neural Networks Learn Symbolic Rewriting? *

Bartosz Piotrowski[1,2], Chad E. Brown[1], Josef Urban[1], and Cezary Kaliszyk[3]

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

Introduction

Neural networks (NNs) turned out to be very useful in several domains. In particular, one of the most spectacular advances achieved with use of NNs has been natural language processing. One of the tasks in this domain is translation between natural languages – neural machine translation (NMT) systems established here the state-of-the-art performance. Recently, NMT produced first encouraging results in the *autoformalization task* [6, 5, 4, 12] where given an *informal* mathematical text in LaTeX the goal is to translate it to its *formal* (computer understandable) counterpart. In particular, the NMT performance on a large synthetic LaTeX-to-Mizar dataset produced by a relatively sophisticated toolchain developed for several decades [1] is surprisingly good [12], indicating that neural networks can learn quite complicated algorithms. This inspired us to pose a question: *Can NMT models be used in the formal-to-formal setting?* In particular: *Can NMT models learn symbolic rewriting?*

The answer is relevant to various tasks in automated reasoning. For example, neural models could compete with symbolic methods such as inductive logic programming [10] (ILP) that have been previously experimented with to learn simple rewrite tasks and theorem-proving heuristics from large formal corpora [11]. Unlike (early) ILP, neural methods can however easily cope with large and rich datasets, without combinatorial explosion. Our work is also an inquiry into the capabilities of NNs as such, in the spirit of works like [3].

Data

To perform experiments answering our question we prepared two data sets – the first consists of examples found in ATP proofs in a mathematical domain (AIM loops), whereas the second is a synthetic set of polynomial terms – they are described below.

The AIM data set: The data consists of sets of ground and nonground rewrites that came from `Prover9` proofs of theorems about AIM loops produced by Veroff [7]. Many of the inferences in the proofs are paramodulations from an equation and have the form

$$\frac{s = t \qquad u[\theta(s)] = v}{u[\theta(t)] = v}$$

where s, t, u, v are terms and θ is a substitution. For the most common equations $s = t$, we gathered corresponding pairs of terms $\big(u[\theta(s)], u[\theta(t)]\big)$ which were rewritten from one to another with $s = t$. We put the pairs to separate buckets (depending on the corresponding $s = t$): in total 8 buckets for ground rewrites (where θ is trivial) and 12 for nonground ones. These constituted training sets for our experiments. Some examples from these data sets are presented in TPTP format in Table 1.

The polynomial data set: This is a synthetically created data set where the examples are pairs of equivalent polynomial terms. The first element of each pair is a polynomial in an

*Supported by the ERC Consolidator grant no. 649043 AI4REASON, and COST Action EUTypes CA15123, STSM no. 42217

Table 1: Examples in the AIM data set.

Rewrite rule:	Before rewriting:	After rewriting:
`b(s(e,v1),e) = v1`	`k(b(s(e,v1),e),v0)`	`k(v1,v0)`
`o(V0,e) = V0`	`t(v0,o(v1,o(v2,e)))`	`t(v0,o(v1,v2))`
`k(V0,k(V1,V2)) = k(V1,k(V0,V2))`	`l(k(v1,k(v0,v2)),k(v0,v2),v3)`	`l(k(v0,k(v1,v2)),k(v0,v2),v3)`

Table 2: Examples in the polynomial data set.

Before rewriting:	After rewriting:
`(x * (x + 1)) + 1`	`x ^ 2 + x + 1`
`(2 * y) + 1 + (y * y)`	`y ^ 2 + 2 * y + 1`
`(x + 2) * ((2 * x) + 1) + (y + 1)`	`2 * x ^ 2 + 5 * x + y + 3`

arbitrary form and the second element is the same polynomial in a normalized form. The arbitrary polynomials are created randomly in a recursive manner from a set of available (non-nullary) function symbols, variables and constants. First, one of the symbols is randomly chosen. If it is a constant or a variable it is returned and the process terminates. If a function symbol is chosen, its subterm(s) are constructed recursively in a similar way. Several data sets of various difficulty were created by varying the number of available symbols and the length of the polynomials. Each data set consists of 300000 examples, see Table 2 for examples.

Experiments

Several experiments were conducted using an established NMT implementation [8] with parameters inherited from [12]. The training terms were given to NMT as linear sequences of symbols. First, NMT models were trained for each of the 20 rewrite rules in the AIM data set. It turned out that the models, as long as the number of examples was greater than 1000, were able to learn the rewriting task with high accuracy – reaching 90% on separated test sets. On the joint set of all rewrite rules (consisting of 41396 examples) the performance was also good – 83%. This means that the task of applying single rewrite step seems relatively easy to learn by NMT. Then experiments on more challenging but also much larger data sets for polynomial normalization were performed. Depending on the difficulty of the data, accuracy on the test sets achieved in our experiments varied between 70% and 99%. Some of the results are shown in Table 3.

Conclusions and future work

NMT is not typically applied to symbolic problems, but surprisingly, it performed very well for both described tasks. The first one was easier in terms of complexity of the rewriting (only one application of a rewrite rule was performed) but the number of examples was quite limited. The second task involved more difficult rewriting – multiple different rewrite steps were performed to construct the examples. Nevertheless, provided many examples, NMT could learn normalizing polynomials.

This motivates us to extend our work in two directions. Firstly, more interesting and difficult rewriting problems for NMT need to be provided for better delineation of the strength of the NMT models. Secondly, we are going to develop and test new kinds of NMT models tailored for

Table 3: Choosen results of experiments with polynomials. (Characteristic of formulas concerns input polynomials.)

Function symbols	Constant symbols	Number of variables	Maximum length	Accuracy on test
$+, *$	$0, 1$	1	30	99.28%
$+, *$	$0, 1$	3	50	88.20%
$+, *$	$0, 1, 2, 3, 4, 5$	5	50	83.47%
$+, *, \hat{}$	$0, 1, 2$	3	50	71.81%

the problem of comprehending symbolic expressions. Specifically, we are going to implement an approach based on the idea of treeNN, which may be another effective approach for this kind of tasks [3, 9, 2].

References

[1] Grzegorz Bancerek, Adam Naumowicz, and Josef Urban. System description: XSL-based translator of Mizar to LaTeX. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2018.

[2] Saikat Chakraborty, Miltiadis Allamanis, and Baishakhi Ray. Tree2tree neural translation model for learning source code changes. *CoRR*, abs/1810.00314, 2018.

[3] Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? In *International Conference on Learning Representations*, 2018.

[4] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. 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.

[5] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora (rough diamond). In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 227–233. Springer, 2015.

[6] 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.

[7] 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 *Lecture Notes in Computer Science*, pages 151–164. Springer, 2013.

[8] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017.

[9] Ning Miao, Hengliang Wang, Ran Le, Chongyang Tao, Mingyue Shang, Rui Yan, and Dongyan Zhao. Tree2tree learning with memory unit, 2018.

[10] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[11] Josef Urban. Experimenting with machine learning in automatic theorem proving. Master's thesis, Faculty of Mathematics and Physics, Charles University, Prague, 1998. English summary at https://www.ciirc.cvut.cz/~urbanjo3/MScThesisPaper.pdf.

[12] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *LNCS*, pages 255–270. Springer, 2018.

Focusing proofs and logics for models of computation

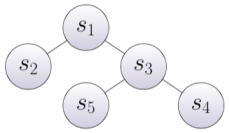
Aleksandra Samonek (UCLouvain)

Keywords: proof theory, linear logic, focusing, computation, logic programming

Linear logic (LL) is particularly successful in expressing the abstract models of computational processes. The completeness of focusing proofs was first shown for LL by [1] and with the following basic principle:

computation = proof search.

In [1] Andreoli observed that the chronology of the computational process is well expressed using the sequent calculus of G. Gentzen. Gentzen-style sequents can be used to easily formalize the history of execution of a computational process during a certain time interval. A sequent system describes the correct inferences in proofs. This description corresponds to allowed process state transitions. Consider the following graph:



The point s_1 represents the state of the computational process at the beginning of the time interval. It corresponds to the root of the tree, or the conclusion of a sequent. s_2 and s_3 are the nodes of the tree representing the intermediate states of the process, while s_4 and s_5 are the leaves of the tree and represent the resulting states at the end of the time interval or the hypotheses. A state is represented not by an atom (as would be default in classical logic), but as a sequent (a multiset of formulae). Unordered multisets allow concurrent access to formulae of the sequent.

However, the most troublesome issue with such Gentzen-style proof is that they proceed very slowly, due to the number of redundant options which a search function needs to consider before finding an appropriate form of the proof. Indded, proofs in a Gentzen-style sequent calculus for LL (and other logics as well) can be *redundant*, meaning that two proofs can differ syntactically although they are identical up to some irrelevant ordering or simplification of application of inference rules (IRs). Consequently, the search procedure makes (computationally) costly choices which turn out to be *irrelevant*.

Focusing proofs is one of the methods used to reduce this redundancy and consequently, speed up the proof search. More specifically, focusing is a strategy in proof searching in which the searching procedure alternates between two phases:

1. **an inversion phase** (when the invertible inference rules are applied exhaustively) and
2. **a chaining phase** (when a selected formula is decomposed as much as possible using non-invertible rules).

In LL **synchronous connectives** are such that the right-introduction inference rules for those connectives are (generally) not invertible, the opposite for **asynchronous connectives**. In focusing proofs the synchronous/asynchronous classification is extended to atoms. This assignment of positive (synchronous) bias or negative (asynchronous) bias is arbitrary and influences the shape and the number of focused proofs, but not the fact of whether a focused proof for a given formula exists in general. For a variety of logics, LL among them, **focusing is complete** and provides a foundation for **developing logics into programming languages**.

Various focusing proofs methods and results have been developed in proof theory and theoretical computer science. However, so far no implementation of focusing proofs to automated theorem proving has been presented. In particular, [1] showed a first focused proof system for a full logic (LLF), which was complete wrt its logic and tractable. Then [2, 3] used Andreoli's completeness result to design and formalize certain logic programming languages. [4] developed focusing proof systems for classical logic ($LKT/LKQ/LK^{\eta}$). [5] developed LJQ which permits the so called forward-chaining in proofs. [6] used both both forward chaining and backward chaining in proofs for full INT (LKF). [7] showed a modal proof of focalization *via* focalization graphs. Finally, [8]: proposed a method for automatic generation of certain focused proof systems *via* permutation graphs based on [7].

During this talk I will (i.) demostrate **how a logic may be turned into a programming language** and (ii) how it can be used to **design a focused proof system** based on the result in [1] and (iii.) discuss how this method of speeding up proof search relates to methods in computer science for **theorem proving optimization**, like using neural networks for premise selection (*cf.* [9]).

References

[1] J.-M. Andreoli, "Logic programming with focusing proofs in linear logic," *Journal of Logic and Computation*, vol. 2, no. 3, pp. 297–347, 1992.

[2] J.-M. Andreoli and R. Pareschi, "Linear objects:logical processes with built-in inheritance," *New Generation Computing*, vol. 9, no. 3-4, pp. 445–473, 1991.

[3] D. Miller, "A multiple-conclusion specification logic," *Theoretical Computer Science*, vol. 165, no. 1, pp. 201–232, 1996.

[4] V. Danos, J.-B. Joinet, and H. Schellinx, "The structure of exponentials: Uncovering the dynamics of linear logic proofs," in *Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pp. 159–171, Springer, 1993.

[5] H. Herbelin, *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes*. PhD thesis, Université Paris-Diderot-Paris VII, 1995.

[6] C. Liang and D. Miller, "Focusing and polarization in linear, intuitionistic, and classical logics," *Theoretical Computer Science*, vol. 410, no. 46, pp. 4747–4768, 2009.

[7] D. Miller and A. Saurin, "From proofs to focused proofs: a modular proof of focalization in linear logic," in *International Workshop on Computer Science Logic*, pp. 405–419, Springer, 2007.

[8] V. Nigam, G. Reis, and L. Lima, "Towards the automated generation of focused proof systems," *arXiv preprint arXiv:1511.04177*, 2015.

[9] A. A. Alemi, F. Chollet, G. Irving, C. Szegedy, and J. Urban, eds., *DeepMath - Deep Sequence Models for Premise Selection*, 2016.

Towards an Efficient Architecture for Intelligent Theorem Provers

Michael Rawson and Giles Reger

University of Manchester, Manchester, UK

High-performance automated theorem provers for first-order logic (e.g. CVC4 [2], E [11], iProver [6], Vampire [7]) include hand-coded heuristics to guide proof search, often exposed as individual prover options. These heuristics perform well, but have a number of disadvantages including a lack of generality over problems (necessitating *portfolio* modes [9, 10]), inability to learn from experience, and maintenance overhead. There is therefore interest in employing machine-learning techniques to guide proof search in automatic theorem provers, with approaches such as FEMaLeCoP [5], ENIGMA [4], or Deep Network Guided Proof Search [8] (DNGPS).

These systems experience a trade-off between the expressivity of their learning algorithms versus the impact of guidance on "raw" prover performance. At extremes:

- The heuristic is fast, but does not take into account the entire proof state (e.g. the MaLeCoP family), restricting the prover to learning from features.
- The heuristic takes into account the entire proof state (usually via neural networks), but is too slow to use all the time. The DNGPS system runs with the heuristic for a fixed amount of time, then reverts back to the old heuristics thereafter.

Ideally, an intelligent system would guide search based on the structure of the current proof state, while also remaining performant enough to run continuously without significantly affecting prover performance. We present a prover architecture which attempts to achieve this ideal, and show that it has several other desirable properties.

Desiderata In such a system we require the following:

1. *Proof state must be small.* Attempting to evaluate large proof states structurally requires a lot of resources. Saturation-based provers such as E or Vampire can have very large proof states, for example.
2. *Evaluation of states must be possible in parallel.* Machine-learning algorithms tend to operate more efficiently in batches. Tree-based approaches (tableau *etc.*) lend themselves to this, whereas saturation provers are inherently sequential.
3. *Subgoals must be independent.* If the prover has a notion of (sub-)goals which must be dispatched (such as in tableau or connection provers), these should be independent of the rest of the search space. Otherwise, the learning system is trying to learn while blind to the context of the search.

Calculus and Algorithm We implement a first-order tableau calculus without unification, with equality, on non-clausal formulae. By using this very "natural" representation, the hope is that inherent proof structure will be more apparent to machine-learning algorithms, which do not have to invert the process of clausification. The tableau space is explored in parallel by

means of a UCT-maximising tree search (similar to that employed by MonteCoP [3]), with new goals placed on a global queue for evaluation in batches by means of arbitrary machine-learning methods, currently a GCN [1, 12].

Advantages This prover architecture satisfies requirements 1–3, but also shows promise in other areas. In terms of reasoning, the calculus used is relatively flexible, allowing for extension to reasoning with theories, induction, and full higher-order logic without modifying the whole prover. In terms of efficiency, such a prover can also make full use of multi-core systems, allowing for linear exploration speedup with the number of available cores, eventually saturating the device or core used for running machine-learned algorithms. The prover is also well-suited to a hybrid approach in which promising subgoals are dispatched to an existing first-order ATP.

Evaluation and Future Work Evaluation and implementation of an example prover system based on this architecture is ongoing, but initial results are promising, with the system appearing to "learn to prove" harder problems based on prior experience with easier problems. Future work includes exploring calculus options, optimisation, further exploration of machine-learning methods, and using the prover as a "pre-processor" for an existing first-order ATP.

References

[1] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[2] 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.

[3] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte carlo connection prover. *arXiv preprint arXiv:1611.05990*, 2016.

[4] Jan Jakubuv and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In *International Conference on Intelligent Computer Mathematics*, pages 292–302. Springer, 2017.

[5] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 88–96. Springer, 2015.

[6] K. Korovin. iProver – an instantiation-based theorem prover for first-order logic (system description). In *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.

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

[8] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. *arXiv preprint arXiv:1701.06972*, 2017.

[9] Michael Rawson and Giles Reger. Dynamic strategy priority: Empower the strong and abandon the weak. In *Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning (PAAR)*, pages 58–71.

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

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

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

Machine Learning for Instance Selection in SMT Solving

Jasmin Christian Blanchette[1,2], Daniel El Ouraoui[2],
Pascal Fontaine[2], and Cezary Kaliszyk[3]

[1] Vrije Universiteit Amsterdam, Amsterdam, the Netherlands
j.c.blanchette@inria.fr
[2] University of Lorraine, CNRS, Inria, and LORIA, Nancy, France
{jasmin.blanchette,daniel.el-ouraoui,pascal.fontaine}@inria.fr
[3] University of Innsbruck, Innsbruck, Austria
cezary.kaliszyk@uibk.ac.at

Abstract

SMT solvers are among the most suited tools for quantifier-free first-order problems, and their support for quantified formulas has been improving in recent years. To instantiate quantifiers, they rely on heuristic techniques that generate thousands of instances, most of them useless. We propose to apply state-of-the-art machine learning techniques as classifiers for instances on top of the instantiation process. We show that such techniques can indeed decrease the number of generated useless instances. We envision that this could lead to more efficient SMT solving for quantified problems.

Satisfiability-modulo-theories (SMT) solvers are among the best backends for verification tools and "hammers" in proof assistants. When proof obligations contain quantified formulas, SMT solvers rely on *instantiation*, replacing quantified subformulas by sets of ground instances. Three main techniques have been designed: enumerative [11], trigger-based [4], and conflict-based [12] instantiation. Among these, only conflict-based instantiation computes instances that are guaranteed to be relevant, but it is incomplete and is normally used in combination with other techniques. Enumerative and trigger-based techniques are highly heuristic and generate a large number of instances, most of them useless. As a result, the search space of the solver explodes. Reducing the number of instances could improve the solver's efficiency and success rate within a given time limit.

We propose to use a state-of-the-art machine learning algorithm as a predictor over the generated set of instances to filter out irrelevant instances, and thus decrease the number of instances given to the ground solver. The predictor is invoked after each instantiation round to rate the potential usefulness of each generated instance. Several strategies are then used to build a subset of potentially relevant instances that are immediately added to the ground solver. Adding the other instances is postponed.

We conducted our experiment in veriT [2], an SMT solver that implements all three instantiation techniques described above. We chose as predictor the XGBoost gradient boosting toolkit [3] with the binary classification objective. This configuration had already been used successfully in the context of theorem proving [6, 10].

Choosing a suitable set of features is crucial for effective machine learning. The features determine how precise the representation of the problem is. Previous works already investigate features for theorem proving [1, 5, 6, 8–10]. Our features are more specifically inspired by ENIGMA [6] and RLCoP [7]. They are basically term symbols and term walks with symbol sequences projected to features using Vowpal Wabbit hashing. Term variables and Skolem constants are translated analogously to constants. The model is further enriched with abstract features such as term size, term depth, and the number of instances.

To encode our problem into sparse vectors, we use three kinds of information available to the solver: the ground part of the formula (set of literals $l_1, \ldots, l_m$), the quantified formula

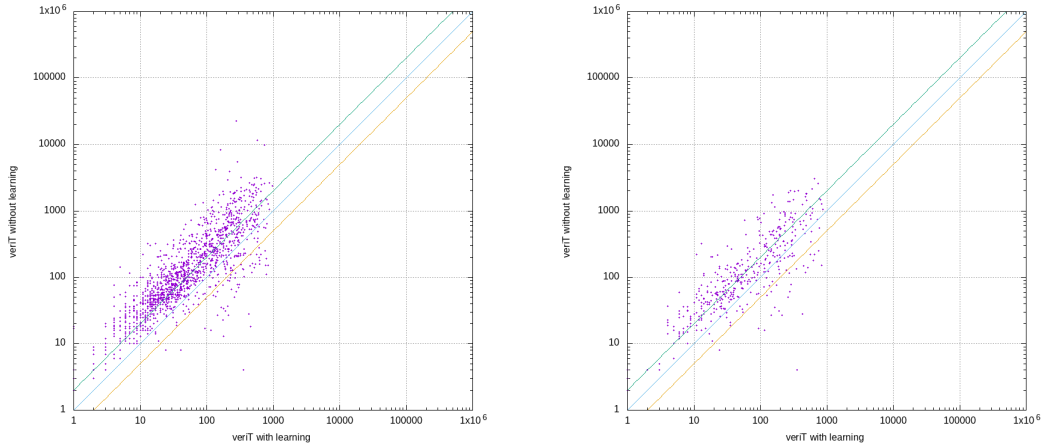


Figure 1: Comparison of veriT configurations on UF SMT-LIB benchmarks

$\psi[\overline{x_n}]$ to instantiate, and the substitution. A query to the predictor is represented by a tuple of the form

$$(\; l_1, \ldots, l_m \; , \; \psi[\overline{x_n}] \; , \; x_1 \mapsto t_1, \ldots, x_n \mapsto t_n \;)$$

In the learning phase, to discriminate useful instances from others in the run of the solver, we consider the pruned proof with only the relevant facts. An instance produced in the run is tagged as useful if it also occurs in the pruned proof. To train the predictor, we used the SMT-LIB benchmarks in the UF category. We ran veriT on 1866 SMT formulas randomly divided into two sets: 560 for the test set (30%) and 1306 for the training set (70%). One run of the solver generates many more useless instances (99%) than useful ones (1%). Consequently, we used undersampling to obtain a balanced data set for learning.

Figure 1 compares the numbers of generated instances for two versions of the veriT solver: with instance selection (x-axis) and without (y-axis). The fewer instances, the better. The left-hand side shows the results of the solvers on the entire data set (training and test), whereas the right-hand side shows the results only on the test set. In both plots, a cluster of points is forming along the line corresponding to the equation $f(x) = 2x$. On average, instance selection allows veriT to find proofs with about half the number of instances. On the other hand, a time comparison would not look good for instance selection, because the current prototype relies on extremely time consuming external calls to the predictor.

These results show that our approach seems to be suitable to substantially reduce the number of generated instances. The implementation of this first prototype is suboptimal. We are now working on an implementation of the predictor inside veriT, hoping to make the prediction cost close to negligible. We hope that the good results in terms of number of instances required will eventually translate into strong performance.

Acknowledgement The work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Experiments were carried out using the Grid'5000 testbed (https://www.grid5000.fr/), supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations.

References

[1] Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In Doina Precup and Yee Whye Teh, editors, *ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 80–88. PMLR, 2017.

[2] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: an open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *CADE–22*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

[3] Tianqi Chen and Carlos Guestrin. XGBoost: a scalable tree boosting system. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi, editors, *KDD 2016*, pages 785–794. ACM, 2016.

[4] Leonardo de Moura and Nikolaj Bjørner. Efficient E-matching for SMT solvers. In Frank Pfenning, editor, *CADE–21*, volume 4603 of *LNCS*, pages 183–198. Springer, 2007.

[5] Geoffrey Irving, Christian Szegedy, Alexander A. Alemi, Niklas Een, François Chollet, and Josef Urban. DeepMath–deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike V. Luxburg, Isabelle Guyon, and Roman Garnett, editors, *NIPS 2016*, pages 2235–2243. Curran Associates, 2016.

[6] 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 2017*, volume 10383 of *LNCS*, pages 292–302. Springer, 2017.

[7] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement learning of theorem proving. In Samy Bengio, Hanna Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *NeurIPS 2018*, pages 8835–8846. Curran Associates, 2018.

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

[9] Sarah Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David Sands, editors, *LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017.

[10] Bartosz Piotrowski and Josef Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *IJCAR 2018*, volume 10900 of *LNCS*, pages 566–574. Springer, 2018.

[11] Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In Dirk Beyer and Marieke Huisman, editors, *TACAS 2018*, volume 10806 of *LNCS*, pages 112–131. Springer, 2018.

[12] Andrew Reynolds, Cesare Tinelli, and Leonardo de Moura. Finding conflicting instances of quantified formulas in SMT. In Koen Claessen and Viktor Kuncak, editors, *FMCAD 2014*, pages 195–202. IEEE, 2014.

Better SMT Proofs for Easier Reconstruction

Haniel Barbosa[1], Jasmin Christian Blanchette[2,3,4], Mathias Fleury[4,5], Pascal Fontaine[3], and Hans-Jörg Schurr[3]

[1] University of Iowa, 52240 Iowa City, USA
haniel-barbosa@uiowa.edu
[2] Vrije Universiteit Amsterdam, 1081 HV Amsterdam, the Netherlands
j.c.blanchette@vu.nl
[3] Université de Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France
{jasmin.blanchette,pascal.fontaine,hans-jorg.schurr}@loria.fr
[4] Max-Planck-Institut für Informatik, Saarland Informatics Campus, Saarbrücken, Germany
{jasmin.blanchette,mathias.fleury}@mpi-inf.mpg.de
[5] Saarbrücken Graduate School of Computer Science, Saarland Informatics Campus, Saarbrücken, Germany
s8mafleu@stud.uni-saarland.de

Proof assistants are used in verification, formal mathematics, and other areas to provide trustworthy, machine-checkable formal proofs of theorems. Proof automation reduces the burden of proof on users, thereby allowing them to focus on the core of their arguments. A successful approach to automation is to invoke an external automatic theorem prover, such as a satisfiability-modulo-theories (SMT) solver [5], reconstructing any generated proofs using the proof assistant's inference kernel. The success rate of reconstruction, and hence the usefulness of this approach, depends on the quality of the generated proofs.

We report on the experience gained by working on reconstruction of proofs generated by an SMT solver while also improving the solver's output. By doing so, we were able to understand some practical constraints of reconstruction systems and find areas that require attention in the documentation of the proof output. We also discovered bugs in the proof generation code.

Proof generation from SMT solvers has attracted attention in the past [3]. The SMT solvers CVC4 [2] and Z3 [9] produce proofs, but CVC4's output format does not record quantifier reasoning, whereas Z3 does not always produce fine-grained steps, notably for skolemization. The SMT solver we work with, veriT [8], was recently extended with a more fine-grained proof-producing module [1] that records skolemization and other preprocessing steps in a detailed fashion. Proofs produced by veriT [6] take the form of a list of steps with optional annotations for term sharing. The syntax is based on SMT-LIB [4].

Many proof assistants reconstruct proofs generated by automatic theorem provers. Examples include the SMTCoq plugin [11], which reconstructs proofs from CVC4 and veriT inside Coq, and Isabelle's *smt* tactic [7], which reconstructs Z3 proofs. We extended this tactic to support veriT proofs as well. The *smt* tactic first translates the current higher-order proof goal to a first-order SMT problem. Then the external SMT solver is invoked. If the solver reports "unsatisfiable," the tactic will attempt to replay the generated proof in Isabelle.

Our experience emphasizes the importance of complete documentation. When veriT is called with the option `--proof-format-and-exit`, it generates a list of proof rules and a description of their semantics. Furthermore, earlier publications [1, 6, 10] provide background documentation on the proof calculus. Nevertheless, this documentation was lacking, especially concerning implicit steps performed by veriT. To replay the proof, the implicit steps must also be performed on Isabelle's side. Such implicit transformations appear in two places. First, veriT ignores the orientation of equalities in the input. The simple solution was to print the input

assertions after this normalization to allow Isabelle to link atoms with their normalized form. Second, veriT performs some simplifications immediately before printing a proof step. This includes eliminating repeated literals and double negation from clauses. Now that this behavior is precisely documented, Isabelle can reconstruct these implicit steps most of the time at the cost of some automatic search. We plan to make this implicit normalization optional in future versions of veriT.

The size of the generated proofs is a practical constraint we initially overlooked. During skolemization, veriT introduces Hilbert choice terms in place of skolemized variables. Thus, $\exists x.\, p(x)$ is skolemized to $p(\varepsilon x.\, p(x))$. While this is elegant in theory, the choice term can be prohibitively large, especially when it is repeated in the output, leading to reconstruction failures. A solution is to use term sharing in the generated proofs: veriT adds a name annotation to every term and subsequently uses the name instead of the term. Sharing can have a dramatic impact on size: a 62 MB proof was compressed to 192 KB.

We encountered some difficulties with the replacement of constants by choice terms. Instead of choice terms, for efficiency reasons veriT uses fresh constants during solving. These constants must be replaced by the corresponding choice terms in the proof output. When choice terms were nested, the proof output did not fully replace constants inside choice terms. Since the choice functions are often quite long, such errors are hard to detect by a human reader, but instantly prohibit reconstruction.

We also observed a phenomenon we call *proof rot*. During the development of an automatic prover, we might inadvertently introduce small discrepancies with respect to the documented behavior. For example, the instantiation rule used by veriT is slightly stronger than published [10] and documented. The documented form is $(\forall x.\, \varphi) \to \varphi[t/x]$, but in practice it sometimes takes the form $(\forall x.\, \varphi_1 \wedge \cdots \wedge \varphi_n) \to \varphi_i[t/x]$. Such changes accumulate and complicate reconstruction. During the implementation of the reconstruction procedure, each change had to be either documented or corrected. Now that it is in place, proof reconstruction serves as a safeguard to prevent such changes from being accidentally reintroduced.

Prospect Proofs are meant to be replayed. Implementing the reconstruction during the development of the proof-producing routines ensures that proofs can be replayed in practice. Given the flexibility of the SMT language, the proofs generated by SMT solvers need to account for a wide variety of theories and language features, which results in complex proofs with many possibilities for errors. These errors can be found by reconstructing proofs.

The quality of veriT's proofs remains unsatisfactory. Simple input problems often produce long, unwieldy proofs; yet, many proof steps are too coarse. A rule for linear arithmetic produces a certificate of the unsatisfiability of a set of inequalities using Farkas's lemma without providing explicit coefficients. This means that reconstruction must rely on a decision procedure to refind the proof, which sometimes fails or is slow. Furthermore, term sharing is required to keep proofs to a reasonable size, but also results in unreadable proofs. A good balance has yet to be found.

A call to an automated prover from inside a proof assistant can fail. Often this is because the prover could not find a proof, but sometimes the proof cannot be reconstructed. This should never happen.

Acknowledgments The work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 713999, Matryoshka). Previous experiments were carried out using the Grid'5000 testbed (https://www.grid5000.fr/), supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities as well as other organizations.

References

[1] Haniel Barbosa, Jasmin Christian Blanchette, Mathias Fleury, and Pascal Fontaine. Scalable fine-grained proofs for formula processing. *Journal of Automated Reasoning*, 2019.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV 2011*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[3] Clark Barrett, Leonardo de Moura, and Pascal Fontaine. Proofs in satisfiability modulo theories. In David Delahaye and Bruno Woltzenlogel Paleo, editors, *APPA 2014*, volume 55 of *Mathematical Logic and Foundations*, pages 23–44. College Publications, 2014.

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

[5] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.

[6] Frédéric Besson, Pascal Fontaine, and Laurent Théry. A flexible proof format for SMT: A proposal. In Pascal Fontaine and Aaron Stump, editors, *PxTP 2011*, pages 15–26, 2011.

[7] Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012.

[8] Thomas Bouton, Diego Caminha B. de Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient SMT-solver. In Renate A. Schmidt, editor, *CADE 2009*, volume 5663 of *LNCS*, pages 151–156. Springer, 2009.

[9] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS 2008*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[10] David Déharbe, Pascal Fontaine, and Bruno Woltzenlogel Paleo. Quantifier inference rules for SMT proofs. In Pascal Fontaine and Aaron Stump, editors, *PxTP 2011*, pages 33–39, 2011.

[11] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq: A plug-in for integrating SMT solvers into Coq. In Rupak Majumdar and Viktor Kunčak, editors, *CAV 2017*, volume 10426 of *LNCS*, pages 126–133. Springer, 2017.

Clause Features for Theorem Prover Guidance*

Jan Jakubův and Josef Urban

Czech Technical University in Prague, Prague, Czech Republic

1 Given Clause Guidance in Theorem Proving

State-of-the-art saturation-based automated theorem provers (ATPs) for first-order logic (FOL), such as E [10], 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, 6] is an *efficient* learning-based method for guiding given 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 or a similar machine learning method, all generated first-order clauses need to be translated to fixed-length numeric *feature vectors*. This is done by translating each clause to a multi-set of *features* which is in turn translated to a numeric vector.

Various possible choices of efficient clause features for theorem prover guidance have been experimented with [5, 6, 7, 8]. So far our goal has been to develop with fast and practically usable methods, allowing E users to directly benefit from our work. Related research in developing neural approaches [2, 9, 11] is of great interest, but so far less practically usable in the context of a fast saturation-style prover. The original ENIGMA [5] uses term-tree walks of length 3 as features, while the second version [6] reaches better results by employing various additional features. Which of the additional features are the main cause of the improvement was however, not investigated. The main goal of this work is to employ experiments with various combinations of features, in order to estimate the value of every possible features choice. Moreover, we propose to use another popular machine learning method instead of logistic regression, namely *decision trees* and their gradient boosted ensembles, effectively implemented by XGBoost library [3]. In this way, we shall be able to estimate, which of the features are useful across various machine learning methods.

2 Experiments with Selection of Clause Features

The following types of features are used in the recent ENIGMA (see [6, Sec.2]):

Vertical Features (V) are (top-down-)oriented term-tree walks of length 3. For example, the unit clause $P(f(a,b))$ contains only features (P,f,a) and (P,f,b).
Horizontal Features (H) are horizontal cuts of a term tree. For every term $f(t_1,\ldots,t_n)$ in the clause, we introduce the feature $f(s_1,\ldots,s_n)$ where s_i is the top-level symbol of t_i.
Symbol Features (S) are various statistics about symbols used in the clause, namely, the

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

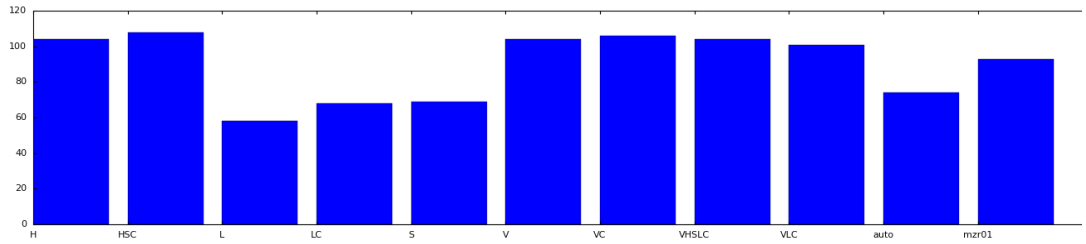


Figure 1: Number of training problems solved by selected variants.

number of occurrences and the maximal depth for each symbol.
Length Features (L) count the clause length and the numbers of positive and negative literals.
Conjecture Features (C) embed information about conjecture being proved into the feature vector. In this way, ENIGMA can provide conjecture-dependent predictions.

ENIGMA uses these features to translate clauses to features vectors. See [6] for how these feature vectors are used for given clause guidance. We experiment with all the possible combinations of features. This gives us 30 possible combinations, for example, "VHS" denotes the variant with vertical, horizontal, and symbol features. The original ENIGMA [5] uses only "V", while the enhanced ENIGMA [6] uses "VHSLC".

We evaluate all possible feature combination on the Mizar MPTP2078 [1] benchmark with 2078 problems as follows. We take a random subset consisting of 200 training problems and a single good-performing E Prover strategy (denoted "mzr01"), we run E with mzr01 on the training problems to obtain training samples, and we create a separate ENIGMA predictor for every possible feature combination. The number of training problems solved by selected variants are presented in Figure 1. From these preliminary results, we can conclude that there are variants that perform slightly better on the training problems than "VHSLC".

Next, we have evaluated all the predictors on all the benchmark (testing) problems with 5 seconds CPU time limit. From these preliminary results, it seems, that there is no clear winner, that is, no single variant outperforms the others. However, different variants solve quite complementary problems. The combined performance of all the variants greatly outperforms both the original strategy and the former ENIGMA. Although for the combined performance we use in general a 30 times higher time limit, we conclude that there is a great potential. Moreover, instead of searching for the best feature variant, it seems better to work simultaneously with several ones. The number of problems solved in time is depicted in graph (1) in Figure 2. Graphs (2) & (3) of the same figure, demonstrate the effect of ENIGMA guidance on the number of processed clauses.[1] ENIGMA guidance heavily shortens the proof search on the training problems and on a large number number of testing problems.

The proposed talk will present extended results, including experiments with additional features (e.g. from [7]), and with decision trees (XGBoost) used instead of logistic regression (LIBLINEAR). Moreover, the talk will present the comparison of the combined performance with the state-of-the-art theorem prover strategies conducted with a comparable time limit.

[1] Point (x, y) in the graph means that x clauses were processed without ENIGMA while y with ENIGMA on the same problem. Hence the more points under the diagonal, the better ENIGMA guides the proof search.

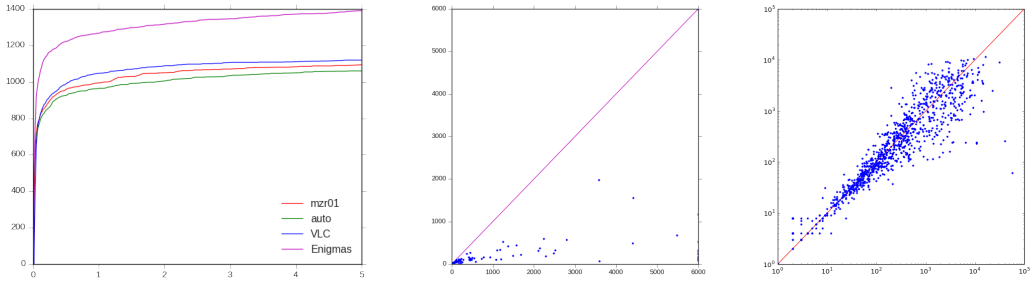


Figure 2: (1) Problems solved by selected strategies in time. Numbers of processed clauses with and without ENIGMA on (2) training and (3) testing problems.

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

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, pages 785–794. ACM, 2016.

[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. Enhancing ENIGMA given clause guidance. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

[7] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olsák. Reinforcement learning of theorem proving. *CoRR*, abs/1805.07563, 2018.

[8] Cezary Kaliszyk, Josef Urban, and Jirí Vyskocil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, pages 3084–3090. AAAI Press, 2015.

[9] Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 85–105. EasyChair, 2017.

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

[11] 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.

Exploration of Machine Translation Techniques in Auto-formalization of Mathematics: Three Experiments

Qingxiang, Wang[1,2], Cezary Kaliszyk[1], Josef Urban[2]

[1] University of Innsbruck
[2] Czech Technical University in Prague

Formalizing all existing mathematical knowledge into a computer-based database and formally checking the correctness of all the mathematical proofs have for a long time been a dream for researchers in interactive and automated theorem proving [1, 2]. Achieving such goals will promote the dissemination of mathematical ideas [3], confirm the validity of complex mathematical proofs [4, 5], and the size of such a database can be an invaluable data source for using machine learning techniques in automated theorem proving [6, 7].

However, despite the existence of various formalization libraries (e.g. Mizar, HOL family, Coq, etc) which contain only a portion of mathematics, extracting information from all mathematics literature is still a task that is too costly to be done by manual labor. We believe that machine learning techniques themselves, especially artificial neural networks, can be adapted to facilitate and automate formalization of mathematics.

As machine learning techniques require training data but the purpose of formalization itself is to obtain data, we run into a dilemma of having to gather enough training data at the very beginning. Previous approaches circumvented the issue by synthesizing informal mathematical statements directly from formal statements [8, 9]. Thanks to the work of Bancerek et al. [10] which generated over 1 million latex-to-mizar aligned statement pairs, we were able to train a recurrent neural network model that achieved promising results for informal-to-formal translation [11].

Our previous work proves that machine learning techniques can be of use in accelerating formalization of mathematics, but the issue of obtaining textbook mathematical data remains. As we have found later, the trained neural network model does not generalize well for arbitrary mathematical statement. Based on this, we propose three experiments to further increase the quality of translation:

1. **Increase generalization power by employing unsupervised machine learning**. Given two corpora which may not need to be aligned, the auto-denoising and back-translation technique described in [12, 13] can transform an unsupervised learning problem into a series of supervised machine learning problems. Through a fixed word embedding [14] during the whole training process, the unsupervised learning is known to achieve reasonable translation in natural language. This paradigm is suitable for our scenario as there are reasonably large datasets in either informal latex (e.g. Proofwiki) or language from a formal proof assistant (e.g. Mizar, HOL-Light). The experiment can be conducted by first only aligning similar subjects such as point-set topology, then proceeding further to all subjects available in the corpora. We have conducted several initial experiments, showing that deploying unsupervised methods is an interesting direction.

2. **Improve translation quality by adding type-checking mechanism during training**. Translation from a trained neural network is well-known to be fluent, but since we are translating to a formal language, it is possible to type-check the decoded formal statements. Successfully type-checked statements can give preference on the weight space

therefore affecting the training process, or the generated correct statements can be added to the training data for a new training pass. For Mizar in particular, the translated statement could be first transformed into an intermediate statement by a probabilistic CKY parser as described in [9] or another pre-trained sequence-to-sequence network, then a custom Mizar elaborator [15] can be invoked to fill in type information on all involving notions of the intermediate statement. Successful elaboration amounts to successful type-checking of the Mizar statement. We have already plugged in our custom elaborator into the learning toolchain, and experimented with several suitable formats (lisp-based parse trees, TPTP, prefix notation) for the neural task that precedes the elaboration. The biggest issue seems to be the verbosity of some of the formats, that seems unsuitable for unmodified neural sequence-to-sequence methods. The best representation however achieves a reasonable perplexity (less than 2) on a dataset of 50000 aligned statements, indicating that it will be possible to set up interesting feedback loops based on elaboration.

3. **Explore various input-output formats and figure out new evaluation metric**. Progress in tree neural networks [16] has made it tempting to be adapted in informal-to-formal translation, as both informal and formal statements are suitable to be represented in tree format. We will see the gain from different input-output combinations and incorporate the result into the above two experiments. Currently we evaluate the quality of translated statements using standard NLP metrics (e.g. BLEU rate, perplexity), new metrics that are more akin to logical statements need to be explored and adapted when the output formats are changed.

We hope the proposed experiments could reveal insights on informal-to-formal translation. Positive results from the above experiments can provide us with more confidence in using machine learning in auto-formalization of mathematics. The talk will report on the several experiments done so far and the results achieved in them.

References

[1] The qed manifesto. In *Proceedings of the 12th International Conference on Automated Deduction*, CADE-12, pages 238–251, London, UK, UK, 1994. Springer-Verlag.

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

[3] Michael Kohlhase and Florian Rabe. Qed reloaded: Towards a pluralistic formal library of mathematical knowledge. *J. Formalized Reasoning*, 9:201–234, 2016.

[4] 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.

[5] Georges Gonthier. Computer mathematics. chapter The Four Colour Theorem: Engineering of a Formal Proof, pages 333–333. Springer-Verlag, Berlin, Heidelberg, 2008.

[6] 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.

[7] Bartosz Piotrowski and Josef Urban. Atpboost: Learning premise selection in binary setting with ATP feedback. *CoRR*, abs/1802.03375, 2018.

[8] 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.

[9] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. System description: Statistical parsing of informalized mizar formulas. 2017. to appear.

[10] Grzegorz Bancerek, Adam Naumowicz, and Josef Urban. System description: Xsl-based translator of mizar to latex. In *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, pages 1–6, 2018.

[11] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *11th International Conference on Intelligent Computer Mathematics (CICM 2018)*, volume 11006 of *LNCS*, pages 255–270. Springer, 2018.

[12] Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Phrase-based & neural unsupervised machine translation. *CoRR*, abs/1804.07755, 2018.

[13] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. *CoRR*, abs/1710.11041, 2017.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[15] Chad E. Brown and Josef Urban. Extracting higher-order goals from the mizar mathematical library. In *Intelligent Computer Mathematics - 9th International Conference, CICM 2016, Bialystok, Poland, July 25-29, 2016, Proceedings*, pages 99–114, 2016.

[16] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

Arithmetical Mini-Games*

Thibault Gauthier and Josef Urban

Czech Technical University in Prague
email@thibaultgauthier.fr josef.urban@gmail.com

Mini-games are playing an important role in the field of artificial intelligence for video games [8, 5]. The principle is that a particularly complex problem can be split it into smaller tasks presenting gentle learning curves. Lessons learned from these easier challenges should prove valuable for solving the original problem.

This divide and conquer strategy could also benefit the automated theorem proving community. That is why we propose three mini-games, each of them being a simplified challenge representative of a larger class of theorem proving problems.

Arithmetic Let $T =_{rec} \{(0, S\, t_1, t_1 + t_2, t_1 \times t_2 \mid (t_1, t_2) \in T^2\}$ and $E =_{def} \{t_1 = t_2 \mid (t_1, t_2) \in T^2\}$. Let x, y be variables and $A =_{def} \{x + 0 = x,\ x \times 0 = 0,\ x + S\ y = S\ (x + y),\ x \times S\ y = x \times y + x\}$. We define R to be rewrite rules for both orientations of equations in A with the exception of $0 \to x \times 0$. For any $e_1, e_2 \in E$ and $t \in T$ the following rules hold:

$$\frac{}{t = t}\ \text{reflexivity} \qquad \frac{e_2}{e_1}\ \text{if } e_1 \to e_2$$

Predictors We compare the performance of two predictors P_{Near} and P_{Tree}. The nearest neighbor algorithm P_{Near} relies on subterm features and is one of the best performing predictor for premise selection tasks in hammers [2]. The tree neural network P_{Tree} mimics the structure of the input formula. Each operator $0, S, +, \times, =$ is replaced by a feed-forward neural network [11] with one hidden layer. The embedding space has dimension four.

Reinforcement Learning The mini-games 2 and 3 are solved by reinforcement learning. We rely on a Monte Carlo tree search [4] to explore different configurations according to their reward potential and progressively improve our predictions form this feedback. Such methodology is described in detail in [7, 10]. Since P_{Tree} is faster than P_{Near}, we use 1600 simulation per move for P_{Tree} and 100 for P_{Near}.

Mini-Game 1: Validity of a Formula

The goal of this mini-game is to decide whether a formula is true or false. By solving it, a predictor will gain the ability to guess the truth value of a formula. Such knowledge could help automated theorem provers avoid spending time searching for a proof of a formula if it is likely to be false. We run a specific instance of this mini-game with a training set of 3200 equations. On a test set of 400 true equations and 400 false equations, we get a percentage of correct guesses of:

53.0% for P_{Near} and 100.0% for P_{Tree}

*Supported by the ERC Consolidator grant no. 649043 AI4REASON and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

The estimation of the truth of a formula given by P_{Near} to estimate is very close to a random guess. For example, the equations $t = u$ and $t \times 0 = u$ may not have the same truth value although they are syntactically close. Moreover, by comparing P_{Tree} to P_{Near}, we can observe that P_{Tree} is not just memorizing the truth values. Otherwise, it would have gotten the same score or a worse score than P_{Near}.

Mini-Game 2: Proof by Rewriting

The goal of this mini-game is to prove a formula through MCTS-guided reinforcement learning using the backward reasoning steps defined by the calculus. The predictor decides at which position in the current term a rewrite step should be applied and the rewrite rule to apply. The choice of a position is made through a series of finite branching choices. Starting from the root position of the equation, the predictor chooses whether to take the left or the right branch and whether to stop or descend further. Therefore, many sub-steps are required to perform a single proof step.

The dataset consists of 100 true equations with shortest proof length ranging from 2 to 6 (20 of each). The number of proven equations after each generation is:

$$0 \times (0 + 0) = 0 \times (0 + S\ 0)$$
$$0 \times (0 + 0) + 0 = 0 \times (0 + S\ 0)$$
$$0 \times (0 + 0) + 0 = 0 \times S\ (0 + 0)$$
$$0 \times (0 + 0) + 0 = 0 \times (0 + 0) + 0$$

P_{Near}	19	32	53	75	81	88	98	99	100	100
P_{Tree}	22	24	32	52	68	88	93	97	93	100

Both P_{Near} and P_{Tree} gradually prove all targets. This shows that this mini-game can be mastered from a dataset of self-generated examples and the syntactical distance given by P_{Near}. Scaling the problem to larger terms and longer proofs may require better generalizations.

Mini-Game 3: Term Synthesis

The goal of this mini-game is to replicate a target term. It provides a simple example of a term building process. Finding a witness (or counter-example) [6, 3] is an natural extension of this mini-game. In order to apply reinforcement learning, we re-frame term synthesis as a search problem. The copy is built starting from the root. At each node, the predictor chooses the operator it wants to imitate. When branching occurs, we first construct the left part of the tree. The replication fails if the copy reaches a size bigger than the target.

The dataset consists of 100 target terms with term size ranging form 6 to 10 (20 of each). The number of replicated term after each generation is:

$$\square,\ S\ \square,\ S\ (\square \times \square),\ S\ (0 \times \square),$$
$$S\ (0 \times (\square + \square)),\ S\ (0 \times (0 + \square)),$$
$$S\ (0 \times (0 + \square)),\ S\ (0 \times (0 + S\ 0))$$

P_{Near}	5	5	8	8	8	9	9	9	9	9
P_{Tree}	11	11	11	11	11	11	11	12	11	11

The discrepancy observed between P_{Near} and P_{Tree} is simply due to the different number of simulations. Since there is exactly one path to the target, any deviation leads to a state where the goal is not reachable anymore. This issue could be solved by adding the possibility to mutate the partially constructed term, making the underlying term rewriting system confluent.

Road Map To conclude, we propose extensions to these mini-games that would move us closer to our final goal of creating stronger automated theorem provers. Our plan is to gradually increase the difficulty of these three problems, test other kind of predictors and change the underlying theory/calculus. At the same time, we would like to design more mini-games, providing a way to evaluate fairly and eventually master other proving [9] and para-proving abilities [1, 12].

References

[1] David Aspinall and Cezary Kaliszyk. What's in a theorem name? (rough diamond). In Jasmin C. Blanchette and Stephan Merz, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 9807 of *LNCS*, pages 459–465. Springer, 2016.

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

[3] Jasmin Christian Blanchette and Tobias Nipkow. Nitpick: A counterexample generator for higher-order logic based on a relational model finder. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2010.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

[5] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

[6] Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of Haskell programs. In Martin Odersky and Philip Wadler, editors, *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000.*, pages 268–279. ACM, 2000.

[7] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olsák. Reinforcement learning of theorem proving. *CoRR*, abs/1805.07563, 2018.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, 2017.

[11] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[12] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2018.

How to Leverage a Large Dataset of Formalized Mathematics with Machine Learning?

Dennis Müller[1], Florian Rabe[1,2], and Michael Kohlhase[1]

[1] Computer Science, FAU Erlangen-Nürnberg
[2] LRI, Université Paris Sud

Abstract

Statistical machine learning techniques have proved very successful recently, including applications in logic. As logic has predominantly been based on exact symbolic methods, the question arises how to combine the strengths of the approaches.
We present MathHub, which aggregates formal libraries including those of most leading proof assistants. All these libraries are available in a standardized and easily machine-readable format, making it an ideal starting point for machine learning applications. Our contribution consists of posing the question given in the title, i.e., we do not provide an answer and instead hope discussions at the workshop will result in insights and collaborations towards future investigations.

Combinations of statistical and symbolic approaches to formal logic offer potential for groundbreaking innovations in artificial intelligence. However, a major impediment to large applications is that the currently most successful statistical methods are based on supervised machine learning and tend to require large sets of training data. And the most successful symbolic approaches to formalizing mathematical and related formal knowledge are based on interactive theorem proving, which requires human input for knowledge creation. Moreover, existing proof assistant are highly incompatible and do not allow easily merging their existing libraries into a single large one. Consequently, current applications have to focus on niches where sufficiently large datasets are in fact available. The most important such example is selecting axioms for reducing the search space of automated provers as in [KU15].

This was part of the motivation of the authors' MathHub project. MathHub collects libraries of mathematical knowledge in all forms. Its scope is not limited to logic, and includes also libraries of computation system, mathematical databases, and informal narrative texts. Technically, it is based on a GitHub-like repository management software with free access for researchers[1].

Its crucial and unique feature is the use of a single representation language for all knowledge, specifically the OMDoc language [RK13]. Thus, all libraries are not only available to be processed through third-party tools, but this processing can be done uniformly for all libraries. Moreover, mature software support is available for managing and reading MathHub repositories.

To make this possible, a huge effort is needed for each library, and we have done that for several major theorem provers, such as for Mizar in [Ian+13], HOL Light in [KR14], PVS in [Koh+17], IMPS [BK18] and very recently for Isabelle.[2] In these translations, great care has been taken to preserve — as much as possible — the original human-authored structure while also including the machine-inferred internal representation. Other MathHub libraries of interest to theorem proving are the LATIN logic library [Cod+11] and Math-in-the-Middle library currently built in the OpenDreamKit project. Figure 1 gives an overview of the sizes of these MathHub libraries. We expect many interesting sets of training data can be gleaned from these

[1] Available at `https://gl.mathhub.info`
[2] To be published. See `https://sketis.net/2018/isabelle-mmt-export-of-isabelle-theories-and-import-as-omdoc-content`.

System	Library	Modules	Declarations/Theorems
MMT	Math-in-the-Middle	183	826
Twelf	LATIN	529	2824
PVS	Prelude	226	3841
PVS	NASA	748	20243
Isabelle	Distribution	2308	484419
Isabelle	AFP	7245	987861
HOL Light	Basic	190	4707
IMPS	Library	64	8573
Mizar	MML	1194	69710

Figure 1: An Overview of the Available Archives on MathHub

and future MathHub libraries. However, transforming such libraries of formal declarations and expressions into the vectorized representations needed by standard machine learning algorithms is itself very difficult and an active research question. We do not offer a solution to this problem, but rather present and offer our library to the community with the hope of engaging in such experiments in future collaborations with machine learning experts.

References

[BK18] J. Betzendahl and M. Kohlhase. "Translating the IMPS theory library to OMDoc/MMT". In: *Intelligent Computer Mathematics (CICM) 2018*. Ed. by F. Rabe, W. Farmer, A. Youssef, and ... LNAI. in press. Springer, 2018. URL: `http://kwarc.info/kohlhase/papers/cicm18-imps.pdf`.

[Cod+11] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. "Project Abstract: Logic Atlas and Integrator (LATIN)". In: *Intelligent Computer Mathematics*. Ed. by J. Davenport, W. Farmer, F. Rabe, and J. Urban. Springer, 2011, pp. 289–291.

[Ian+13] M. Iancu, M. Kohlhase, F. Rabe, and J. Urban. "The Mizar Mathematical Library in OMDoc: Translation and Applications". In: *Journal of Automated Reasoning* 50.2 (2013), pp. 191–202.

[Koh+17] M. Kohlhase, D. Müller, S. Owre, and F. Rabe. "Making PVS Accessible to Generic Services by Interpretation in a Universal Format". In: *Interactive Theorem Proving*. Ed. by M. Ayala-Rincon and C. Munoz. Springer, 2017, pp. 319–335.

[KR14] C. Kaliszyk and F. Rabe. "Towards Knowledge Management for HOL Light". In: *Intelligent Computer Mathematics*. Ed. by S. Watt, J. Davenport, A. Sexton, P. Sojka, and J. Urban. Springer, 2014, pp. 357–372.

[KU15] C. Kaliszyk and J. Urban. "HOL(y)Hammer: Online ATP Service for HOL Light". In: *Mathematics in Computer Science* 9.1 (2015), pp. 5–22.

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

Translating from Higher-Order to Higher-Order*

Chad E. Brown, Thibault Gauthier, and Josef Urban

Czech Technical University, Prague

Introduction

A hammer [5] for an interactive theorem prover (ITP) [13] typically translates an ITP goal into a formalism used by an automated theorem prover (ATP) [16]. Since the most successful ATPs have so far been first-order, the focus has been on first-order translations. There is however interest in producing ATPs working in richer formalisms, such as THF0 [3], THF1 [10], and TFF1 [6]. An interesting related task is to create a (grand) unified large-theory benchmark that would allow fair comparison of such systems and their integration with premise-selectors [1] across the different formalisms. As a step towards creating such benchmarks we would like to use translations that are TD-abstractions [11] and behave similarly on first-order formulas.

HOL4 [17], like many other ITPs, is based on an extension of Church's simple type theory [8] that includes prefix polymorphism and type definitions [12]. Without these extensions it would be possible to directly translate HOL4 terms and propositions into the THF0 format for higher-order ATPs such as Satallax [7] and LEO [2, 18]. It is nevertheless possible to give a translation from a goal in HOL4 into a THF0 problem that takes some advantage of the HOL4 higher-order constructs. The essential idea is to give axioms for a higher-order set theory and translate the HOL4 goal into the higher-order set theory. We propose such a translation here and briefly compare the result to a first-order translation. We have implemented these translations for HOL4 and plan to use them for the first (grand) unified benchmarks, generalizing the existing ones (CakeML [15], HOL4 standard library) used in the CASC LTB competition [19, 20].

Translation to THF0

In order to translate HOL4 into THF0 we begin by including a few basic constants and axioms. Note that base types o (for propositions) and ι (for individuals) are built into THF0. We will think of elements of type ι as being sets. The basic constants we include are as follows:

- $\mathsf{mem} : \iota \to \iota \to o$ corresponds to the membership relation on sets.
- $\mathsf{ne} : \iota \to o$ is for nonemptiness. HOL4 types will be mapped to nonempty sets.
- $\mathsf{ap} : \iota \to \iota \to \iota$ corresponds to set theory level application.
- $\mathsf{lam} : \iota \to (\iota \to \iota) \to \iota$ is used to construct set bounded λ-abstractions as sets.
- $\mathsf{bool} : \iota$ is used for a fixed two element set.
- $\mathsf{arr} : \iota \to \iota \to \iota$ is used to construct the function space of two sets.
- $\mathsf{p} : \iota \to o$ is a predicate which indicates whether or not an element of bool is true or not.

We then include a number of basic axioms summarized as follows: $\mathsf{arr}\ A\ B$ is nonempty when A and B are nonempty, lam and ap satisfy typing properties relative to arr and mem, boolean extensionality, functional extensionality and a beta axiom. If ι is interpreted using a model of ZFC, then the constants above can be interpreted in an obvious way as to make the basic axioms true.

Given this theory, our translation from HOL4 to THF0 can be informally described as follows. We map each HOL4 type α (including type variables) to a term $\hat{\alpha}$ of type ι for which

*Supported by the ERC Consolidator grant no. 649043 AI4REASON, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

we should always know $\mathsf{ne}\ \hat{\alpha}$ in the context in which α is used. The invariant can be maintained by always having a hypothesis $\mathsf{ne}\ \hat{\alpha}$ when α is a type variable or constant. HOL4 type variables (constants) are mapped to THF0 variables (constants) of type ι. For the remaining cases we use bool and arr. We map each HOL4 term $s : \alpha$ to a THF0 term $\hat{s}$ of type ι for which we should always know $\hat{s} \in \hat{\alpha}$ in the context in which s is used. Again, the invariant can be maintained by including the hypothesis $\hat{x} \in \hat{\alpha}$ whenever x is a variable or a constant. The ap and lam constants are used to handle HOL4 application and λ-abstraction. The axioms corresponding to typing rules maintain the invariant. Finally HOL4 propositions (which may quantify over type variables) are translated to THF0 propositions in an obvious way, using p to go from ι to o when necessary. As an added heuristic, the translation makes use of THF0 connectives and quantifiers as deeply as possible, only using p and $\hat{s}$ when necessary.

Translation of HOL4 to FOF for Large-theory Benchmarks

Our translation to first-order follows approximately [14] and keeps the same type encoding [4]. However, there are two major differences: We create new constant symbols and give independent definitions for lambda-abstraction and nested predicates as in [9, 21]. The same constant c used with two different arities i, j is translated to two different constants c_i and c_j. Arity equations relating c_i and c_j to c_0 are added and can be used to recover the dependency between c_i and c_j. Thanks to these modifications, the translation of a formula to first-order does not depend anymore on formulas co-occuring in the same problem.[1] This is essential to export large HOL4 theories in a consistent manner for the first-order LTB competition.

Example and Discussion

As a small example, suppose a HOL4 constant $\mathsf{B} : (\alpha \to \alpha) \to (\alpha \to \alpha) \to \alpha \to \alpha$ were defined so that $\forall\alpha.\ \forall f, g : \alpha \to \alpha.\ \forall x : \alpha.\ \mathsf{B}\ f\ g\ x = f\ (g\ x)$ were a HOL4 theorem (we call this Bdef). From this theorem we could prove $\forall\alpha.\ \forall x : \alpha.\ \mathsf{B}\ (\lambda x.x)\ (\lambda x.x)\ x = x$ (we call this Bid) To translate this to a THF0 problem, we would translate Bdef as the axiom

$$\forall A.\ \mathsf{ne}\ A \Rightarrow \forall f, g, x : \iota.\ \mathsf{mem}\ f\ (\mathsf{arr}\ A\ A) \Rightarrow \mathsf{mem}\ g\ (\mathsf{arr}\ A\ A) \Rightarrow \mathsf{mem}\ x\ A \Rightarrow$$

$$\mathsf{ap}\ (\mathsf{ap}\ (\mathsf{ap}\ \hat{\mathsf{B}}\ f)\ g)\ x = \mathsf{ap}\ f\ (\mathsf{ap}\ g\ x)$$

and Bid as the conjecture

$$\forall A.\ \mathsf{ne}\ A \Rightarrow \forall x : \iota.\ \mathsf{mem}\ x\ A \Rightarrow \mathsf{ap}\ (\mathsf{ap}\ (\mathsf{ap}\ \hat{\mathsf{B}}\ (\mathsf{lam}\ A\ (\lambda x.x)))\ (\mathsf{lam}\ A\ (\lambda x.x)))\ x = x.$$

Disregarding the type encoding the translation of Bdef to FOF is essentially

$$\forall f, g, x.\ \mathsf{B_3}(f, g, x) = \mathsf{ap}(f, \mathsf{ap}(g, x))$$

where $\mathsf{B_3}$ is an arity 3 function. To translate Bid to FOF, we create two new constants c_0 and c_1, give a first-order definition of $c_1 = \lambda x.x$ as $\forall x.\ c_1(x) = x$ and an arity equation $\forall x.\ c_1(x) = \mathsf{ap}(c_0, x)$. After this the conjecture can be expressed as $\forall x.\ \mathsf{B_3}(c_0, c_0, x) = x$.

One advantage of the THF0 translation over first-order translations is that there is no need to deanonymize λ-abstractions inside terms. This means no new names need to be created simply to represent the problem. Since all names used will be common across a collection of problems, this may help techniques which learn to do premise selection.

The talk will include results comparing first-order and higher-order provers on HOL4 problems and discuss examples of possible benefits of using the proposed THF0 translation.

[1] With the exception of the counter used for generating new fresh constants

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] Christoph Benzmüller, Lawrence C. Paulson, Frank Theiss, and Arnaud Fietzke. LEO-II - a cooperative automatic theorem prover for classical higher-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[3] Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 – the core of the TPTP language for classical higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *LNCS*, pages 491–506. Springer, 2008.

[4] Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013.

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

[6] Jasmin Christian Blanchette and Andrei Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. In Maria Paola Bonacina, editor, *CADE*, volume 7898 of *LNCS*, pages 414–420. Springer, 2013.

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

[8] Alonzo Church. A formulation of the simple theory of types. *J. Symbolic Logic*, 5:56–68, 1940.

[9] Lukasz Czajka. Improving automation in interactive theorem provers by efficient encoding of lambda-abstractions. In Jeremy Avigad and Adam Chlipala, editors, *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*, pages 49–57. ACM, 2016.

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

[11] Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.

[12] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[13] John Harrison, Josef Urban, and Freek Wiedijk. History of interactive theorem proving. In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014.

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

[15] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: a verified implementation of ML. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 179–192. ACM, 2014.

[16] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.

[17] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of

LNCS, pages 28–32. Springer, 2008.

[18] Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning. IJCAR 2018*, volume 10900 of *LNCS*, pages 108–116. Springer, Cham, 2018.

[19] Geoff Sutcliffe. The 9th IJCAR automated theorem proving system competition - CASC-J9. *AI Commun.*, 31(6):495–507, 2018.

[20] Geoff Sutcliffe and Josef Urban. The CADE-25 automated theorem proving system competition - CASC-25. *AI Commun.*, 29(3):423–433, 2016.

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

Automated Reasoning for the Andrews-Curtis Conjecture

Alexei Lisitsa

University of Liverpool, Liverpool, UK
a.lisitsa@liverpool.ac.uk

Abstract

We present recent developments in the applications of automated theorem proving and disproving in the investigation of the Andrews-Curtis conjecture. We answer negatively open question from [9] on extensions of AC-transformations and demonstrate trivializations of the cases reported in *op.cit.* We outline further directions of the research on the borders between Mathematics, Automated Reasoning and, more generally, AI.

Introduction and Outline

The Andrews-Curtis conjecture is a well-known open conjecture [1] in low-dimensional topology and combinatorial group theory. In terms of the latter it states that every balanced presentation of the trivial group can be transformed into a trivial one by a series of simple transformations including Nielsen transformations and conjugations of relators. Many authors expressed belief that ACC is likely to be false and there are series of trivial group presentations for which conjectured trivializations remain unknown. Various computational techniques have been applied for the search of the required trivializations (i.e. for elimination of potential counterexamples for ACC), including the methods from *computational group theory* [2]; *genetic algorithms* [8, 10, 5]; *systematic breadth-first search algorithms* [3] to name a few.

In [6, 7] we have proposed to use automated deduction in first-order logic in the search of trivializations and have shown that the approach is very competitive. It was able to trivialize any known to us example tackled by any alternative method reported in the literature, and in [7] we demonstrated new examples of simplifications, including for group presentations of dimension[1] 3 and 4. In our approach we formalized the AC transformations (commonly presented at the meta-level) at the object level of term rewriting (modulo group theory) and first-order deduction. The problem of finding AC-simplifications is reduced here to the problem of proving first-order formulae, which is then delegated to the available automated theorem provers. We also have shown that the disproving by automated finite countermodel finding can be used to show the impossibility of trivializations by subsystems of AC transformations.

In recent work [9][2] the authors proposed a novel algorithmic approach to AC simplifications which relies on the use of generalized moves and a strong equivalence relation on group presentations. Further they have shown that for the famous open series of Akbulut-Kirby presentations $AK(n), n \geq 3$ extending the AC rules by automorphisms of free group F_2 does not increase the sets of reachable presentations. They notice that in general "It is not known if adding these transformations to AC-moves results in an equivalent system of transformations or not".

In this paper:

- We answer the open question from [9] negatively, that is adding automorphic transformations to AC rules leads indeed to non-equivalent systems of transformations. We utilize the approach from [7] and use automated disproving by finite countermodel finding

[1] that is the number of generators (= number of relators for balanced presentations)
[2] which we neglected to refer to and compare with in [7]

- We show that all 12 novel AC trivialization cases reported in [9], Table 4 can also be tackled by our automated theorem proving method,
- We report that we failed to find[3] AC-transformations from trivial group presentations to their images under F_2 automorphisms, reported in [9], Table 3, by our automated theorem proving method. That means that unlike for trivializations finding general AC-transformations by generic theorem proving may not be that efficient as by specialized methods of [9].

Discussion

We have shown that generic automated first-order proving and disproving can be used in combinatorial group theory, both in tackling open questions and as a competitive alternative to specialized algorithms. That places the reported research at the borders between Mathematics, Automated Reasoning and more generally, Artificial Intelligence. Further directions include development of automated extraction of the simplifying move sequences from proofs; machine learning applied to AC-proofs, in a spirit of, for example [4]; investigation of how first-order proving and disproving methods and strategies affect the efficiency of automated reasoning in this area.

We have published the extended version of this paper and all computer-generated proofs and countermodels online[4].

References

[1] J. Andrews and M.L. Curtis. Free groups and handlebodies. *Proc. Amer. Math. Soc.*, 16:192–195, 1965.

[2] George Havas and Colin Ramsay. Andrews-Curtis and Todd-Coxeter proof words. Technical report, in Oxford. Vol. I, London Math. Soc. Lecture Note Ser, 2001.

[3] George Havas and Colin Ramsay. Breadth-first search and the Andrews-Curtis conjecture. *International Journal of Algebra and Computation*, 13(01):61–68, 2003.

[4] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Mirek Olšák. Reinforcement Learning of Theorem Proving. *ArXiv e-prints*, page arXiv:1805.07563, May 2018.

[5] Krzysztof Krawiec and Jerry Swan. AC-trivialization proofs eliminating some potential counterexamples to the Andrews-Curtis conjecture. `www.cs.put.poznan.pl/kkrawiec/wiki/uploads/Site/ACsequences.pdf`, 2015.

[6] Alexei Lisitsa. First-order theorem proving in the exploration of Andrews-Curtis conjecture. *TinyToCS*, 2, 2013.

[7] Alexei Lisitsa. The Andrews-Curtis Conjecture, Term Rewriting and First-Order Proofs. In *Mathematical Software - ICMS 2018 - 6th International Conference, South Bend, IN, USA, July 24-27, 2018, Proceedings*, pages 343–351, 2018.

[8] Alexei D. Miasnikov. Genetic algorithms and the Andrews-Curtis conjecture. *International Journal of Algebra and Computation*, 09(06):671–686, 1999.

[9] Dmitry Panteleev and Alexander Ushakov. Conjugacy search problem and the Andrews-Curtis conjecture. *ArXiv e-prints*, page arXiv:1609.00325, September 2016.

[10] Jerry Swan, Gabriela Ochoa, Graham Kendall, and Martin Edjvet. Fitness Landscapes and the Andrews-Curtis Conjecture. *IJAC*, 22(2), 2012.

[3] with the timeout 1000s

[4] https://zenodo.org/record/2555481 DOI: 10.5281/zenodo.2555481

Making Set Theory Great Again: The Naproche-SAD Project

Steffen Frerix and Peter Koepke

University of Bonn, Germany
Email: s6stfrer@uni-bonn.de and koepke@math.uni-bonn.de

John Harrison, at AITP 2018, gave a programmatic talk "Let's make set theory great again!" [Har18] in which he proposes to take standard set theory (and classical first-order logic) as a basis for automated theorem proving, enriched conservatively and along current mathematical practice by a soft type system à la Freek Wiedijk [Wie07] and by syntactic sugar. This should lead to a standard hierarchy of number systems, a principled treatment of undefinedness, and generally to a "closer correspondence with informal texts".

SAD. A small system which embraces the Harrison approach is the *System for Automated Deduction* (SAD) by Andrei Paskevich et.al. [Pas07]. SAD combines natural language input with first-order proof checking. Mathematical texts are expressed in the controlled mathematical language ForTheL, and checked for logical correctness by a "reasoner" together with a standard automated theorem prover like E.

Naproche-SAD. In 2017 we began our work with the SAD system, based on experiences with our earlier Naproche system [KCKS09]. We have made the code more efficient and added set theoretical mechanisms [FKW18]. At AITP 2018, we reported on our progress. Meanwhile we are able to deal with chapter-sized texts at the level of first-year undergraduate mathematics. We are working on a LaTeXinterface, and, together with Makarius Wenzel, on a jedit-PIDE for Naproche-SAD (see [Wen18]).

Naturally Enriched First-Order Logic. ForTheL signature definitions like

Signature 1. *A* real number *is a notion. Let* x, y, z *denote real numbers.*

Signature 2. $\mathbb{R}$ *is the set of* real numbers.

Definition 1. x *is* positive *iff* $x > 0$.

Signature 3. *An* integer *is a real number. Let* a, b *denote integers. Let* m, n *denote positive integers.*

serve to set up convenient first-order and set-theoretical environments. ForTheL allows many natural language constructs of ordinary mathematics. Proof methods like case splits, contradiction and induction are supported by automatically generating and checking their implicit proof obligations. The example also indicates that hierarchical number systems can easily be set up in ForTheL.

Soft Typing and Undefinedness. SAD allows soft dependent types via notions and adjectives, as in the above example. As in [Wie07], types are internally translated into obvious first-order predicates. Type checking is turned into an *ontological check* at "runtime" during proving to ensure that all presuppositions are fulfilled. Usually these obligations are considerably simpler than the main proof task. This approach also encompasses a correct treatment of undefinedness: the ontological check of the notorious fraction $\frac{1}{x}$ and hence the checking of the entire text fails if the system cannot prove $x \neq 0$ in the proof context of the term.

Set Theory. The well-known difficulties of set theory in automated theorem proving stem from its vast infinite axiom system and the deep iterations of the $\in$-relation in set-theoretically defined notions like numbers. It is essential to keep the proof search away from arbitrary axioms and expansions of notions. In Naproche-SAD, instances of the problematic infinite

axiom schemes have to be explicitly invoked, e.g., by the use of abstraction terms or function definitions. The reasoner of Naproche-SAD has a restrained strategy for definition expansions which benefits set theory in particular.

Correspondence with Informal Texts. Several of our point are exemplified in the following fragment of our formalization of Rudin's [Rud76] which we compare to the original statement of Theorem 120 *a*) in [Rud76]. One could obtain even stronger natural language resemblances by adding more argumentative phrases to ForTheL.

Theorem [Naproche-SAD] If $x \in \mathbb{R}$ and $y \in \mathbb{R}$ and $x > 0$ then there is a positive integer n such that

$$n \cdot x > y.$$

Proof. Define $A = \{n \cdot x \mid n \text{ is a positive integer}\}$. Assume the contrary. Then y is an upper bound of A. Take a least upper bound α of A. $\alpha - x < \alpha$ and $\alpha - x$ is not an upper bound of A. Take an element z of A such that not $z \leq \alpha - x$. Take a positive integer m such that $z = m \cdot x$. Then $\alpha - x < m \cdot x$ (by 15b).

$$\alpha = (\alpha - x) + x < (m \cdot x) + x = (m + 1) \cdot x.$$

$(m + 1) \cdot x$ is an element of A. Contradiction. Indeed α is an upper bound of A. □

Theorem [Rudin's original text] (a) If $x \in \mathbb{R}$, $y \in \mathbb{R}$, and $x > 0$, then there is a positive integer n such that

$$nx > y.$$

Proof. Let A be the set of all nx, where n runs through the positive integers. If (a) were false, then y would be an upper bound of A. But then A has a *least* upper bound in $\mathbb{R}$. Put $\alpha = \sup A$. Since $x > 0$, $\alpha - x < \alpha$, and $\alpha - x$ is not an upper bound of A. Hence $\alpha - x < mx$ for some positive integer m. But then $\alpha < (m + 1)x \in A$, which is impossible, since α is an upper bound of A. □

Comparisons with Mizar. Whilst a majority of proof assistants employ some strong type theory for fundamental reasons or to narrow down proof search, systems like Isabelle/ZF [Isa], Metamath [Met] or Mizar [Miz] are based on first-order set theory. As we share the aim of modeling ordinary mathematical practice with Mizar in particular, there are similarities concerning language and text structuring. Mizar is commited to Tarski-Grothendieck set theory, whereas in Naproche-SAD the specific foundations are variable and depend on which abstraction terms are declared to be sets. Mizar and Naproche-SAD reflect the general mathematical practice of soft typing by "types" and "notions", respectively, which are interpreted as set-theoretic predicates. Both systems have mechanisms to deal with proof obligations spawned by soft-typing. The type systems are however different, as Mizar, e.g., requires types to be non-empty.

A decisive difference between Mizar and Naproche-SAD lies in the degree of proof automation. Mizar texts are required to specify detailed proof steps which leads to a legible, yet computer orientated input language. Naproche-SAD uses strong automated theorem proving to find implicit proof steps. This allows proof granularities similar to textbook proofs, and supports the use of a (restricted) natural language as proof language. The prospect of formal mathematical texts written in natural language is a main driving force of the Naproche-SAD project.

Kelley Morse Class Theory. If one allows *class* quantifiers in the defining properties φ of abstraction terms $\{x \mid \varphi\}$ one is working in Kelley Morse class theory (KM) which is somewhat stronger than Zermelo-Fraenkel set theory. In KM, sets are those classes which are elements of some class. The abstraction term mechanism of Naproche-SAD corresponds to Kelley Morse terms. This has motivated our current formalization of the Appendix of [Kel55] in which the theory KM was introduced. Working with the Appendix has shown the necessity of splitting larger texts into chapters and using ideas of small theories and theory morphisms [Koh14] to control ontological manueuvers like turning the formation of Kuratowski ordered pairs into a basic function.

In our **Talk** we shall illustrate the above set theory orientated principles by excerpts and demonstrations of the mentioned formalisations.

References

[FKW18] Steffen Frerix, Peter Koepke, and Makarius Wenzel. Naproche-SAD. https://github.com/Naproche/Naproche-SAD, 2018.

[Har18] John Harrison. Let's make set theory great again! http://aitp-conference.org/2018/slides/JH.pdf, 2018.

[Isa] Isabelle. https://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[KCKS09] Daniel Kühlwein, Marcos Cramer, Peter Koepke, and Bernhard Schröder. The naproche system. *Intelligent Computer Mathematics, Springer LNCS, ISBN*, 978:3–642, 2009.

[Kel55] John L Kelley. General topology. *Graduate Texts in Mathematics*, 27, 1955.

[Koh14] Michael Kohlhase. Mathematical knowledge management: Transcending the one-brainbarrier with theory graphs. *EMS Newsletter*, 92:22–27, 2014.

[Met] Metamath. http://us.metamath.org/index.html.

[Miz] Mizar. http://mizar.org/.

[Pas07] Andriy Paskevych. *Méthodes de formalisation des connaissances et des raisonnements mathématiques: aspects appliqués et théoriques*. PhD thesis, Université Paris 12, 2007.

[Rud76] Walter Rudin. Principles of mathematical analysis. *McGraw-Hill, Inc.*, 1976.

[Wen18] Makarius Wenzel. Isabelle/jEdit as IDE for domain-specific formal languages and informal text documents. *arXiv preprint arXiv:1811.10819*, 2018.

[Wie07] Freek Wiedijk. Mizar's soft type system. In *Theorem Proving in Higher Order Logics*, pages 383–399. Springer, 2007.

First Experiments with Data Driven Conjecturing*

Karel Chvalovský[1], Thibault Gauthier[1], and Josef Urban[1]

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

An essential part of mathematics and the work of a mathematician is to produce conjectures. This is also an important problem in automated theorem proving. (Un)fortunately, already humans have different opinions on what is a good conjecture and hence an objective function for ranking conjectures is hard to specify. It is even less clear how conjectures are discovered.

There have been various attempts to produce conjectures automatically. Well known examples like Lenat's AM (Automated Mathematician) [9], a more specialized Graffitti by Fajtlowicz [4], and Colton's HR [3] are based on human curated rules for generating conjectures. In small domains, exhaustive brute force generation can be useful, in particular when controlled by a type system and further semantic pruning [7].

Using Distributed Representations for Conjecturing: Our approach is different. We do not want to write down rules describing interesting conjectures directly, but we would like to *learn* meaningful conjecturing from a large corpus of mathematical proofs. For that, a better semantic understanding of such corpora is needed. It is possible to use distributional semantics approach, where we try to learn semantic similarities among concepts solely based on their co-occurrences in a corpus. This has proven to be very successful in computational linguistics [10]. A *notion* (concept) is then represented by a low-dimensional vector. One of the interesting aspects of such a representation are analogies via linear algebra. Let $\mathbf{v}_\cap$, $\mathbf{v}_\cup$, and $\mathbf{v}_\wedge$ be the vector representations of $\cap$, $\cup$, and $\wedge$, respectively. Then we can answer a question *"What is* $\wedge$ *as* $\cup$ *is to* $\cap$*?"* by finding $\mathbf{v}$ such that $\mathbf{v}_\wedge - \mathbf{v}$ is most similar to $\mathbf{v}_\cap - \mathbf{v}_\cup$. Such analogies can be used for free-style conjecturing similar to [6].

A straightforward application of this idea is to learn such representations over a large formal library, in our case we use the Mizar [2] Mathematical Library (MML). Given a statement s, for example, $x \cap y = x \to x \cup y = y$, we can identify an important notion in s that we would like to shift, e.g., $\cap$ represented by $\mathbf{v}_\cap$. Now we look for a vector that is close to $\mathbf{v}_\cap$ such that it is a binary function. If we are lucky $\mathbf{v}_\wedge$ is close and hence we would like to replace $\cap$ in s by $\wedge$. We should also replace $\cup$ by a binary function represented by a vector that is to $\mathbf{v}_\cup$ as $\mathbf{v}_\wedge$ is to $\mathbf{v}_\cap$. It could be $\mathbf{v}_\vee$ and hence we obtain $x \wedge y = x \to x \vee y = y$ as a new statement.

However, here we have made several decisions and it is rather unclear how to make them automatically. Before we start to discuss them, it is worth mentioning that our situation is significantly different from the situation in natural language processing (NLP). We use the Mizar formal library so for every statement we have a parse tree. Moreover, if we produce a new statement from an old one, we can try to check by an automated theorem prover (ATP) whether it is provable or disprovable, because we can work directly with a TPTP [11] translation of the Mizar statement [12]. Although it is generally very difficult to disprove a statement, in our case it is possible to do that for trivially invalid statements, which we will often produce. Similarly, we can filter trivially valid statements.

Now back to our problem. We can use a distributed representation of notions and statements such as [1]. Given a statement s we can find an important notion N in it and shift it (i.e., its vector). Here N should be a predicate, function, or a constant. Once we do that, we can

*Supported by the ERC Consolidator grant no. 649043 AI4REASON and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

look for a semantically similar notion of the same type. This search can be unrestricted, or we can look for notions that, e.g., appear only in different Mizar articles. When we find a suitable notion (or more notions), we can start to shift notions in our statement from the most important to the least important. It is unlikely that a vector for a new notion is exactly at the position where we expect it, therefore we can use the previous shifts to correct the new ones. The question when to stop shifting and keep the rest of the statement intact can be left open, because we can generate all possible variants and remove those that are trivially (in)valid.

This procedure can be improved in various ways, e.g., by using a beam search with the possibility to keep a notion intact even when less important notions are shifted. However, so far we have obtained only weak results with this method. It probably suffers from the fact that it is hard to keep all parts synchronized. A single error can spoil the whole translation, and even more importantly, it is usually necessary to shift different parts of statements differently.

Consistency by NMT: The recently developed neural machine translation (NMT) architectures provide a different and possibly better approach. It was shown recently that we can use NMT for simple informal to formal translations [14]. Here, the above mentioned semantic relations between the notions are learned as part of the training process. Moreover, the inner consistency of the translated result is controlled directly by NMT. That is even incorrect results are likely to parse and the notions are combined meaningfully. For example, in the encoder-decoder neural architectures a hidden state (vector) characterizing the translation done so far is updated after each decoding step, and the choice of the next decoded symbol is statistically conditioned on the state, making the resulting combinations of symbols statistically plausible.

We can formulate our conjecturing task as a translation problem—translate an already known statement s into a conjecture t. How can we produce a sufficient amount of training data $\{\,(s,t)\colon s \text{ translates into } t\,\}$ for such a task? Assume we have a statement s and we can say that statements $t_1,\ldots,t_n$ in our library are somehow relevant to s. We can then try to confuse NMT by adding n training examples $\{(s,t_1),\ldots,(s,t_n)\}$ and hence NMT will then attempt to translate s into a statement that is most similar to all $t_1,\ldots,t_n$.

For an initial experiment, we produce abstracted common patterns (e.g. commutativity, associativity, etc.) from all Mizar toplevel statements using Gauthier's patternizer [5] used previously for concept alignment and conjecturing based on them [6]. The patternizer finds about 16000 patterns that generalize at least two statements. From them we create a corpus of about 1.3 million (non-unique) translation pairs by making an input-output pair from all statements that are instances of the same pattern. This means that NMT will be trained to analogize on many examples, and due to the large non-determinism in the training data it may produce a new formula that will likely be syntactically consistent. This is indeed often the case on a test set of about 30000 unique statements that after the training result in about 16000 formulas that do not appear in MML. A very simple example generated by this conjecturing approach is $(X \cap Y) \setminus Z = (X \setminus Z) \cap (Y \setminus Z)$ produced from $(X \cup Y) \setminus Z = (X \setminus Z) \cup (Y \setminus Z)$.

Although it is a trivial duality statement, it should be noted that it was produced completely automatically without any intervention from outside and it is not in the Mizar library. Moreover, there is no need to check for a correct substitution, cf. [5], this part is handled by NMT itself. This statement can be proved automatically by the MizAR hammer [13, 8]. Examples of false but syntactically consistent conjectures generated automatically in this way include:

```
for n, m being natural numbers holds n gcd m = n div m;
for R being Relation holds with_suprema(A) <=> with_suprema(inverse_relation(A));
```

In this initial experiment, we say that two statements with a common pattern are relevant to each other. There are many other untested options, for example, we can say that a statement t is relevant to s if t occurs in a proof of s, or vice versa.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR 2017*, 2017.

[2] Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015.

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

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

[5] Thibault Gauthier and Cezary Kaliszyk. Aligning concepts across proof assistant libraries. *J. Symb. Comput.*, 90:89–123, 2019.

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

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

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

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

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[11] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

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

[13] Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50:229–241, 2013.

[14] Qingxiang Wang, Cezary Kaliszyk, and Josef Urban. First experiments with neural translation of informal to formal mathematics. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2018.

Project Proposal: Neural Modelling of Mathematical Structures*

Martin Smolík and Josef Urban

[1] Charles University
[2] Czech Technical University, Prague,

Introduction and planned experiments

In this project we will focus on the usage of deep learning in building a "mathematical intuition" for a machine. The ultimate goal is to make a neural network that will attempt to distinguish between true and false statements about a mathematical structure, based on training on a provided knowledge base of true and false statements. For more complicated mathematical structures such as natural and real numbers, set-theoretical universes, or just any very large finite object, our knowledge base may be incomplete, infinite and/or non-computable. However, we will first mainly focus on a few specific cases, where we will try to build a network that "understands" group operations or real numbers.

Initially, we will focus on small finite structures, in particular groups, and try to build networks that will emulate the group operations $\cdot$ and $^{-1}$, as well as the constant e. The sequence of planned experiments is roughly as follows:

- Learn finite structures directly from their multiplication/interpretation table(s).
- Learn finite structures from sets of (universally) quantified sentences.
- Learn finite structure from sets of randomly chosen or generated sentences.
- Learn infinite structures in a similar way.

At each point we can evaluate the usefulness of the trained neural approximations directly on a knowledge base of true/false statements, but also in more complex tasks such as estimating the truth of conjectures generated automatically by various conjecturing methods.

Experiments done on finite groups so far

Short introduction to groups: A group is a mathematical structure that has 2 operators (in our case called "composition" - denoted $\cdot$ and "inverse" - denoted $^{-1}$) and one constant, called "unit" - e. These operators and the constant must satisfy some axioms (for any elements).

- Associativity: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
- Unit: $x \cdot e = x = e \cdot x$
- Inverse: $x \cdot x^{-1} = e = x^{-1} \cdot x$

Learning of the composition operator: Our main focus are now on small permutation groups. Composition operator is learned from the multiplication table. We construct a multilayer feedforward neural network that tries to estimate the result. For this we choose a grounding - a representation of the group elements as vectors of a chosen size. We construct training data by taking groundings of pairs of random elements and we pre-compute the result of the composition. The network processes the input pairs, we measure the error and backpropagate. Because we use the multiplication table, our data will always satisfy the associativity axiom.

*Supported by the ERC Consolidator grant no. 649043 AI4REASON and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

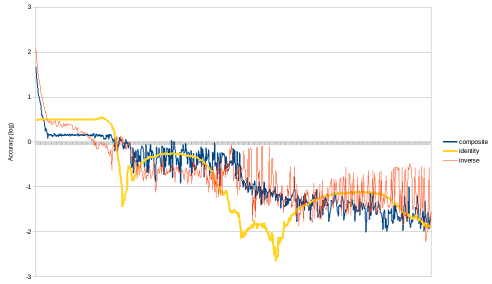


Figure 1: Accuracy during training for S_3.

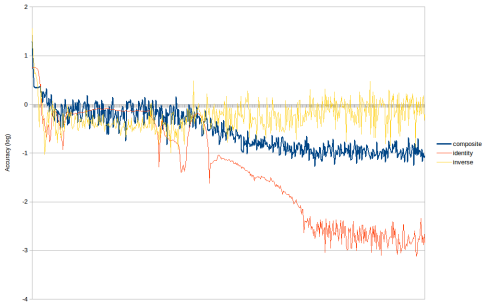


Figure 2: Accuracy for S_4.

Learning of the unit and inverse operator: The unit and the inverse operator are trained differently. We try to learn them by trying to satisfy universally quantified formulas. We pre-train the composition operator, and we use it to train the networks for the unit and the inverse operator. Unit is represented as a single (learned) tensorflow variable. Using the composition operator and a random element x, we estimate $x \cdot e$. This should always be equal to x. We measure the error and adjust our estimation of e accordingly. Here we do not change any variables in the pre-trained composition operation. For the inverse operator we also use this method with the inverse axiom. Inverse operator, like composition, is a multilayer feedforward neural network. For a random element x we estimate $x \cdot x^{-1}$. This should be equal to our estimated value of e. We therefore measure the error and backpropagate in the inverse's network. We once again do not touch the composition nor unit.

Results

We have built a framework in tensorflow [1] that supports learning from both tables and universally quantified formulas. In Figure 1 we can see the accuracy of operations in a group S_3 in one of the early implementations. Training consisted of almost 100 000 epochs. All operations were trained at the same time, although optimizers were able to alter only their specific variables. Because of implementation difficulties in tensorflow the identity and inverse were trained in batches of 1. That is why in case of the inverse operator the accuracy has such a high variation.

So far we have used a very basic grounding, where each permutation is represented in $\mathbb{R}^3$ by its one-line notation, e.g. the identity is represented by the vector $[0, 1, 2]$. The loss function that was used was $(|v_c - v_e|_{L_1})^2$ where v_c is computed and v_e is the expected value. Such a grounding and loss function are however not optimal and may result in large inaccuracies when working with larger groups. The accuracy is computed as $|v_c - v_e|_{L_1}$ with v_c and v_e as above. The metric used is the L_1 metric. We use base 10 logarithm, which means that in the end the error was close to 0.01.

Figure 2 shows the graph for training of the group S_4 with similar groundings and loss functions as for S_3. The training went for 1 000 000 epochs. We can see that the inverse operator is severely limited by the size of the training batch. We can also see that the identity is very accurate despite having the same disadvantage. The talk will discuss further experiments with training groups and other mathematical structures.

The networks described here use 4 hidden layers with 9 nodes each. The activation function is leaky ReLu. The optimizer is a default tensorflow Adam optimizer, therefore the learning rate is quite low. Our project's goal also includes finding better architectures.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 265–283. USENIX Association, 2016.

SMAC and XGBoost Your Theorem Prover

Edvard K. Holden and Konstantin Korovin

The University of Manchester, U.K.

Abstract

Heuristic selection for automated theorem provers [6, 7, 11] has received considerable attention in recent years [1, 4, 5, 8, 9, 10, 12]. Various heuristics for proof searching yield dramatically different solving times for different problems. In this paper we introduce methods for learning good heuristics for sets of diverse problems via parameter optimisation and dynamic clustering. We also propose a method for predicting the optimal heuristic and estimating the solving time of a given problem. We divide the system into two phases: the heuristic learning phase (HLP), and the heuristic mapping phase (HMP).

Heuristic Learning Phase: HLP

The goal of this phase, presented in Algorithm 1, is to learn diverse heuristics using SMAC [3] and at the same time cluster problems based on heuristics performance. SMAC (Sequential Model-based Algorithm Configuration) is a hyperparameter optimiser which can optimise both numerical and categorical parameters. It builds a model for selecting promising configurations, by creating an ensemble of regression trees over the space of the parameter options.

We can run SMAC over a collection of problems with the goal of optimising the number of problems solved. However, problem sets are usually diverse: different problems require different heuristics, and learning a single heuristic which is globally optimal does not necessarily result in a useful heuristic for specific problems. We propose to cluster similar problems and optimise heuristics for each cluster separately. To create collections of problems with some similarity, we cluster the problems based on the syntactic problem features and in addition we use dynamic features representing the heuristics performance on the problems. These features are normalised, weighted and combined to create a feature vector which is used to produce K problem clusters by applying the K-means algorithm.

A syntactic feature is a feature representing syntactic properties of the problem. Such features can be the number of EPR or Horn clauses. A selection of syntactic features constructs $problem\_feature\_vector$. The $heuristic\_evaluation\_vector$ consists of the solving times for the problem when ran over the set of heuristics. A special value is used for time outs. We combine $problem\_feature\_vector$ and $heuristic\_evaluation\_vector$ to cluster problems and run the SMAC heuristic optimisation over these clusters separately (the inner for loop in Algorithm 1). After optimising the heuristics over the clusters, we evaluate the best local heuristics over the whole problem set and re-cluster problems based on the new heuristics performance.

Heuristic Mapping Phase: HMP

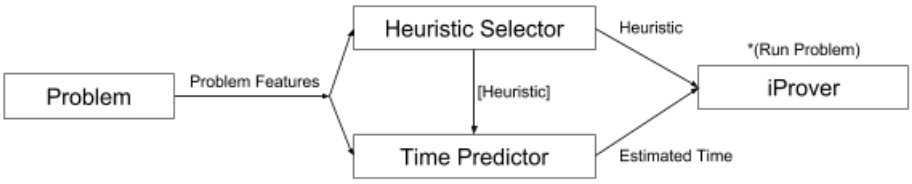
The second phase consists of building an automatic heuristic selector which selects the optimal heuristic concerning the solving time from a set of heuristics. This heuristic set is the result of the HLP phase as it has discovered optimal heuristics for subsets of problems. Based on this data we construct the dataset $D = [(x_1, y_1), ..., (x_n, y_n)]$ where x is the $problem\_feature\_vector$ and y is the optimal heuristic for the problem. Hence, we can represent the heuristic mapping by the function $f : x \to y$. We can approximate this function by utilising supervised machine learning methods. In particular, we can use XGBoost [2] which is an implementation of Gradient

Algorithm 1 Heuristic Learning Phase

```
best_heuristics ← get_initial_heuristics()
problem_feature_vector ← get_problem_features()
repeat
    heuristic_evaluation_vector ← compute_proving_times(best_heuristics)
    problem_clusters ← kmeans(problem_feature_vector, heuristic_evaluation_vector)
    new_heuristics ← ∅
    for cluster ∈ problem_clusters do
        problem_sample ← get_problem_sample(cluster)
        run_SMAC(best_heuristic(cluster), problem_sample)
        new_heuristics ← new_heuristics ∪ get_optimal_heuristics_from_SMAC_run()
    end for
    new_heuristics ← get_top_heuristics(problem_clusters)
    best_heuristics ← best_heuristics ∪ new_heuristics
until Timeout
```

Boosting Machines. This model is known to be quite fast, have state-of-the-art performance and to control overfitting better than most alternatives. It is therefore reasonably to assume that the model will perform well on this dataset, even though it is likely to be imbalanced.

Figure 1: Heuristic Mapping and Regression Overview



There may be the case that a heuristic cannot solve a problem. This occurs either if the model predicted the wrong heuristic or the problem is previously unseen, in which case a heuristic which can solve the problem does not exist in the set. These situations can waste an unnecessary amount of resources, as the prover will not be able to solve the problem within a global time constraint. To reduce this resource waste, we propose to create a regression model which estimates the solving time of a problem. The problems can be represented by the $problem\_feature\_vector$. As the runtime is dependent on the heuristic, we can encode the heuristic as a one-hot vector to get a proper representation of the input to the prover. This regression model can be constructed by non-linear regression using XGBoost or neural networks.

Conclusion

The HLP phase is implemented as a Python wrapper around the SMAC framework which runs iProver as its target function. SMAC has an option for sharing the model between multiple SMAC instances which allows us to optimise the same model over several server nodes. All the experiment data is stored in a database. The HMP phase interacts with the database and the machine learning models, both for training and evaluation.

Experimental results over the CASC'18 FOF data set show that the HLP-prediction phase manages to increase the number of solved problems by 24% in 30 hours, starting with the

default iProver heuristic. The HMP phase reduces the average solving time over jointly solved problems by 40%.

One of the related methods is BlistrTune [5] which interleaves a search for high-level parameters with fine-tuning for the heuristic invention. Our approach differs as it does not interleave during the search but utilises a model based configuration algorithm, as well as it optimises over subsets of problems from dynamically generated clusters.

In [1] we see automatic heuristic selection using Support Vector Machines and Gaussian Processes on a general set of heuristics, whereas we utilise a tree boosting algorithm on a specialised set of heuristics. The system E-MaLeS [8] uses a Gaussian Kernel to perform heuristic selection by selecting the best time estimate of a problem for each candidate heuristic. Our approach differs by treating heuristic selection as a classification task and time estimation as a regression problem. The separation of concerns may lead to better performing components which can significantly improve the overall system.

References

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

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

[3] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Proc. of LION-6*, pages 55–70, 2012.

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

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

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

[7] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV. Proceedings*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

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

[9] Giles Reger, Martin Suda, and Andrei Voronkov. New Techniques in Clausal Form Generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016.

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

[11] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR-19*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

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

Can a Failed Strategy be Useful? (A Research Proposal)

Giles Reger[1] and Martin Suda[2]

[1] University of Manchester, Manchester, UK
[2] Czech Technical University in Prague, Czech Republic

Modern automatic theorem provers for first-order logic (ATPs), such as E [7][1] or Vampire [2][2], provide many *options* for setting up and tuning the deduction algorithm, thus giving rise to a huge number of possible *strategies* that can be used to attack a problem. To achieve their best performance, these ATPs employ *strategy scheduling*, a technique in which they execute a *sequence* of strategies of complementary strengths, each for a certain fraction of the allotted time. There are usually about a dozen of such pre-selected sequences and the prover picks one based on a number of simple problem features such as the size of the signature, presence of equality, the shape of clauses, etc. Apart from this initial choice, however, the strategies are executed in a fixed sequential order (or, optionally, in parallel) until one of them finds a solution or an overall time limit is reached. In particular, there is no sharing of information between the strategies, which can thus be viewed as independent attempts at solving the problem. In this work, we want to investigate to what degree could this simple setup be improved by extracting information from the failed runs of earlier strategies and making use of this information during the execution of the later strategies.

There seem to be at least two conceptually distinct ways of elaborating this idea. In the first one, we ask whether the *proof search characteristics* of a failed run could be used as additional features of the given problem, helping to choose the next strategy to try on the problem in a more informed way. In this view, running a strategy is not just an attempt at solving the problem, but also a probe aimed at learning more about it. In the second, we ask to what degree it would be possible to retain certain parts of *logical information* derived from the problem and successfully reuse them in the subsequent strategy runs in the form of lemmas. These lemmas should ideally be non-obvious (i.e. hard to derive) consequences of the problem formulation to represent enough invested work to be worth transferring. At the same time, they should be of sufficiently simple form not to overload or distract the freshly starting strategy. Ultimately, these considerations lead to an architecture with a large room for experimental tuning (subject to a particular target distribution of problems). We eventually want to investigate how to make this tuning fully automatic by posing the task as an instance of *reinforcement learning*.

Focusing on failures The second aspect of our proposed improvement, i.e. that of sharing logical information between individual strategies, has already been studied in the past in various incarnations, see e.g. [1, 6, 9, 12]. Although encouraging results were always reported, the fact that none of the current "mainstream"[3] ATPs is employing the idea suggests that the ensuing architectures might be hard to maintain or, perhaps, that further theoretical understanding is needed before the idea can be applied in a sufficiently principled way.

That is why we would first like to further our general understanding of the behaviour of strategies by focusing on the first aspect mentioned above, i.e. the question of to what extent can the information about a failed strategy be useful to eventually solving a given problem.

[1] http://www.eprover.org/
[2] https://vprover.github.io/
[3] Such as the ones participating in the annual CASC competition [10].

Proof search characteristics that could be readily used here include the number of generated and selected clauses, the number of performed inferences and reductions of each kind or the amount of time spent by the prover in the individual proving sub-routines such as parsing, preprocessing, reductions, or the maintenance of individual data structures. An encouragement that such runtime statistics can be informative comes from our previous work on the evolution of simplification orderings [5] as well as from the work on predicting the success of strategies at runtime [4]. We even believe that these characteristics could completely replace the more classical static problem features used for problem clustering in the past [3].

A data-mining experiment The strategy scheduling mode (a.k.a. CASC mode) of Vampire 4.3.0, consists of approximately 1200 strategies in 39 scheduled sequences dispatched according to simple syntactic properties of the given problem. As our initial experiment, we intend to evaluate these strategies on a feasible subset of the TPTP library [11], collecting the information about successful runs as well as proof search characteristics of the failed runs. This will provide a data-set for the evaluation of our hypothesis that the characteristics of the failed runs, when used as features of a problem, can be used to improve the selection process of a strategy that will succeed on the problem.

We would also like to use the collected data to further explore and statistically quantify previously established phenomena such as 1) the inherent fragility of the space of strategies: Are there certain options parametrising a strategy the change of which tends to have a continuous effect on the strategy's behaviour or is the behaviour almost always chaotic? 2) the effect of sub-linearity [8], usually expressed as the observation that: If a problem can be solved by a strategy then it typically can be solved within a short amount of time. Ultimately, we would like to shed more light on the question 3) to what degree is predicting a good strategy for a particular problem actually feasible or practical and to what degree is "just quickly trying sufficiently orthogonal strategies in succession" enough to solve all the solvable problems.

References

[1] J. Denzinger and M. Kronenburg. Planning for distributed theorem proving: The teamwork approach. In *KI-96: Advances in Artificial Intelligence, 20th Annual German Conference on Artificial Intelligence, Dresden, Germany, September 17-19, 1996, Proceedings*, vol. 1137 of *Lecture Notes in Computer Science*, pp. 43–56. Springer, 1996.

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

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

[4] M. Rawson and G. Reger. Dynamic strategy priority: Empower the strong and abandon the weak. In *PAAR 2018, Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning co-located with Federated Logic Conference 2018*, 2018.

[5] G. Reger and M. Suda. Measuring progress to predict success: Can a good proof strategy be evolved? (extended abstract). In *AITP 2017, The Second Conference on Artificial Intelligence and Theorem Proving, Abstracts of the talks*, pp. 20–21, 2017.

[6] G. Reger, D. Tishkovsky, and A. Voronkov. Cooperating proof attempts. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, vol. 9195 of *Lecture Notes in Computer Science*, pp. 339–355. Springer, 2015.

[7] S. Schulz. System Description: E 1.8. In *Proc. of the 19th LPAR, Stellenbosch*, vol. 8312 of *LNCS*. Springer, 2013.

[8] G. Stenz and A. Wolf. E-SETHEO: design, configuration and use of a parallel automated theorem prover. In *Advanced Topics in Artificial Intelligence, 12th Australian Joint Conference on Artificial Intelligence, AI '99, Sydney, Australia, December 6-10, 1999, Proceedings*, vol. 1747 of *Lecture Notes in Computer Science*, pp. 231–243. Springer, 1999.

[9] G. Sutcliffe. The design and implementation of a compositional competition-cooperation parallel ATP system. In *Proceedings of the 2nd International Workshop on the Implementation of Logics*, pp. 92–102, 2001.

[10] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[11] 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.

[12] A. Wolf and M. Fuchs. Cooperative parallel automated theorem proving. Technical report, SFB Bereicht 342/21/97, Technische Universität München, 1997.

Tactic Learning for Coq

Lasse Blaauwbroek*

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

Abstract

We present ongoing work being performed to utilize Artificial Intelligence for proof search in the Coq theorem prover. In a similar vein as the TacticToe project for HOL4 [4], we are working on a system that finds proofs of goals on the tactic level, by learning from previous tactic scripts. Learning on the level of tactics has several advantages over more low-level approaches. First, this allows for much coarser proof steps, meaning that during proof search more complicated proofs can be found. Second, it allows for the usage of custom built, domain specific tactics that where previously defined and used in the development. This will allow for better performance of the system in very specialized domains. Currently our system is not yet capable of proof search but does support live tactic learning and provides feedback to the user in the form of tactic suggestions. The rest of this abstract will describe the required components of our system and an evaluation of our prediction system.

Proof Recording The first component of the system is the recording of previous proofs. As said, this is done on the level of tactics. When a tactic script is executed, we record the goal state before and after the execution of each tactic. The diff between the state before and after the tactic then represents the action that has been performed by a tactic. By recording many of these instances for a tactic, we create a database that contains an approximation of the semantic meaning of tactics.

A major question here is what exactly constitutes a tactic in Coq. On a low level, Coq utilizes a backtracking monad in which tactics can be written [8]. This monad is not immediately accessible by end-users. For that, a number of tactic languages exist that are then compiled into the backtracking monad. The most used one is the Ltac language [3]. In an ideal world, we would record tactics on the level of the proof monad since that would allow us to record tactics from all existing tactic languages. However, this turns out to be a major technological challenge. Therefore we have chosen to only record tactics of the Ltac language.

Withing the Ltac language, it is also not immediately clear what a tactic is. One option is to decompose a script into a series of primitive tactic invocations, and record those. On the other side of the extreme, one could view every vernacular command as one whole tactic. The first option means that the advantages of the system are greatly diminished, because then we are working on a very low level and no custom tactics will be recorded. The second option means that almost all tactics will be unique. The best solution is likely to lie somewhere in between. At the moment, we see every vernacular command as one tactic, with the exception of tactic composition and tactic dispatching. In order to record a tactic script, conceptually we replace the tactic with a custom recording tactic that receives the original tactic as an argument. For example, the tactic script `tac1; [tac2 | tac3]; tac4` will be converted to `r (tac1); [r (tac2) | r (tac3)]; r (tac4)`, where `r` is the recording tactic. The recording tactic first records the proof state before the tactic, then executes the original tactic and

*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)

finally records the proof state after the tactic. Our current approach also means that if tactics require arguments, then every instance of this tactic with a different argument will be seen as a unique tactic. In the future we intend to change this by also performing some machine learning on the parameters of tactics.

Tactic recording can happen both in batch mode and interactive mode. When recording in batch mode, an entire Coq file is processed and a file containing the recorded data of all tactic invocations is outputted. This file can later be used for predictions in other Coq files. In interactive mode our system follows the actions of the user. This means that when a tactic is executed, the system immediately learns from this. When a tactic is removed from the script the system also automatically unlearns that tactic. This ensures that what we learn is consistent with what would be learned is the script gets executed in batch mode.

In order to achieve this, our system is rather deeply integrated with the Coq system itself. The code is writted as a Coq plugin, but also requires a few minor modifications to the Coq source code. This level of integration is a distinguishing feature compared to existing machine learning systems for Coq. Our system is fully implemented in Ocaml and has no external dependencies. The ML4PG system [9] also provides tactic predictions and other statistics but is instead integrated with the Proof General [1] proof editor and requires connections to Matlab or Weka to function. The SEPIA system [5] provides proof search using tactic predictions and is also integrated with Proof General. It should be noted, however that their proof search is only based on tactic traces and does not make predictions based on the proof state. Finally, GamePad [6] is a framework that integrates with the Coq codebase and allows machine learning to be performed in Python. Similar to our system this is in the early stages of development and no evaluation of tactic prediction has been performed yet.

Tactic Prediction After creating a database of tactics and recorded proof states, we now wish to predict the correct tactic to use in order to make progress in a new, unseen proof state. For this, we must find a proper characterization of the proof states. For our initial prototype, we have opted to reuse feature characterization that is already present in the CoqHammer system [2]. CoqHammer characterizes a formula as a vector containing all identifiers and pairs of adjacent identifiers in the abstract syntax tree. To find a list of likely matches for a new goal, a fast k-nearest neighbor algorithm is run on the vectors. We compare neighbors using the Jaccard similarity with TF-IDF weighted features as described by Kaliszyk and Urban [7].

In order to evaluate our predictions, we use Coq's standard library. During the compilation of the library, before every tactic invocation we try to predict the correct tactic using the database collected until that point. We then check if that tactic corresponds with the tactic used in the source file. The standard library consists of 145866 tactic invocations. If one where to predict a random tactic from the database a success rate of 4.9% can be expected. The best possible success rate lies at 67.2%. This is because the correct tactic is not always present in the database. Using the machine learning method described above, we are able to predict the correct tactic 21.7% of the time. When we look for the correct tactic in the top 10 predictions we can increase this number to 47.7%. We also performed an evaluation using a naive Bayes classifier. The results are rather similar with a prediction rate of respectively 20.8% and 45.3%. We expect that even simpler machine learning techniques will also perform reasonably well.

Proof Search From the previous paragraph it becomes clear that it is unlikely that the tactic prediction system will predict the correct tactic every time. For this reason, a proof search must be performed. In the TacticToe system for HOL4, initially an A*-style algorithm was used to guide the search. Later, taking inspiration from AlphaGo Zero [10] a Monte Carlo Tree Search algorithm was used. We are currently working on a similar search system. This has turned out to be a challenge because a tactic that was intended for one location in a script can not always easily be executed in a different location of a script.

References

[1] David Aspinall. Proof general: A generic tool for proof development. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 38–42. Springer, 2000.

[2] Łukasz Czajka and Cezary Kaliszyk. Hammer for coq: Automation for dependent type theory. *Journal of Automated Reasoning*, pages 1–31, 2018.

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

[4] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Tactictoe: Learning to reason with HOL4 tactics. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, pages 125–143, 2017.

[5] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: search for proofs using inferred automata. In Amy P. Felty and Aart Middeldorp, editors, *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.

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

[7] Cezary Kaliszyk and Josef Urban. Stronger automation for flyspeck by feature weighting and strategy evolution. In Jasmin Christian Blanchette and Josef Urban, editors, *Third International Workshop on Proof Exchange for Theorem Proving, PxTP 2013, Lake Placid, NY, USA, June 9-10, 2013*, volume 14 of *EPiC Series in Computing*, pages 87–95. EasyChair, 2013.

[8] Florent Kirchner and César A. Muñoz. The proof monad. *J. Log. Algebr. Program.*, 79(3-5):264–277, 2010.

[9] Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in proof general: Interfacing interfaces. In Cezary Kaliszyk and Christoph Lüth, editors, *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012.*, volume 118 of *EPTCS*, pages 15–41, 2012.

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