AITP 2016

The First Conference on
Artificial Intelligence and Theorem Proving

Abstracts of the talks

April 3–7, 2016, Obergurgl, Austria

Preface

This volume contains the abstracts of the talks presented at AITP 2016: The First Conference on Artificial Intelligence and Theorem Proving held on April 3–7, 2016 in Obergurgl, Austria.

We have organized the first 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 2016 consists of three focused sessions on AI for ATP, ITP and mathematics, a (tutorial) session on modern AI and big-data methods, and three sessions with 9 contributed talks. The focused sessions are based on the following 11 invited talks and discussion oriented.

- AI and large-theory ATP/ITP. Speakers: Thomas C. Hales, Cezary Kaliszyk
- AI and internal guidance of ATP. Speakers: Robert Veroff, Stephan Schulz, Martin Suda
- AI and automated understanding of informal and semi-formal mathematics. Speakers: Noriko Arai, Deyan Ginev, Takuya Matsuzaki, Jiří Vyskočil
- Modern AI and big-data methods (tutorials and connections to ATP, ITP and math). Speakers: Sean Holden, Christian Szegedy

We would like to thank the University of Innsbruck for hosting AITP 2016 at its conference center in Obergurgl. Many thanks also to Andrei Voronkov and his EasyChair for their support with paper reviewing, proceedings creation, and registration of participants.

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

March 29, 2016
Pittsburgh
Innsbruck
Stuttgart
Prague

Thomas C. Hales
Cezary Kaliszyk
Stephan Schulz
Josef Urban

Table of Contents

Program Committee

Marcos Cramer	Universiy of Luxembourg
Thomas Hales	University of Pittsburgh
Tom Heskes	Radboud University Nijmegen
Sean Holden	University of Cambridge
Cezary Kaliszyk	University of Innsbruck
Michael Kohlhase	Jacobs University
Ramana Kumar	University of Cambridge
John Lafferty	University of Chicago
Lawrence Paulson	University of Cambridge
Stephan Schulz	DHBW Stuttgart
Geoff Sutcliffe	University of Miami
Josef Urban	Czech Technical University in Prague

Automation in the Formal Proof of the Kepler Conjecture

Thomas C. Hales

University of Pittsburgh

Abstract. The Kepler conjecture asserts that no packing of congruent balls in Euclidean space has density greater than the face-centered cubic packing (the familiar pyramid packing that is used to display oranges at the fruit-stand). This conjecture was established as a theorem in 1998 by a lengthy computer-assisted proof. In 2014, this computer proof was formally verified in the HOL Light proof assistant.

This talk will discuss some of the forms of automation that were used in the formal proof of the Kepler conjecture and will mention further forms of automation that might have been useful. The formal proof required an estimated 20-work years to complete (beyond all the years that it took to discover a proof of the Kepler conjecture in the first place). The formal proof takes about 5000 CPU-hours to check.

It is not a practical matter to spend 20 years on the formalization of every major theorem. A long-term goal in theorem proving is to develop technology that would allow a large proportion of mathematical proofs to be formally checked as a routine part of the publication process. With all this in mind, I hope that this talk will lead to discussions about concrete steps to automate more of the process of proof formalization.

An Overview of Deep Learning

Christian Szegedy

Google

Abstract. Deep machine learning models have the interesting property of being able to learn complex, hierarchical feature representations from data alone. In the recent years, deep learning methods have lead to new approaches and a large number of practical applications in a variety of domains, especially in machine perception. These methods include vision systems that rival humans in terms of object detection and recognition, voice recognition systems that are widely used in cell phones and deep learning systems make inroads in machine translation and even in computer go. In this talk, I will give a short overview of the methods behind the breakthrough results of the last few years and will give a sample a few of the most exciting recent developments and challenges in the field.

Modular Architecture for Proof Advice

Cezary Kaliszyk

cezary.kaliszyk@uibk.ac.at
University of Innsbruck, Austria

1 Introduction

Proof assistant formalization usually involves justifying many small proof steps. Many of such proof steps are so simple, that they would not be mentioned or explained in informal proofs. Nevertheless state-of-the-art proof assistants require the user to spend a significant part of the formalization time on them. Various automation techniques have been proposed, that aim to reduce this effort. This includes includes automated reasoning techniques, as well as domain specific solutions.

The most powerful general propose automation technique available for proof assistants today is provided by "hammers" [3]. Hammers combine machine learning from previous proofs with automated reasoning techniques to automatically search for proofs of user given goals. Hammers typically consist of three main components: (i) a heuristic lemma selector (also called relevance filter or premise selector) that chooses a subset of the accessible facts, that are likely useful for the given conjecture, (ii) an encoding (translation) of the user given goal together with the chosen facts to the logics and input formats of automated theorem provers (ATPs), and (iii) reconstruction of proofs which uses the found ATP proofs to re-prove the user goal in the logic of the proof assistant.

Robust hammers exist today for proof assistants based on higher-order logic (Sledgehammer [5] for Isabelle/HOL [17], HOLyHammer [11] for HOL Light [8] and HOL4 [16]), as well as set theory (MizAR [12] for Mizar [6]) and are able to solve 40–50% of the top-level goals in the various developments [3], as well as more than 70% of the user-visible subgoals [4], and as such has been found very useful in various proof developments [7]. For proof assistants based on other foundations, including ACL2 [13], Coq [2], Matita [1], Isabelle/ZF [14,15], only initial hammering experiments and parts of hammers have been constructed [9,10].

In this talk, we will discuss the components needed to provide proof advice for the various proof assistants and their foundational logics. We will propose a more modular architecture for proof advice, where the lemma selection, proof encoding, and reconstruction are further divided into smaller components. For the learning phase we will propose and specify feature extraction, statement similarity, and lemma learning components. For the encoding we will separate the translation of dependent products, lambda-abstrations, partiality, predicate-subterms, from the translation of the type systems. Finally we will discuss the reconstruction options available for the various proof assistant foundations, and propose shared components.

References

1. Andrea Asperti, Wilmer Ricciotti, and Claudio Sacerdoti Coen. Matita tutorial. *J. Formalized Reasoning*, 7(2):91–199, 2014.
2. Yves Bertot. A short presentation of Coq. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 12–16. Springer, 2008.
3. Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.
4. Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based relevance filter for Isabelle/HOL. J. Autom. Reasoning, to appear. http://cl-informatik.uibk.ac.at/cek/mash2.pdf, 2016.
5. Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Jürgen Giesl and Reiner Hähnle, editors, *International Joint Conference on Automated Reasoning (IJCAR 2010)*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.
6. Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.
7. Thomas Hales. Developments in formal proofs. *Séminaire Bourbaki*, 1086, 2013–2014. abs/1408.6474.
8. John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2009)*, volume 5674 of *LNCS*, pages 60–66. Springer, 2009.
9. Sebastiaan Joosten, Cezary Kaliszyk, and Josef Urban. Initial experiments with TPTP-style automated theorem provers on ACL2 problems. In Freek Verbeek and Julien Schmaltz, editors, *ACL2 Theorem Prover and its Applications (ACL2 2014)*, volume 152 of *EPTCS*, pages 77–85, 2014.
10. Cezary Kaliszyk, Lionel Mamane, and Josef Urban. Machine learning of Coq proof guidance: First experiments. In Temur Kutsia and Andrei Voronkov, editors, *Symbolic Computation in Software Science (SCSS 2014)*, volume 30 of *EPiC*, pages 27–34. EasyChair, 2014.
11. Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014.
12. Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.
13. Matt Kaufmann and J. Strother Moore. An ACL2 tutorial. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 17–21. Springer, 2008.
14. Lawrence C. Paulson. Set theory for verification: I. From foundations to functions. *J. Autom. Reasoning*, 11(3):353–389, 1993.
15. Lawrence C. Paulson. Set theory for verification: II. Induction and recursion. *J. Autom. Reasoning*, 15(2):167–215, 1995.
16. Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar, editors, *TPHOLs 2008*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.
17. Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The Isabelle framework. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 33–38. Springer, 2008.

Machine learning for automatic theorem proving: the story so far

Dr Sean B Holden

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD, UK
sbh11@cl.cam.ac.uk

Keywords: Machine learning, automatic theorem proving, satisfiability (SAT), first-order logic (FOL), higher-order logic (HOL), naïve Bayes, linear regression, support vector machine (SVM).

Abstract

One's ability to produce proofs improves with practice. Thus it is natural to ask whether automatic theorem provers might be strengthened by incorporating the ability to learn using machine learning techniques. This question has been widely addressed in the simplest form of theorem-prover — that aimed at solving propositional satisfiability (SAT) problems — usually using quite basic learning algorithms. There has until recently been relatively little success in applying learning to first-order or higher-order provers, but in recent years work has begun to appear. We provide an introduction to the work to date, including a brief introduction to the most commonly employed learning algorithms.

Induction Controlling Deduction

Stephan Schulz

DHBW Stuttgart
schulz@eprover.org

Abstract

First-order theorem provers search for proofs of a conjecture in an infinite and highly branching search space. This search critically depends on good heuristics. Unfortunately, designing good heuristics for the different choice points and classes of problems for has proved to be very hard. Indeed, even classification of proof problems into classes with similar search behaviour is a largely open research question.

One way to address the difficulty of controlling search is to use inductive approaches, i.e. to try to find good heuristics by observing and generalizing examples of successful and failing proof searches. Here we discuss the three major choice points for superposition-based provers, and how good heuristics can be found via inductive processes.

We distinguish two different learning paradigms. In the first case, only the performance of different heuristics on a test set is used as input for the inductive process. Two examples for this are the automatic generation of automatic modes for theorem provers, and the improvement of clause evaluation heuristics via parameter optimization or genetic algorithms. The second paradigm not only considers the performance of a heuristic, but tries to find new heuristics by analyzing proofs, or, more generally, a graph of the proof search.

We give some results on established work, and discuss some preliminary progress and open questions from ongoing work.

References

[1] J. Denzinger, M. Fuchs, C. Goller, and S. Schulz. Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München, 1999.

[2] J. Denzinger and S. Schulz. Automatic Acquisition of Search Control Knowledge from Multiple Proof Attempts. *Journal of Information and Computation*, 162:59–79, 2000.

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

[4] S. Schulz. Learning Search Control Knowledge for Equational Theorem Proving. In F. Baader, G. Brewka, and T. Eiter, editors, *Proc. of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, volume 2174 of *LNAI*, pages 320–334. Springer, 2001.

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

[6] S. Schulz, A. Küchler, and C. Goller. Some Experiments on the Applicability of Folding Architecture Networks to Guide Theorem Proving. In D.D. Dankel II, editor, *Proc. of the 10th FLAIRS, Daytona Beach*, pages 377–381. Florida AI Research Society, 1997.

[7] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[8] Josef Urban. Blistr: The blind strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Proc. of the Global Conference on Artificial Intelligence, Tibilisi, Georgia*, volume 36 of *EPiC*, pages 312–319. EasyChair, 2015.

CLAUSE SELECTION IN RESOLUTION-STYLE THEOREM PROVERS

ROBERT VEROFF

Abstract. The search for a proof in a resolution-style theorem prover can effectively be reduced to the problem of selecting clauses for the application of inference rules. This talk will be an overview of clause selection methods that are common to automated theorem provers such as Prover9. Such methods include weighting, attribute-based selection, model-based selection and subsumption-based selection.

Weighting (heuristic evaluation) assigns a score to a clause predicting the likelihood that the clause will participate in a proof of a candidate theorem. User-defined weighting functions allow a user to impose intuition, knowledge or preferences on a search. Attribute-based selection, such as the set of support strategy, can be used to restrict the application of inference rules, effectively narrowing a search. In model-based selection, the evaluation of clauses under user-supplied models can be used to further focus a search. Subsumption-based selection is another way for a user to impose intuition, knowledge or preferences on a search. In this case, the priority of clauses can be adjusted if they match user-supplied hint clauses.

The hints mechanism is the foundation of a higher-level strategy–the method of proof sketches–where sequences of theorem proving runs are used to systematically narrow the search for a proof. Proof sketches have been used effectively to solve numerous open questions in mathematics and logic.

The proof sketches method is still evolving. For example, we find that in larger ongoing studies, as the number of accumulated hints increases, the prover can get bogged down and become ineffective. We are beginning to look at machine learning and other methods to help manage and prioritize hints.

Department of Computer Science, University of New Mexico, Albuquerque, New Mexico 87131, U.S.A.

E-mail address: veroff@cs.unm.edu

URL: http://www.cs.unm.edu/~veroff

When Should We Add Theory Axioms And Which Ones?

Giles Reger and Martin Suda

University of Manchester, Manchester, UK

Abstract. One approach to first-order reasoning in the theory of arithmetic is to partially axiomitise the theory. For example, if an input problem uses the interpreted sum operation then one could add axioms stating that the operation is symmetric and has identity zero. For obvious reasons it is not possible to add all necessary axioms. However, it may also be counter-productive to add all obvious axioms. For example, the symmetry axiom for sum is highly prolific in saturation-based methods and may not be necessary for solving the input problem. This talk presents a work-in-progress study applying machine learning to the problem of selecting which theory axioms to add given an input problem. The work is being carried out within the context of the Vampire theorem prover. The problem and the approach taken so far will be explained and some initial results given.

Can a machine solve university entrance exam math problems automatically?

Noriko H. Arai
arai@nii.ac.jp
National Institute of Informatics

"Todai Robot Project (Can an AI get into the University of Tokyo?)" was initiated by National Institute of Informatics in 2011 as an AI grand challenge. The goal of the project is to create an AI system that answers real questions of university entrance examinations. The task naturally requires the interdisciplinary research synthesis. For example, our math problem solving machine requires novel fusion of natural language processing, automated theorem proving and computer algebra.

In 2015, our software took a mock test of the National Center Test (standardized multiple-choice style test) with more than 0.4 million high school students. The results showed that its ability was still far below the average entrants of the University of Tokyo. However, it was among top twenty percentages of all the examinees: it was competent to pass the entrance exams of 474 out of 750 universities in Japan.

Solving Natural Language Math Problems

Takuya Matsuzaki
Nagoya University

Abstract

We have been developing a system that is capable of solving pre-university level math problems written in natural language. The system takes a problem text encoded as an XML as the input. Then it translates a whole problem to a logical formula through several steps of natural language processing (NLP). The formal representation of a problem is further rewritten to find its representation in a local theory such as real closed field (RCF) and peano arithmetic (PA). Finally, an answer is deduced by applying automated reasoning (AR) techniques including computer algebra and theorem proving. In the talk, I will explain why it matters both to NLP and AR and describe several technical problems that have come about from the intersection of (or, the gap between) the two worlds.

Math-rich Natural Language Processing (NLP) on Billion Token Corpora

Deyan Ginev

Jacobs University Bremen

March 21, 2016

Abstract

Analyzing mathematical natural language has a high entry barrier, due to challenges of licensing, representation, and processing at scale.

Additionally, the interplay between the modality of mathematical symbolism and natural language often requires redesign of any existing state-of-the-art solutions. Examples are core problems in the field of computational linguistics, which are largely considered solved for newswire and biomedical texts, such as part-of-speech (POS) tagging and named entity recognition (NER).

A third outstanding core challenge is the lack of availability of "gold standard" datasets, traditionally used for the training and evaluation of learned models and analysis techniques. Here again, the family of classic annotation tasks needs to be extended to mathematical expressions.

In this talk I will share the experience the KWARC research group has had in working on these problems over the last decade, and suggest potential next steps to ensure open collaborations and reproducible and verifiable results in the domain of math-rich NLP.

Our main corpus of investigation has been Cornell's e-Print archive, arXiv.org. The HTML5 conversion of arXiv tentatively contains over 4.2 billion words and over a third of a billion formulas, found in the paragraphs of just under a million scientific articles, as of March 2016.

Probabilistic Parsing of Mathematics

Jiří Vyskočil*

Czech Technical University in Prague

Abstract. One of the first big hurdles that mathematicians encounter when considering writing formal proofs is the necessity to get acquainted with the formal terminology and the parsing mechanisms used in the large ITP libraries. This includes the large number of formal symbols, the grammar of the formal languages and the advanced mechanisms instrumenting the proof assistants to correctly understand the formal expressions in the presence of ubiquitous overloading.
In this work we start to address this problem by developing approximate probabilistic parsing techniques that autonomously train disambiguation on large corpora. Unlike in standard natural language processing, we can filter the resulting parse trees by strong ITP and AR semantic methods such as typechecking and automated theorem proving, and even let the probabilistic methods self-improve based on such semantic feedback. We describe the general motivation and our first experiments, and build an online system for parsing ambiguous formulas over the Flyspeck library.

* Supported by ERC Consolidator grant nr. 649043 *AI4REASON*. This talk describes joint work with Cezary Kaliszyk and Josef Urban.

Revisiting Paulson's Theory of the Constructible Universe with Isar and Sledgehammer

Ioanna M. Dimitriou H. and Peter Koepke

Mathematics Insitute, University of Bonn, Germany
dimitri@math.uni-bonn.de
koepke@math.uni-bonn.de

Lawrence Paulson's Isabelle formalization of Gödel's relative consistency result for the Axiom of Choice stands out as an early formalization of a long and sophisticated theory including a general introduction to Zermelo-Fraenkel set theory. The metamathematical aspects of the theorem and its proof present veritable challenges. We are currently pursuing a project to understand and revise the formalization into a more "readable" form, in parallel with modern introductions to set theory and constructibility theory. Eventually methods of the Naproche-project (Natural proof checking) will be brought in to obtain a natural language layer on top of the new Isabelle text. We hope that such endeavours will facilitate the public acceptance of formal mathematics.

Isabelle and the formalisation of the relative consistency of AC. In 1938, Gödel proved the relative consistency of the Axiom of Choice (AC) [Göd40] relative to the axioms of Zermelo-Fraenkel (ZF) set theory. He introduced the constructible universe, L, which is a minimal model of the standard axioms of set theory, and proved that the Axiom of Choice and the Generalised Continuum Hypothesis (GCH) are valid in L. The proofs involve sophisticated combinatorial as well as metatheoretic arguments. In 2003, Paulson formalised Gödel's proof of the relative consistency of AC in Isabelle [Pau03], thus providing a formalised proof of a large scale metamathematical theorem.

Isabelle is a powerful modern interactive proof assistant, which includes the proving language Isar (Intelligible semi-automatic reasoning) that covers a great distance towards a more naturalised style for formal proofs. Isabelle's core is based on a very minimal higher order logic (Pure), but it is distributed with and supports a wide variety of logics, its main branch being higher order logic (HOL). Since its creation in 1986, Isabelle has become an established system for formal proving and has a large library of formalised mathematics [AFP].

In order to formalise Gödel's proof in first order logic (FOL), Paulson together with Coen, Grabczewski, and Nipkow, developed a ZF-set theory. Building upon the non-AC theories of this ZF-branch, Paulson formalised the key concept of the proof, the reflection metatheorem, which he presents as five theorems in the ZF-language, corresponding to the five steps of an induction on formula complexity, and he also uses relational counterparts for complex set theoretic "terms", and other intricate constructions. This ZF-branch and, as a result, the theories around this seminal metamathematical formal proof, are not longer at the centre of the Isabelle development. Instead, the HOL-branch has gained a lot of attention, and Isabelle's community has developed useful and powerful tools, to aid novices as well as experts in formal proving.

Naturalising Isabelle with Sledgehammer as AI. In Bonn we are currently pursuing the project of "naturalising Isabelle". This involves reformalising existing theories in Isar, with a

view towards natural mathematical text flow and argumentation, and combining the Isar language with our linguistic experiences from the Naproche project [Nap] [Cra13] on shorter texts, in order to provide an expressive mathematical CNL, typeset naturally for a mathematician in LaTeX. In this setting we develop a set theory within Isabelle/HOL, in which we will reformalise Paulson's proof of the relative consistency of AC. It is very important to pay close attention to certain metamathematical aspects involving AC. What we need to prove is that ZF proves in FOL that L is a model of AC, and that the background HOL does not introduce fragments of AC implicitly in the proof.

For Isabelle/HOL, the highly successful tool Sledgehammer [PB10] is available, which can find relevant theorems, translate typed higher order statements into first order logic statements and feed them to powerful first order theorem provers, such as E, Vampire, and SPASS. By dropping the strict requirement of type soundness, Sledgehammer returns suggestions faster than the actual complexity of this problem would allow, thus modelling, in a way, a "human reader", allowing the steps between the explicitly given statements in a proof to become bigger, like in standard textbook proofs. With this tool, the intelligible proving language Isar, our Naproche experiences, and with the concise HOL-arguments, we aim to create a natural presentation of ZF, of Paulson's formalisation, and of further "natural formal mathematics".

We view our project as a contribution towards the QED manifesto [QED94], about which Wiedijk said in [Wie07]:

> The other reason that there has not been much progress on the vision from the QED manifesto is that currently formalized mathematics does not resemble real mathematics at all. Formal proofs look like computer program source code. For people who do like reading program source code that is nice, but most mathematicians, the target audience of the QED manifesto, do not fall in that class.

The Naproche project has shown that mathematical texts can be reformulated naturally and formally at the same time. In the current project we want to extend this experience to large texts, supported by a strong established system.

References

[AFP] The archive of formal proofs. `http://afp.sourceforge.net`.

[Cra13] Marcos Cramer. *Proof-checking mathematical texts in controlled natural language*. PhD thesis, Rheinisch Friedrich-Wilhelms-Universität Bonn, 2013.

[Göd40] Kurt Gödel. The consistency of the axiom of choice and of the generalized continuum hypothesis with the axioms of set theory. *Annals of Mathematics Studies*, 3:33–101, 1940.

[Nap] The naproche project. `http://naproche.net/`.

[Pau03] Lawrence Paulson. The relative consistency of the axiom of choice mechanized using Isabelle/ZF. *LMS Journal of Computation and Mathematics*, 6:198–248, 2003.

[PB10] Lawrence Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In *8th International Workshop on the Implementation of Logics (IWIL-2010)*, 2010.

[QED94] The QED manifesto. Automated Deduction – CADE 12, Springer-Verlag, Lecture Notes in Artificial Intelligence, 1994.

[Wie07] Freek Wiedijk. The QED manifesto revisited. In *From insight to proof : Festschrift in honour of Andrzej Trybulec*, pages 121–134. University of Białystok, 2007.

Machine learning in Satallax

Michael Färber[1], Chad Brown[2]

[1] Universität Innsbruck, Austria
michael.faerber@uibk.ac.at
[2] Czech Technical University in Prague, Czech Republic

Abstract

We discuss options to integrate machine learning into the automated theorem prover Satallax and show some preliminary experimental results.

Satallax is a higher-order automated theorem prover developed by Chad Brown [Bro12]. Its proof search puts commands with a certain delay onto a priority queue and executes the commands sorted by ascending delay. The delay for each command is calculated via a set of customisable flags. Our goal is to reduce the time needed to find a proof by learning from previous proof data and influencing the command priorities accordingly.

Related work includes the theorem provers MaLeCoP [UVŠ11] and FEMaLeCoP [KU15], which are versions of leanCoP [Ott08] extended by internal guidance. FEMaLeCoP influences the selection of contrapositives at every tableau extension step, based on the current path and which contrapositives were useful at which paths in previous proofs. Such an approach is not directly applicable in Satallax, because Satallax does not use paths, but rather a monotonically growing set of terms. Furthermore, where in leanCoP's calculus we only need to choose between a constant set of input contrapositives for extension steps, in Satallax the set of commands to evaluate is not known at the beginning of the proof.

While there are 11 types of commands in Satallax 2.7 (such as mating and confrontation), the command to process a term (called "ProcessProp1") usually makes for more than 90% of commands generated during proof search. Therefore, we concentrated on this command to guide proof search.

In a first approach, we implemented term weighting, a technique present for example in E-prover [Sch13] that gives higher relevance to smaller terms. In a first evaluation, this made the proving process slower and actually reduced the number of proven problems, but a change of the parameters and of the weighting scheme might improve this situation in the future.

In a second approach, we regarded the symbols (i.e. constants) of all previously processed terms as *features* and all symbols of the current term as *labels*. We recorded all features and labels for every term processed in a proof, and differentiated whether the term contributed to the proof. For this, we chose a two-pass solution: In a first pass, we run Satallax with a timeout and record the set of refutation terms when a proof was found. If a proof was found during the first pass, we rerun Satallax without timeout (to prevent that recording learning information makes Satallax time out) and store features and labels for all terms, marking those present among the refutation terms as positive and the others as negative examples.

We then convert the learning data to a format that allows fast evaluation of new terms via a Naive Bayesian classifier with inverse document frequency (IDF), similarly as done in FEMaLeCoP: In contrast to a usual Bayesian classifier, where one wishes to retrieve a set of labels fitting an input set of features, our classifier is optimised to quickly answer the question how well a certain given label fits a set of features.

The Bayesian classifier is then loaded into Satallax and influences the priority of terms by obtaining the features of previously processed terms (which are cached for performance reasons) and running a Bayesian classification of all term labels, summing up the results. Based on the score from the classifier, we give a higher or smaller delay to the term. We chose the parameters

of the classifier via an off-line evaluation of existing proofs, by calculating for every proof-relevant term the number of irrelevant terms with a higher priority assigned by the classifier.

On a first evaluation on a higher-order version of the Mizar dataset (kindly provided by Josef Urban), this second approach did not fare so well, only proving a few more problems than without learning. Furthermore, a new problem emerged, namely that the Bayesian priorities assigned to the terms were so high that they completely pushed other commands, such as mating, to the very end, thus delaying them too much.

For these reasons, we tried a third approach, where we completely disregard the current prover state and base internal guidance just on the terms encountered in previous proofs: Based on how often a term appeared in previous proofs and how often it contributed to a proof, we determine its priority. An important observation that led to this approach was that the terms appearing in Satallax proofs were in almost all cases either always relevant or always irrelevant – there were hardly any "controversial" terms. We evaluated on the HOL version of the Mizar dataset (902 problems) with the single best-performing Satallax mode (mode483) and a timeout of 1 second. The parameters of the machine learning were optimised via a particle swarm [KE95]. This approach has been so far the most successful, yielding up to 19 new solved problems to a set of 529 solved problems (+3,6%).

The presented approaches learn only data for given symbols, therefore proof knowledge of a certain theory does not help in a different theory whose symbols are all different. It would be interesting to abstract terms and symbols such that proof knowledge could be shared among different theories. We will continue improving the machine learning techniques presented, as well as trying new methods such as Bayesian classification of terms instead of term symbols.

Supported by the ERC Consolidator grant nr. 649043 *AI4REASON*.

References

[Bro12] Chad E. Brown. Satallax: An automatic higher-order prover. In *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, pages 111–117, 2012.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Nov 1995.

[KU15] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'15)*, LNCS, 2015. to appear.

[Ott08] Jens Otten. leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, pages 283–291, 2008.

[Sch13] Stephan Schulz. System description: E 1.8. In *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, pages 735–743, 2013.

[UVŠ11] Josef Urban, Jiří Vyskočil, and Petr Štěpánek. MaLeCoP machine learning connection prover. In *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings*, pages 263–277, 2011.

Conjecturing over Large Corpora

Thibault Gauthier[1], Cezary Kaliszyk[1], Josef Urban[2]*, and Jiří Vyskočil[2]

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

Abstract

We propose automated methods, learning from large libraries of formalized theorems, to generate plausible formulas. The discovery of interesting conjectures in multiple domains demonstrates the generality of our approach for exploring mathematical theories.

A critical part of the work of a mathematician is the process of conjecture-making during which the researcher observes and find patterns in and between mathematical objects. The observed regularity is transformed into formalized conjectures that describe or better explain the behavior of the objects.

Computer scientists have since long tried to reproduce this process automatically, but the most successful methods were restricted to a specific domain such as graph theory [1], number theory [3] and algebra [2]. During the last four decades, a large number of theories were formalized inside interactive theorem provers(ITP). The most popular ones, Mizar [10], Isabelle/HOL [11] and Coq [4], provide many automated tools for theorem proving, however the assistance users can obtain for making conjectures in the large-theory context is still limited to non-existent.

We believe that good conjecturing is an essential step for automation of harder proofs in ITPs, therefore we propose several methods for generating conjectures over large libraries. In particular, the methods need to answer two questions: (i) what are interesting conjectures, and (ii) how to find them efficiently and quickly reject bad conjectures.

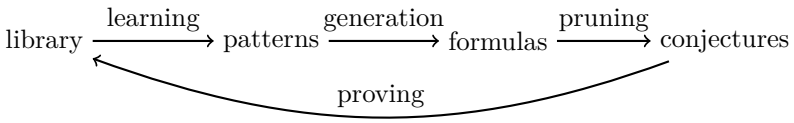

Figure 1: Self-improving conjecture generation loop

A frequent drawback of existing general-purpose conjecturing methods [7, 12] is that they try to define exactly what *interesting* means only by a series of hand-crafted rules that are often used exhaustively. Large-theory reasoning methods are today usually *data-driven*, i.e., using guidance extracted from the large libraries. We would like to find such data-driven/learning methods useful for guiding the conjecturing process. Such methods should be combined with fast pruning mechanisms that will allow to quickly focus on the nontrivial conjectures. Once the number of conjectures is small enough, an automatic proof search can be performed on each of them. The newly discovered theorems can enrich the libraries that can be further learned from. This will likely influence the next learning and pruning steps. A schematic representation of this self-improving loop is shown in Fig 1.

*Supported by ERC Consolidator grant nr. 649043 *AI4REASON*.

Conjecture generating methods:

Statistical Analogy: We have developed a procedure for statistical/semantic matching of concepts between different libraries based on the HOL logic [5]. The concepts are characterized by the (statistically weighted) vector of abstracted patterns of lemmas that hold about them. The closer such characteristic vectors, the stronger the match between the concepts. This matching allows to heuristically translate and transfer lemmas from one terminology to another, and to try to prove such conjectures in the other library. We have initially tried to apply this procedure also to *just one* (Mizar) library, by forbidding the obvious strongest match of each concept to itself. The initial experiment generated some interesting provable conjectures that were not yet in the library, e.g., by exploring the strong duality between $\cup$ and $\cap$. We want to further develop this method by abstracting not just over symbols but also over whole subterms, and running it over the whole Mizar and other libraries [6].

Exploring concept hierarchies: Mizar has large subtyping hierarchies, e.g., every real number is also complex. We can statistically measure the likelihood that a lemma generalizes to the wider concept, and use such likelihoods for conjecturing. The hierarchies can be more complicated and in general concern all (monadic) predicates in logics that do not allow subtyping.

Probabilistic formula generation by (dis)ambiguation: We have recently started to extract probabilistic context-free grammars (PCFGs) and context-aware grammars from the large libraries that allow statistical parsing of ambiguous formulas [9]. This has already produced a number of "wrongly parsed" ambiguous formulas that are actually provable (and new). This can be leveraged, e.g. in the following way. A formula is first *ambiguated*, e.g. by using `+` instead of `real_plus`, forgetting variable types etc. Such ambiguous symbols are often *high-level concepts* such as *abstract group addition*, etc. When we vary the interpretation of various parts of such ambiguous formulas (e.g., interpreting all variables as abstract group elements) and run the probabilistic parser on such partially disambiguated formulas, we will quite naturally produce interpretation of the ambiguous formula ("idea") in the changed context [8].

Genetic modification: The "wildest" mechanism that requires good fitness functions and fast evaluation are genetic algorithms for evolving formula trees. As in the PCFG-based mechanisms, we can apply a lot of *semantic* pruning when doing crossover and mutation – for example a crossover with wrongly typed arguments might not be generated or repaired. The genetics can also be seeded in various ways by the probabilities obtained in the previous methods.

Pruning methods:

Types: A simple pruning method is to require type consistency, using the various notion of types in the particular proof assistants. This helps a lot already in the probabilistic disambiguation task.

Finite Models: The MaLARea system [13] has a finite-model generating loop, which can relatively quickly generate a large number of useful models fo a large library. Evaluation of formulas in such pool of models is typically fast, and the resulting characteristic vectors (of evaluations) be used for quick checking of the validity and semantic similarity of the generated conjectures.

Theories as Models: When a conjecture concerns abstract fields, it is useful to check whether it holds in the fields of real or complex numbers. Often a conjecture can be quickly rejected or repaired by checking in particular theory instantiations. Again, the large libraries provide a number of such theory instantiations – this kind of checking is in some sense dual to the hierarchy-based conjecture generation (which goes in the opposite direction – from an instance to its generalization).

PseudoModels: Even when we do not have a particular finite model or a quickly decidable instance of a theory, there might be imprecise methods such as deep neural networks trained on the large libraries that can quickly estimate the (in)validity of a particular conjecture. It would be quite interesting to see how good such methods can be in various domains.

References

[1] M. Aouchiche and P. Hansen. A survey of automated conjectures in spectral graph theory. *Linear Algebra and its Applications*, 432(9):2293 – 2322, 2010. Special Issue devoted to Selected Papers presented at the Workshop on Spectral Graph Theory with Applications on Computer Science, Combinatorial Optimization and Chemistry (Rio de Janeiro, 2008).

[2] Bruno Buchberger, Adrian Crăciun, Tudor Jebelean, Laura Kovács, Temur Kutsia, Koji Nakagawa, Florina Piroi, Nikolaj Popov, Judit Robu, Markus Rosenkranz, and Wolfgang Windsteiger. Theorema: Towards computer-aided mathematical theory exploration. *Journal of Applied Logic*, 4(4):470 – 504, 2006. Towards Computer Aided Mathematics.

[3] Simon Colton. Automated conjecture making in number theory using hr, otter and maple. *Journal of Symbolic Computation*, 39(5):593 – 615, 2005. Automated Reasoning and Computer Algebra Systems (AR-CA)AR-CA.

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

[5] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen M. Watt, James H. Davenport, Alan P. Sexton, Petr Sojka, and Josef Urban, editors, *Conference on Intelligent Computer Mathematics (CICM 2014)*, volume 8543 of *LNCS*, pages 267–281. Springer, 2014.

[6] Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2015)*, 2015.

[7] Moa Johansson, Lucas Dixon, and Alan Bundy. Conjecture synthesis for inductive theories. *Journal of Automated Reasoning*, 47(3):251–289, 2011.

[8] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Improving statistical linguistic algorithms for parsing mathematics. In Boris Konev, Stephan Schulz, and Laurent Simon, editors, *The 11th International Workshop on the Implementation of Logics (IWIL'15)*, EasyChair Proceedings in Computing. EasyChair, 2015. to appear.

[9] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 227–233, 2015.

[10] Roman Matuszewski and Piotr Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4:3–24, 2005.

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

[12] Yury Puzis, Yi Gao, and Geoff Sutcliffe. Automated generation of interesting theorems. In *FLAIRS Conference*, pages 49–54, 2006.

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

Machine Learning of Given Clause Selection in E Prover

Jan Jakubův and Josef Urban

Czech Technical University, CIIRC, Prague {jakubuv,josef.urban}@gmail.com

Abstract

Many saturation based theorem provers are build around an inference loop where new clauses are produced by applications of inference rules. Selection of clauses participating in the inference provides the main source of non-determinism and an important choice-point of the loop. E prover is a state-of-the-art theorem prover fitting the above description. We propose to extend the E prover by adopting methods of clause selection successfully used in practice, namely, *semantic clause features* and *learning-based guidance*.

1 Large Theories and Given Clause Selection

First-order automated theorem provers (ATPs) aim to decide the validity of an input conjecture C within a given theory T. *Saturation-based* ATPs search for a contradiction by generating clauses deducible from T extended with the negation of C. Deducing the contradiction from $T \cup \{\neg C\}$ is equivalent to proving conjecture C in theory T.

Many state-of-the-art saturation-based ATPs use the *given clause algorithm* exemplified by *Otter* [3]. The input problem $T \cup \{\neg C\}$ is translated into a logically equivalent set of clauses. Then the search for a contradiction, represented by the empty clause, is performed maintaining two sets: the set P of *processed clauses* and the set U of *unprocessed* clauses. Initially, all the input clauses are unprocessed. The algorithm repeatedly selects a *given clause* g from U and generates all possible inferences using g and the processed clauses from P. Then, g is moved to P, and U is extended with the newly produced clauses. This process continues until the empty clause is inferred, or P becomes *saturated*, that is, nothing new can be inferred ($U = \emptyset$).

The search space of this loop grows quickly. Several methods can be used to make the proof search more efficient. The search space can be narrowed by adjusting (typically restricting) the inference rules, pruned by using *forward* and *backward subsumption*, reduced by pre-selecting relevant input clauses, or otherwise simplified. One of the main sources of non-determinism affecting effectiveness of the search is the selection of the given clause. Clever selection mechanism can improve the search dramatically: in principle, one only needs to do the inferences that participate in the final proof. So far, this is often only a tiny portion of all the inferences done by the ATPs during the proof search.

In our work, we consider E [5], a high-performance *equational* ATP for first-order logic with equality. E uses *resolution* and *paramodulation* inference rules within the given clause algorithm. We propose to use methods based on *clause features* and *learning-based guidance* to improve the given clause selection in E.

2 Clause Features and Learning-based Guidance

There are already several ATP methods that characterize clauses by their specific properties called *features* and use such features in various ways. E prover uses *feature vectors* [6] for clause indexing in an efficient implementation of forward and backward subsumption. Given clause c,

E uses the following clause features: the number of positive or negative literals in c (denoted $|c^+|$ and $|c^-|$), the number of occurrences of symbol f in positive or negative literals ($|c^+|_f$ and $|c^-|_f$), and the depth of the deepest occurrence of symbol f in positive or negative literals ($d_f(c^+)$ and $d_f(c^-)$).

Another successful use of features is due to Kaliszyk *et al.* [2] where features are used to train a machine learner for selection of relevant axioms from a large theory. Such features include the set of symbols of a clause (SYM), the set of all subterms with unified variables (TRM_0), the set of all subterms with de Bruijn normalized variables (TRM_α), various sets of generalized terms (MAT_), term walks (PAT), *substitutions tree* nodes (ABS), and the set of all unifying terms (UNI).

Another feature-aware method related to our purposes is the (Fairly Efficient) Machine Learning Connection Prover (FE)MaLeCoP [1, 7]. (FE)MaLeCoP is a *Tableaux*-based theorem prover where the main source of non-determinism is the selection of the best clause in a Tableaux expansion step. The importance of this step is comparable to the selection of a given clause in saturation-based theorem provers. (FE)MaLeCoP is equipped with a machine learning system, which can be used to advice the clause selection for Tableaux expansion. The machine learning system is trained on the pairs of proof state features and the corresponding clauses that were part of the previous successful proofs.

3 Proposed Given Clause Selection Refinements

We propose to implement *heuristics* for E prover, which utilize various clause features and learning-based guidance. E prover comes with various heuristics, which can be used to influence the given clause selection algorithm by assigning different *weights* to the participating clauses. E prover then prefers clauses with lower weights. One of the very useful heuristics is `ConjectureRelativeSymbolWeight`, which uses goal-directed evaluation to prefer clauses that have a stronger symbolic connection to the conjecture.

As the first step, we propose to extend the `ConjectureRelativeSymbolWeight` heuristics to use more advanced clause features. Currently, the heuristic measures the connection of a clause to the conjecture by considering symbols the clause has in common with the conjecture. In the large-theory axiom-selection problem, symbol-based metrics are quite significantly improved by the more advanced (sub)term-based features, especially if they also consider semantic relations like matching [2]. E already has a lot of the necessary infrastructure: (sub)terms in clauses are normalized and shared in *termbanks*, making it easy to detect (sub)term overlap. Also, E has efficient *discrimination trees*, whose nodes we plan to use (as in [2]) as the matching features.

As a second step, we propose to extend the above heuristics with learning-based guidance analogous to (FE)MaLeCoP. Such guidance seems more challenging in the saturation-based setting, because the notion of a *proof state* is not as clear and compact as in the tableaux setting. We plan to experiment with various efficient definitions of the proof state. The simplest one is the static one already used by the conjecture-oriented E heuristics – the "proof state" is just the conjecture. But instead of just measuring the feature overlap with the conjecture, we will learn clause relevance for the particular conjecture features from many related proofs. There are (at least) two ways how to practically do this. (1) Putting the learning-based clause evaluator directly into E as in (FE)MaLeCoP or in Schulz's PhD work, which used pattern-based guidance [4] (similar also to Veroff's hints [8]). Or (2) just estimating the feature (symbol, (sub)term, etc.) weights before the loop starts, based on the conjecture features and the previous proof traces, and using (possibly further modified) weighting mechanisms already available in E. This can be further extended to estimating good term orderings and other important

parameters. Later we would like to consider also more dynamic proof-state features for the learning and clause selection. This could be various metrics based either on absolute criteria such as the proportion of short ("good") clauses and their distribution, or also more learning-based metrics, such as the number/proportion of already derived clauses that were needed in related proofs, etc.

Acknowledgments

Supported by the ERC Consolidator grant nr. 649043 *AI4REASON*.

References

[1] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly Efficient Machine Learning Connection Prover. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 88–96. Springer, 2015.

[2] Cezary Kaliszyk, Josef Urban, and Jirı Vyskocil. Efficient semantic features for automated reasoning over large theories. In *IJCAI*, volume 15, pages 3084–3090, 2015.

[3] William W McCune. *Otter 3.0 reference manual and guide*, volume 9700. Argonne National Laboratory Argonne, IL, 1994.

[4] Stephan Schulz. *Learning search control knowledge for equational deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000.

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

[6] Stephan Schulz. Simple and efficient clause subsumption with feature vector indexing. In *Proc. of the IJCAR-2004 Workshop on Empirically Successful First-Order Theorem Proving*. Citeseer, 2004.

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

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

Loops and the AIM Conjecture: History and Progress

Michael Kinyon

December 23, 2015

The fact that the inner automorphism group $\mathrm{Inn}(G)$ of a group G is isomorphic to the quotient of G by its center $Z(Q)$ implies the essentially trivial observation that $\mathrm{Inn}(G)$ is abelian if and only if G is nilpotent of class at most 2. For a loop Q with inner mapping group $\mathrm{Inn}(Q)$, class 2 nilpotency is indeed sufficient for $\mathrm{Inn}(Q)$ to be abelian, but the existence of what are now known as loops of Csörgő type show that the condition is not necessary. All known examples of loops Q of Csörgő type have certain features in common: the quotient $Q/\mathrm{Nuc}(Q)$ is an abelian group and $Q/Z(Q)$ is a group. These properties certainly imply that Q has nilpotency class at most 3, but in fact, they are stronger properties. In addition, if we define the (conventional) commutator $[\cdot,\cdot]$ by the equation $xy \cdot [y,x] = yx$, then, when $\mathrm{Inn}(Q)$ is abelian, this binary operation is associative. In groups, it is a classical result of Levi that associativity of the commutator is equivalent to nilpotence of class 2.

These observations about the known examples have led to a conjecture which I have been working on and publicizing for several years:

AIM Conjecture: Let Q be a loop. The following are equivalent:

1. $\mathrm{Inn}(Q)$ is an abelian group;
2. $Q/\mathrm{Nuc}(Q)$ is an abelian group, $Q/Z(Q)$ is a group and the commutator is associative.

What is not obvious from this high level discussion is that the formulation of the problem can, in fact, be given in purely equational terms. The efforts of myself and Bob Veroff have been aimed (pun intended) at resolving this conjecture using Prover9 and other tools.

"(2) implies (1)" is now known to hold in general. For "(1) implies (2)," the associativity of the commutator has been proven, but the rest remains open in general. For many classical varieties of loops, such as Moufang, Bol, automorphic and others, "(1) implies (2)" has been established.

This talk will be about the AIM Conjecture, its history and some of its interesting generalizations and extensions, and its current status. I will speak about this from a mathematician's perspective, and no background in loop/quasigroup theory will be assumed. For the expected audience, I will describe how the problem is formulated in the ATP context.

The different facets of this project are joint work with various combinations of people, especially Bob Veroff who has been involved in all of it.

Towards Knowledge-Based Assistance for Scholarly Editing

Jana Kittelmann[1] and Christoph Wernhard[2]

[1] Martin-Luther-Universität Halle-Wittenberg, Halle (Saale), Germany
info@janakittelmann.de
[2] Technische Universität Dresden, Dresden, Germany
info@christophwernhard.com

Abstract

We investigate possibilities to utilize techniques of computational logic for scholarly editing. Semantic Web technology already makes relevant large knowledge bases available in form of logic formulas. There are several further roles of logic-based reasoning in machine supported scholarly editing. KBSET, a free prototype system, provides a platform for experiments.

1 Introduction – Knowledge Bases, TEI and Beyond

Much of the background material used today in scholarly editing is available electronically in form of large knowledge bases. Some of these emerge from the archive, library and museum communities, for example *Gemeinsame Normdatei (GND)*[1], which provides about 120 million facts on approximately 11 million entities such as persons, institutions and works, and *Kalliope*[2], which provides meta data on collections of personal papers and manuscripts, linking them with *GND*. Other relevant large knowledge bases have a more general scope, for example *GeoNames* about locations or *YAGO* [6] and *DBpedia* [9], which combine extracts from various sources, including *Wikipedia*. Outcomes of scholarly editing include electronic documents that may again be considered as providing knowledge bases. The prevailing technology to realize all these knowledge bases is the Semantic Web as advocated by the W3C, characterized by ontologies, global identifiers (*URIs*) and subject-predicate-object statements with *RDF*.

The XML-based document markup conventions *Text Encoding Initiative (TEI)* [11] represents the best developed machine support for scholarly editing today. Main objectives are rendering for different media and extraction of metadata. URIs as attribute values of markup elements are supported to provide rudimentary connections with knowledge bases. From today's perspective, several demands by scholarly editing are out of the scope of *TEI*. This includes in particular the incorporation of advanced and automated semantics related techniques such as named entity recognition or statistics-based text analysis, the advanced use of relationships to external knowledge bases and to formal semantics, and the coupling of object text with associated information in ways that are more flexible than in-place markup. Also the generation of high-quality print or hypertext presentations based on *TEI* are quite expensive undertakings.

We approach these demands here from the view of computational logic. With Semantic Web technology, large fact bases can simply be considered as sets of logic facts. Logic languages have various further potential roles in machine supported scholarly editing, such as specifying properties and values associated with texts, specifying collections of text and pieces of text to which these relate, specifying knowledge sources and their combination and specifying inferences involved in automated computation of information to be associated with texts. We discuss various points that are arising there. Aspects of this interdisciplinary work have been recently presented to the scholarly editing community [7]. To clarify precise requirements of machine support for scholarly editing and to experiment with advanced techniques, the authors implemented the experimental platform *KBSET*.

[1] http://www.dnb.de/gnd.
[2] http://kalliope-verbund.info/.

2 Arising Issues

Three phases can be identified for machine assisted scholarly editing: (1) Creating the enhanced object text; (2) Generating intermediate representations for inspection by humans or machines; (3) Generating consumable presentations. Support for all of them should be of high quality, professional and at the state of the art, implying the inclusion of specialized techniques and systems as well as the combination of automated techniques with information and adjustments provided by humans. The latter aspect is an essential difference from conventional programming and query languages. Relevant techniques include non-monotonic reasoning, semantics-based knowledge partitioning [14, 5, 3, 8] and the provision and use of explanations for inferred information, as exemplified by proofs in mathematical knowledge bases [13].

Statistics-based techniques, which are essential for many natural language processing operations, have to be combined with a symbolic framework. "Ranked sets", sets whose elements are ranked on the basis of associated feature tuples – like the familiar result sets returned by Web search engines – seems useful as data structure to mediate between the two paradigms.

Annotations in external documents – instead of in-place markup – are facilitated by powerful techniques to identify places in text – based on syntactic as well as semantic properties.

Scholarly editing involves the association of various forms of epistemic status with facts, which is interesting to model formally. For example, a creation date associated with written communication can be given by its author or can be inferred – by the editor or by a machine, can be only partially specified by the author, can be specified with a certain precision, etc.

Efficient access to large knowledge bases requires caching and preprocessing, which ideally should be performed automatically on the basis of the queries performed by the knowledge processing engine. Relevant techniques come from optimization in databases [12] and first-order model computation [10]. In particular, recent approaches to view-based query processing [2] with variants of Craig's interpolation and second-order quantifier elimination [12, 1, 15] seem useful to automate the conversion from application neutral representations such as *RDF* fact bases to access-oriented representations.

3 The KBSET System – An Experimental Platform

The objective of the *KBSET*[3] system is to make requirements precise and to investigate the discussed issues and possibilities. Currently these are realized only in part. So far, the system has been applied for a draft edition of a 19th century German book, which accompanies the system as comprehensive example. A future version will be used for a forthcoming electronic edition of the correspondence of Swiss philosopher and polymath Johann Georg Sulzer (1720–1779) prepared at University Halle-Wittenberg.

The system takes the following inputs: (1) An object text document, possibly in LaTeX; (2) Documents with annotations, where the associated places in the object text are specified abstractly; (3) Large fact bases, like *GND*, *GeoNames*, *YAGO* and *DBpedia*, from which extracts are preprocessed and cached; (4) An *assistance document*, that is, a configuration file, where information is given to bias or override automated inferencing such that fully correct results are obtained. A user interface is provided that integrates the system into the *Emacs* editor. A facility for named entity recognition is included, which identifies individual persons, locations and dates, based on *GND* and *GeoNames* as gazetteers. Conventional recognition systems such as *Stanford Named Entity Recognizer* [4] just associate entity types with phrases.

[3] `http://cs.christophwernhard.com/kbset/`, free software under GNU General Public License.

A variety of outputs can be produced, including LaTeX documents with annotations and merged-in inferred information, and *Emacs* text buffers where recognized identifiers are highlighted and candidate entities can be inspected along with a presentation of the respective rationale for choosing them. A typical application is the development of an annotated essay or book, where the object text is edited in LaTeX and the assistance document evolves step-by-step until the inferred information is fully correct.

Acknowledgments. This work was supported by *Alexander von Humboldt-Professur für neuzeitliche Schriftkultur und europäischen Wissenstransfer* and by *DFG grant WE 5641/1-1*.

References

[1] M. Benedikt, B. ten Cate, and E. Tsamoura. Generating low-cost plans from proofs. In *PODS'14 – Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems*, pages 200–211. ACM, 2014.

[2] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoret. Comp. Sci.*, 371(3):169–182, 2007.

[3] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular reuse of ontologies: Theory and practice. *J. Artif. Intell. Res.*, 31:273–318, 2008.

[4] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proc. 43nd Ann. Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 363–370. ACL, 2005.

[5] S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? A case for conservative extensions in description logics. In *Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 187–197. AAAI Press, 2006.

[6] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artif. Int.*, 194:28–61, 2013.

[7] J. Kittelmann and C. Wernhard. Knowledge-based support for scholarly editing and text processing. In *DHd 2016 – Digital Humanities im deutschsprachigen Raum: Modellierung – Vernetzung – Visualisierung. Die Digital Humanities als fächerübergreifendes Forschungsparadigma. Konferenzabstracts*, pages 176–179. nisaba verlag, 2016.

[8] R. Kontchakov, F. Wolter, and M. Zakharyaschev. Logic-based ontology comparison and module extraction, with an application to DL-Lite. *Artif. Int.*, 174(15):1093–1141, 2010.

[9] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia – A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[10] B. Pelzer and C. Wernhard. System description: E-KRHyper. In *Automated Deduction (CADE-21)*, volume 4603 of *LNCS (LNAI)*, pages 503–513. Springer, 2007.

[11] The TEI Consortium. *TEI P5: Guidelines for Electronic Text Encoding and Interchange, Version 2.8.0*. Text Encoding Initiative Consortium, 2015. `http://www.tei-c.org/Guidelines/P5/`.

[12] D. Toman and G. Weddell. *Fundamentals of Physical Design and Query Compilation*. Synthesis Lectures on Data Management. Morgan and Claypool, 2011.

[13] J. Urban, P. Rudnicki, and G. Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50(2):229–241, 2013.

[14] C. Wernhard. Semantic knowledge partitioning. In *Logics in Artificial Intelligence (JELIA 04)*, volume 3229 of *LNCS (LNAI)*, pages 552–564. Springer, 2004.

[15] C. Wernhard. Expressing view-based query processing and related approaches with second-order operators. Technical Report KRR 14–02, TU Dresden, 2014. `http://www.wv.inf.tu-dresden.de/Publications/2014/report-2014-02.pdf`.

Automation and Computation in the Lean Theorem Prover

Robert Y. Lewis[1] and Leonardo de Moura[2]

[1] Carnegie Mellon University, Pittsburgh, PA, USA
rlewis1@andrew.cmu.edu
[2] Microsoft Research, Redmond, WA, USA
leonardo@microsoft.com

Abstract

We describe some details of the Lean theorem prover, focusing on the aspects that make Lean a good environment for automation and AI. We also outline a novel automated procedure for proving inequalities over $\mathbb{R}$ that is currently being implemented.

The Lean theorem prover. The Lean theorem prover is a new proof environment being developed at Microsoft Research ([3, 4]). While somewhat similar to Coq [2] in foundation and syntax, Lean aims to improve on the status quo in a number of ways. In dependent type theory, complex and subtle techniques are needed to elaborate terms in an intuitive way. Lean combines an extremely efficient elaboration process with a powerful type class inference mechanism. This setting allows for a clean, uniform development of the algebraic hierarchy and number structures, making the system conducive to generalized automated methods. Machine-learning techniques are being developed to choose relevant theorems for use in ATP, making use of learned strategies for problem-solving.

Indeed, Lean has been developed from the beginning with automation in mind. While we will treat Lean as an interactive proof environment in this document, the system can also be seen as a body of verified mathematical claims and an API for connecting them; in effect, it is a setting for automated tools to assemble proofs. Lean aims to bridge the gap – or blur the line, if one likes – between user-centric interactive proving and machine-centric automated proving.

Type class inference. Lean allows any family of inductive types to be marked as a *type class*. A term that instantiates one such type can be marked as an *instance* of the type class. The elaborator can then synthesize terms of this type by searching through declared instances. This search is not simply a lookup table, since instances can depend on other instances. In the following example, square brackets instruct the elaborator to infer the term using type class inference. The proof in the final line is inferred as well.

```
inductive decidable [class] (p : Prop) : Type := ...
definition dec_and [instance] (p q : Prop) [Hp : decidable p] [Hq : decidable q] : decidable (p ∧ q) := ...
constant P : ℕ → Prop
axiom Pn [instance] : ∀ n, decidable (P n)
example : decidable (P 2 ∧ P 3 ∧ P 4) := _
  -- inferred: @dec_and (P 2) (P 3 ∧ P 4) (Pn 2) (@dec_and (P 3) (P 4) (Pn 3) (Pn 4))
```

The type class inference tool employs a λ-Prolog-style [7] recursive, backtracking search, and caches results.

Algebraic hierarchy. The algebraic hierarchy serves two purposes in a proof assistant. First, these structures are objects of study in their own right: users prove theorems about groups, vector spaces, or fields in the abstract without intended applications. Second, these structures are instantiated by the concrete number structures: $\mathbb{R}$ forms a field, and theorems proved about fields should also apply to $\mathbb{R}$.

As in similar systems, Lean's algebraic structures are defined as record types indexed over a base type.

```
structure monoid [class] (A : Type) extends semigroup A, has_one A :=
  (one_mul : ∀a, mul one a = a) (mul_one : ∀a, mul a one = a)
```

The `extends` syntax ensures that all data fields required for a semigroup are also required for a monoid, and automatically constructs a definition of the form

```
definition monoid.to.semigroup [instance] [H : monoid A] : semigroup A := ...
```

by finding the appropriate projections. When one structure A' extends another structure A, all theorems proved about types instantiating A will also apply to types instantiating A'. For instance, the associativity of multiplication in a monoid follows from the corresponding theorem about semigroups. No duplication or instantiation of theorems is necessary. Since projections reduce definitionally, when type class inference finds different terms witnessing the same property, they will be definitionally equal.

The effect of this development is an algebraic hierarchy that behaves exactly as a mathematician would expect: structures are cumulative, and when using a certain theorem or property, there is no need to remember at what point of the hierarchy that property entered.

Number structures. The number structures $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ are all defined and instantiate the appropriate algebraic structures. Once `int_is_comm_ring [instance] : comm_ring int` has been defined, all theorems applicable to commutative rings are immediately available for $\mathbb{Z}$. Importantly, this process happens via exactly the same mechanism as in the algebraic hierarchy. Automation designed for, say, ordered fields will not distinguish between $\mathbb{Q}$ and $\mathbb{R}$, making the system clean and uniform.

Lean's number structures have been developed constructively, through showing that $\mathbb{R}$ is an ordered ring. The remainder of the construction of $\mathbb{R}$ and $\mathbb{C}$ is classical.

Numerical computation. Binary numerals can be used in Lean in any structure that has 0, 1, and +.

```
definition bit0 {A : Type} [s : has_add A] (a : A)                : A := add a a
definition bit1 {A : Type} [s : has_one A] [t : has_add A] (a : A) : A := add (bit0 a) one
definition add2 (n : ℕ) : ℕ := n + 2
```

The type of a numeral is inferred from context. Here, 2 is shorthand for `bit0 nat nat_has_add (one nat nat_has_one)`, where `nat_has_add : has_add nat` and `nat_has_one : has_one nat` are found by type class inference. In structures with the appropriate properties of addition, multiplication, subtraction, and division, numeric computations can be performed efficiently by binary arithmetic algorithms.

Uniformity for automation. The uniform approach using type class inference is conducive to both shallow and deep automation. Treating numerals generally makes arithmetic work identically across structures: computations with + and − in $\mathbb{Z}$ are exactly the same as in $\mathbb{R}$ and as in any ring. This allows Lean to have a powerful and flexible simplifier. Type class inference makes it trivial for automated techniques to detect what algebraic structures are present in a goal and what methods may be applicable. Since the same general theorem – not separate instantiations of one general theorem – asserts the associativity of addition in $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, etc., machine learning techniques do not need to be trained separately on examples for each structure.

Verifying non-linear inequalities. In interactive theorem proving, one often wishes to discharge simple arithmetic hypotheses that follow from premises in the context. Proof assistants implement solvers over various domains, for example linear constraints over $\mathbb{Q}$ or $\mathbb{Z}$. Higher-powered methods – e.g. CAD methods for RCFs – can only generate proofs with much effort [6]. A class of relatively simple but highly nonlinear problems over $\mathbb{R}$ has proven very difficult to attack automatically. These problems show up often in mathematical analysis and in the verification of physical systems. For example, consider the inference

$$0 < x < y,\ u < v \ \Rightarrow\ 2u + \mathtt{exp}(1 + x + x^4) < 2v + \mathtt{exp}(1 + y + y^4).$$

This inference is tight, nonlinear, outside the theory of RCF, and difficult to capture systematically with backchaining methods; nevertheless, it looks "simple" to any mathematician. In [1, 5], Avigad, Lewis, and Roux describe a system, based on a Nelson-Oppen style architecture [8], that solves many such problems by saturating arithmetic facts and heuristically instantiating lemmas. While not a complete decision procedure, a prototype has been shown to perform well on real-life examples from ITP and system verification.

This system – nicknamed Polya – is currently being implemented as a tactic in Lean. One requirement of this system is that terms be rewritten to a normal form; Lean's flexible rewriter allows us to achieve this with minimal overhead. Polya makes many simple numerical calculations over $\mathbb{Q}$, which is efficient to verify in Lean. Once Polya is combined with a linear solver and general-purpose AI methods, we believe Lean will support a suite of automated tools exceeding that of other theorem provers.

References

[1] J. Avigad, R. Y. Lewis, and C. Roux. A heuristic prover for real inequa. *Journal of Automated Reasoning*, Forthcoming.

[2] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: The calculus of inductive constructions*. Springer-Verlag, Berlin, 2004.

[3] L. de Moura, J. Avigad, S. Kong, and C. Roux. Elaboration in dependent type theory. *http://arxiv.org/pdf/1505.04324v1.pdf*, 2014.

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

[5] R. Y. Lewis. Polya: a heuristic procedure for reasoning with real inequalities. *MS Thesis, Dept. of Philosophy, Carnegie Mellon University*, 2014.

[6] A. Mahboubi. Programming and certifying a cad algorithm inside the coq system. *Mathematics, Algorithms, Proofs*, 2006.

[7] D. Miller and G. Nadathur. *Programming with Higher-Order Logic*. Cambridge University Press, 2012.

[8] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions of Programming Languages and Systems*, 1:245–257, 1979.

Commonsense Reasoning meets Theorem Proving

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

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

Abstract

This paper describes the use of automated reasoning methods for tackling commonsense reasoning benchmarks. For this we use a benchmark suite introduced in the literature. Our goal is to use general purpose background knowledge without domain specific hand coding of axioms, such that the approach and the result can be used as well for other domains in mathematics and science.

Benchmarks for Commonsense Reasoning For a long time, no benchmarks in the field of commonsense reasoning were available and most approaches were tested only using small toy examples. Recently, this problem was remedied with the proposal of various sets of benchmark problems. There is the Winograd Schema Challenge [6] whose problems have a clear focus on natural language processing whereas background knowledge has an inferior standing. Another example is the Choice Of Plausible Alternatives (COPA) challenge[1] [10] consisting of 1000 problems equally split into a development and a test set. Each problem consists of a natural language sentence describing a scenario and a question. In addition to that two answers are provided in natural language. The task is to determine which one of these alternatives is the most plausible one. The following example presents a problem from this benchmark suite:

My body cast a shadow over the grass. What was the CAUSE of this?

1. The sun was rising.
2. The grass was cut.

Even though for the COPA challenge capabilities for handling natural language are necessary, background knowledge and commonsense reasoning skills are crucial to tackle these problems as well, making them very interesting to evaluate cognitive systems.

Another set of benchmarks is the Triangle-COPA challenge[2] [7]. This is a suite of one hundred logic-based commonsense reasoning problems which was developed specifically for the purpose of advancing new logical reasoning approaches. The structure of the problems is the same as in the COPA Challenge however the problems in the Triangle-COPA challenge are not only given in natural language but also in first-order logic. All previous systems tackling the COPA benchmarks focus on linguistic and statistical approaches by calculating correlational statistics on words. Until now only one logic based system is able to tackle the Triangle-COPA benchmarks: [7] use abduction together with a set of hand-coded axioms.

We refrain from using hand-coded knowledge and suggest to use knowledge bases containing commonsense knowledge like OpenCyc [5], Sumo [9], and Yago [11] together with a theorem prover instead.

Tackling Backround Knowledge When combining an example of the COPA benchmarks with background knowledge, several problems have to be solved: 1. The problems are given in natural language and have to be transformed into a logical representation. 2. The predicate symbols used in the formalization of the example are unlikely to coincide with the predicate symbols used in the background knowledge. 3. The background knowledge is too large to be considered as a whole. The first problem can be solved using the Boxer [1] system which is able to transform natural language into first-order logic formulae. We address the second problem by using WordNet [8] to find synonyms and hypernyms of the

*Work supported by DFG FU 263/15-1 'Ratiolog'.

[1] Available at http://people.ict.usc.edu/∼gordon/downloads/COPA-questions-dev.txt.

[2] Available at https://github.com/asgordon/TriangleCOPA/.

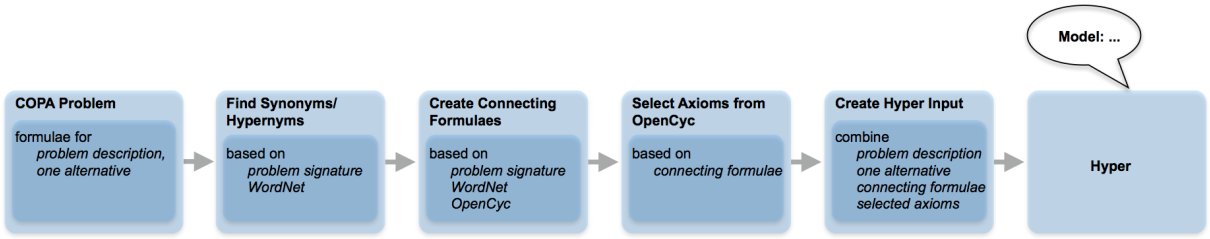

Figure 1: Workflow of the entire task. The starting formulae are generated by transforming the natural language problems of COPA into logic by using the Boxer system. I case of Triangle-COPA they are given already as the problem description

predicate symbols used in the formalization of the example. Note that the formalization of the example consists both of the formulae describing the situation as well as the formulae for one of the alternatives. In the next step, predicates symbols used in OpenCyc, which are similar to these synonyms and hypernyms are determined. With the help of this information, a connecting set of formulae is created. In this step, it is also necessary to adjust the arity of predicate symbols which is likely to differ since Boxer only creates formulae with unary or binary predicates and all predicates in Triangle-COPA contain an argument for so called eventualities.

The third problem is addressed using selection methods. For this, all predicate symbols occurring in the formalization of the example and in the connecting set of formulae are used. As selection methods, SInE as well as k-NN as they are implemented in the E.T. metasystem [4] come to use. Axioms resulting from this selection are combined with the connecting set of formulae and the formalization of the example. The result of this combination serves as input for a theorem prover, in our case the Hyper prover.

Figure 1 shows the entire workflow, which is necessary in order to transform the given benchmark problem into a theorem proving task. We did a very preliminary experiment to test this workflow. From the COPA benchmark set we selected 75 problems together with the respective two alternatives. Feeding these examples into the workflow resulted finally in 150 prove tasks for Hyper and we learned a lot — about problems which have to be solved. Hyper found 48 proofs and 71 models; the rest are time-outs. One problem we encountered is that some contradictions leading to a proof are introduced by selecting too general hypernyms from WordNet. E.g. the problem description of our example from the previous section as given by Boxer is

$$\begin{aligned}\exists A,B(\exists C,D,E,F(rover(D,B) \wedge \exists G(rtopic(G,A) \wedge arisingC(G)) \wedge rthat(B,C) \wedge \\ rpatient(D,E) \wedge ragent(D,F) \wedge vcast(D) \wedge nshadow(E) \wedge \\ nbody(F) \wedge rof(F,E) \wedge nperson(E)) \wedge nsun(A) \wedge ngrassC(B)).\end{aligned}$$

From WordNet the system extracted the information, that ‘individual’ is a hypernym of ‘shadow’ and ‘collection’ is a hypernym of ‘person’ leading to the two connecting formulae:

$$\forall X(nshadow(X) \rightarrow individual(X))$$
$$\forall X(nperson(X) \rightarrow collection(X)).$$

The selection from OpenCyc resulted among others in the axiom

$$\forall X \neg(collection(X) \wedge individual(X)).$$

These formulae together lead to a closed tableau because of a contradiction between a shadow which stems from a person, whereas WordNet gives that a shadow is an individual, a person is a collection

and together with the Cyc axiom we get the contradiction, which has nothing to do with the examined alternative that the sun is rising. In order to avoid this, we have to implement a kind of depth-bound for selecting hypernyms from WordNet in future versions.

Other contradictions stem directly from inconsistencies in the knowledge base used as source for background knowledge (in our case OpenCyc). E.g. the two formulae

$$\forall X\, speed(fqpquantityfnspeed(X))$$
$$\forall X \neg speed(X)$$

were selected immediately leading to a contradiction which again does not have to do anything with the examined alternative about the sun rising. This illustrates that we have to find a way to deal with inconsistent background knowledge.

Ranking of Proofs and Proof Attempts When using the Hyper prover within the Deep Question Answering system LogAnswer ([2]) we already had to tackle the problem that the prover nearly never found a proof of the given problem. Instead we had to delete some subgoals of the proof-task, which could not be solved within a given time-bound — we called this relaxation. In order to find a best answer of the system we had to compare several proofs, or better proof attempts, because of the relaxations. For this ranking we used machine learning to find the best proof rsp. answer. We are planning to use a similar approach for the afore-mentioned benchmarks. The situation is such, that we have a problem description P, in our COPA example above the logical representation of 'My body cast a shadow over the grass.' together with two possible answers $E1$ and $E2$.

We try to solve the two proof tasks $P \cup BG \models E1$ and $P \cup BG \models E2$, where BG is the background knowledege as discussed in the previous section. After timeouts and relaxations we get for the two tasks two tableaux, which may contain open and closed branches. The closed branches are parts of a proof and the open branches either are a model (and hence no closed tableau exists) or they are only open because of a time-out for this branch.

The task is to find out which of the two explanations $E1$ and $E2$ are 'closer' to be a logical consequence. In the LogAnswer system we gave the two answers to humans to decide this and we then used this information to train a machine learning system. For the scenario of the COPA and Triangle-COPA benchmarks we designed a preliminary study, which aims at using the information about the two tableaux created for $P \cup BG \models E1$ and $P \cup BG \models E2$ respectively together with information from formulae of the problem and the background $P \cup BG$ to generate examples for training.

We restricted our preliminary study to propositional logic and analyzed tableaux created by the Hyper prover for randomly created sets of clauses. For each pair of propositional logic variables p and q occurring in a clause set, we were interested in the question if p or q is 'closer' to a logical consequence. We reduced this question to a classification problem: for each pair of variables p and q, the task is to learn if $p < q$, $p > q$ or $p = q$, where $p < q$ means, that q is 'closer' to a logical consequence then p and $p = q$ means that p's and q's 'closeness' to a logical consequence is equal. Consider the following set of clauses:

$$p0$$
$$p4 \rightarrow p2 \vee p3 \vee p7$$
$$p0 \rightarrow p4$$
$$p3 \wedge p5 \rightarrow p6$$
$$p3 \wedge p5 \wedge p8 \rightarrow p1$$
$$p2 \rightarrow \bot$$

$<$ (predicted)	$>$ (predicted)	$=$ (predicted)	
$<$ (actual)	5,595	78	33
$>$ (actual)	90	5,589	27
$=$ (actual)	9	5	772

Table 1: Confusion matrix for classifying the test set with the learnt decision tree. The numbers occurring in the diagonal represent all correctly classified instances, whereas the other cells list incorrectly classified instances.

Clearly, $p0$ and $p4$ are logical consequences of this clause set. Therefore $p0 = p4$ and $p0 > q$ for all other variables q. On the other hand, from $p2$ it is possible to deduce a contradiction, which leads to $p2 < q$ for all other variables q. Comparing $p6$ and $p1$ is a little bit more complicated. Neither of these variables is a logical consequence. However assuming $p3$ and $p5$ to be true, would allow to deduce $p6$ but not $p1$. In oder to deduce $p1$ it is necessary to assume not only $p3$ and $p5$ to be true but also $p8$. Therefore $p1 < p6$.

To use machine learning techniques to classify these kind of examples, we represent each pair of variables (p,q) as an instance of the trainings examples and we provide the information, which of the three relations $<,>,=$ is right for p and q. Each of these instances contains 22 attributes. Some of these attributes represent information on the clause set like the proportion of clauses with p or q in the head as well as rudimentary dependencies between the variables in the clause set. In addition to that, we determine attributes representing information on the hypertableau for the set of clauses like the number of occurrences of p and q in open branches. Furthermore, we determine an attribute mimicking some aspects of abduction by estimating the number of variables which have to be assumed to be true in order to deduce p or q respectively, which allows us to perform comparisons like the one between $p1$ and $p6$ in the above example. Of course we also took into account whether one of the two variables is indeed a logical consequence.

For the first experiments, 1,000 sets of clauses each consisting of about 10 clauses and containing about 12 variables were randomly generated. These sets of clauses were analyzed and used to create a training set. For each pair of variables occurring in one of the clause sets, an instance was generated. All in all this led to 123,246 examples for training purposes. In these examples, the classes $<$ and $>$ each consists of 57,983 examples and the class $=$ of 7,280 examples. We used several classifiers implemented in the Weka [3] system. For example we learnt a decision tree from these training examples. We tested this decision tree with a test set which was generated from 100 randomly generated sets of clauses different from the clause sets used for the training examples. This resulted in a test set consisting of 12,198 instances. The constructed decision tree correctly classified 98.02 % instances of our test set. Table 1 provides information on correctly and incorrectly classified instances of the different classes.

We are aware that automatically classifying the test set might introduce errors into the test set and therefore tampers the results. Since it is very labor-intensive to manually generate test data, we only classified all pairs of two clause sets manually. For this much smaller test set, depending on the classifier used, we reached percentages of correctly classified instances of up to 80 %. In the next step, we are planning to expand our experiments to clause sets given in first-order logic.

References

[1] J. R. Curran, S. Clark, and J. Bos. Linguistically motivated large-scale NLP with C&C and Boxer. In *Proceedings of the ACL 2007 Demo and Poster Sessions*, pages 33–36, Prague, Czech Republic, 2007.

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

[3] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[4] C. Kaliszyk, S. Schulz, J. Urban, and J. Vyskocil. System description: E.T. 0.1. In A. P. Felty and A. Middeldorp, editors, *Proceedings of CADE-25, Berlin, Germany, 2015*, volume 9195 of *LNCS*. Springer, 2015.

[5] D. B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[6] H. J. Levesque. The Winograd Schema Challenge. In *Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11-06, Stanford, California, USA, March 21-23, 2011*. AAAI, 2011.

[7] N. Maslan, M. Roemmele, and A. S. Gordon. One hundred challenge problems for logical formalizations of commonsense psychology. In *Twelfth International Symposium on Logical Formalizations of Commonsense Reasoning, Stanford, CA*, 2015.

[8] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, 1995.

[9] I. Niles and A. Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9. ACM, 2001.

[10] M. Roemmele, C. A. Bejan, and A. S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, 2011.

[11] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from Wikipedia and WordNet. *Web Semant.*, 6(3):203–217, Sept. 2008.

Proof Engineering of Higher Order Logic: Collaboration, Transformation, Checking and Retrieval

Shuai Wang*

ILLC, University of Amsterdam
The Netherlands
shuai.wang@student.uva.nl

Abstract

Higher Order Logic has been used in formal mathematics, software verification and hardware verification over the past decades. Recent developments made sharing proofs between some theorem provers possible. This paper first gives an introduction and an overview of related recent advances, followed by the proof checking benchmarks of a proof sharing repository, namely OpenTheory. Finally, we introduce ProofCloud, the first proof retrieval engine for higher order logic proofs.

1 Introduction

Higher order logic is also known as simple type theory. It is an extension of simply typed λ-calculus with additional axioms and inference rules [4]. Interactive Theorem Provers (ITPs) of higher order logic have been playing an important role in software verification, hardware verification and formal mathematics. Among them, the HOL family consists of HOL Light [8], HOL4 [13] and ProofPower [11], etc. These ITPs implemented the same higher order logic, namely Church's simple type theory [9]. However, they each contain significant theory formalizations that are not accessible to each other. HOL Light [8] has a formalization of complex analyses while HOL4 has a formalization of probability theory [13]. In contrast, ProofPower has a formalization of the Z specification language [11]. Proof libraries from one ITP may contribute to the proof automation of another ITP [7, 5]. For the sake of proof sharing between ITPs with different theories, OpenTheory [9] was developed as a cross-platform proof package manager for proofs in the HOL family. It consists of a standard library and many proof packages including lists, natural numbers, functions, etc. Taking advantage of these packages, OpenTheory has inspired further exploration of theory management [10, 6] as well as the development of several projects [1, 2, 14].
ITPs may not be bug-free and may lead to errors in generated proofs while not being apparent within the proof systems themselves. Together with possible mistakes in the process of importing and exporting, OpenTheory is not guaranteed to be reliable. Even worse, proofs can be huge, making them difficult or even impossible to be checked by hand. The demand of reliability of such systems leads to the necessity of proof checking, especially independently from the ITPs involved. Taking advantage of the similarity of the logic and design between these systems, OpenTheory [9] has developed a standard format for serialising proofs [9]. One way to verify these proofs (also known as proof articles) is to export them to the OpenTheory format followed by the proof checking process by Dedukti [12].

*The author was supported by the MPRI-INRIA scholarship.

2 Proof Transformation and Checking with Holide and Dedukti

Dedukti [12] is a logical framework based on $\lambda\Pi$-calculus Modulo [3] for defining logics and checking proofs. It has been widely used as a universal proof checker for ITPs including Coq, Matita, HOL, FoCaLiZe, etc[1]. To transform higher order logic proofs from the OpenTheory format to the Dedukti format, we need a translator, namely Holide [1]. Holide uses a modular translation of higher order logic which makes the translation possible to extend [1]. Recent updates of the OpenTheory resulted in an upgrade of the Holide program. More specifically, OpenTheory expanded its logic kernel and added some new inference rules and some other features[2]. Holide is therefore upgraded to version 2 to capture corresponding features and the new format.

3 Proof Retrieval with ProofCloud

ProofCloud[3] is a search engine of higher order logic proofs customised from Swiftype. While OpenTheory groups proofs up, ProofCloud unpacks them and displays a description of each package (on the index pages) as well as all the theorems contained individually. It has been populated by over 1,800 proofs from 6 packages of OpenTheory. As far as the author knows, it is the only online proof search engine of its kind. ProofCloud presents also the proof checking results by Holide and Dedukti. Further more, it tracks the origin of classicism. With the ability to classify classical proofs, it illustrates the axioms and constructive/classical lemmas involved for each theorem as well as the amount of constructive/classical proofs for the package page.

4 Evaluation and Conclusion

The standard library in OpenTheory grouped theorems into packages, including the standard library and theorems of booleans, sets, lists, etc. Previous work of Holide has checked only the standard library. Following the upgrade of Holide, for the first time, Holide and Dedukti performed proof checking on all packages of OpenTheory. Table 1 illustrates the size of OpenTheory proof article files and the time taken for translation as well as the size of translated files and the time taken for proof checking by Dedukti. Both article files and Dedukti files are compressed by *gzip* to reduce the effect of syntax formatting and white-space. These benchmarks were generated on a 64-bit Intel Core i5-4590 CPU @3.30GHz $\times 4$ machine with 3.8GB RAM. This provides evidence that OpenTheory is a reliable platform for higher order logic proofs and validated the upgrade of Holide. In addition, the structural proof analyses by ProofCloud shows that the proportion of constructive theorems varies from package to package. For example, the *natural-divides* package has only 10 constructive theorems out of 136 theorems, making only 7.35% proofs constructive in the package. Apart from maintaining Holide, future work also includes adding more packages to ProofCloud and further improve the user interface and the searching accuracy.

[1] `https://www.rocq.inria.fr/deducteam/software.html`

[2] More details between version 5 and version 6 are included in the announcement: `http://www.gilith.com/pipermail/opentheory-users/2014-December/000461.html`

[3] `airobert.github.io/proofcloud/`

Package	Proof Translation with Holide		Proof Checking with Dedukti	
	Size (KB)	Time (s)	Size (KB)	Time (s)
base	1,194	19.42	4,440	9.74
cl	313	5.56	1,219	2.46
empty	0	0.00	0	0.00
gfp	112	1.35	375	0.65
lazy-list	1,391	31.78	5,717	13.11
modular	37	0.37	111	0.17
natural-bits	132	1.39	419	0.68
natural-divides	157	1.94	566	0.99
natural-fibonacci	108	1.24	354	0.60
natural-prime	116	1.34	388	0.65
parser	204	3.15	776	1.69
probability	23	0.23	69	0.11
stream	63	0.73	211	0.38
word10	71	0.62	216	0.29
word12	72	0.75	220	0.35
word16	107	0.77	364	0.36
word5	64	1.56	192	0.72
Total	**4,377**	**72.21**	**15,637**	**32.95**

Table 1: Benchmarks of OpenTheory with Holide and Dedukti

References

[1] Ali Assaf and Guillaume Burel. Translating HOL to dedukti. In *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, pages 74–88, 2015.

[2] Ali Assaf and Raphaël Cauderlier. Mixing HOL and coq in dedukti (extended abstract). In *Proceedings Fourth Workshop on Proof eXchange for Theorem Proving, PxTP 2015, Berlin, Germany, August 2-3, 2015.*, pages 89–96, 2015.

[3] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In *Typed lambda calculi and applications*, pages 102–117. Springer, 2007.

[4] William M Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.

[5] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Proc. of the 7th Conference on Intelligent Computer Mathematics (CICM'14)*, volume 8543 of *LNCS*, pages 267–281. Springer Verlag, 2014.

[6] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across hol libraries. In *Intelligent Computer Mathematics*, pages 267–281. Springer, 2014.

[7] Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'15)*, LNCS, 2015. to appear.

[8] John Harrison. Hol light: An overview. In *Theorem Proving in Higher Order Logics*, pages 60–66. Springer, 2009.

[9] Joe Hurd. The OpenTheory standard theory library. pages 177–191.

[10] Cezary Kaliszyk and Alexander Krauss. Scalable lcf-style proof translation. In *Interactive Theorem Proving*, pages 51–66. Springer, 2013.

[11] Marcel Oliveira, Ana Cavalcanti, and Jim Woodcock. Unifying theories in proofpower-z. In *Unifying Theories of Programming*, pages 123–140. Springer, 2006.

[12] Ronan Saillard. Dedukti: a universal proof checker. In *Foundation of Mathematics for Computer-Aided Formalization Workshop*, 2013.

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

[14] Makarius Wenzel. Interactive theorem provers from the perspective of isabelle/isar.