AITP 2017

The Second Conference on Artificial Intelligence and Theorem Proving

Abstracts of the talks

March 26 – 30, 2017, Obergurgl, Austria

Preface

This volume contains the abstracts of the talks presented at AITP 2017: The Second Conference on Artificial Intelligence and Theorem Proving held on March 26 – 30, 2017 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 2017 consists of several sessions discussing connections between modern AI, ATP, ITP and (formal) mathematics. The sessions are discussion oriented and based on 12 invited talks and 13 contributed talks.

We would like to thank the University of Innsbruck for hosting AITP 2017 at its conference center in Obergurgl. 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).

Finally, we are grateful to all the speakers, participants and PC members for their interest in discussing and pushing forward these exciting topics!

March 24, 2017
Pittsburgh
Innsbruck
Stuttgart
Prague

Thomas C. Hales
Cezary Kaliszyk
Stephan Schulz
Josef Urban

Table of Contents

Program Committee

Noriko Arai	National Institute of Informatics
Jasmin Christian Blanchette	Inria Nancy & LORIA
Ulrich Furbach	University of Koblenz
Deyan Ginev	Jacobs University Bremen
Thomas Hales	University of Pittsburgh
Sean Holden	University of Cambridge
Geoffrey Irving	Google
Cezary Kaliszyk	University of Innsbruck
Jens Otten	University of Oslo
Claudio Sacerdoti Coen	University of Bologna
Stephan Schulz	DHBW Stuttgart
Geoff Sutcliffe	University of Miami
Josef Urban	Czech Technical University in Prague

What Has Artificial Intelligence Ever Done For Us (Formalizers)?

John Harrison

Intel

Abstract. In this talk I will first give a historical overview of the role, or lack of it, that artificial intelligence has played during the development of automated and interactive theorem proving. I will then discuss the (still mostly relatively recent) work in applying learning for interactive provers. I will first talk a little about some of the existing successes of these methods and then speculate freely, but I think plausibly, about how they might lead to even more exciting applications in the future. I will focus particularly on using such techniques to help in the formalization of mathematical analysis, geometry and topology, since this is an area in which I have a particular personal interest and experience.

Deep Reasoning - A Vision for Automated Deduction

Stephan Schulz

DHBW Stuttgart
schulz@eprover.org

Abstract

Artificial intelligence and automated theorem proving have been co-developing and partially overlapping since the late 1950s, when general purpose computers became reasonably available to researchers. During that time, theorem proving has mostly kept a low but consistent profile. AI in general, on the other hand, has seen a number of waves, characterized by new technologies raising high hopes, but sooner or later reaching its limits and losing momentum. And while each new wave often resulted in significant achievements, they all fell short of the original expectations, leaving the impression of disappointment and failed goals.

The latest resurgence of AI, mostly driven by Deep Learning, is characterized by a number of features: It is data-driven, it is compute-intensive, and it is mostly relying on sub-symbolic, connectionist representations. It has achieved massive successes in fields that have long been considered to be relatively easy for humans, but very hard for computers, such as image recognition, voice recognition, or heuristic evaluation.

The deep learning approach corresponds to intuitive human learning, to acquired experience and maybe to a degree of "common sense". This kind of knowledge has significant drawbacks: It is hard to explain, it is hard to communicate, it is hard to review and revise, and it is hard to build on. In human society, it is complemented by the scientific approach to formulate, test, and refine abstract hypotheses that can be explicitly taught, explained, and communicated. I will argue that a similar integration of symbolic reasoning as exemplified by automated theorem proving and sub-symbolic, data-driven approaches like deep learning will enable us to step beyond the obvious limits of either of the two individual approaches.

Can Deep Learning Help Formal Reasoning?

Sarah Loos

Google Research

Abstract. Deep learning techniques lie at the heart of several significant AI advances in recent years including object recognition and detection, image captioning, machine translation, speech recognition and synthesis, and playing the game of Go. Automated theorem provers can aid in the formalization and verification of mathematical theorems and play a crucial role in program analysis, theory reasoning, security, interpolation, and system verification. We discuss deep learning based premise selection and guidance in the proof search of the theorem prover E. We train and compare several deep neural network models on the traces of existing ATP proofs of Mizar statements and use them to select processed clauses during proof search. We also discuss a newly released dataset based on Higher-Order Logic (HOL) proofs, for the purpose of developing new machine learning-based theorem-proving strategies. We make this dataset publicly available under the BSD license.

Deep Prolog: End-to-end Differentiable Proving in Knowledge Bases

Tim Rocktäschel

University College London

Abstract. In this talk, I will present a class of deep neural networks that we termed Neural Theorem Provers (NTPs): end-to-end differentiable counterparts of symbolic theorem provers that perform first-order reasoning on vector representations of symbols using function-free, possibly parameterized, rules. NTPs are recursively constructed via a differentiable version of the backward chaining algorithm used in Prolog. Specifically, in NTPs symbolic unification is replaced by a differentiable computation on vector representations of symbols.

NTPs follow a long tradition of neural-symbolic approaches to automated knowledge base inference but differ in that they are differentiable with respect to vector representations of symbols, as well as rules of predefined structure. They can be trained to infer new facts from a given incomplete knowledge base using gradient-based optimization methods. By doing so, they learn to (i) place representations of similar symbols in close proximity in a vector space, (ii) make use of such similarities to generalize to unseen facts, (iii) induce latent logic programs, and (iv) use either provided or induced logic programs for complex multi-hop reasoning.

On a knowledge base containing information about country locations and their neighbors, we demonstrate that NTPs outperform existing neural link prediction methods, while at the same time often providing interpretable logic programs.

Formalising mathematical conventions

Georges Gonthier

INRIA

F-ABSTRACTS (formal abstracts): Project Introduction and Discussion

Thomas C. Hales

University of Pittsburgh

Abstract. The talk will introduce the idea of F-ABSTRACTS (formal abstracts) – a service that would abstract a formal statement (including formal definitions) of the main result of each (or most) newly published mathematical papers. No formal proofs would be involved, just definitions and statements. This idea has been included into a proposal submitted by Stephen Watt and Ingrid Daubechies to the Sloan Foundation, which has recently been funded. After the introduction, we can hold a discussion about how to design and implement F-ABSTRACTS.

Tapping Sources of Mathematical (Big) Data

Michael Kohlhase

FAU Erlangen-Nürnberg

Abstract. Many technologies that drive modern science are data-intensive, complementing the human intellect which is capable of deep insights with the breadth of large scale data processing. One of the few areas, where this trend has scarcely taken hold is mathematics, which is largely still an enterprise of deep deliberation supported by pen, paper, blackboards, and LaTeX. Arguably one of the reasons for that is the lack of large, machine-actionable data sets.

In this talk I will survey our efforts to tap – i.e. make machine-actionable by extracting the inherent structure and meaning – of some of the large data sets available, and the services we have built on them.

- We have converted 1.050.000 preprints in the Cornell ePrint archive (arXiv) and 2.8 M abstracts from ZBMath from LaTeX to XML/HTML5 and provide a formula search engine on this corpus.
- We have heuristically parsed the Online Encyclopedia of Integer Sequences (OEIS: ca 280.000 sequences with references, formulae, and inter-sequence relations) and have derived the same order of new arithmetic relations between them (and a formula search engine).
- We are converting 12 theorem prover libraries (Mizar, Hol Light, Isabelle, Metamath, IMPS, ...) into a joint syntactic format semantic framework, aligning them semi-automatically, and provide joint browsing.

This is just a first step, and we need to discuss ways of identifying more data sets (e.g. the math genealogy, author identification, math linguistics, ...) to build a resource which can unleash data-intensive methods on mathematical knowledge.

Number-theoretic conjecturing via zeta types and Tannakian symbols

Andreas Holmstrom and Torstein Vik

Abstract. We introduce two new data formats for representing information about objects from number theory, representation theory, and algebraic geometry. The first data format, which we call a zeta type, is an two-dimensional array of elements of some commutative ring (typically integers) indexed in one direction by the primes and in the other direction by the positive integers. The second data format is a pair of finite disjoint multisets of elements of some monoid (typically the multiplicative monoid of nonzero complex algebraic numbers), and we call this a Tannakian symbol. By giving many examples and some theoretical results, we want to show how these data formats can be used for pattern-detection and conjecturing. If time allows, we also describe an application to automatic generation of identities between multiplicative functions (such as the Euler phi function and the divisor functions).

Logic Tensor Networks (Extended Abstract)

Artur d'Avila Garcez[1] and Luciano Serafini[2]

[1] City, University of London, UK, a.garcez@city.ac.uk
[2] Fondazione Bruno Kessler, Trento, Italy, serafini@fbk.eu

Abstract. *Logic Tensor Networks* provide a well-founded integration of deductive reasoning and relational learning using first-order logic. Logical constants are interpreted as feature vectors of real numbers, logical formulas have truth-value in the interval [0,1] and semantics defined in the domain of real numbers, all implemented in deep tensor neural networks with the use of Google's TENSORFLOW™. Keywords: Knowledge Representation, Relational Learning, Tensor Networks, Neural-Symbolic Computing, Knowledge Completion.

1 Introduction

The recent availability of large-scale data combining multiple modalities, such as image, text and audio, has opened various research and commercial opportunities, underpinned by machine learning methods and techniques [1, 7]. Recent work in machine learning has sought to combine logical services, such as knowledge completion, approximate inference, and goal-directed reasoning with data-driven statistical and neural network-based approaches. We argue that there are great possibilities for improving the current state of the art in machine learning and artificial intelligence (AI) thought the principled combination of knowledge representation, reasoning and learning [9].

Logic Tensor Networks (LTN) integrate learning based on tensor networks [10] with reasoning using first-order many-valued logic [2], all implemented in TENSORFLOW™ [6]. This enables a range of knowledge-based tasks using rich symbolic knowledge representation in first-order logic (FOL) to be combined with efficient data-driven machine learning based on the manipulation of real-valued vectors. In practice, FOL reasoning including function symbols is approximated through the usual iterative deepening of clause depth. Given data available in the form of real-valued vectors, logical soft and hard constraints and relations which apply to certain subsets of the vectors can be specified compactly in FOL. With the use of *Real Logic* and *learning as approximate satisfiability*, reasoning can help improve learning, and learning from new data may revise the constraints thus modifying reasoning.

2 Real Logic

An agent has to manage knowledge about an unbounded, possibly infinite, set of objects $O = \{o_1, o_2, \dots\}$. Some of the objects are associated with a set of quantitative attributes, represented by an n-tuple of real values $\mathcal{G}(o_i) \in \mathbb{R}^n$, which we call *grounding*. For example, a person may have a grounding into a 4-tuple containing some numerical representation of the person's name, height, weight, and number of friends in some social network. The same person may have a different grounding in a different context, and two or more people may have very similar groundings within a context. It

is this similarity that can be explored by the tensor networks for learning, subject to the constraints given by the logical relations. It is also possible that some of the attributes of an object (in the above example, a person) are unknown or missing. Object tuples can participate in a set of relations $\mathcal{R} = \{R_1, \ldots, R_k\}$, with $R_i \subseteq O^{\alpha(R_i)}$, where $\alpha(R_i)$ denotes the arity of relation R_i. We presuppose the existence of a latent (unknown) relation between the above numerical properties, i.e. groundings, and partial relational structure $\mathcal{R}$ on O. Starting from this partial knowledge, an agent is required to: (i) infer new knowledge about the relational structure on the objects of O; (ii) predict the numerical properties or the class of the objects in O. Classes and relations are not normally independent. For example, it may be the case that if an object x is of class C, $C(x)$, and it is related to another object y through relation $R(x, y)$ then this other object y should be in the same class $C(y)$. In logic: $\forall x \exists y((C(x) \wedge R(x, y)) \rightarrow C(y))$. Whether or not $C(y)$ holds will depend on the application: through reasoning, one may derive $C(y)$ where otherwise there might not have been evidence of $C(y)$ from training examples only; through learning, one may need to revise such a conclusion $C(y)$ once examples to the contrary become available. The vectorial representation permits both reasoning and learning as exemplified above and detailed in the next section.

The above forms of reasoning and learning are integrated in a unifying framework, implemented within tensor networks, and exemplified in relational domains combining data and relational knowledge about the objects in [9]. It is expected that, through an adequate integration of numerical properties and relational knowledge, differently from the immediate related literature [5, 3], LTNs will be capable of combining in an effective way first-order logical inference on open domains with efficient relational multi-class learning using tensor networks.

3 Learning as approximate satisfiability

Suppose that $O = \{o_1, o_2, o_3\}$ is a set of documents defined on a finite dictionary $D = \{w_1, ..., w_n\}$ of n words. Let $\mathcal{L}$ be the language that contains the binary function symbol $concat(x, y)$ denoting the document resulting from the concatenation of documents x and y. Let $\mathcal{L}$ contain also the binary predicate Sim which is supposed to be *true* if document x is deemed to be similar to document y. An example of grounding is the one that associates to each document its bag-of-words vector [4].

To check satisfiability on a subset of all the functions on real numbers, recall that a grounding should capture a latent correlation between the quantitative attributes of an object and its relational properties. For example, whether a document is to be classified as from being in the field of Artificial Intelligence (AI) depends on its bag-of-words grounding. If the language $\mathcal{L}$ contains the unary predicate $AI(x)$ standing for "x is a paper about AI" then the grounding of $AI(x)$, which is a function from bag-of-words vectors to [0,1], should assign values close to 1 to the vectors which are close semantically to AI. Furthermore, if two vectors are similar (e.g. according to the cosine similarity measure) then their grounding should be similar.

When a grounded theory is inconsitent, that is, there is no grounding $\mathcal{G}$ that satisfies it, we are interested in finding a grounding which satisfies *as much as possible* of it. For any formula and real number interval, this is a grounding $\mathcal{G}$ that minimizes a *satisfiability error*: an error occurs when a grounding $\mathcal{G}$ assigns a value $\mathcal{G}(\phi)$ to a clause ϕ which is outside the interval $[v, w]$ prescribed by knowledge-base $\mathcal{K}$.

LTNs were implemented as a Python library called `ltn` using Google's TENSORFLOW™. To illustrate knowledge completion in `ltn`, the well-known *friends and smokers* example [8] was used in [9]. Normally, a probabilistic approach is taken to solve this problem, and one that requires instantiating all clauses to remove variables, essentially turning the problem into a propositional one; `ltn` takes a different approach.

The facts contained in the friends and smokers knowledge-base (K) should have different degrees of truth, and this is not known. Otherwise, K would be inconsistent. Our main task is to complete K, that is, find the degree of truth of the facts. Hence, we use `ltn` to find a grounding that best approximates K. Results show that more facts can be learned with the inclusion of background knowledge than through reasoning alone. Similarly, using the symmetry of the friendship relation, `ltn` derives new facts through reasoning. The axioms in the background knowledge are satisfied with a degree of satisfiability higher than 90%.

4 Conclusion and future work

We have proposed *Real Logic* and the use of approximate satisfiability as a uniform framework for the integration of learning and reasoning, whereby knowledge and data are mapped onto real-valued vectors. With such an inference-as-learning approach, relational knowledge and state-of-the-art data-driven approaches can be combined. We are in the process of making the implementation of LTN available in TENSORFLOW™ to enable comparisons with statistical relational learning, neural-symbolic computing, and (probabilistic) inductive logic programming approaches.

References

1. Y. Bengio. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.
2. M. Bergmann. *An Introduction to Many-Valued and Fuzzy Logic: Semantics, Algebras, and Derivation Systems*. Cambridge University Press, 2008.
3. T. R. Besold, A. d'Avila Garcez, G. Marcus, and R. Miikulainen (eds.). Cognitive computation: Integrating neural and symbolic approaches. Workshop at NIPS 2015, Montreal, Canada, April 2016.
4. D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.
5. A. d'Avila Garcez, M. Gori, P. Hitzler, and L. Lamb. Neural-symbolic learning and reasoning (dagstuhl seminar 14381). *Dagstuhl Reports*, 4(9):50–84, 2014.
6. M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
7. D. Kiela and L. Bottou. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proc. EMNLP 2014*, Doha, Qatar, 2014.
8. M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, February 2006.
9. L. Serafini and A. d'Avila Garcez. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *CoRR*, abs/1606.04422, 2016.
10. R. Socher, D. Chen, C. Manning, and A. Ng. Reasoning With Neural Tensor Networks For Knowledge Base Completion. In *NIPS 26*. 2013.

Mathematics, Ontology and Reference

David McAllester
TTI-Chicago

Ontology — the study of what kinds of things there are — is a central question of philosophy, cognitive science and artificial intelligence. This talk addresses mathematical ontology and its relationship to cognitive science. In mathematics we have concepts such as graphs, groups, topological spaces, various kinds of manifolds, group representations, and on and on. In spite of the rich variety of concepts discussed in mathematics, it seems that all mathematical concepts can be identified with types in a fairly simple formal universal type system — a single type-theoretic foundation for mathematics. It will be argued that this type-theoretic foundation, or some closely related system, seems be an innate aspect of human cognition.

Of central interest is the common sense or intuitive notion of isomorphism. The notion of isomorphism seems related to the notion of an application programming interface (API) in computer software. An API specifies what information and behavior an object provides. Two different implementations can produce identical behavior when interaction is restricted to that allowed by the API. For example, textbooks on real analysis typically start from axioms involving multiplication, addition, and ordering. Addition, multiplication and ordering define an abstract interface — the well-formed statements about real numbers are limited to those that can be defined in terms of the operations of the interface. We can implement real numbers in different ways — as Dedekind cuts or Cauchy sequences. However, these different implementations provide identical behavior as viewed through the interface — the different implementations are isomorphic as ordered fields. The axioms of real analysis specify the reals up to isomorphism for ordered fields. The second order Peano axioms similarly specify the structure of the natural numbers up to isomorphism.

Isomorphism and Dependent Pair Types. The general notion of isomorphism is best illustrated by considering dependent pair types. A dependent pair type is typically written as $\Sigma_{x:\sigma}\ \tau[x]$ where the instances of this type are the pairs (x, y) where x is an instance of the type σ and y is an instance of the type $\tau[x]$. The type of directed graphs can be written as $\Sigma_{\mathcal{N}:\mathbf{Set}}\ (\mathcal{N} \times \mathcal{N}) \to \mathbf{Bool}$. The instances of this type are the pairs $(\mathcal{N}, P)$ where $\mathcal{N}$ is the set of nodes of the graph and P is a binary predicate on the nodes giving the edge relation. Two directed graphs $(\mathcal{N}, P)$ and $(\mathcal{M}, Q)$ are isomorphic if there exists a bijection from $\mathcal{N}$ to $\mathcal{M}$ that carries P to Q. Some bijections will carry P to Q while others will not. The class of all groups and the class of all topological spaces can each be written as subclasses of dependent pair classes.

Observational Equivalence. Intuitively, isomorphic objects are "observationally equivalent" — they have the same observable properties when access to the objects is restricted to those operations allowed by the type system. In MoTT this observational equivalence is expressed by the following substitution rule.

$$\frac{\Gamma \vdash \sigma, \tau : \mathbf{Class} \quad \Gamma; x:\sigma \vdash e[x]:\tau \quad \Gamma \vdash u =_\sigma w}{\Gamma \vdash e[u] =_\tau e[w]}$$

Here $=_\sigma$ is the isomorphism relation associated with the class σ and similarly for $=_\tau$. As an example suppose we have $G:\mathbf{Graph} \vdash \Phi[G]:\mathbf{Bool}$. Intuitively this states that $\Phi[G]$ is a graph-theoretic property. For $G =_{\mathbf{Graph}} H$ we then have $\Phi[G] \Leftrightarrow \Phi[H]$. We do not require that $\Phi[G]$ is a first order formula. For example, $\Phi[G]$ might state that the spectrum of the graph Laplacian of G contains a gap of size λ. As another example we might have $G:\mathbf{Topology} \vdash e[G]:\mathbf{Group}$ where $e[G]$ might be an expression denoting the fundamental group of a topological space. The substitution rule then says that isomorphic topological spaces have isomorphic fundamental groups.

Voldemort's Theorem. There are many situations in mathematics where a type can be shown to be inhabited (have a member) even though there is no natural or canonical member of that type. A classical example is that for every finite dimensional vector space V there is an isomorphism (a linear bijection) between V and its dual V^*. However, there is no natural or canonical isomorphism. This is closely related to the fact that a vector space has no natural or canonical basis. A simpler example is that there is no natural or canonical point on the topological circle S^1. Voldemort's theorem states that if no natural element of a type exists then no well-typed expression can name an element.

Cryptomorphism. Two classes σ and τ are cryptomorphic in the sense of Birkoff and Rota [3] if they "present the same data". For example a group can be defined as a four-tuple of a set, a group operation, an identity element and an inverse operation satisfying certain equations. Alternatively, a group can be defined as a pair of a set and a group operation such that an identity element and an inverse elements exist. These are different classes with different elements (four-tuples vs. pairs). However, these two classes present the same data. Rota was fond of pointing out the large number of different ways one can formulate the concept of a matroid. Any type theoretic foundation for mathematics should account formally for this phenomenon. Here we take two classes σ and τ to be crytomorphic if there exist natural functions $f:\sigma \to \tau$ and $g:\tau \to \sigma$ such that $f \circ g$ and $g \circ f$ are the identity functions on τ and σ respectively.

A particular type system and its semantics (morphoid type theory [2]) will be briefly discussed and contrasted with Martin-Löf type theory. The more general notion of "reference" — how do we refer and what are we referring to — will also be discussed in the broader context of natural language and deep learning models of reading comprehension [1].

References

[1] Zewei Chu, Hai Wang, Kevin Gimpel, and David A. McAllester. Broad context language modeling as reading comprehension. *CoRR*, abs/1610.08431, 2016. To appear in EACL 2017.

[2] David A. McAllester. Morphoid type theory. *CoRR*, abs/1407.7274, 2014.

[3] G.C. Rota. *Indiscrete Thoughts*. Birkhuser Boston, Inc., 1997.

PRIORS ON PROPOSITIONS: TOWARDS PROBABILISTIC PROGRAMMING FOR THEOREM PROVING

CAMERON FREER

Abstract. There has recently been exciting practical progress towards improving proof search and exploring mathematical theories. In this talk, we will consider these problems from a more theoretical perspective — in particular, how they can be viewed through the lens of probabilistic programming. We will first describe proof search and theory exploration in terms of probabilistic inference with respect to rich and complex priors over formulas. This will lead us to questions about the computability and complexity of conditional distributions, the formation of random complete theories, and the possible distributions over theories that give rise to symmetric models. Joint work with Nathanael Ackerman, Rehana Patel, and Daniel Roy.

Remine, Falls Church, VA 22043
E-mail address: cameron@remine.com
URL: http://cfreer.org/

Measuring progress to predict success: Can a good proof strategy be evolved? (Extended Abstract)

Giles Reger[1] and Martin Suda[2]

[1] University of Manchester, Manchester, UK
[2] TU Wien, Vienna, Austria

One of the main parameters of the superposition calculus employed by Automated Theorem Provers (ATPs) is the simplification ordering, and the choice of an ordering can have a huge impact on the success of a theorem proving attempt. However, it is difficult to choose a good ordering in advance and ATPs typically provide only a few heuristical schemes for determining an ordering from a large space of possibilities. The aim of this work is to establish to what extent the space of possible orderings can be better utilised during the construction of new successful proving strategies.

There is a well known principle in automated deduction which states that a strategy that leads to a slowly growing search space will likely be more successful (at finding a proof in reasonable time) than a strategy that leads to a rapidly growing one. We propose to employ this principle and search for a strategy which, for a given problem, minimises the number of derived clauses after a certain number of iterations of the saturation loop. Focusing on the search for a good ordering as a simplifying restriction on the set of available strategies, we experimentally investigate the practical potential of this idea.

Simplification orderings in superposition-based theorem proving. The superposition calculus [7] used by modern ATPs such as E [9], SPASS [13], and Vampire [4] is parametrised by a simplification ordering on terms. The two most commonly used classes of orderings are the Lexicographic Path Ordering (LPO) [2] and the Knuth-Bendix Ordering (KBO) [3]. To get a concrete instance of an ordering from either class one needs to specify a symbol *precedence*, a total order on the predicate and function symbols occurring in the given problem.[1]

It is well known that the choice of a concrete ordering can have a huge impact on the success or failure of the subsequent proof attempt. For example, giving a high precedence to symbols introduced during clausification as names of sub-formulas [8], will effectively lead to their immediate elimination during saturation and thus give rise to an exponential clause set. Nevertheless, since there is no obvious general way to choose a good ordering in advance, ATPs typically provide only a few heuristical schemes for the automatic selection of the precedence. This leaves the majority of the $n!$ possibilities on how to choose a precedence (for a problem with a signature of size n) inaccessible to a typical theorem proving strategy.

The role of strategies in modern ATPs. Theorem provers typically have a large number of options for organising proof search and a *strategy* is obtained by fixing concrete values for these options. Since there cannot be a universally best strategy for solving all problems, an ATP may try to predict the best strategy for a given problem based on the problem's features [6, 11, 9]. An arguably more robust approach is to split the allotted time amongst a *schedule of strategies* and execute them one by one or in parallel [10, 5].[2] The success of strategy

[1] To fully determine a KBO, one also needs to specify an admissible weight function.
[2] Vampire [4] also heavily relies on strategy scheduling, but the details are currently unpublished.

scheduling can be explained by the observation pertaining to first-order theorem proving: *if a strategy solves a problem then it typically solves it within a short amount of time*. Thus there is a demand for methods for discovering new good strategies, especially strategies able solve previously unsolved problems [12].

Slow search space growth as a guiding principle. Previous work on literal selection heuristics [1] experimentally established that strategies which lead to a slowly growing search space tend to be more successful at finding proofs (within the allotted time limit). Here we propose to use this observation as a general principle for finding good strategies for solving a particular problem. The idea is to look for strategies which minimize the number of derived clauses after a certain number of iterations of the saturation loop. In contrast to random sampling of the strategy space, this approach promises to be a *directed* search for strategies that can solve previously unsolved problems!

As a proof of concept, we focus here on the space of possible orderings and, more specifically, on precedences specifying an ordering, and try to establish the extent to which the proposed idea can be useful in practice. There is a limiting factor in the fact that a precedence must be fixed in advance and cannot be changed during proof search. Thus, optimizing the precedence (exploration) and utilising a precedence shown to perform well to actually solve the problem (exploitation) must be understood as separate activities, which leads to a refinement of the question from the title: Can a good proof strategy be evolved *in time*?

References

[1] K. Hoder, G. Reger, M. Suda, and A. Voronkov. Selecting the selection. In *IJCAR 2016*, pp. 313–329. Springer International Publishing, 2016.

[2] S. Kamin and J.-J. Levy. Two generalizations of the recursive path ordering. Departement of Computer Science, University of Illinois, Urbana, IL, 1980.

[3] D. E. Knuth and P. B. Bendix. *Simple Word Problems in Universal Algebras*, pp. 342–376. Springer Berlin Heidelberg, 1983.

[4] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In *CAV 2013*, vol. 8044 of *Lecture Notes in Computer Science*, pp. 1–35, 2013.

[5] D. Kühlwein, S. Schulz, and J. Urban. E-MaLeS 1.1. In *CADE-24, 2013.*, vol. 7898 of *Lecture Notes in Computer Science*, pp. 407–413. Springer, 2013.

[6] W. McCune and L. Wos. Otter - the CADE-13 competition incarnations. *J. Autom. Reasoning*, 18(2):211–220, 1997.

[7] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, vol. I, chapter 7, pp. 371–443. Elsevier Science, 2001.

[8] G. Reger, M. Suda, and A. Voronkov. New techniques in clausal form generation. In *GCAI 2016*, vol. 41 of *EPiC Series in Computing*, pp. 11–23. EasyChair, 2016.

[9] S. Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.

[10] G. Stenz and A. Wolf. E-SETHEO: an automated^3 theorem prover. In *TABLEAUX 2000*, vol. 1847 of *Lecture Notes in Computer Science*, pp. 436–440. Springer, 2000.

[11] T. Tammet. Gandalf. *J. Autom. Reasoning*, 18(2):199–204, 1997.

[12] J. Urban. BliStr: the blind strategymaker. In *GCAI 2015*, vol. 36 of *EPiC Series in Computing*, pp. 312–319. EasyChair, 2015.

[13] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski. SPASS version 3.5. In *CADE-22, 2009. Proceedings*, vol. 5663 of *LNCS*, pp. 140–145. Springer, 2009.

An Isabelle Formalization of the Expressiveness of Deep Learning (Extended Abstract)

Alexander Bentkamp[1], Jasmin Christian Blanchette[23], and Dietrich Klakow[4]

[1] Universität des Saarlandes, Saarbrücken, Germany
s8albent@stud.uni-saarland.de
[2] Inria Nancy – Grand Est & LORIA, Villers-lès-Nancy, France
jasmin.blanchette@inria.fr
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[4] Spoken Language Systems (LSV), Universität des Saarlandes, Saarbrücken, Germany
dietrich.klakow@lsv.uni-saarland.de

Deep learning has had a profound impact on computer science in recent years, with applications to search engines, image recognition and language processing, bioinformatics, and more. However, on the theoretical side only little is known about the reasons why these deep learning algorithms work so well. Recently, Cohen et al. [4] presented a theoretical approach using tensor theory that can explain the power of one particular deep learning algorithm called convolutional arithmetic circuits.

We present a formalization [1, 2] of their mathematical proof using the Isabelle/HOL proof assistant. This formalization simplifies and generalizes the original proof, while working around the limitations of the Isabelle type system. To support the formalization, we developed and extended reusable libraries of formalized mathematics.

The expressiveness of deep convolutional arithmetic circuits. Sum-product networks (SPNs) are a deep learning architecture [8], also known as arithmetic circuits. SPNs consist of a rooted directed acyclic graph with input variables as leaf nodes and two types of interior nodes: sum nodes and product nodes. The incoming edges of sum nodes are labeled with real weights, which have to be learned during training.

Convolutional arithmetic circuits (CACs) impose the structure of the frequently used convolutional neural networks (ConvNets) on SPNs. CACs consist of alternating convolutional and pooling layers, which are realized as collections of sum nodes and product nodes, respectively. CACs are equivalent to SimNets, which have been demonstrated to perform as well as these state of the art networks, even outperform them when computational resources are limited [3]. Moreover, Cohen et al.'s analysis of CACs allows to deduce properties of ConvNets with rectified linear unit (ReLU) activation [5].

The formalized theorem states that CACs enjoy complete depth efficiency, i.e., except for a negligible set S (a Lebesgue zero set in the weight space of the network), a deep network of polynomial size implements functions that require a network of super-polynomial size when the network is constrained to be shallow.

To simplify the formalization work, we restructured the original proof to obtain a more modular version, which generalizes the result as follows: The set S is not only a Lebesgue null set, but in particular the zero set of a multivariate polynomial $\not\equiv 0$. This stronger theorem gives a clearer picture of how superior deep CACs are opposed to shallow ones in terms of expressiveness.

Formalization of the result in Isabelle/HOL. Isabelle/HOL is a natural choice for the formalization of this result because its strength lies in the high level of automation it provides. Moreover, it has an active community and includes a large library of formalized mathematical theories including measure theory, linear algebra, and polynomial algebra.

Our formalization provides a formal proof of the fundamental theorem from Cohen et al.'s original work for a network with non-shared weights. The formalization does not rely on any axioms beyond those that are built into Isabelle/HOL, and has an approximate total size of 7000 lines.

This project led to the development of general libraries that can be used in future formalizations about possibly completely different topics. Most prominently, we developed a library for tensors and their properties including the tensor product, tensor rank, and matricization. Moreover, we extended several libraries: We added a the matrix rank and its properties to Thiemann and Yamada's matrix library [9], adapted the definition of the Lebesgue measure by Hölzl and Himmelmann [7] to our purposes, and extended Lochbihler and Haftmann's polynomial library [6] by various lemmas, including the theorem that zero sets of multivariate polynomials $\not\equiv 0$ are Lebesgue null sets.

This formalization is a case study of applying proof assistants to a field where they have been barely used before, namely machine learning. It shows that the functionality and libraries of state of the art proof assistants such as Isabelle/HOL are up to the task. Admittedly, even the formalization of such relatively short proofs is labor-intensive. On the other hand, the process can not only lead to a computer verification of the result, but can also reveal new ideas and results. The generalization and simplifications we found demonstrate how formal proof development can also benefit the research outside the world of formal proofs.

References

[1] Alexander Bentkamp. Expressiveness of deep learning. *Archive of Formal Proofs*, November 2016. http://isa-afp.org/entries/Deep_Learning.shtml, Formal proof development.

[2] Alexander Bentkamp. An Isabelle formalization of the expressiveness of deep learning. Master's thesis, Universität des Saarlandes, 2016. http://matryoshka.gforge.inria.fr/bentkamp_msc_thesis.pdf.

[3] Nadav Cohen, Or Sharir, and Amnon Shashua. Deep simnets. *arXiv preprint arXiv:1506.03059*, 2015.

[4] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. *CoRR*, abs/1509.05009, 2015.

[5] Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. *arXiv preprint arXiv:1603.00162*, 2016.

[6] Florian Haftmann, Andreas Lochbihler, and Wolfgang Schreiner. Towards abstract and executable multivariate polynomials in Isabelle. In *Isabelle Workshop*, volume 201, 2014.

[7] Johannes Hölzl and Armin Heller. Three chapters of measure theory in Isabelle/HOL. In *Interactive Theorem Proving*, pages 135–151. Springer, 2011.

[8] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 689–690. IEEE, 2011.

[9] René Thiemann and Akihisa Yamada. Matrices, Jordan normal forms, and spectral radius theory. *Archive of Formal Proofs*, August 2015. http://isa-afp.org/entries/Jordan_Normal_Form.shtml, Formal proof development.

Higher-Order Problems from Landau's Grundlagen

Chad E. Brown

Czech Technical University in Prague

Abstract

We describe a collection of problems for higher-order theorem provers derived from translating the AUTOMATH formalization of Landau's *Grundlagen der Analysis*. We briefly describe the formal development obtained by translating into higher-order set theory. The theorem prover Satallax has been tested on the problem sets and we present some results. In addition we discuss some preliminary experiments using machine learning techniques to do premise selection on the problems.

Landau's *Grundlagen der Analysis* [6] gives a mathematical development constructing the rational numbers, the real numbers and the complex numbers (along with the corresponding operations, relations and basic theorems) starting from the natural numbers. In total the book contains 73 definitions and 301 theorems. The contents of the book were formalized in AUTOMATH in the 1970s by van Benthem-Jutting [5]. The AUTOMATH formalization contains 32 primitives and 6878 abbreviations. Primitives and abbreviations can be classified as types, predicates (including propositions), objects or proofs.

Among the 6878 abbreviations 5991 can be viewed as theorems, where the type of the abbreviation gives the proposition and the definiens gives a proof of the proposition. Among these 5991 "theorems" only 850 correspond to Landau's theorems. (Some of Landau's theorems are represented by multiple AUTOMATH theorems.) The remaining 5141 correspond either to logical preliminaries or to steps of proofs of Landau's theorems (with the final step proving the theorem).

There were type theories implemented by versions of AUTOMATH. The type theory used for the *Grundlagen* formalization was called AUT-QE. AUT-QE is in some ways similar to modern type theories, e.g., including polymorphism and dependent types. In other ways, it is quite different. For example, suppose $M : B$ (M has type B) under the assumption $x : A$ (where x is a variable and A is a type). In modern type theories, $\lambda x : A.M$ would have type $\Pi x : A.B$. In AUT-QE, $\lambda x : A.M$ would have type $\lambda x : A.B$. The first step to extracting problems was to translate the AUTOMATH formalization into a type theory where $\Pi x : A.B$ is used instead of $\lambda x : A.B$ for function types. Here we take as a starting point a version of the formalization in such a type theory with Π types.

There are further problems if we wish to obtain problems in simple type theory. Simple type theory (in particular the THF0 language for higher-order theorem provers [3]) supports neither polymorphism nor dependent types. We have avoided this problem by translating the formalization into higher-order set theory. AUTOMATH types and objects are mapped to sets. Additionally, objects are mapped to proofs that the set corresponding to the object is in the set corresponding to the type. For example, the AUTOMATH type `nat` is mapped to the set $\omega \setminus \{0\}$ while the AUTOMATH object `one` (of type `nat`) is mapped to both the ordinal 1 and a proof that $1 \in \omega \setminus \{0\}$. Due to the proof component of the translation of AUTOMATH objects, the higher-order formalization includes an additional 581 "sorting" theorems, for a total of 6572 theorems. However, some of these theorems are duplicates. After removing the duplications, there remain a total of 6238 theorems, 847 of which correspond to Landau's theorems.

Starting with axioms for higher-order Tarski-Grothendieck set theory, basic building blocks can be constructed including pairs, sets of pairs, functions, sets of functions and the natural

numbers. Once we have functions and sets of functions in the higher-order set theory, we can interpret λ abstractions as set theoretic functions and Π types as set theoretic function spaces. We can define van Benthem-Jutting's primitives (e.g., the natural numbers) and prove van Benthem-Jutting's axioms (e.g., induction) in the set theory. Once this has been done we obtain a development starting from axioms of set theory and including all of the formalization of Landau's *Grundlagen*.

Given this formal development in higher-order set theory, we can extract various higher-order theorem proving problems in a manner similar to the way MPTP was used to extract theorem proving problems from Mizar articles [7]. We have created four different problem sets according to two dimensions: granularity and relevance. By *granularity* we refer to the choice of whether we consider all of the 6238 non-duplicate "theorems" from the AUTOMATH formalization (called *by* theorems) or only the 847 non-duplicate "top level theorems" corresponding to Landau's theorems (called *top* theorems). By *relevance* we refer to the choice of whether we only include all the previous definitions and results in the problem or include only those used in the known proof. We follow [1] in calling the harder version with all previous information the *Chainy* version and the easier version with selected information the *Bushy* version. Consequently the four problem sets consist of the *Bushy-by* problems, the *Bushy-top* problems, the *Chainy-by* problems and the *Chainy-top* problems.

The higher-order prover Satallax [4] has been run on each of these problem sets with a timeout of 1 second on 54 selected modes. Preliminary results indicate that Satallax can prove roughly 69% of the Bushy-by problems and roughly 21% of the Bushy-top problems. Only 2 of the Chainy-by problems were proven and none of the Chainy-top problems were proven.

It is clear that to attack the Chainy problems better premise selection is required. Preliminary testing has been done using k-NN with the dependency information from the Bushy-by problems along with features extracted from the axioms and definitions (e.g., the occurrences of constant names) to suggest and rank premises in the Chainy-by problems. If we only give Satallax the top 128 k-NN ranked axioms and definitions in the Chainy-by problems, then it can prove 49% of the problems. We plan to apply more recent machine learning technology such as [2] in the hopes of improving premise selection.

References

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

[2] Alex A. Alemi, François Chollet, Geoffrey Irving, Christian Szegedy, and Josef Urban. Deepmath - deep sequence models for premise selection. *CoRR*, abs/1606.04442, 2016.

[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] Chad E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. *Journal of Automated Reasoning*, 51(1):57–77, Mar 2013.

[5] L.S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System*. PhD thesis, Eindhoven University of Technology, 1977.

[6] E. Landau. *Grundlagen der Analysis*. Leizig, 1930.

[7] J. Urban. MPTP 0.2: Design, Implementation, and Initial Experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006.

DeepAlgebra - an outline of a program

Przemysław Chojecki
prz.chojecki@gmail.com
December 2, 2016

In [1] authors name two main bottlenecks for formalizing mathematics:

(1) lack of automated methods for formalization (*autoformalization*);
(2) lack of strong automated methods to fill in the gaps in already formalized human-written proofs.

A basis for the research in [1], where the authors deal with the second bottleneck, is Mizar Mathematical Library (MML) which contains over 50,000 lemmas. The authors have used deep learning methods to select premises for the automatic theorem proving and automatically prove as many theorems as possible using at each step all previous theorems and proofs. To some extent this solves (2), though much more optimalization work is needed to attain level of human effectiveness.

In our work, we focus on the first bottleneck. We propose a program to automate a formalization of large parts of modern algebraic geometry using deep learning techniques run on well-chosen repositories of human-written mathematical facts (The Stacks Project [6]). The main problem is the construction of a dictionary between human-written proofs in (La)TeX and Interactive Theorem Provers. Here we outline our program and lay a theoretical basis for it. We report on our progress on implementing it in subsequent papers and in [3]. Our main theoretical input is to use syntactic parsers run on The Stacks Project to correctly translate [6] into COQ, and then use hammers to verify its validity in COQ/E, and possibly reprove some of it by automated methods. Eventually this approach should lead to proving new theorems in algebraic geometry purely by machines.

1 The Stacks Project

Algebraic geometry is one of the pillars of modern mathematics. Its main object of study is algebraic varieties (vanishing loci of sets of polynomials) and maps between them. It builds upon standard algebra and study of rings and modules, and as such is highly symbolic. Because of that, we believe it is easier to formalize it rather than more analytic parts of mathematics where reasoning is more ad hoc and proofs can use tricky approximations, which seem hard to teach to a computer. On the other hand, the problem with algebraic geometry is the level of abstraction and the amount of terms which one needs to formulate the problems correctly.

When trying to formalize human-written proofs with a goal of training neural networks on them, one has to be sure that proofs are correct in the first place. In other words, one has to choose sources well. Mathematicians often write proofs in informal ways, leaving gaps to readers which are assumed to be experts in a given field as well.

That is why we propose as our training source the Stacks Project ([6]). This is a repository of facts and theorems in algebra and algebraic geometry, which starts from the basic material and goes up to the current developments in the field. It is still actively developped with the help of dozens of volunteers and currently contains 509,794 lines of code and 5,347 pages (as we write).

Its huge advantage is that it is completely self-contained. There are no references to outside sources. Every fact is proved and can be traced to the axioms introduced at the beginning. The only problem for our goal is that it is written by humans and for humans.

2 Dictionary

In order to formalize this amount of mathematics one needs to automate the process. Our goal is to develop an AI which could prove new theorems in algebraic geometry by itself. To do that, one firstly needs to translate the source (The Stacks project in our case) to one of **Interactive Theorem Provers** (ITPs) such as COQ [2] or Mizar [4] , and then use an **Automatic Theorem Prover** (ATP), for example E [5] , together perhaps with some deep learning techniques to facilate proving more complex theorems.

The first step of our DeepAlgebra program is to construct a **dictionary** between (La)TeX files and COQ/Mizar, which would translate .tex files into ITPs and vice versa. While one direction is relatively easy (ITP to .tex) as one can use fairly simple, hard-coded language, the other direction is much less trivial. This is due to the fact to which we have alluded before, that human-written mathematics happens in general to be informal with many gaps left to readers. Computer needs to be taught which sentences and words can be ignored as informal discussions and which are crucial to on-going proofs, as mathematical papers are not constructed in the form of "theorem, proof, theorem, proof", but often contains a valuable discussion outside of it.

The other problem is to correctly implement the abstraction in the form of Types (COQ). Algebra tends to use many words for the same thing and correctly storing it is crucial. For example computer needs to know that a "reduced scheme of finite type over a field" is the same thing as an "algebraic variety", while both characterizations give slightly different perspectives.

When trying to formalize The Stacks Project, most of these problems do not appear (or are easier) as the text itself is written in the form of definitions, followed by lemmas, followed by theorems. Thus translating .tex file into COQ in this case needs much less work than with the general mathematical paper, and we shall report on our progress elsewhere. However this does not solve the general problem, which is needed in order to formalize current mathematical work. General mathematical papers tend to be worse structured with respect to machines.

References

[1] A. Alemi, F. Chollet, G. Irving, C. Szegedy, J. Urban, "DeepMath - Deep Sequence Models for Premise Selection", arXiv:1606.04442

[2] COQ proof assistant, https://coq.inria.fr

[3] DeepAlgebra, https://przchojecki.github.io/deepalgebra/

[4] Mizar, http://mizar.org

[5] S. Schulz, "E - A Brainiac Theorem Prover" AI Commun., 15(2-3):111-126, 2002.

[6] The Stacks Project, http://stacks.math.columbia.edu

math.wikipedia.org: A vision for a collaborative, semi-formal, language independent math(s) encyclopedia

J. Corneli and M. Schubotz

[1] Goldsmiths, University of London, London, UK
j.corneli@gold.ac.uk
[2] Universität Konstanz, Konstanz, Germany
moritz.schubotz@uni.kn

Introduction

It has been argued that the right representation is the key to successful reasoning [Bun13, p. 16]. We would broaden this observation to point out that while both representation and reasoning are crucial, they are not in themselves sufficient for most applied cognitive and computational workflows. In particular, the processes by which representations are created are logically prior. With large-scale projects, human and HCI factors must be considered, and suitable management strategies devised. Below, we consider the representation and workflow issues surrounding the creation of a large machine-readable mathematics knowledge base. Such a knowledge base would differ from a repository of human-readable texts, in that it would support a range of automated reasoning tasks. For example, one such task is automated tutoring. An ongoing master's thesis [Dud16] on a related simpler task, automated question answering, has helped to motivate the current inquiry.

1 What has been done?

As far as human-readable mathematics goes, one of the most successful and visible projects is Wikipedia, the world's largest encyclopedia. The Mathematics WikiProject indexes 31444 articles at the time of writing.[1] For some sub-fields of STEM, Wikipedia also contains a significant amount of related domain knowledge. Until recently, the main – and still most familiar – way of representing this knowledge was in encyclopedic human readable articles in different languages, which were to some degree linked to each other. With the implementation of Wikidata this has begun to change. Now, there is a centralized database that has unique IDs for each semantic concept. From those concepts, there are links to the individual language versions of the Wikipedia articles. In addition, Wikidata contains links between concepts, and properties of the items of various data types, including mathematical formulae [SS16]. In this way, Wikidata forms the backbone of a knowledge graph. The project, then, exists somewhere between two domains: *digital libraries* and *artificial intelligence*.

2 Our plan for the future

We propose to rely on Wikidata, and build "math.wikipedia.org" as a frontend to that. This service will allow users to conveniently store and retrieve formulas, terms, and other mathematical objects. Building a knowledge base with machine-readable formulas and links between

[1] https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Mathematics/Count

concepts will produce a richer domain for reasoning than we get from existing projects or other mathematical digital library proposals that lack semantic models.

Working with an existing popular platform should help bootstrap the social side, and allow us to focus our development effort on domain-specific issues. State of the art technology now exists for mining relevant knowledge from existing knowledge resources. Wikidata and a handful of other projects have proposed knowledge representation formats for mathematics. One current example, *Gamma function* on Wikidata is pictured in Figure 1. The page is quite terse. More details would be needed to bring the Wikidata page closer to feature parity with the Wikipedia pages in English or German. Indeed, better-articulated frameworks would be required to adequately represent more complex mathematical concepts, e.g., *adjunction in a category*.[2] Whatever the concept, the definition and any formulas should be accompanied by framing information (e.g., who invented it, where has the concept been discussed?).

3 How will the public benefit?

Improvements to Wikipedia: If all of this information is available within Wikidata, we and others can use it to make a better Wikipedia using semi-automated methods. For example, article placeholders, autolinking of entries, automatic provision of references, and so on, could all be provided. By using the knowledge graph, such services can be delivered with contextual sensitivity; e.g., to point out that "this article is missing a basic explanation of $X/Y/Z$.")

Reasoning and computational methods available by default: Having a computational "frame" over the material, we would be able to infer, e.g, "Here is an example; here's how you compute with it." An implementation of the key Pólya heuristic "If you can't solve a problem, there is an easier problem you can solve: find it" would be valuable for tutoring applications.

Integration with other knowledge bases and AI systems: Thinking of the system as a data consumer, we should be able to use it to index and centrally store concepts from the broader literature. Thinking of the system as a data provider, if the Wikidata representations are done right, we should be able to use this data in AI applications, for example, as a resource to use in more general question-answering systems that make use of mathematical reasoning.

4 Related work

The NIST Digital Library of Mathematical Functions [Mil13] and the derived Digital Repository of Mathematical Formulae [CMS+14, CSM+15] are inspiring examples of formula-centric storage. Contentmine is an interesting current project that has had success mining knowledge out of research papers (for now, mostly in chemistry and bioscience). It would be interesting to apply the methods to the mathematics and physics content stored in arXiv. Babar is another project for extracting knowledge out of Wikipedia, specifically [dL12]. The authors of the current paper have been highly involved with representing mathematics on the web in a format suited for human readers, notably through work with the MathML Association and PlanetMath. Representing mathematical knowledge in a format suitable for both human and machine interaction requires a somewhat different approach. Michael Kohlhase's recent work on a Semantic Multilingual Glossary for Mathematics is relevant prior art, as are the earlier OpenMath Content Dictionaries. A range of more formal representations exists in the context of projects like HOL [GK15], Isabelle, Coq [CFGW04], and Mizar, where contributions to the Mizar Mathematical Library correspond to articles in the journal *Formalized Mathematics*.

[2] https://en.wikipedia.org/wiki/Adjoint_functors

5 Conclusions

If we compare the few high-level properties of the gamma function that are expressed in Figure 1 with those that are captured in connection with the function's formalisation in HOL [SH14], it becomes even more starkly clear that the current Wikidata treatment lacks detail. In addition to listing more properties, the formal treatment naturally provides the proofs of these properties. At the moment, Wikidata even lacks the property '`demonstrates`' that could be used to connect the statement of a theorem with a proof.[3] In principle, one could be added – which would then beg the question as to how proofs would be represented in the system. However, proofs are just one of many challenging topics to address. A simple example: how should Wikidata answer the question "What is the area of a circle?" – given that 'disk' would be correct in mathematical English, but *Kreisfläche* (lit. *circle area*) is the relevant concept in German.[4]

More broadly: what role will semantic representation and AI software play in the development of the online mathematics ecosystem? What workflows will support the use and further development of the constituent online resources? This is the upstream part of the "collaborative development of open services based on representations of mathematical knowledge" [Ion16]. The fruits of repository- and KB-building efforts depend on due attention to these roots.

Acknowledgment Corneli's work was supported by an EPSRC-funded project studying "The Integration and Interaction of Multiple Mathematical Reasoning Processes" (EP/N014758/1).

References

[Bun13] Alan Bundy. The interaction of representation and reasoning. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 469(2157), 2013.

[CFGW04] Luís Cruz-Filipe, Herman Geuvers, and Freek Wiedijk. C-CoRN, the constructive Coq repository at Nijmegen. In *International Conference on Mathematical Knowledge Management*, pages 88–103. Springer, 2004.

[CMS+14] H. S. Cohl, A. McClain M, B. V. Saunders, M. Schubotz, and J. C. Williams. Digital Repository of Mathematical Formulae. In *CICM*, LNCS, 2014.

[CSM+15] H. S. Cohl, M. Schubotz, M. A. McClain, B. V. Saunders, C. Y. Zou, A. S. Mohammed, and A. A. Danoff. Growing the DRMF with Generic LaTeX Sources. In *CICM*, 2015.

[dL12] Pierre Raymond de Lacaze. BABAR: Wikipedia Knowledge Extraction, 2012.

[Dud16] Kaushal Dudhat. Teaching Mathematics to Question Answering System. Master's thesis, Universität Konstanz, 2016.

[GK15] Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 372–386. Springer, 2015.

[Ion16] Patrick Ion. The Effort to Realize a Global Digital Mathematics Library. In *International Congress on Mathematical Software*, pages 458–466. Springer, 2016.

[Mil13] Bruce R. Miller. Three years of DLMF: web, math and search. In J. Carette and D. Aspinall, editors, *CICM*, volume 7961 of *LNCS*, 2013.

[SH14] Umair Siddique and Osman Hasan. On the formalization of gamma function in HOL. *Journal of automated reasoning*, 53(4):407–429, 2014.

[SS16] M. Schubotz and A. P. Sexton. A Smooth Transition to Modern mathoid-based Math Rendering in Wikipedia with Automatic Visual Regression Testing. In M. Kohlhase, editor, *WiP at CICM*, Aachen, 2016.

[3] https://www.wikidata.org/wiki/Wikidata_talk:WikiProject_Mathematics#Modelling_proofs_and_properties_of_some_objects

[4] https://www.wikidata.org/wiki/Wikidata_talk:WikiProject_Mathematics#Consistent_naming_convention

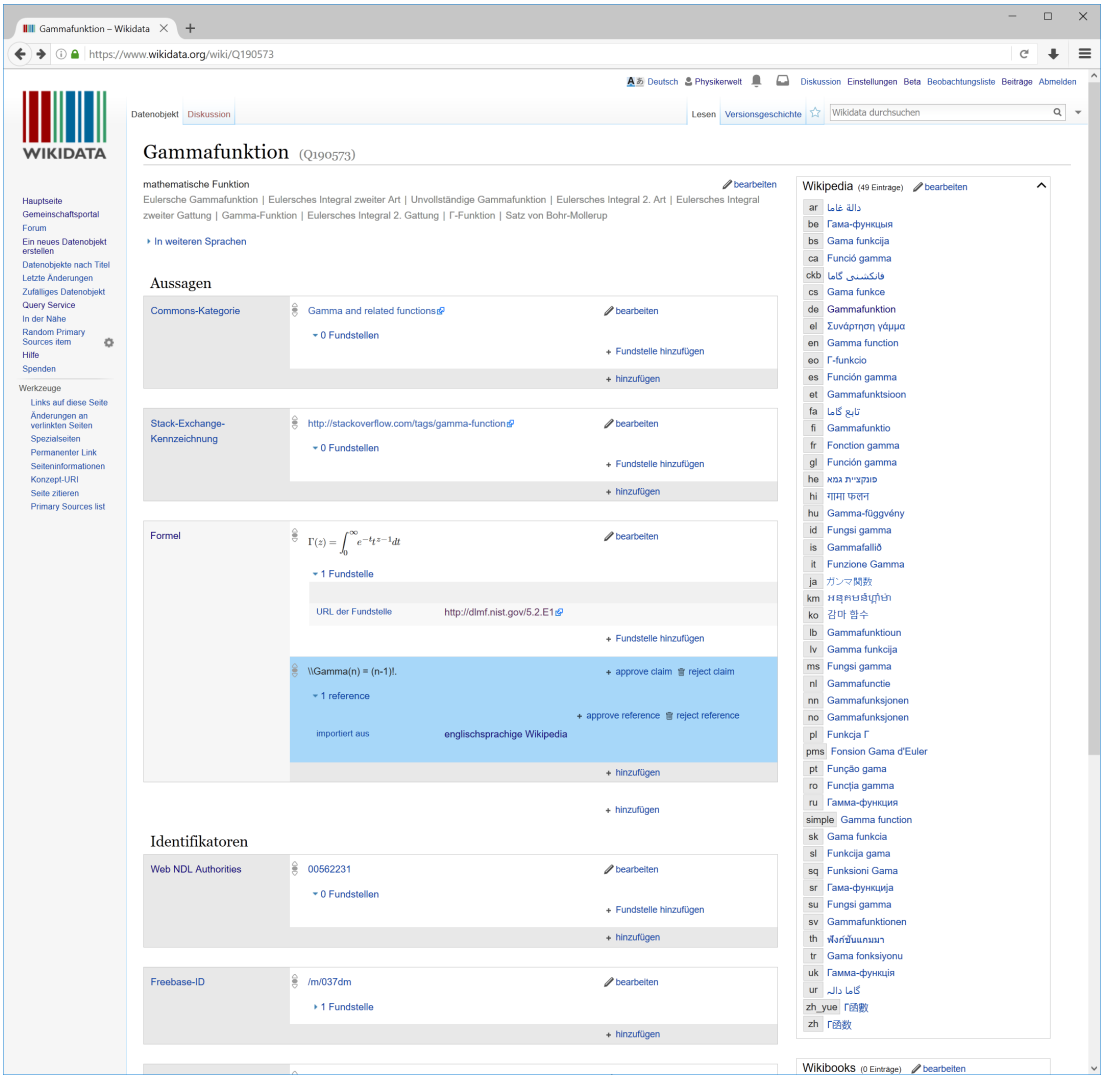

Figure 1: Screenshot of the Wikidata page for Euler's gamma function https://www.wikidata.org/wiki/Q190573, with German language localization. On the top of the page is the title and the id. Below there is a natural language description and a list of aliases. In the block on the right there are links to the various Wikipedia pages about the gamma function. One of the blocks on the left contains a definition that was manually entered, and originates from the NIST Digital Library of Mathematical Functions. The formula in the blue box, $\Gamma(n) = (n-1)!$, was automatically extracted from the English Wikipedia by Yash Nager (a master's student at Universität Konstanz), and currently waits for human verification. Links to external identifiers and other related resources are provided.

Monte Carlo Connection Prover

Michael Färber[1], Cezary Kaliszyk[1], Josef Urban[2]

[1] Universität Innsbruck, Austria
michael.faerber,cezary.kaliszyk@uibk.ac.at
[2] Czech Technical University in Prague, Czech Republic
josef.urban@gmail.com

Abstract

Monte Carlo Tree Search (MCTS) is a technique to guide search in a large decision space by taking random samples and evaluating their outcome. The talk will discuss our recent experiments with using MCTS to guide the search of an automated theorem prover based on the connection calculus [FKU16].

Monte Carlo Tree Search (MCTS) [BPW+12] is a method to bias search based on the outcomes of random simulations, i.e. random traversals of the search tree. For some games such as Go, MCTS algorithms have become state of the art.

A single iteration of the MCTS algorithm works as follows: First, a node of the search tree having at least one unvisited child is chosen. Then, a random simulation is started from the child. When the random simulation ends, its reward is evaluated and back-propagated to the child and all its ancestors.

One of the most popular variants of MCTS, UCT (upper confidence bounds applied to trees) [KS06], steers search towards promising areas, while ensuring that also less promising parts are eventually explored. To combine exploration and exploitation, the initial choice of the node in UCT balances the average reward received from random simulations against the number of times the node has been visited.

We use MCTS to guide the search of an automated theorem prover. In fair two-player games, random simulations have good chances of coming up with final states that feature both winning and losing outcomes, therefore allowing over time to distinguish between good and bad moves with increasing confidence. In automated theorem proving, however, one can not even expect to reach a final state most of the time, and losing the game by not finding a proof is the rule rather than the exception. For this reason, the main problem is to come up with a suitable reward function that rates the quality of proof attempts. Furthermore, this reward function must be bounded by a constant to ensure that UCT will eventually select every potential node in the search tree, thus preserving completeness of the proof search.

We came up with a machine-learnt reward function that estimates the provability of a proof obligation to be the ratio of successful and tried proofs of similar proof obligations in previous proof attempts. The total provability of multiple proof obligations is then a combination of the individual provabilities, either by taking the product or the minimum. A second heuristic rewards proof states that have fewer open proof obligations with respect to the total number of previously opened proof obligations.

In random simulations, the next proof step in a proof state is selected with a probability that is a combination of: a) the inverse of the number of proof obligations opened by the proof step, and b) a probability calculated by a Naive Bayesian classifier trained on previous proofs similar to [KU15]. We stop a random simulation if either a maximal number of steps has been performed or no proof step can be applied to the last proof state.

The resulting system can be used as a stand-alone automated theorem prover or as an oracle for a *base* prover that drives the proof search: When the base prover has multiple proof steps to choose from, it calls the Monte Carlo prover with each of the potential proof steps and then

tries the proof steps in the order determined by their Monte Carlo rewards. Furthermore, if the Monte Carlo prover finds a proof during random simulation, the base prover can use it.

We implemented the described system on top of the automated theorem prover leanCoP [Ott08] for the reason that its connection calculus is very simple and the prover's very small implementation facilitates extensions and experiments. Both base and Monte Carlo prover are modified versions of leanCoP, returning lazily calculated streams of proofs. This allows for example backtracking to different proofs found by the Monte Carlo prover.

We evaluated our system using the MPTP library [AKT+11]. On the 2078 bushy MPTP problems, leanCoP running for 100s per problem proves 475 problems with Monte Carlo guidance and 624 problems without. Of the 475 problems, 26 problems are unique. On the other hand, for the problems that both leanCoP with and without guidance solved, unguided leanCoP took more than 13 times as many inferences as guided leanCoP.

The reduction of number of proven problems can be explained by the large overhead incurred by calculating every single proof step possibility during random simulations, whereas in standard leanCoP, proof step possibilites are only calculated when needed. The search for better reward functions remains challenging.

This work has been supported by the ERC Consolidator grant nr. 649043 *AI4REASON* and the Austrian Science Fund (FWF) grant P26201.

References

[AKT+11] Jesse Alama, Daniel Kühlwein, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Premise selection for mathematics by corpus analysis and kernel methods. *CoRR*, abs/1108.3446, 2011.

[BPW+12] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Trans. Comput. Intellig. and AI in Games*, 4(1):1–43, 2012.

[FKU16] Michael Färber, Cezary Kaliszyk, and Josef Urban. Monte Carlo connection prover. *CoRR*, abs/1611.05990, 2016.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou, editors, *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18-22, 2006, Proceedings*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[KU15] 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.

[Ott08] Jens Otten. leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International*

Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings, volume 5195 of *Lecture Notes in Computer Science*, pages 283–291. Springer, 2008.

Towards AI Methods for Prover9*

Jan Jakubův[1], Josef Urban[1], and Robert Veroff[2]

[1] Czech Technical University, Prague, Czech Republic
{jakubuv,josef.urban}@gmail.com
[2] University of New Mexico, Albuquerque, New Mexico, U.S.A.
veroff@cs.unm.edu

1 Introduction

Theorem provers such as Prover9 [3] are regularly being applied to math problems that require substantially more prover resources than ever before. We have had numerous successful proof searches that generated and kept tens of millions of clauses and found proofs that are several tens of thousands of steps in length.

Many of the successful searches have relied on a type of reasoning by analogy, where knowledge about proofs of related theorems is used to help guide the search. In Prover9, this capability is supported by a *hint* mechanism [4], where a user includes hint clauses as input–typically proof steps from related problems–and the search is biased to prefer clauses that match hint clauses. Successful searches have included hundreds of thousands of hints clauses, with a corresponding increase on the demand for prover resources.

In this talk, we will discuss some of the work that is being done to address the difficulties of working with such large problems. This work includes implementation issues (for runtime efficiency), new program features and evolving usage guidelines for working on especially large and difficult problems. Finally, we will give a brief summary of some of the latest math results, focusing in particular on the role of hints (analogy) in the searches.

2 Performance Issues

As the number of hint clauses increases, and the proof searches become longer and more complex, runtime performance degrades significantly. The most straightforward way to handle larger sets of hint clauses in Prover9 is to improve the performance of the current implementation. By improving the time and memory complexity of algorithms and data structures used in the prover, not only can we expect faster run times, but also new problems can be solved because requirements on time and memory resources are decreased.

A crucial operation in Prover9 regarding hints is *instance retrieval*, that is, the operation to retrieve all the hints which are instances of a generated clause. To represent a set of hints, Prover9 currently uses *path indexing* [2], which effectively supports instance retrieval. Path indexing constructs a tree, where each branch represents a *path* in a term.

An important consideration is the depth of the tree. A deeper tree will have fewer clauses at the leaves, resulting in fewer instance candidates that have to be checked explicitly. On the other hand, an unlimited depth can significantly degrade performance [2], so it is desirable to limit the depth of the tree. The default depth limit in Prover9 currently is set to 10. This value was previously determined experimentally to provide reasonable results. With a large increase in the number of hints, a natural idea is to increase this depth limit to further reduce

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

the number of candidate clauses at the leaves. As was done previously, we can attempt to determine an optimal value for this context experimentally.

We have experimented with a benchmark set of 10^6 hints and different settings of indexing path depth limits. The result is that increasing the depth limit from 10 to 45 can provide a speedup of 6.6% in total run time without an impact on memory consumption.

For future work, we would like to experiment with different indexing methods. For example, using *discrimination* or *substitution trees* [2, 1] to represent hints could give us further speedup, allowing us to make use of even larger numbers of hints.

3 Search Issues

Even with good search structures, the number of input hints can get prohibitively large. The prover can spend much of its time exploring generated clauses that match hints but are not helpful in finding a proof for the current problem. One way to deal with this is to limit the input hints to come from proofs that seem to be the most closely related to the current problem. Although it may be easy to identify the most and least similar problems, there can be a large number of proofs for which it is too difficult to decide. A better way to deal with this is to prioritize hints and to have a selection bias for clauses that match the highest priority hints.

We have recently added support for hint prioritization in Prover9 and are studying ways to use hint statistics from previous proofs and learning methods to determine hint priorities. Initial evidence is primarily anectodal but strongly implies that hint prioritization is effective. We already have several cases where the difference between runs with and without hint prioritization was the difference between success and failure.

In addition, we currently are looking at methods to change the priorities of hints based on heuristics that measure progress that is being made during a search. For example, when hints are taken from several related proofs, we can ask how well each of these proofs is being covered in the current search and increase the priority of hints from proofs that are close to being completely covered.

4 Math Results

For several years, we have been working on a project in loop theory that includes many related problems of interest. Over time, we have solved several of the problems and have accumulated a large set of hints. As we prove more theorems, we accumulate new hints, which in turn help us get even more results, results that were previously completely out of reach, both because of the lengths of the proofs and the complexity of the clauses that participate in the proofs. We can conclude that the continued development of AI methods for selecting and prioritizing hints will help us continue to make progress on this project and on the overall effectiveness of automated theorem proving.

References

[1] Peter Graf. Substitution tree indexing. volume 914 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 1995.

[2] William McCune. Experiments with discrimination-tree indexing and path indexing for term retrieval. *Journal of Automated Reasoning*, 9(2):147–167, 1992.

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

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

Logic-Independent Premise Selection for Autormated Theorem Proving

Eugen Kuksa[1]

Otto-von-Guericke University of Magdeburg, Germany
kuksa@iks.cs.ovgu.de

In recent years, formalised large theories in form of ontologies, system specifications or mathematical libraries with thousands of sentences have become more and more common. Automatically proving properties in such theories is an important task in research as well as in industry, for instance, as part of system specifications. However, resolution-based automated theorem provers (ATPs) struggle with combinatorial explosion and need too many resources to find a proof. One way to overcome this problem is to reduce the knowledge to only the presumably relevant axioms before the ATP is set to find a proof. This approach, the premise (pre-) selection, has been successfully applied before with, for instance, heuristics such as the SUMO Inference Engine (SInE) [5], naïve Bayes-based machine-learning techniques [7], advanced kernel-methods [1] and even deep neural networks [6].

Such premise selection implementations improved proving performance significantly. Compared to the case without preprocessing, they increased the number of provable problems in well-known benchmarks (such as SUMO [12], CYC [9] and the Mizar-based [14] MPTP-2078 [18]) by about 50%. SInE, for instance, is a heuristic that is basically applicable to any logic, but the known implementations are always tied to a specific logic and serialisation (such as first-order logic serialised in TPTP [16]) or to a specific prover (such as SInE in the Vampire Prover [13] or in the E Theorem Prover [15] [17]).

We implemented the SInE algorithm in a way that is independent of the underlying logic. The base of this implementation is the Heterogeneous Tool Set (Hets) [10] and the concepts of the Distributed Ontology, Model and Specification Language (DOL) [11]. It builds on consequence relations [2] and institutions [4] to abstract from symbols and sentences of a logic. The sentences and symbols of a concrete logic are analysed and abstracted from by Hets. These abstract forms are then fed to the web application Ontohub [8]. The SInE algorithm is implemented in Ontohub as a wrapper for Hets. Ontohub selects the premises based on the abstract sentences and symbols and passes the selection back to Hets. In the next step, Hets selects an appropriate prover and serialises the modified theory (having only the premises which are selected by Ontohub) to a file and calls the selected prover.

Our experiments with this implementation covered both, first-order problems serialised in TPTP and higher-order problems in THF [3] and confirmed the improvements of proving performance that have been made with logic-specific implementations of SInE. Some of the utilised provers implement SInE themselves, but the usage of the prover-internal implementation was disabled in our experiments.

This result is proof of concept of logic-independent premise selection. In future work, we will implement other premise selection algorithms in a logic-independent way for use in Ontohub.

References

[1] Jesse Alama et al. 'Premise Selection for Mathematics by Corpus Analysis and Kernel Methods'. In: *Journal of Automated Reasoning* 52.2 (2014), pp. 191–213.

[2] Arnon Avron. 'Simple consequence relations'. In: *Information and Computation* 92.1 (1991), pp. 105–139.

[3] Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. 'THF0 – The Core of the TPTP Language for Higher-Order Logic'. In: *International Joint Conference on Automated Reasoning*. Springer. 2008, pp. 491–506.

[4] Joseph A Goguen and Rod M Burstall. 'Institutions: Abstract Model Theory for Specification and Programming'. In: *Journal of the ACM (JACM)* 39.1 (1992), pp. 95–146.

[5] Krystof Hoder and Andrei Voronkov. 'Sine Qua Non for Large Theory Reasoning'. In: *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*. 2011, pp. 299–314.

[6] Geoffrey Irving et al. 'DeepMath – Deep Sequence Models for Premise Selection'. In: *Advances In Neural Information Processing Systems*. 2016, pp. 2235–2243.

[7] Daniel Kühlwein et al. 'MaSh: Machine Learning for Sledgehammer'. In: *Interactive Theorem Proving*. Springer, 2013, pp. 35–50.

[8] Eugen Kuksa and Till Mossakowski. 'Ontohub — Version Control, Linked Data and Theorem Proving for Ontologies'. In: *FOIS 2016*.

[9] Douglas B Lenat. 'CYC: A Large-Scale Investment in Knowledge Infrastructure'. In: *Communications of the ACM* 38.11 (1995), pp. 33–38.

[10] Till Mossakowski. *Heterogeneous Specification and the Heterogeneous Tool Set*. University of Bremen Germany. Habilitation thesis. 2005.

[11] Till Mossakowski et al. 'The Distributed Ontology, Modeling, and Specification Language - DOL'. In: *The Road to Universal Logic*. Ed. by A. Koslow and A. Buchsbaum. Vol. II. Studies in Universal Logic. Springer, 2015.

[12] Ian Niles and Adam Pease. 'Towards a Standard Upper Ontology'. In: *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*. ACM. 2001, pp. 2–9.

[13] Alexandre Riazanov and Andrei Voronkov. 'The Design and Implementation of VAMPIRE'. In: *AI communications* 15.2, 3 (2002), pp. 91–110.

[14] Piotr Rudnicki. 'An Overview of the Mizar Project'. In: *Proceedings of the 1992 Workshop on Types for Proofs and Programs*. 1992, pp. 311–330.

[15] Stephan Schulz. 'System Description: E 1.8'. In: *Proc. of the 19th LPAR, Stellenbosch*. Ed. by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. LNCS. Springer, 2013, pp. 735–743.

[16] G. Sutcliffe. 'The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0'. In: *Journal of Automated Reasoning* 43.4 (2009), pp. 337–362.

[17] Geoff Sutcliffe. 'The 4th IJCAR automated theorem proving system competition – CASC-J4'. In: *AI Communications* 22.1 (2009), pp. 59–72.

[18] Josef Urban. 'MPTP 0.2: Design, Implementation, and Initial Experiments'. In: *Journal of Automated Reasoning* 37.1-2 (2006), pp. 21–43.

Applying Formal Verification to Reflective Reasoning

Ramana Kumar[1] and Benya Fallenstein[2]

[1] Data61, CSIRO, and UNSW, Sydney, Australia
ramana@intelligence.org
[2] Machine Intelligence Research Institute, Berkeley, CA, U.S.A.
benya@intelligence.org

There has been a recent wave of interest in ensuring that increasingly capable AI systems will be robust and beneficial (e.g., [5]). Here, we consider the following problem [2] in creating robust, highly capable self-modifying AI. Design a system that is capable of (almost) arbitrary self-improvement while always satisfying a safety property. The difficulty is in ensuring that the system is constrained enough to avoid creating successors that violate the safety property, but free enough to find highly effective successors amongst the safe ones.

In reality, creating successors (or self-modifying) are not clearly distinguished from other actions, so we consider the problem in the general setting of finding effective safe actions. However, successors are of particular interest because ensuring a successor's actions are safe runs up against fundamental difficulties analogous to proving consistency of a formal system from within that system.

We want to include the possibility that successors might self-modify again, which means reasoning about the safety of self-modification at one remove (i.e., reasoning about the safety of the process that reasons about the safety of the process of making a modification). Obviously, this extends to arbitrarily long chains of successors. Can we guarantee safety through a long chain of modifications without imposing blunt limits on the kinds of modification allowed?

The suggester-verifier architecture

A concrete approach to safety without unnecessary limits is the suggester-verifier architecture [7]. Systems following this architecture are divided into two parts. The first part, the suggester, is arbitrarily sophisticated but untrusted code for finding and proposing actions to take, with the caveat that it must produce evidence that its proposed action is safe in the form of a proof. The second part, the verifier, is simple code that ensures that the proof produced by the suggester is correct. In case the suggester cannot find a proof, the system takes a fallback action: a simple action that has been verified to be safe but is likely suboptimal.

We do not expect this architecture to be sufficient for real-world systems, as reasoning about the consequences of real-world actions seems likely to require methods beyond formal proof. However, proof-based formal reasoning is successful at verifying formal properties of software systems, and in this context we expect it to play a role in reliable real-world self-modifying systems. We use the suggester-verifier architecture as a test bed to explore issues of self-reference that may arise in such systems.

Can we establish correctness of the suggester-verifier architecture? In particular, can we show that a system following this architecture is safe but also capable of self-modification to another system following the same architecture? Self-modifying into the same architecture is a baseline test of capability, and highlights the problem of trusting the reasoning of other systems.

Now the difficulty with chains of self-modification can be made more precise. Suppose the suggester proposes actions that would result in a successor that also follows the suggester-verifier architecture. To prove that such a successor is safe, the suggester would like to establish

that the successor's verifier is sound, i.e., that a safety proof verified by the successor is valid. Formally, supposing the successor uses proof system S', the system needs to establish

$$\forall a.\, \Box_{S'} \ulcorner \mathsf{Safe}(\bar{a}) \urcorner \implies \mathsf{Safe}(a)$$

where $\Box_{S'} \ulcorner \varphi \urcorner$ represents a proof of φ in the proof system S', and $\bar{a}$ within the quotes represents anti-quotation. If the successor's proof system, S', is the same as the initial proof system, then by Löb's theorem [3], the above statement would imply that $\mathsf{Safe}(a)$ holds for all actions a, which renders the safety predicate useless. The argument that the successor system is safe because it follows the suggester-verifier architecture fails for the same reason that a logical system cannot internalise its own consistency proof.

Partial solutions Fallenstein and Soares [2] describe three partial solutions to the problem above; we consider two of them. The first is obvious: make the successor's proof system weaker than the original. Then there is no problem with internalising the consistency of the successor's proof system. However, this solution only allows for a finite chain of successors. Fortunately, we can do better.

The second solution, called *model polymorphism*, adds a new constant κ to the language and is based on the insight that the following axiom schema is sound no matter how κ is interpreted (the proof is in [2]):

$$0 < \kappa \implies \forall n.\, \Box \ulcorner \varphi(\bar{n}) \urcorner \implies \varphi[\kappa - 1/\kappa](n)$$

Now we assume discrete timesteps and modify the safety predicate so that rather than $\mathsf{Safe}(a)$ indicating action a is safe in all future timesteps, we have instead $\mathsf{Safe}(a, n)$ indicating action a is safe after n timesteps. If we further assume the system has access to the current timestep t, then we can produce an infinite chain of safe successors via the following theorem:

$$\forall a.\, \Box \ulcorner \forall n \leq \overline{t+1} + \kappa.\, \mathsf{Safe}(\bar{a}, n) \urcorner \implies \forall n \leq t + \kappa.\, \mathsf{Safe}(a, n)$$

The theorem follows from the model polymorphism axiom schema, taking $\varphi(\cdot)$ to be $\mathsf{Safe}(a, \cdot)$. Outside the system, we note that since κ can be arbitrarily high, safety is established for all timesteps despite the bounded quantification of n.

Botworld

To explore these ideas more concretely, Fallenstein and Soares have developed a cellular-automaton based environment called Botworld [6] in which robots perform actions (moving around, picking up objects, etc.) according to computer programs stored inside them. Importantly, robots are able to assemble parts to construct and program new robots, and the computations are run within the environment. Botworld robots are thereby embedded in their environment in a sense identified by Orseau and Ring [4]. This makes Botworld a suitable model for studying systems that can reason about other reasoners, including their own successors.

Using model polymorphism in HOL for Botworld robots

Building on the above ideas, the contribution to be presented at AITP is a formal model of Botworld in higher-order logic (HOL), aimed at verifying the suggester-verifier architecture using model polymorphism (implemented previously [1]). The point is to see how far theorem provers are from capturing the kind of reasoning self-modifying AI systems may need to entertain.

References

[1] Benja Fallenstein and Ramana Kumar. Proof-producing reflection for HOL - with an application to model polymorphism. In Christian Urban and Xingyuan Zhang, editors, *ITP*, volume 9236 of *LNCS*, pages 170–186. Springer, 2015.

[2] Benja Fallenstein and Nate Soares. Vingean reflection. Technical Report 2015–2, MIRI, 2015.

[3] M. H. Löb. Solution of a problem of Leon Henkin. In *JSL*, volume 20, pages 115–118, 1955.

[4] Laurent Orseau and Mark Ring. Space-time embedded intelligence. In Joscha Bach, Ben Goertzel, and Matthew Ikl, editors, *AGI*, number 7716 in LNAI, pages 209–218. Springer, 2012.

[5] Stuart Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, pages 105–114, Winter 2015.

[6] Nate Soares and Benja Fallenstein. Botworld. Technical Report 2014–2, MIRI, 2014.

[7] Eliezer Yudkowsky and Marcello Herreshoff. Tiling agents for self-modifying AI, and the Löbian obstacle. Early draft, MIRI, 2013.

Towards Smart Proof Search for Isabelle

Yutaka Nagashima[1]

Data61, CSIRO, Sydney, New South Wales, Australia
first_name.last_name@data61.csiro.au

Abstract

Despite the recent progress in automatic theorem provers, proof engineers are still suffering from the lack of powerful proof automation. In this position paper we first report our proof strategy language based on a meta-tool approach. Then, we propose an AI-based approach to drastically improve proof automation for Isabelle, while identifying three major challenges we plan to address for this objective.

1 PSL and Meta-Tool Approach

In the last decade, we have seen the successful application of automated theorem provers to assist interactive theorem proving. Despite the popularity of these so-called "hammer-style" tools, their performance is still suffering from the gap between underlying logics [1].

To circumvent this problem, we introduced a proof strategy language, PSL [4], to Isabelle/HOL [5], taking a meta-tool approach. A proof strategy is an abstract description of how to attack proof obligations. Users write strategies in PSL based on their intuitions on a conjecture. Using a strategy, PSL's runtime system generates many invocations of Isabelle's native proof tools, called *tactics*. The runtime tries to find out the appropriate combination of tactics for each goal by trial-and-error. This way, PSL reduces the domain specific procedure of interactive theorem proving to the well-known dynamic tree search problem. The meta-tool language approach brought the following advantages:

- Domain specific procedures can be added as new sub-tools.
- Sub-tools (including Sledgehammer) can be improved independently of PSL.
- Generated efficient-proof scripts are native Isabelle proofs.

We provided a default strategy, `try_hard`. Our evaluation shows that PSL based on `try_hard` significantly outperforms Sledgehammer for many use cases [4]. However, PSL's proof search procedure is still mostly brute-force. For example, when PSL generates many tactics, even though each of these is tailored for the proof goal utilizing the runtime information, it is still the statically written strategy, `try_hard`, that decides:

- what kind of tactics to generate,
- how to combine them, and
- in which order to apply generated tactics.

2 Meta-Tool Based Smart Proof Search

On a fundamental level, this lack of flexibility stems from the procedural nature of Isabelle's tactic language, which describes how to prove conjectures in a step-by-step manner rather than what to do with conjectures. However, a conventional declarative language would not be a good solution either, since in many cases we cannot determine what to do with a given proof goal with certainty. Our guess may or may not be right: given a proof goal, we cannot tell which tactics to use until we complete the goal.

Probabilistic Declarative Proof Strategy Language. We propose the development of a probabilistic declarative proof strategy language (PDPSL), which allows for the description of feasible tactics for *arbitrary* proof obligation. For example, when we apply a proof strategy, `str`, written in PDPSL, to a proof search graph, `pgraph`, PDPSL's runtime:

(1) picks up the most promising node in `pgraph`,
(2) applies the most promising tactic that has not been applied to that node yet, and
(3) adds the resultant node to `pgraph`, connecting them with a labelled edge representing the tactic application and weight indicating how promising that tactic application is.

The runtime keeps applying this procedure until it reaches a solved state, in which the proof is complete. This way, PDPSL executes a best-first search at runtime.

The major challenge is the design of PDPSL. A strategy written in PDPSL is applied repeatedly during a best-first search against emerging proof obligations, and we cannot predict how these intermediate goals look like prior to the search; therefore, weighting of tactics against concrete proof goals is not good enough. PDPSL should only describe the meta-information on proof goals and information in Isabelle's standard library. Such meta-information includes:

- Which ITP mechanism was used to define constants that appear in the proof obligation?
- Does the conjecture involve recursive functions?

This may sound too restrictive; however, when proof engineers use ITPs, we often reason on the meta level. For instance, Isabelle's tutorial [5] introduces the following heuristics:

> "*Theorems about recursive functions are proved by induction.*"
> "*The right-hand side of an equation should (in some sense) be simpler than the left-hand side.*"

These are independent of user-defined formalisation.

Posterior Proof Attempt Evaluation. Evaluating a proof goal to write promising tactics is only half the story of typical interactive proof development. The other half is about evaluating the results of tactic applications. We presume that posterior evaluations on tactic application will improve proof search when combined with prior expectations on tactics described in PDPSL. The main challenge is to find a useful measure by:

- discovering (possibly multiple) useful measures that indicate progress in proof search, and
- investigating how to integrate these measures into the best-first search of PDPSL.

Of course it is not possible to come up with a heuristic that is guaranteed to work for all kinds of proof obligations. But when focusing on a particular problem domain such as systems verification, it is plausible to find a useful measure to narrow the search space at runtime.

Reinforcement Learning of PDPSL using Large Proof Corpora. PDPSL is primarily for engineers to encode their intuitions. But at the next step, we plan to improve this hand-written heuristics using reinforcement learning on the existing proof corpora. The Isabelle community has a repository of formal proofs, called the Archive of Formal Proofs (AFP) [3]. As of 2015, the size of the AFP is larger than one million lines of code. Additionally, the seL4 project open-sourced roughly 400,000 lines of Isabelle proof script [2]. We speculate that these large proof corpora will work as the basis for reinforcement learning of strategies written in PDPSL. The lack of concrete information on proof obligations can be an advantage at this stage: since PDPSL cannot describe concrete proof obligations, it is less likely to cause over-fitting.

References

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

[2] Gerwin Klein. Proof engineering challenges for large-scale verification. http://www.ai4fm.org/ai4fm-2014/, May 2014.

[3] Gerwin Klein, Tobias Nipkow, and Larry Paulson. The archive of formal proofs. https://www.isa-afp.org/, 2004–2016.

[4] Yutaka Nagashima and Ramana Kumar. A proof strategy language and proof script generation for isabelle. *CoRR*, abs/1606.02941, 2016.

[5] 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.

Greedy Algorithms for Finding Maximum Number of Then Step in Reasoning*

Karol Pąk[1] and Aleksy Schubert[2]

[1] Institute of Informatics, University of Białystok,
ul. K. Ciolkowskiego 1M, 15-245 Bialystok, Poland
pakkarol@uwb.edu.pl
[2] Institute of Informatics, University of Warsaw
ul. S. Banacha 2, 02–097 Warsaw, Poland
alx@mimuw.edu.pl

Abstract

As in the case of computer programs, proof scripts written according to declarative formats of proof assistants such as Mizar or< Isabelle can sometimes be more and sometimes be less readable. However, quite frequently, smart pieces of reasoning that are conducted in a less readable way are re-analyzed by human readers to make them stronger, more easily applicable and so on. Unfortunately, the existing optimization methods that try to improve the representation of the information flow in a given reasoning are NP-complete. We present an algorithm that improves proof readability in accordance with Behaghel's First Law. We show that our algorithm achieves satisfactory performance ratio, which results from the fact that the maximal degree of the proof graph generated for a reasoning is small.

Formal proof checking systems such as Mizar[1] or Isabelle/Isar[5] can verify the correctness of proof scripts regardless of their legibility. However, in the case of human readers, the legibility has substantial significance. Generally, shorter reasoning should be more readable. However, if we shorten a given proof script, using the proof construction schemes available in the proof assistant at hand or external tool developed with system in the Mizar case, the operation may strongly impair readability.

As an orthogonal direction, we can improve proof legibility, reshaping the linearization of information flow in a given reasoning to better fit into human short-term memory. This manipulation is based upon the observation that every premise used in a justification has to be assumed or derived before in the proof; however the particular location of the premise in a reasoning is not fixed. The Behaghel's law prescribes that elements intellectually close one another should be located in the text next to each other. This idea is realized when we keep the rule that for each step at least part of required premises is available in n directly preceding steps of the proof script, or just in the previous step where we can refer to using the *then* construction. It is important to note that the problem of finding the maximum number of such steps is NP-hard, for each positive n [4].

There have been several attempts to build a tool that can automatically enhance a given linearization of an information flow. We can develop a transformation method where the information flow and a selected determinant of legibility are together formulated as a solving goal for an SMT solver [3]. In an alternative approach we can create dedicated algorithms or adapt existing ones to realize multi-criteria optimization of determinants. However, our problems of legibility are relatively new and not enough explored.

We propose an algorithm that maximizes the number of *then* steps, one of possible legibility determinants. In our approach the main goal is obtained based on an arbitrary heuristics that

*The paper has been financed by the resources of the Polish National Science Centre granted by decision no DEC-2012/07/N/ST6/02147

approximates the optimal solution of the Maximal Independent Set problem (MIS). Note that the MIS problem is well explored and there exists algorithms that have a performance ratio of $\frac{\Delta+2}{3}$, $\frac{2\overline{d}+3}{5}$ for approximating independent sets in graphs with degree bounded by Δ and the average degree $\overline{d}$, respectively [2].

To sketch the idea of our algorithm, let R be a reasoning and denote by F_R the DAG that represent the information flow in R. A topological sorting τ of F_R is a τ-linearization of R if τ describes the linear ordering of proof steps in R. A fragment of a reasoning is a *τ-reasoning path* when it is a sequence of steps such that all steps excluding at most the starting one are *then*-steps in τ. Additionally, τ-reasoning path is maximal, means simply that it is not a subsequence of any other τ-reasoning path. Note that the number of *then*-steps in a given τ-linearization determines uniquely an acyclic partition of the information flow for maximal τ-reasoning path of reasoning. Moreover the number of non *then*-steps in τ is equal to the length of the maximal τ-reasoning path. Consequently we can view the problem to maximize the number of *then*-steps as the problem to minimize the acyclic partition of a graph into directed paths.

The main idea of our algorithm is that we concatenate pairs of directed paths that fulfill chosen criteria keeping as invariant the acyclicity of the resulting partition.

We start the computation of the algorithm with a partition π_0 of the graph into singleton sets. Subsequently, we generate in an iteration i a set V_i of pairs (a, b) of elements in the partition π_{i-1} such that the first step in b uses the last element of a as its premise, but at the same time joining the two elements of the partition does not introduce cycle (equivalently, the longest path that connects the joined elements in the partition graph has length 1). Then we construct an instance of the MIS problem based on V_i. We use V_i as the vertices in the input graph and its elements are connected with an edge when corresponding pairs of elements in π_{i-1} have common element (pairs as sets have non-empty intersection). The independent set of vertices that results from an MIS algorithm indicates the set of those pairs in π_{i-1} that are joined to obtain π_i. We continue this iteration until the set V_i gets empty.

It is worth mentioning here that after the first iteration we obtain at most every second edge in each *then*-sequence. If we use the known minimum degree-greedy algorithm on top of the above mentioned procedure, we obtain the maximal number of *then*-steps in 93.97% of one-level reasonings in MML, which offers a much satisfactory compromise between the complexity of the procedure and the quality of the final result.

References

[1] G. Bancerek, C. Bylinski, A. Grabowski, A. Korniłowicz, R. Matuszewski, A. Naumowicz, K. Pąk, and J. Urban. Mizar: State-of-the-art and Beyond. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015.

[2] M. M. Hallorsson and J. Radhakrishnan. Greed is Good: Approximating Independent Sets in Sparse and Bounded-degree Graphs. *Algorithmica*, 18(1):145–163, 1997.

[3] K. Pąk. Automated Improving of Proof Legibility in the Mizar System. In S. M. Watt, J. H. Davenport, A. P. Sexton, P. Sojka, and J. Urban, editors, *CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *LNCS*, pages 373–387. Springer, 2014.

[4] K. Pąk. Improving Legibility of Formal Proofs Based on the Close Reference Principle is NP-Hard. *Journal of Automated Reasoning*, 55(3):295–306, 2015.

[5] Makarius Wenzel. The Isabelle/Isar Reference Manual, 2015.

Chemical Discovery as a Knowledge Graph Completion Problem

Marwin Segler[1]* Mark P. Waller[2]

[1] Institute of Organic Chemistry & Center of Multiscale Theory and Computation
Westfälische Wilhelms-Universität Münster, Germany
marwin.segler@wwu.de
[2] Department of Physics & International Centre for Quantum and Molecular Structures
Shanghai University, China
waller@shu.edu.cn

Abstract

The ability to reason beyond established knowledge allows Organic Chemists to invent novel chemical reactions. Recently, we proposed a model which mimics chemical analogical reasoning and formalises discovery as knowledge graph completion. Paths in the knowledge graph indicate missing reaction nodes, while missing molecule nodes can be predicted with a formal grammar. Importantly, paths also correspond to logical explanations of why our model makes certain predictions. Using a graph containing 14.4 million molecules and 8.2 million binary reactions, which represents the bulk of all chemical reactions ever published in the scientific literature, we demonstrate that our model outperforms classical rule-based expert systems in the reaction prediction task. Using only data until 2013, we show that our model is capable of (re-) discovering 35% of the novel transformations published in 2014/15.

1 Introduction

The deliberate synthesis of molecules is a main contributor to human well-being and provides goods from materials to medicines. However, despite tremendous achievements in the past 150 years, chemists still cannot make any molecule that needs to be made.[2] Therefore, thousands of research groups worldwide pursue the discovery of novel ways to make molecules, which is called methodology research in chemistry. New methodology gets frequently published in high-profile journals like Science and Nature and awarded with Nobel prizes, which might serve as an indicator that the scientific community considers this subject as relevant.

The published knowledge of chemistry grows exponentially. Chemists would therefore highly benefit from systems that not just allow information retrieval, but also assist in the creative parts in methodology research. Currently, hypothesis generation for novel reactions is the domain of the human expert. There have been a few reports about systems which aid in hypothesis generation, however, they have been limited to specific reaction classes. To allow computers to invent novel reactions, we have recently proposed to combine knowledge graphs with a formal grammar on molecules.[4]

Chemical knowledge can be naturally represented as a bipartite graph $G = (M, R, E)$ of molecules $m_i \in M$ and reactions $r_j \in R$. Each molecule m_i has a molecular structure S_i. Edges $e_{ij} = (m_i, r_j, \varrho) \in E$ represent the role ϱ of a molecule m_i in a reaction r_j as a reactant, product, catalyst or solvent. Then, reaction discovery can be generally be described as a knowledge graph completion problem. More specifically, missing subgraphs in the graph have to be found. While link prediction is well-established in the study of knowledge graphs, node or subgraph prediction is less common.[3] E.g. in a social network, inferring the friendship between two existing people

is straightforward. Predicting a person missing in a social network, and his/her name, address and a photo of him/her is more difficult. To discover a reaction r_k of two molecules m_a, m_z, we have to predict a subgraph $g = \{r_k, e_{ak}, e_{zk}, m_p, e_{pk}, E_{ck}, E_{sk}\}$, where m_p is the product of r_k, and the molecular structure S_p of m_p. E_{ck} and E_{sk} are the sets of edges connecting r_k to the catalysts and solvents needed for the reaction, which can be empty. While g can be inferred from the knowledge graph alone, the molecular structure S_p of the product m_p cannot, even though it is of central importance.

Thankfully, we can lean on another symbolic model of chemistry to predict the molecular structure of the formed products: Molecules can be represented as *molecular* graphs (which do not have anything to do with our knowledge graph!), in which atoms are labeled nodes and bonds are labeled edges. Chemical reactions can then be described by graph rewriting, and correspond to productions in the framework of graph grammars.[1]

To predict g, paths $\pi_t = (m_a, r_\alpha, ..., r_\omega, m_z)$ between our two molecules m_a, m_z are retrieved. In our talk, we will describe how g and S_p can be inferred from r_α and r_ω, and why the paths directly correspond to logical explanations for the made prediction, which allows the model to explain its hypotheses to the user.

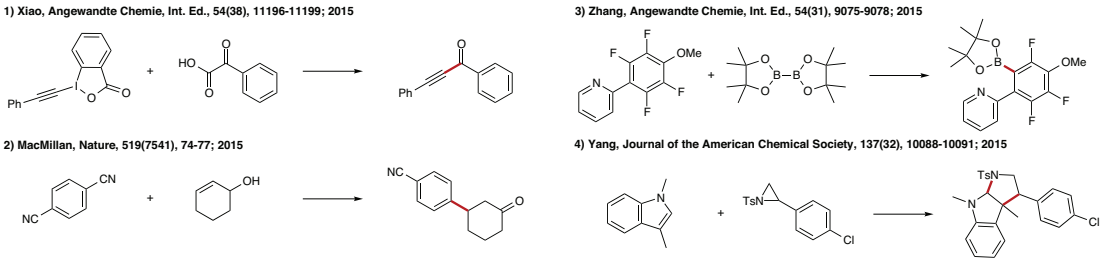

Figure 1: Examples of chemical reactions "rediscovered" by our algorithm with their respective references.

To validate our model, we constructed a knowledge graph containing 14.4 million molecules and 8.2 million binary reactions published until 2013. These are essentially all chemical reactions ever published in the literature. We could show that our model is capable of (re-) discovering 35% of the novel transformations published in 2014/15. A few examples of the rediscovered reactions are depicted in 1. Currently, we cooperate with experimental chemists to apply our model for "real" discovery in the wet lab.

In our talk, besides describing our model and its limitations and possible extensions, we will also discuss why chemistry is a fruitful playground for computer-assisted scientific discovery.

References

[1] Jakob L. Andersen, Christoph Flamm, Daniel Merkle, and Peter F. Stadler. Generic strategies for chemical space exploration. *Int. J. Comp. Bio. Drug Des.*, 7(2-3):225–258, 2014.

[2] Steven V. Ley, Daniel E. Fitzpatrick, Richard Ingham, and Rebecca M. Myers. Organic synthesis: march of the machines. *Angew. Chem. Int. Ed.*, 54(11):3449–3464, 2015.

[3] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Tech.*, 58(7):1019–1031, 2007.

[4] Marwin H.S. Segler and Mark P. Waller. Modelling chemical reasoning to predict and invent reactions. *Chemistry-A European Journal*, DOI: 10.1002/chem.201604556, 2016.

Progress in Automating Formalization

Josef Urban*
Czech Technical University in Prague

Jiří Vyskočil*
Czech Technical University in Prague

1 Introduction and Summary of Our Approach

The talk will describe progress in the project of automatically formalizing informal mathematics by using statistical parsing methods and large-theory automated reasoning. In the first part of the talk we will summarize the overall autoformalization approach and its performance as recently described in [4]. Its main components are:

1. Using aligned informal/formal corpora to train statistical disambiguation. In particular, the Flyspeck corpus of about 20000 toplevel lemmas has been previously *informalized* (ambiguated) by introducing overloaded symbols, forgetting types, brackets and casting functors. After alignment with formal HOL parses, this provides a *treebank* to learn from.
2. Learning (augmented) probabilistic context-free grammar (PCFG) from the treebank.
3. Using the grammar in a CYK parser which is modified by semantic checks and by more involved probabilistic (context-aware) processing. Fast discrimination trees are used to collect deeper subtrees from the treebank and to efficiently match them and boost their probabilities when parsing new formulas.
4. Passing the top-ranking parse trees to HOL type-checking and large-theory reasoning components based on the HOL(y)Hammer system.

2 Parsing Informalized Mizar

A distinctive feature of Mizar [1] is human-style and natural-language-like representation of formal mathematics. This includes Jaśkowski-style natural deduction for proofs (re-invented in the informal ProofWiki[1], soft-typing Prolog-like mechanisms propagating implicit knowledge using Mizar *adjectives* and *registrations* [8, 6], hidden arguments, syntactic macros, and both parametric and ad-hoc overloading [1]. This poses interesting challenges, but also takes our project closer to parsing true natural-language corpora such as ProofWiki.

Treebank creation for Mizar: For Flyspeck we have ambiguated the parsed HOL Light formulas by several transformations applied on the formal representation. For exporting Mizar we apply transformations to the Mizar internal XML layer [5] used previously to produce both the HTML representation of the articles and the semantic (MPTP [7]) representation for ATP systems. The transformation (producing now about 60000 parse trees from MML) is based on the HTML-ization code, mainly modifying the hyperlinks into disambiguating nonterminals. The alignment of the informal (user-level) and formal (semantic/MPTP) representation is nontrivial and there are several interesting issues involving Mizar's *expandable modes* (essentially macros) and *hidden arguments* where we prefer to use theorem proving search rather than search in the PCFG parser. This will be described in detail in the talk. For example the theorem `RCOMP_1:5`:[2] "`for s, g being real number holds [.s,g.] is closed`" which in TPTP is "`![A]: v1_xreal_0(A) => ! [B]: (v1_xreal_0(B) => v2_rcomp_1(k1_rcomp_1(A, B)))`" is transformed by us to the following parse tree, which can be turned into an MPTP-like format and subjected to the MizAR [2] hammer:

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

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

[2] `http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/rcomp_1.html#T5`

```
(Bool for (Varlist (Set (Var s)) , (Varlist (Set (Var g)))) being
(Type ($#v1_xreal_0 real) ($#m1_hidden number)) holds (Bool (Set ($#k1_rcomp_1 [.)
(Set (Var s)) , (Set (Var g)) ($#k1_rcomp_1 .])) is ($#v1_rcomp_1 compact)))
```

Mizar types: In the HOL setting used by Flyspeck, types are unique and do not intersect. This allows their simple use in the PCFG setting as "semantic categories" corresponding, e.g., to word senses when using PCFG for word-sense disambiguation. Such categories seem useful for learning parsing rules from the treebank [4] that strike some balance between generality and concreteness. In Mizar, each term has in general many *adjectives* (soft types) like `finite`, `ordinal`, `natural`, `Function-like`, `non empty`, etc., which are computed during the type analysis. Only some of them are usually needed to allow a term to be an argument of a particular function or predicate. Since it is more involved to learn such complex typing rules statistically, in the first version we use only the top of the type hierarchy – the Mizar type `Set` – as the result type of all terms, and types occur only in quantification. This corresponds to the (untyped) MPTP encoding, where type guards (encoded as predicates) occur only in quantification, and type-hierarchy formulas delegate the type computation and checking to the ATP systems.

A problem with this approach is that allowing any term to be an argument of any function/predicate may lead to great proliferation of ill-typed terms during parsing. This was indeed an issue in our first untyped export of Flyspeck [3, 4] which used just raw HOL parse trees and only context-free parsing rules. It however seems that the recently introduced deeper (context-aware) parsing rules quite significantly reduce such proliferation. When using a combination of subtrees of depth 4–8, the top-20 success rate in parsing Mizar (100-fold cross-validation) is already over 60%, while it is only about 30% for the simple context-free approach (Table 1).

3 Improving the Parsing

The parsing methods have been improved in several ways. We have implemented a *three-pass* algorithm, speeding up Mizar parsing by about 30%. In the first bottom-up pass, we use CYK to compute only the set of reachable nonterminals for every cell in the parsing chart (instead of computing all partial parses). In the second top-down pass we prune from the chart all nonterminals that cannot be reached from the top, and only in the third (bottom-up) parse we run the standard (full) CYK, however avoiding the unreachable nonterminals detected before.

There are also two improvements of the parsing precision (Table 1). First, we use a form of Occam's Razor to prefer simpler parses, where *simpler* means that the parse was constructed using fewer parsing rules. This discourages e.g. from formulas that parse the very overloaded symbol + in many different ways. It is done by multiplying the probability $P(Q)$ of a partial parse Q by the inverse value of the number of all different parsing rules used in Q. Second, for each parsing rule R we compute from the treebank the probability of R appearing more than once on the same branch of a parse tree, and again use this for modifying the parsing probabilities.

parser setup	Flyspeck		Mizar	
	top-20 success	avrg. rank	top-20 success	avrg. rank
subtree depth 2 (context-free)	41.5%	3.81	32.9%	4.02
subtree depth 4-8	77.1%	**1.95**	63.7%	2.64
subtree depth 4-8 + new improvements	**82.5%**	1.99	**64.6%**	**2.61**

Table 1: Comparison of the parsing methods on the Flyspeck and Mizar corpora. top-20 success is the number of examples where the correct parse appears among the top 20 parses proposed by the parser.

References

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

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

[3] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. 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.

[4] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Semantic parsing of mathematics by context-based learning from aligned corpora and theorem proving. *CoRR*, abs/1611.09703, 2016.

[5] Josef Urban. XML-izing Mizar: Making semantic processing and presentation of MML easy. In Michael Kohlhase, editor, *MKM*, volume 3863 of *LNCS*, pages 346–360. Springer, 2005.

[6] Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15(1):109–130, 2006.

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

[8] Freek Wiedijk. Mizar's soft type system. In Klaus Schneider and Jens Brandt, editors, *Theorem Proving in Higher Order Logics, 20th International Conference, TPHOLs 2007, Kaiserslautern, Germany, September 10-13, 2007, Proceedings*, volume 4732 of *LNCS*, pages 383–399. Springer, 2007.

Proof Search in Conflict Resolution

John Slaney and Bruno Woltzenlogel Paleo
The Australian National University
Canberra, Australia

Abstract

The conflict resolution calculus **CR** shown in Fig. 1 is a first-order hybrid resolution and clausal natural deduction calculus inspired by (and extending to first-order logic) the main ideas used in modern propositional sat-solvers. The calculus restricts the resolution inference rule to unit propagation and provides a mechanism for assuming *decision literals* and a rule for inferring new clauses after a conflict is reached (a.k.a. *conflict driven clause learning*). From a natural deduction point of view, a unit propagating resolution can be regarded as a chain of implication eliminations taking unification into account, whereas decision literals and conflict driven clause learning are reminiscent of, respectively, assumptions and implication introductions, also generalized to first-order through unification.

Unit-Propagating Resolution:

$$\frac{\ell_1 \quad \ldots \quad \ell_n \quad \overline{\ell'_1} \vee \ldots \vee \overline{\ell'_n} \vee \ell}{\ell\,\sigma}\ \mathbf{u}(\sigma)$$

where σ is a unifier of ℓ_k and ℓ'_k, for all $k \in \{1,\ldots,n\}$.

Conflict:

$$\frac{\ell \quad \overline{\ell'}}{\bot}\ \mathbf{c}(\sigma)$$

where σ is a unifier of ℓ and ℓ'.

Conflict-Driven Clause Learning:

$$\frac{\begin{array}{cc}[\ell_1]^{i}_{1} & [\ell_n]^{i}_{n} \\ \vdots\ (\sigma^1_1,\ldots,\sigma^1_{m_1}) & \vdots\ (\sigma^n_1,\ldots,\sigma^n_{m_n}) \\ & \vdots \\ & \bot\end{array}}{(\overline{\ell_1}\sigma^1_1 \vee \ldots \vee \overline{\ell_1}\sigma^1_{m_1}) \vee \ldots \vee (\overline{\ell_n}\sigma^n_1 \vee \ldots \vee \overline{\ell_n}\sigma^n_{m_n})}\ \mathbf{cl}^i$$

where σ^k_j (for $1 \le k \le n$ and $1 \le j \le m_k$) is the composition of all substitutions used on the j-th path[a] from ℓ_k to $\bot$.

[a] Since a proof DAG is not necessarily tree-like, there may be more than one path connecting ℓ_k to $\bot$ in the DAG-like proof.

Figure 1: The Conflict Resolution Calculus **CR**

We defined this calculus in [1], where we proved its soundness and refutational completeness through, respectively, its simulation by natural deduction and the simulation of the usual resolution calculus. Moreover, we showed that **CR** derivations are isomorphic to implication graphs, which are an essential abstract data structure used by CDCL sat-solvers. In [2], **CR** was presented with a focus on its hybrid resolution natural-deduction status and on how it may reduce the gap between the diverging proof-theoretical foundations of fully automatic theorem provers and interactive proof assistants and facilitate their inter-operability.

In this talk, we go further and discuss the issues that arise when we try to establish proof search strategies for **CR**. In contrast to propositional logic, unit propagation does not terminate in first-order logic. Therefore, we must be willing to assume decision literals even before unit propagation terminates. On the other hand, if we assume decision literals too eagerly and fully backtrack on any conflict, we also lose completeness.

Subsumption is also slightly more complicated in first-order **CR**. In propositional conflict driven clause learning, a new derived (learned) clause c is never[1] subsumed by a previously derived clause c'. If it were, then assume that $c' = \ell_1, \ldots, \ell_n, \ell$ and $c' \subseteq c$ and assume, without loss of generality, that at some point during the search the duals of all literals of c' except ℓ have been assumed as decision literals. Then ℓ would be immediately propagated and hence $\overline{\ell}$ would never have become a decision literal. Therefore, ℓ could never be in c and, contradicting the assumption, $c' \not\subseteq c$. In the first-order case, on the other hand, there are at least two problems. Firstly, because propagation does not terminate and we need to decide eagerly, $\overline{\ell}$ may become a decision literal before c' propagates ℓ. Secondly, because of variables and unification, decision literal may not be exact duals of propagated literals and they may even unify with more the duals of more than one literal in the same clause. For instance, on the clause set $\{p(X) \vee q\ ,\ \neg p(a) \vee \neg p(b)\}$, deciding $p(X)$ would lead to a conflict where the learned clause is exactly the same as the second clause in the given set. Although it may seem obvious that we should not have decided $p(X)$ in this simple example, it is not clear whether how to block such decisions in general, when the conflict is not so immediate. And even if such blocking would be possible at all, it is unlikely that it could be done efficiently, and certainly not as efficiently as in the propositional case.

References

[1] John Slaney and Bruno Woltzenlogel Paleo. Conflict Resolution: a First-Order Resolution Calculus with Decision Literals and Conflict-Driven Clause Learning. *Submitted. (Preprint: https://arxiv.org/pdf/1602.04568.pdf)*, 2016.

[2] Bruno Woltzenlogel Paleo. Conflict Resolution: Between Resolution and Natural Deduction. In *Dagstuhl seminar on the Universality of Proofs*, 2016.

[1] Assuming that derived clauses are never deleted.