Learning-Assisted Reasoning within Proof Assistants

cumulative dissertation

by

Thibault Gauthier

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements for the degree of academic degree

advisors: Ass.-Prof. PD Dr. Cezary Kaliszyk

Innsbruck, 30 April 2019

universität
innsbruck

cumulative dissertation

Learning-Assisted Reasoning within Proof Assistants

Thibault Gauthier (1415002)
email@thibaultgauthier.fr

30 April 2019

advisors: Ass.-Prof. PD Dr. Cezary Kaliszyk

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.
Die vorliegende Arbeit wurde bisher in gleicher oder ähnlicher Form noch nicht als Magister-/Master-/Diplomarbeit/Dissertation eingereicht.

Datum	Unterschrift

Abstract

One of the aims of artificial intelligence is to be able to solve problems through learning and automated reasoning. Famous examples of intelligent systems can be found in molecular chemistry and games such as chess and Go. The current most successful approaches in these domains combine statistical models trained by machine learning methods with search strategies. However, more abstract problems, generally out of reach of current methods, constructed from abstract principles such as induction appear commonly in mathematics. And proving new theorems often requires to exploit patterns hidden in complex structures.

The aim of this thesis is to improve automated reasoning techniques in mathematics. This would help to reduce the time needed for formally proving complex theorems such as the Kepler conjecture. In order for our system to learn from a large number of examples, we align the formal libraries of interactive theorem provers (ITPs) by recognizing similar concepts. The shared knowledge is processed by a statistical model which infers relations between mathematical objects. We exploit those links to produce better strategies for proving theorems.

In practice, our project combines ITP knowledge with the search power of automated theorem provers (ATPs). Therefore, if a mathematician states a conjecture in an ITP, our system understands through the statistical model the relation of the conjecture to previously formalized knowledge. It selects theorems relevant for proving the conjecture, simplifies the proof by conjecturing intermediate lemmas and may also guide the proof search by choosing suitable reasoning methods. If successful, it produces a computer-verified proof of the conjecture.

Abstract in German

Eines der Ziele künstlicher Intelligenz ist die Fähigkeit, Probleme durch Lernen und automatische Deduktion zu lösen. Bekannte Beispiele intelligenter Systeme sind in Chemie und Spielen wie Schach und Go zu finden. Die aktuell erfolgreichsten Ansätze in diesen Bereichen kombinieren Suchstrategien mit von maschinellen Lernmethoden gelernten statistischen Modellen. Allerdings involvieren mathematische Probleme abstraktere Prinzipien wie Induktion, die automatisierte logische Deduktion erschweren. Der Beweis neuer Theoreme erfordert häufig die Ausnutzung von Mustern, die in komplexen Strukturen eingebettet sind.

Das Ziel unseres Projektes ist die Verbesserung automatischer Beweissuchmethoden in der Mathematik. Diese ermöglicht die Reduktion der Zeit, die zum formalen Beweisen komplexer Theoreme wie der Kepler'schen Vermutung benötigt wird. Um unser System von einer großen Menge an Beispielen lernen zu lassen, identifizieren wir ähnliche Konzepte in verschiedenen formellen Bibliotheken von interaktiven Theorembeweisern (ITP). Das kombinierte Wissen wird von einem statistischen Modell verarbeitet, welches Beziehungen zwischen mathematischen Objekten ableitet. Wir nutzen diese Beziehungen aus, um bessere Strategien zum Beweis von Theoremen zu erzeugen.

Unser System kombiniert Wissen von ITPs mit der Suchfähigkeit von automatischen Theorembeweisern (ATPs). Dies erlaubt es, die Beziehung zwischen neuen Vermutungen und vorhergegangenem formalisierten Wissen mittels des statistischen Modells zu verstehen. Unser System wählt für den Beweis einer Vermutung relevante Theoreme aus, vereinfacht den Beweis durch die Annahme von Lemmata und lenkt die Beweissuche durch den Vorschlag von erfolgversprechenden Beweisschritten. Wenn das System erfolgreich ist, erzeugt es einen automatisch verifizierten Beweis der Vermutung.

Abstract in French

Un des buts de la recherche en intelligence artificielle, c'est de pouvoir résoudre des problèmes par l'apprentissage et la réflexion automatique. Des célèbres exemples de systèmes intelligents peuvent être trouvés dans la chimie moléculaire et les jeux comme les échecs et le Go. L'approche qui rencontre le plus de succès actuellement dans ces domaines combine des modèles statistiques entrainés par des méthodes d'apprentissage machine avec des stratégies de recherche. Cependant, des problèmes plus abstrait, globalement hors de portée des méthodes actuelles, construit par des principes abstraits tel que l'induction apparaissent souvent mathématiques. Et démontrer de nouveaux théorèmes nécessite d'exploiter des schémas cachés dans des structures complexes.

Le but de cette thèse est d'améliorer le raisonnement automatique en mathématiques. Cela aiderait à réduire le temps nécessaire à la démonstration formelle de théorèmes difficiles tels que la conjecture de Kepler. Pour que notre système puisse apprendre d'un grand nombre d'exemples, nous alignons les libraries formelles des assistants de preuve par la reconnaissance de concepts similaires. Les connaissances partagées sont analysées par un modèle statistique ce qui permet d'inférer des relations entre les objets mathématiques. Nous exploitons ces liens pour créer des stratégies plus efficaces pour la preuve de théorèmes.

En pratique, notre projet combine les connaissances des assistants de preuve et la puissance de recherche des prouveurs automatiques. Ainsi, quand un mathématicien propose une conjecture dans un assitant de preuve, notre système comprend à travers son modèle statistique la relation entre la conjecture et les connaissances précédemment formalisées. Il sélectionne des théorèmes adaptés pour prouver la conjecture, puis il peut simplifier la preuve en proposant des lemmes intermédiaires et enfin il guide la recherche de preuves en choisissant des méthodes de raisonnement efficaces. En cas de succès, il produit une preuve vérifiée de la conjecture.

Acknowledgements

I would like to thank everyone that supported me during these four years of research. Most notably, my supervisor Ass.-Prof. PD Dr. Cezary Kaliszyk is the guide that lead me through the hardship of publications. Giving advice in a timely manner and readily available for discussion, Cezary is the kind of supervisor that knows how to let the ideas of a candidate flourish. Secondly, my appreciation goes to Josef Urban for all the informal chats we shared on proof automation. His excitement about the subject is contagious. Many thanks go to my second and third supervisors Univ.-Prof. Dr. Aart Middeldorp and assoz. Prof. Dr. René Thiemann. Both contribute to the friendly and enjoyable environment of the Computational Logic group. To Michael Färber with whom I share a laugh or two during lunches: un grand merci! As a PhD candidate himself, he understands the ups and downs one can go through during this research endeavor. Last but not least, I am grateful to my parents Claire and Jean-Michel, who have provided me with moral and emotional support through my entire life.

Contents

Chapter 1

Introduction

Natural sciences are creating models of the real world and checking them against experimental evidence. Those models are made of abstract objects and mathematicians study the relation between these abstractions. Moreover, the truths of mathematics are absolute and a theorem is an abstract property that has been verified beyond doubt. The process by which this verification is performed in mathematics is called proving. In contrast, a theory in natural sciences can never be fully verified because the real world may not always follow the mathematical model of the theory. Mathematicians give precise definitions for theoretical objects such as numbers, functions and geometrical objects. They prove indisputable truths about them like the Pythagorean theorem that holds for every right triangle. Yet, the process by which we can guarantee that the proof is correct was not fully understood before the 20th century. Therefore, in an effort to make the proofs of mathematical statements impermeable to skeptics, logicians worked toward the creation of a logical language that is unambiguous and that could be used to form any kind of arguments. A successful foundation for mathematics was laid by Russell and Whitehead in Principia Mathematica [WR27] and published in three volumes in 1910, 1912 and 1913. With the advent of computer age, different systems were implemented in the 1980s in interactive theorem provers (ITPs). Nowadays, there are dozens of different ITPs that build upon two competing logical foundations: set theory and type theory.

Thanks to the computerization of mathematics, ITPs are able to automatically check if each step of a proof follows the rules given by their formal system. If the verification succeeds, it guarantees the truth of the proven formula. But most of the time, a mathematician omits intermediate steps needed for a complete formal proof [Wie06]. Indeed, providing these steps would be very tedious and they would obfuscate the core ideas of the proof to the reader. There are two possible solutions to this problem, either an expert mathematician provides the missing intermediate steps or a computer scientist develops proof automation techniques that are powerful enough to propose a valid sequence of reasoning steps to bridge the gaps.

The work conducted here follows the second approach and improves general purpose proof automation for developments of formal libraries in ITPs. In this introduction, I give an overview of the systems that I build upon in this thesis.

1.1 Interactive Theorem Provers

ITPs, also called proof assistants, were created to supply rigorous checking of formal proofs, freeing them of any human errors. The need for automated certification is even more obvious when a proof is discovered with the help of a computer and thus is too long to be verified manually. For instance, the proofs of the 4-color theorem and the Kepler conjecture are computer-assisted proofs, which were never fully manually checked, but formalized [Gon08, HHM+10] in Coq [Ber08] and HOL Light [Har09] respectively. ITPs are also well-suited to verify the properties of critical software, such as the kernel of an operating system [KAE+10] whose safety was certified in Isabelle/HOL [WPN08].

The library of each ITP contains mathematical objects and theorems from multiple theories. For example, the Mizar [GKN10] Mathematical Library, which is one of the largest proof libraries, contains more than 12,000 formal definitions of mathematical objects and more than 65,000 human-named theorems (also called top-level theorems). The logic of each ITP is adapted to the need of its users. It can range from first-order set theory preferred by mathematicians to type theory more adequate for logicians and computer scientists. Defined by each logic, there is a basic set of rules that can be applied to create new theorems from the axioms.

During the process of formalizing a proof in an ITP, the user must provide every basic rule needed to complete the proof. The user can do so manually or with the help of automation. The ITP automatically checks that they are correctly applied according to its logical system. The process of manually finding a proof composed of basic rules is laborious. Therefore, ITPs provide a way to combine these rules into strategies, effectively defining a language in which you can create new more complex proof methods from the basic ones. Two examples of strategies found in most ITPs are a rewrite tactic that simplifies a goal based on a set of directed equalities and an arithmetic tactic that solves sets of linear equalities on integers (omega test, Cooper method, etc.). More examples are given in Section 1.2.

HOL4 In the thesis, I rely on many different ITPs for my developments. The ITP in which I have the most expertise is HOL4 [SN08] and many of my projects depend on this ITP. HOL4 is a direct descendant of the first interactive theorem provers based on a LCF-style kernel characterized by the presence of a reserved type for theorems. The size of its kernel which consists the trusted base of the system is small which makes HOL4 proofs highly reliable. Its expressive logic, which is a higher-order logic, allows the user to state theorems in a natural way. The definition of new concepts (types and constants) is facilitated by the presence of many constructors for these objects such as definitions by induction. Moreover, thanks to the compositional nature of its proof language, it is easy for a user to implement new tactics (or proof methods). After review by a developer, these new tactics are included in the HOL4 main development, thus the assistance that HOL4 provides increases at each new release.

HOL4 contains a standard library of about ten thousand theorems and many contributions. The most prominent one is the CakeML [KMNO14] project. Its developers created a compiler for the SML-like language CakeML and produced a verified x86-64

implementation of a read-eval-print loop for CakeML. One of their goal is to develop ITPs that can check their own code, adding another layer of trust to proof systems. The HOL4 library is also developed for pure mathematical domains such as arithmetic, complex analysis and probability. Example 1.1 is a formally proven theorem in the arithmetic theory.

Example 1.1. (HOL4 statement on the infiniteness of prime numbers)

$$\forall n.\ \exists p.\ n < p \wedge prime\ p$$

where *prime* is a predicate testing the primality of a number p.

1.2 ITP Proof Automation

Tactics are one of the important ways to accomplish automation in modern proof assistants. A tactic is a function that takes a mathematical formula called a goal and returns a list of goals. If this list is proven then the tactic guarantees that the original goal is proven too. Contrarily to the standard way of reasoning from the axioms and definitions to the conjecture, producing formal proofs with the help of tactics requires the user to reason in a backward manner from the conjecture (or goal) which is less intuitive but more effective in practice. Next, I give examples of the most common tactics in ITPs and reference them by their HOL4 names. They are ordered from the lowest degree of automation (respectively highest degree of control) to the highest (respectively lowest).

- `SPEC_TAC`: This function enables the user to instantiate an existential quantifiers by a witness of its choice.
- `INDUCT_TAC`: Reasoning by induction is necessary to prove properties about structured types such as natural numbers, ordinals, lists, trees and graphs.
- `RW_TAC`: This tactic belongs to the family of simplification functions. They are probably the most commonly used. Simplifying an expression is helpful in almost every domain of mathematics whether it be abstract models, calculus, algebra or analysis.
- `COOPER_TAC`: This is a complete decision procedure for linear integer arithmetic.
- `METIS_TAC`: This is an automated theorem prover (see Section 1.3) implemented as a tactic. Given a goal and a list of theorems (also called premises in this context), it searches for a proof. It is complete for first-order logic but most translations from higher-order logic introduce incompleteness. This means that its search algorithm is able to find a proof for any theoretically provable formula as long as the necessary mathematical facts are selected. However in practice, if the tactic is given too many premises or if the proof is too large, the search will most likely time out.

Since tactics are functions, it is possible to compose their effect thanks to higher-order functions called tacticals. The simplest tactical `THEN` takes two tactics t_2, t_1 and a goal g as arguments and calls the tactic t_2 on each of the goals returned by the application of t_1 on g.

Example 1.2. (Formal proof of the statement on the infiniteness of prime numbers)

```
CCONTR_TAC THEN
‘∃n. ∀p. n < p ⇒ ¬(prime p)‘ by METIS_TAC[] THEN
‘~(FACT n + 1 = 1)‘          by RW_TAC arith_ss
                                  [FACT_LESS,NOT_ZERO_LT_ZERO] THEN
‘∃p. prime p ∧
     divides p (FACT n + 1)‘ by METIS_TAC [PRIME_FACTOR] THEN
‘0 < p‘                      by METIS_TAC [PRIME_POS] THEN
‘p ≤ n‘                      by METIS_TAC [NOT_LESS] THEN
‘divides p (FACT n)‘         by METIS_TAC [LEQ_DIVIDES_FACT] THEN
‘divides p 1‘                by METIS_TAC [DIVIDES_ADD_2] THEN
‘p = 1‘                      by METIS_TAC [DIVIDES_ONE] THEN
‘¬(prime p)‘                 by METIS_TAC [NOT_PRIME_1]
```

This proof starts by a classical contradiction step which is followed by a formalized version of Euclid's argument where each step is justified by a tactic.

1.3 ATPs

Automated theorem provers (ATPs) concentrate solely on the task of searching for a proof of a formal statement. In particular, they do not manage and organize any mathematical knowledge like ITPs do. That is why an ATP usually operates within first-order logic where it is easier to implement a search algorithm that respects among others the desirable properties of soundness and completeness. ATPs come with a lot of bells and whistles and their parameters can be manually or automatically tuned for particular sets of problems. A set of parameters is called an ATP strategy. Strategy scheduling is the art of sharing the allocated time between the different strategies to solve a goal. On a usual set of problems, these schedules are much stronger than the ATP default strategy. ATPs are very efficient general purpose search procedures and they contain many optimizations such as term indexing. However, since their code base is large and intricate, it is difficult to eliminate undesirable effects and some of those may affect the soundness of the prover. To counter that, ATPs often produce proof certificates that can be checked by an ITP.

E prover This ATP is an open source prover that performs very well at the annual CASC competition [Sut14]. It belongs to the family of superposition provers which has dominated the world of ATPs in the last decade. New strategies for E prover [Sch02] are continuously created either by the manual addition of new parameters or by the automatic discovery of new strategies. Because of its strength and frequent updates, I use E prover in almost all the experiments in this thesis.

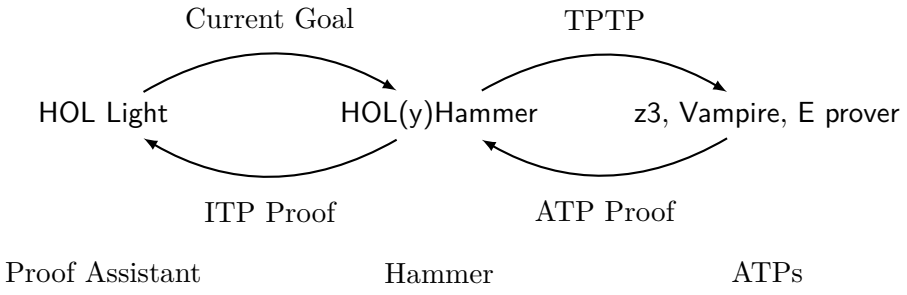


Figure 1.1: HOL(y)Hammer interaction loop

Satisfiability Modulo Theories (SMT) Solvers Compared to a pure ATP such as E prover, a SMT solver relies on additional axiomatic knowledge and decision procedures for a set of theories. The most common theory objects supported by SMT solvers are integers, reals, bit vectors, lists and arrays. The most prominent SMT solvers are z3 [DMB08] and CVC4 [BCD+11]. In this thesis, I also conduct experiments with Beagle [BW13] because it supports the TPTP (TFA) format [SSCB12].

1.4 Hammers

Hammers are the strongest kind of proof automation developed so far in ITPs. They speed up the development of ITP libraries by discharging a large number of goals automatically. Compared with other tactics, these proof methods have the following characteristics: they are general purpose, they demand no user interaction and they are very effective as demonstrated by experiments yielding a 40 percent success rate [BKPU16] on average on top-level theorems. They accomplish this feat by calling many ATPs in parallel on an ITP goal. The hammers acts as an interface between the two kind of systems resolving two expected issues that arise naturally in this setup: the discrepancy between the logics and the lack of theory knowledge in ATPs.

In the following, I present the internals of a hammer. The process of solving an ITP goal via a hammer can be divided in three phases: premise selection, translation to ATPs and reconstruction. The interaction between the three parties (ITP, hammer, ATPs) is summarized in Figure 1.1. These three steps are completely automated and invisible to the ITP user, which can then build upon the newly certified theorem to perform the next formalization step.

Premise Selection Given an ITP library and a goal, the premise selection algorithms select a limited set of suitable lemmas that could help to prove the goal. This will give to the ATP the knowledge necessary to solve the goal. The number of selected lemmas has to be carefully determined. Too few and the premise selection algorithm might not be accurate enough to choose the right lemmas. Too many and the ATP search may have too many possible proofs to explore. In the machine-learning based premise

selection algorithms, the relevance of a lemma is determined by its syntactic and semantic similarity with the goal. However, only syntactic features are considered in this thesis. Furthermore, the selection considers also theorems that are directly needed to prove a theorem similar to the goal. Combining proof dependencies and syntactical similarity, the predictor gives an evaluation of the relevance of a theorem relative to the goal.

Translation from an ITP to an ATP Generally, the ITP problems (goal + predictions) contain constructions from an expressive logic that have no direct equivalent in the first-order logic of ATPs usually written in the widespread TPTP format [Sut09]. Therefore, a translation from the ITP logic to the ATP is necessary. As an example, I discuss some of the required steps for a translation from the higher-order logic of HOL Light to untyped first-order logic. First, an encoding of polymorphic types inside first-order terms is required. Second, λ-abstractions should be eliminated. This can usually be achieved by performing λ-lifting or by expressing the λ-expressions in terms of combinators. Last, no function should appear as argument of another function or under a quantifier in first-order logic. This constraint can be fulfilled by the introduction of apply operators. Each translation can be judged in terms of soundness, completeness, efficiency, scalability and readability. Beyond hammers, why3 [FP13] is a platform for deductive program verification that offers many such translations.

Reconstruction The ATP proof of the conjecture from the top theorems, if found, has to be transformed into an ITP proof to be certified. This transformation is called reconstruction. One approach is to extract the theorems used by the ATP proof and reprove the conjecture from these theorems using internal ATPs. To increase the success rate of the reconstruction mechanism, a step by step translation of the ATP proof is necessary [KU13a, BBF+16].

1.4.1 Comparison of Existing Implementations

There are currently four implementations:

- Sledgehammer [PB10] for Isabelle/HOL,
- HOL(y)Hammer [KU14] for HOL Light.
  Additional support for HOL4 [GK15a] is provided by this thesis.
- MizAR [KU15d] for Mizar,
- CoqHammer [CK18] for Coq.

Many of the different techniques for premise selection, translation and proof reconstruction were transferred from one system to the other. But I would like to highlight what research area is investigated in each project. The Sledgehammer development team experiments with many different translations and optimized each of the steps. Its integration with Isabelle/HOL makes it the most popular hammer so far. In contrast, HOL(y)Hammer and MizAR developers explore ways of making premise selection stronger. They re-implement

existing machine learning techniques, such as k-nearest neighbor and naive Bayes, and adapt those techniques to work on sets of mathematical formulas. Notably, they test different sets of features for these learning models. Moreover, the Mizar standard library and the Flyspeck [HHM+10] project in HOL Light provide a large enough data collection for training predictors. Finally, CoqHammer is the most recent development and it proposes solutions to the additional challenges posed by the intuitionistic logic and dependent types of Coq during premise selection, translation and reconstruction.

1.5 Interoperability

With the diversity of interactive theorems provers [HUW14], the lack of interoperability is a growing issue. Formalized proofs originating from one prover are hardly reusable in a different one. To solve this issue, two avenues have been considered: translations and frameworks.

Bridges between ITPs A number of translations between formal mathematical libraries were developed to bridge the gap. All these translation rely on a mapping of concepts that ensures that each concept is translated to its equivalent in the other library. All these concept maps have been found manually so far.

The first translation that introduced maps between concepts was the one of Obua and Skalberg [OS06]. Due to the complexity of finding such existing concepts and specifying the theorems which do not need to be translated, Obua and Skalberg were able to map only a small number of concepts like booleans and natural numbers, leaving integers or real numbers as future work.

The translation of Keller and Werner [KW10] was the first one, which was able to map concepts between systems based on different foundations. The translation from HOL Light to Coq proceeds in two phases. First, the HOL proofs are imported as a defined structure. Second, using the *reflection* mechanism, native Coq properties are built. It is the second phase that allows mapping the HOL concepts like natural numbers to the Coq standard library type `N`.

The translation that maps so far the biggest number of concepts has been done by Kaliszyk [KK13]. The translation process consists of three phases, an exporting phase, offline processing and an import phase. The offline processing provides a verification of the (manually defined) set of maps and checks that all the needed theorems will be either skipped or mapped. This allows to quickly add mappings without the expensive step of performing the actual proof translation, and in turn allows for mapping 70 HOL Light concepts to their corresponding Isabelle/HOL counterparts.

Frameworks Another approach is to create a framework where all theorems can be mapped to. The disadvantage is that tactics and tools available in the framework may be different from those present in an ITP. So an ITP developer would be reluctant to port her formal proofs to the framework in order to benefit from formalized theorems in another library.

Isabelle [WPN08] is one of the first attempts at creating a single logical framework under which developments over different logical foundations could be build. The most popular one Isabelle/HOL is based on higher-order logic. The second most developed one is Isabelle/ZF, which is based on first-order set theory. However, the possibility to transfer or port libraries from one logic to the other inside Isabelle remains limited [KS10, BJL06].

Hurd's OpenTheory [Hur11] aims to share specifications and proofs between different HOL systems (HOL4, HOL Light, Isabelle/HOL) by defining small theory packages. In order to write and read such theory packages by theorem prover implementations, a fixed set of concepts is defined that each prover can map to. This provides highest quality standards among the HOL systems, however since the procedure requires manual modifications to the sources and inspection of the libraries in order to find the mappings, so far only a small number of constants and types could be mapped.

The use of theory morphisms and concept mappings is one of the basic features of the MMT framework [Rab13]. Therefore, manual mappings between concepts across different logics are possible within this framework.

The Dedukti proof checker [DHK03], based on the $\lambda\Pi$-modulo calculus, can import and verify developments from Coq and HOL systems. An example Coq proof has been shown to be translatable to Dedukti and to be instantiated with HOL natural numbers [AC15]. One of the main challenges was to match the different typing levels of Coq and HOL into a common structure in the logic of Dedukti.

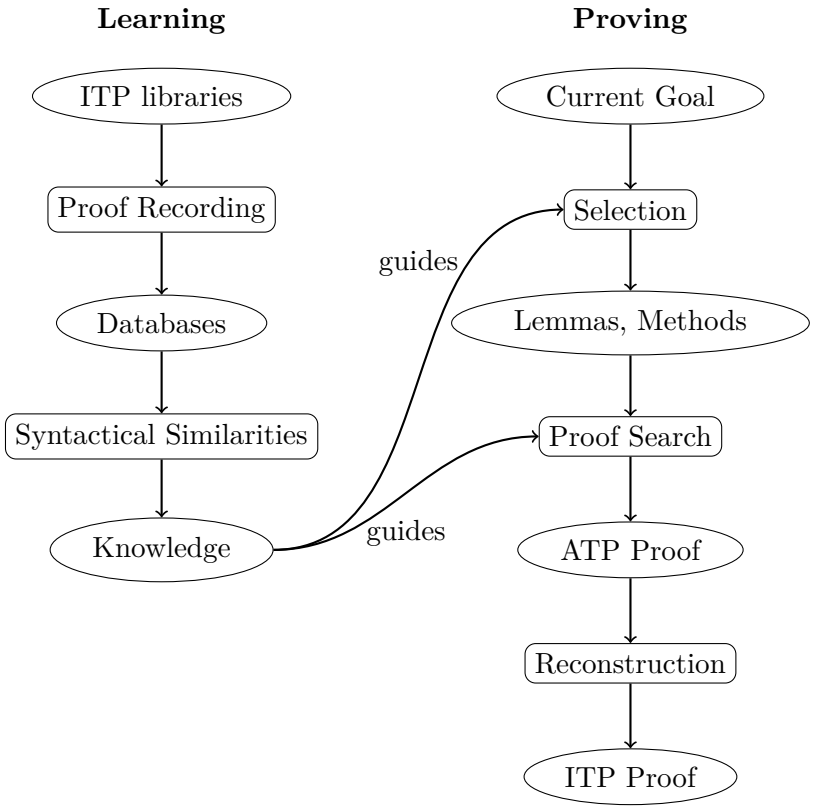
1.6 Aim of this Thesis

The aim of this project is to improve proof automation in ITPs. All the work in this thesis can be understood as a way to overcome the limitation of the current hammers by extending their capabilities. Indeed, I do not rely on any user interaction and intend to automatically find a proof for most of the top-level theorems of ITP's libraries.

I **record** the libraries of multiple ITPs based on different logics. I design an intermediate format onto which the translation and premise selection algorithms of HOL(y)Hammer can be plugged in. I make use of the kernel steps or proof objects to record information about the dependencies between theorems. I also investigate recording proofs at the tactical level for HOL4. I then **learn** from the collected data by recognizing patterns thanks to statistical machine learning techniques. The relation between these patterns allow our systems to get an intuition of the relation between syntactical representation of theorems and proof methods. This statistical analysis enables us to predict the **reasoning** methods and intermediate lemmas suitable for **proving** a particular conjecture. I connect external automation and re-use proof methods already present in ITPs. Their complementary strength covers a large range of ITP problems. To increase the readability of the proofs produced by our system, I rely on two complementary techniques: proof reconstruction and proof minimization. To finalize my project, I integrate the most successful developments inside ITPs creating **user-friendly** and efficient push-button automation. The collaboration between the different modules of the final system is

illustrated in Example 1.3. The details of the individual contributions will be discussed in the next chapter.

Example 1.3. (Flow chart showing the relation between modules of the learning-assisted reasoning system developed in this thesis)



Chapter 2

Contributions

I am the main contributor of all the papers included in this thesis. I performed almost all of the implementation, experiments and interpretation of the results associated with these first-authored papers, and wrote the majority of their content. Therefore, after the presentation of each paper in this chapter, with the assurance that I am the principal contributor, I will only mention my decisive insights and implementation ideas.

The papers corresponding to Sections 2.1–2.6 will be included in this thesis as published in Chapters 3–8 respectively. This series of papers follows the two main axes of our thesis in order to foster stronger proof automation: extending proof knowledge and developing reasoning abilities. I start by showing how the HOL(y)Hammer framework can be connected to HOL4 in Section 2.1. Then I investigate the benefits of a translation from higher-order to typed first-order logic with a manual mapping for arithmetic constants in Section 2.2. In an effort to improve interoperability, I present a way to discover concepts mappings (also called alignments) automatically. The alignment algorithm supports six ITPs based on diverse logical foundations and is presented in Section 2.3. In combination with premise selection, alignments found across ITPs can be used to share proof knowledge and thus provide better advice. This is demonstrated on the HOL4-HOL Light pair in Section 2.4. With the inclusion of imperfect alignments, and by matching a prover's library with itself, it is possible to create conjectures from inferred substitutions. This method is tested on the Mizar library in Section 2.5. All the methods so far were concerned with improving proof automation guidance by analyzing the syntactical structure of terms. As a final note for this thesis, I present a system in HOL4 that builds upon standard machine learning techniques used for premise selections and additionally propose tactics learned from human proof scripts. This technique is accompanied by an A*-search algorithm described in Section 2.6.

In addition, I discuss two contributions that naturally follow and extend some of the ideas laid out in this thesis. First, a description of the different type of alignments and a standard for sharing them across multiple systems, including informal ones, is introduced in Section 2.7. It is a step forward toward the creation of a library of all mathematical knowledge but is not included in this thesis as I am not the main developer of this project. Second, the exploration procedure of TacticToe presented in Section 2.6 is rewritten as a Monte Carlo Tree Search (MCTS) algorithm and the prediction method is extended to select tactic arguments independently. An overview of the ameliorations provided in this additional contribution is given in Section 2.8. I explain them in detail in a first-authored journal paper. However, this paper does not constitute a chapter of this thesis because it

has not been peer-reviewed yet.

Each of these papers contains theoretical insights, algorithm descriptions and experiments. Since most papers are evaluating improvements to our system by re-proving ITP libraries, I discuss the methodology and evaluation settings shared by these experiments in Section 2.9.

2.1 Premise Selection and External Provers for HOL4

Publication Details

[1] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In Xavier Leroy and Alwen Tiu, editors, *Conference on Certified Programs and Proofs (CPP)*, pages 49–57. ACM, 2015. URL `http://doi.org/10.1145/2676724.2693173`

In this chapter we make HOL(y)Hammer available to HOL4 users. Thanks to the integration of HOL(y)Hammer, external ATPs (such as E prover) can now be called to discharge HOL4 goals. Since HOL(y)Hammer was first conceived for HOL Light users only, we adapt its infrastructures to accept a more generic input format. We choose a format close to TPTP (THF1) [KSR16]. This way, it is easy to express and export HOL Light and HOL4 formulas in this common format. HOL(y)Hammer then parses the exported formulas back into HOL Light terms which are feeded to the premise selection and translation modules. We also track dependencies between theorems through kernel rules using HOL4's tagging system as this information has been shown to improve the accuracy of the premise selection algorithm on HOL Light developments.

We evaluate the performance of HOL(y)Hammer for HOL4 by re-proving the standard library with multiple provers, suitable number of premises and different accessible set of theorems. In the most successful setting, a combination of three provers E prover, Vampire and z3 running in parallel for 30 seconds is able to prove 50% of HOL4 theorems.

My contribution to this paper was first to implement dependency tracking in HOL4 and export HOL4 terms to the HOL(y)Hammer input format. Then, I transformed pre-parsed OCaml data structures of the input formulas into HOL Light terms. I also adapted with the help of my co-author the output of the HOL Light extraction mechanism to match the input format of HOL(y)Hammer. Finally, I packaged HOL(y)Hammer as a HOL4 tactic, gave a detailed description of the system, presented the different evaluation settings and ran the re-proving experiments on the HOL4 standard library.

2.2 Beagle as an External ATP Method

Publication Details

[2] Thibault Gauthier, Cezary Kaliszyk, Chantal Keller, and Michael Norrish. Beagle as a HOL4 external ATP method. In Stephan Schulz, Leonardo De Moura, and Boris Konev, editors, *Workshop on Practical Aspects of Automated Reasoning (PAAR)*,

volume 31 of *EPiC*, pages 50–59. EasyChair, 2015. URL http://doi.org/10.29007/8xbv

In this work we investigate the usability of an SMT solver (Beagle) as an automated theorem proving component of a hammer. SMT solvers are capable of reasoning modulo theories (arithmetic, lists), so they have built-in decision procedures adapted to these theories and an input language with reserved types and constants for those domains. In the case of Beagle, this language is TFA (type first-order arithmetic) which is part of the TPTP family.

For this purpose, we modify the existing translation from higher-order to first-order and target TFA instead. First, a full type encoding is not necessary anymore because TFA supports unit types. The type transformation can be summarized as an application of the monomorphization algorithm to instantiate the polymorphic types. Second, the arithmetic theory in HOL4 is manually mapped to its representation in TFA. In practice, this includes a direct substitution of the HOL4 arithmetic constants and types to their counterparts in TFA. As many problems in HOL4 are stated about natural numbers and TFA only supports integers, we also convert theorems about naturals to theorems about integers as a pre-processing step.

I implemented the translation steps in HOL4 for the translation of HOL4 formulas to TFA: monomorphization, λ-lifting and formula extraction. For arithmetic support, I also added a mapping for arithmetic types and constants and created a conversion form naturals to integers. In order to test some reconstruction techniques, I tweaked Beagle so that it outputs a minimal trace.

2.3 Aligning Concepts across Proof Assistant Libraries

Publication Details

[3] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 8543 of *LNCS*, pages 267–281. Springer, 2014. URL http://doi.org/10.1007/978-3-319-08434-3_20

[4] Thibault Gauthier and Cezary Kaliszyk. Aligning concepts across proof assistant libraries. *Journal of Symbolic Computation*, 90:89–123, 2019. URL https://doi.org/10.1016/j.jsc.2018.04.005

This chapter discusses research ideas published in the Journal of Symbolic Computation [4] which extends a conference paper [3]. Since the journal paper subsumes the conference paper, I only include the journal publication in this thesis.

Here, our goal is to increase the interoperability between ITPs by recognizing which concepts are isomorphic or similar. Aligning concepts originating from different ITP theories is arguably the first step toward sharing mathematical knowledge across ITPs. This kind of concept matching was already mentioned in our HOL4-Beagle translation

for the few theories Beagle supported. However, by applying an automatic alignment method to the large libraries of ITPs, tens of thousands of concepts can be analyzed, leading to thousands of matches.

We observe and overcome three challenges which come up when trying to define a flexible and accurate alignment algorithm. First, each ITP may represent its formulas with a particular logical language. For this, we manually align the small set of logical constructions. Our algorithm could recognize these alignments automatically but it would add some ambiguity which would reduce the accuracy of our results. Second, there are many choices for what constitutes a concept on the syntactical level. It can for example be a constant, a type, a subterm or any λ-abstraction generalized from subterms. A more general (or abstract) concept typically appears in more theorems which favors statistical learning. But when enough data is available, more precise concepts are preferable since finding such mappings gives more information. This leads us to our third point which is to specify what constitutes a match between two concepts and to devise a procedure to recognize them.

The three issues have been resolved in the implementation as follows. We extract theorems from six ITPs based on different logics to the same datatype. We create patterns (also called properties) by abstracting mathematical theorems. We made a statical analysis of the concepts sharing the most number of properties (i.e. the concepts $+$ and $\times$ share the commutativity property). Rare properties were given a higher weight, making the concepts sharing this property more similar. Since properties may contain more than one concept, the probability of an alignment has to be understood in the context of other possible alignments. This gives rise to a dynamical system that recursively improves the quality of the matches. On top of that, in order to tell apart the best matches, disambiguation methods are applied when a concept matches multiple other concepts with a high probability.

We evaluate the approach on six ITPs: HOL4, HOL Light, and Isabelle/HOL for higher-order logic, Coq and Matita for intuitionistic type theory, and the Mizar Mathematical Library for set theory. Comparing the structures available in these libraries our algorithm automatically discovers hundreds of isomorphic concepts (which corresponds to the same mathematical object) and thousands of highly similar ones. Although the goal of this chapter is to find the highest number of isomorphic matches, approximate matches are also useful for any of the applications envisioned: translation, premise selection and conjecturing.

My contributions were the extraction of Coq theorems and constants to the intermediate OCaml datastructures. I also parsed the XML export of Matita and the typed version of Mizar into these structures. All these exports include implicit logical mappings. Next, I created modules that can extract patterns from theorems modulo a set of normalizations (CNF conversion, rewriting modulo associativity and commutativity) and abstractions (subterm abstraction, type abstraction). Then, I implemented a dynamical system composed of concept pairs with evolving interdependent similarity scores. To get a theoretical confirmation of the approach, I proved convergence and uniqueness theorems for this dynamical system.

2.4 Sharing HOL Proof Knowledge

Publication Details

[5] Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 9450 of *LNCS*, pages 372–386. Springer, 2015. URL `http://doi.org/10.1007/978-3-662-48899-7_26`

To demonstrate that alignment discovery can be used to share proof knowledge across ITPs, we evaluate the quality of the knowledge transfer by trying to re-prove theorems in one prover (target ITP) with the help of additional knowledge from the other library (helping ITP).

A direct way to take advantage of the mathematical knowledge present in the helping ITP is to translate every theorem from the helping ITP to the target ITP by substituting isomorphic concepts. Then, we can observe how these additional theorems contribute to an increase of the success rate of a hammer (or other proof automation) on the target library. Nevertheless, it can be particularly difficult to re-prove these theorems in the target library automatically. The biggest hurdles are small changes in the concept definitions, incompatible logics and different kernel rules. That is why we forgo importing theorems in our approach and only use the additional knowledge to guide premise selection of a hammer. From dependencies between theorems, we derive a relation between concepts found in these theorems. These concepts are mapped from the helping ITP to the target ITP. The mapped concept relation helps the premise selection algorithm to select lemmas that contains suitable for concepts. We propose four scenarios for this method depending on the kind of theorem dependencies used and the sets of theorems accessible. In our experiments, we re-prove the HOL4 library with the help of HOL Light knowledge and conversely. When re-proving HOL Light, the additional knowledge extracted from HOL4 increases the success rate of the single best HOL(y)Hammer strategy from 30 % to 40 %.

My contributions to this paper were the conception and implementation of proving schemes (called scenarios here) that would benefit from the interplay of the premise selection and alignment algorithms.

2.5 Statistical Conjecturing

Publication Details

[6] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Initial experiments with statistical conjecturing over large formal corpora. In Andrea Kohlhase, Paul Libbrecht, Bruce R. Miller, Adam Naumowicz, Walther Neuper, Pedro Quaresma, Frank Wm. Tompa, and Martin Suda, editors, *Work in Progress at the Conference on Intelligent Computer Mathematics (CICM-WiP)*, volume 1785, pages 219–228. CEUR-WS.org, 2016. URL `http://ceur-ws.org/Vol-1785/W23.pdf`

In this work, we attempt to improve proof automation by conjecturing intermediate lemmas. Theorem proving searches for a path of formal transformations from existing theorems to the conjecture. When the length of the path grows, it becomes exponentially harder for an ATP to find a proof.

Our solution is to propose candidate intermediate steps (i.e. conjectured lemmas), splitting the proof into smaller parts, making it easier for a hammer. To create the intermediate conjectures, we use approximate concepts alignments, enabling us to map via concept substitutions (also called analogies) theorems from one domain to another (e.g. from reals to complex numbers). When mapping theorems within a library, it is not obvious which concept substitution should be applied. Indeed, substitutions where some concepts are left unchanged should also be considered. To heuristically evaluate the potential of each substitution, we take into account the similarity score of concept pairs and the correlation between concept pairs (i.e. approximately how often two pairs co-occur in the same pattern).

Experiments are conducted on the totality of the Mizar library. In a first experiment, we evaluate our conjecturing method in a non-targeted way leading to:

- the discovery of tens of thousands of approximate matches,
- the generation of 73535 conjectures,
- 10% of these conjectured lemmas can be proven by the hammer MizAR,
- 6% are non-trivial in the sense that their proof requires at least two lemmas.

The second experiment consists of evaluating the effect of the inclusion of the conjecturing process on the success rate of MizAR over the Mizar library. However, the results show no improvement over the state-of-the-art. This is probably due this technology's early stage of development.

My principal contribution to this work was the design of an algorithm for creating and ranking context-dependent substitutions. I also programmed a conjecture generator with heuristics for deciding which theorems should be transformed by highly-ranked substitutions.

2.6 Learning to Reason with Tactics

Publication Details

[7] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors, *Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR)*, volume 46 of *EPiC*, pages 125–143. EasyChair, 2017. URL `http://www.easychair.org/publications/paper/340355`

One important issue of tools like Sledgehammer or HOL(y)Hammer noticed by users is they can be quite brittle. The premise selection is very sensible to the shape of the

goal and a slight increase in complexity of the proof might make the goal too challenging for ATPs. Most of the time, some pre-processing of the goal by a human is required: unfolding of definitions, case splitting or induction. After applying one or more of these steps, proving the goal becomes much easier for a hammer. These steps are typically achieved through the application of tactics.

Therefore, we propose a method for automatically suggesting suitable tactics for a given goal. Tactics are ranked according to how successful they have been on similar goals. The similarity between goals is given by a nearest neighbor algorithm whose database is build by recording humans proofs at the tactic level. Since more than one tactic application could be needed to complete the proof, we implement a proof search algorithm. It is an A*-algorithm with cost and heuristic functions determined by the tactic selection algorithm. To avoid repeating the same transformations on the same goals, we introduce an orthogonalization procedure. Furthermore, to increase the potential of the tactical proof search system TacticToe, we use a small hammer approach. The internal ATP Metis is run with 16 selected premises on each intermediate goals during the search. In this way, we combine the flexibility of general purpose ATPs and the precision of specialized tactics.

In our experiments, we tune the following parameters: feature generation, prediction algorithm and tactic timeout, number of selected premises for Metis and different heuristic functions for the A*-algorithm. We run a full-scale evaluation with best parameters and a time limit of 5 seconds to compare TacticToe and HOL(y)Hammer approaches. TacticToe re-proves 39% of the HOL4 library whereas the best single HOL(y)Hammer strategy (based on E prover) solves 32%.

I contributed to the implementation of TacticToe: parsing HOL4 proofs at the tactic level, ported the premise selection algorithm to SML and adapted it to predict tactics and designed the proof search algorithm.

2.7 Contributions beyond this Thesis: Standard for Alignments

[8] Dennis Müller, Thibault Gauthier, Cezary Kaliszyk, Michael Kohlhase, and Florian Rabe. Classification of alignments between concepts of formal mathematical systems. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 10383 of *LNCS*, pages 83–98. Springer, 2017. URL `http://doi.org/10.1007/978-3-319-62075-6_7`

This chapter proposes a standard format for alignments of formal concepts. It is not included in this thesis because I am not the main author of the paper. Only formal systems were considered in Section 2.3. Here, we also aim to classify alignments between formal (e.g. Mizar) and informal systems (e.g. Wikipedia).

Some alignments are perfect in the sense that two aligned concepts are syntactically and semantically equivalent. But approximate alignments are much more common. That is why we explicitly classify different types of approximate alignments: up to permutation of arguments, up to totality of functions, up to generalization, etc. Our list of types

of alignments is not exhaustive and many of them were found manually. Exploring these types of alignments could give us an idea how to automatically discover matching concepts between formal and informal libraries.

The format for a concept pair specified in this chapter is a simple URI pair with an additional tag describing the type of alignment. The URI gives the particular location from which the concept was extracted. The tags can be made quite precise and may contain a level of confidence in case the alignment was discovered automatically. For formal libraries, it is the theory where it was introduced and for informal libraries it can either be the directory where it is located in the database or the webpage of its definition. We use of a list of adjectives following the concept pair to characterize the alignment.

My contribution in this paper was to propose some classes of alignments based on my matching experiments as well as to adapt the automatically found alignments (already presented in Section 2.3) to the formats and specifications proposed in the paper.

2.8 Contributions beyond this Thesis: Tactical Proof Search

Publication Details

[9] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Learning to prove with tactics. *CoRR*, 2018. URL `http://arxiv.org/abs/1804.00596`

This paper presents the most recent update on the tactic-based theorem prover TacticToe (introduced in Section 2.6). It has been submitted in March 2018 to the Journal of Automated Reasoning and has not undergone peer-review yet. It is therefore not included in this thesis. Thanks to these updates, the success rate of TacticToe has increased from 39% to 66% on the HOL4 standard library. In the following, we discuss the ideas and implementation changes that contributed to this large improvement. We explain how they affect each of the components of TacticToe: learning and predicting, proof search and integration with HOL4.

Learning and Predicting The effectiveness of the prediction algorithms is strongly correlated with the quantity and quality of the training examples. We increase the size of the tactic database by recording more proofs (e.g. proofs created by the SML function `prove`) and supporting more SML constructions (e.g. pattern matching). This produces a larger set of raw data that is filtered by an improved orthogonalization algorithm. The decision as which tactic to keep between tactics with similar effects is now determined by the coverage of a tactic, which is the number of times a tactic appears in the recorded database. A tactic abstraction mechanism is introduced. It creates more general and flexible tactics where tactic arguments can be dynamically and independently predicted. In practice, it means that the tactical predictor can now choose theorem arguments adapted for the current goal. Tactic abstraction considerably extends the action space of TacticToe and solves one of its blind spots by proposing theorems that never appeared in previous tactic calls.

Proof Search Monte Carlo tree search (MCTS) replaces A* as our search algorithm. This change is natural as the A*-algorithm works on graphs whereas MCTS is designed for trees and the datastructure generated by tactical search is a proof tree. As a consequence, the cost and heuristic functions of A* are replaced by the policy and evaluation functions of MCTS. In an effort to make stronger ATPs available during the search, we build the necessary interface for TacticToe to call HOL(y)Hammer.

Integration with HOL4 In this work, we focus a lot more on how to improve the user interaction with TacticToe. In particular, the proofs returned by the search algorithm can be minimized in two ways: some of the tactic steps may be redundant and lists appearing as tactic arguments may contain unnecessary elements. The proof can be further embellished by omitting module prefixes if these modules are opened in the HOL4 session. After the application of many other embellishment steps, we then return a readable proof to the user that is very similar to previous human proofs in its presentation.

I designed and applied all the algorithm changes to TacticToe and wrote a concise and precise description for each of them. In order to make the structure of TacticToe more visible, I gave an overview of the relation between its modules. To improve the success rate of TacticToe, I tuned the hyper-parameters of TacticToe during training experiments. In a final full-scale re-proving experiment, I evaluated TacticToe by comparing its success rate with E prover. In order to give ideas of the strength and weaknesses of TacticToe, I observed how the provability of a theorem by TacticToe is affected by its origin or the length of its original proofs. Finally, I compared the proof discovered by TacticToe with the original human proofs.

2.9 Methodology and Evaluation

The success of our proof automation techniques, developed in each of the presented papers, is measured by the percentage of theorems than can be re-proven among a set of top-level theorems (typically the whole standard library). We mention here three points that show the robustness of our methodology and confirm the validity of this approach.

2.9.1 Fairness

The evaluation imitates the construction of the library: For each theorem only the previous human proofs are known. These are used as the learning base for the predictions. To achieve this scenario we re-prove all theorems during a modified build of an ITP library. As theorems are proved, their human proofs are recorded and included in the training examples.

2.9.2 Comparison with other Systems

Our results are compared with the state-of-the-art techniques for automatically proving theorems in each ITP. Since we are the pioneers in this kind of experiments, this often

means that we have to compare our new techniques to the most successful ones we have developed so far. Nevertheless, because we have chosen standard datasets, it will invite future research to use them in their evaluations in order to measure their progress.

2.9.3 Reproducibility

To increase the credibility of our experiments, we make sure that they are reproducible by providing the source code used for our experiments under a permissible license. The resources can be downloaded at:
`https://github.com/HOL-Theorem-Prover/HOL/src/holyhammer` for Chapter 3,
`https://github.com/barakeel/HOLtoTFF` for Chapter 4,
`http://cl-informatik.uibk.ac.at/users/tgauthier/alignments` for Chapter 5-6,
`http://thibaultgauthier.fr/conjecturing.tar.gz` for Chapter 7,
`https://github.com/HOL-Theorem-Prover/HOL/src/tactictoe` for Chapter 8.
In this way, other teams can replicate our experiments to confirm our findings. Our results can be kept up-to-date by evaluating our programs against the latest ITP libraries without major modifications to our programs. Indeed, only modifications to an ITP kernel would require an update of our framework and this update would be limited to the import and export modules for the particular ITP.

Chapter 3

Premise Selection and External Provers for HOL4

Abstract

Learning-assisted automated reasoning has recently gained popularity among the users of Isabelle/HOL, HOL Light, and Mizar. In this paper, we present an add-on to the HOL4 proof assistant and an adaptation of the HOL(y)Hammer system that provides machine learning-based premise selection and automated reasoning also for HOL4. We efficiently record the HOL4 dependencies and extract features from the theorem statements, which form a basis for premise selection. HOL(y)Hammer transforms the HOL4 statements in the various TPTP-ATP proof formats, which are then processed by the ATPs.

We discuss the different evaluation settings: ATPs, accessible lemmas, and premise numbers. We measure the performance of HOL(y)Hammer on the HOL4 standard library. The results are combined accordingly and compared with the HOL Light experiments, showing a comparably high quality of predictions. The system directly benefits HOL4 users by automatically finding proofs dependencies that can be reconstructed by Metis.

3.1 Introduction

The HOL4 proof assistant [SN08] provides its users with a full ML programming environment in the LCF tradition. Its simple logical kernel and interactive interface allow safe and fast developments, while the built-in decision procedures can automatically establish many simple theorems, leaving only the harder goals to its users. However, manually proving theorems based on its simple rules is a tedious task. Therefore, general purpose automation has been developed internally, based on model elimination (MESON [Har96]), tableau (blast [Pau99]), or resolution (Metis [Hur03]). Although essential to HOL4 developers, the methods are so far not able to compete with the external ATPs [Sch02, KV13] optimized for fast proof search with many axioms present and continuously evaluated on the TPTP library [Sut09] and updated with the most successful techniques. The TPTP (Thousands of Problems for Theorem Provers) is a library of test problems for automated theorem proving (ATP) systems. This standard enables convenient communication between different systems and researchers.

On the other hand, the HOL4 system provides a functionality to search the database for theorems that match a user chosen pattern. The search is semi-automatic and the resulting lemmas are not necessarily helpful in proving the conjecture. An approach that combines the two: searching for relevant theorems and using automated reasoning methods to (pseudo-)minimize the set of premises necessary to solve the goal, forms the basis of "hammer" systems such as Sledgehammer [PB10] for Isabelle/HOL, HOL(y)Hammer [KU14] for HOL Light or MizAR for Mizar [KU15d]. Furthermore, apart from syntactic similarity of a goal to known facts, the relevance of a fact can be learned by analyzing dependencies in previous proofs using machine learning techniques [Urb07], which leads to a significant increase in the power of such systems [KBKU13].

In this paper, we adapt the HOL(y)Hammer system to the HOL4 system and test its performance on the HOL4 standard library. The libraries of HOL4 and HOL Light are exported together with proof dependencies and theorem statement features; the predictors learn from the dependencies and the features to be able to produce lemmas relevant to a conjecture. Each problem is translated to the TPTP FOF format. When an ATP finds a proof, the necessary premises are extracted. They are read back to HOL4 as proof advice and given to Metis for reconstruction.

An adapted version of the resulting software is made available to the users of HOL4 in interactive session, which can be used in newly developed theories. Given a conjecture, the SML function computes every step of the interaction loop and, if successful, returns the conjecture as a theorem:

Example 3.1. (HOL(y)Hammer interactive call)

```
load "holyHammer";
  val it = (): unit
holyhammer ``1+1=2``;
  Relevant theorems: ALT_ZERO ONE TWO ADD1
  metis: r[+0+6]#
  val it = |- 1 + 1 = 2: thm
```

The HOL4 prover already benefits from export to SMT solvers such as Yices [Web11], z3 [BW10] and Beagle [GKKN15]. These methods perform best when solving problems from the supported theories of the SMT solver. Comparatively, HOL(y)Hammer is a general purpose tool as it relies on ATPs without theory reasoning and it can provide easily[1] re-provable problem to Metis.

The HOL4 standard distribution has since long been equipped with proof recording kernels [Won95, KH12]. We first considered adapting these kernels for our aim. But as machine learning only needs the proof dependencies and the approach based on full proof recording is not efficient, we perform minimal modifications to the original kernel.

Contributions We provide learning assisted automated reasoning for HOL4 and evaluate its performance in comparison to that in HOL Light. In order to do so, we :

[1] reconstruction rate is typically above 90%

- Export the HOL4 data

  Theorems, dependencies, and features are exported by a patched version of the HOL4 kernel. It can record dependencies between theorems and keep track on how their conjunctions are handled along the proof. We export the HOL4 standard libraries (58 types, 2305 constants, 11972 theorems) with respect to a strict name-space rule so that each object is uniquely identifiable, preserving if possible its original name.

- Re-prove

  We test the ability of a selection of external provers to re-prove theorems from their dependencies.

- Define accessibility relations

  We define and simulate different development environments, with different sets of accessible facts to prove a theorem.

- Experiment with predictors

  Given a theorem and a accessibility relation, we use machine learning techniques to find relevant lemmas from the accessible sets. Next, we measure the quality of the predictions by running ATPs on the translated problems.

The rest of this paper is organized as follows. In Section 3.2 we describe the export of the HOL4 and HOL Light data into a common format and the recording of dependencies in HOL4. In Section 3.3, we present the different parameters: ATPs, proving environments, accessible sets, features, and predictions. We select some of them for our experiments and justify our choice. In Section 3.4 we present the results of the HOL4 experiments, relate them to previous HOL(y)Hammer experiments and explain how this affects the users. Finally in Section 3.5 we conclude and present an outlook on the future work.

3.2 Sharing HOL Data between HOL4, HOL Light and HOL(y)Hammer

In order to process HOL Light and HOL4 data in a uniform way in HOL(y)Hammer, we export objects from their respective theories, as well as dependencies between theorems into a common format. The export is available for any HOL4 and HOL Light development. We shortly describe the common format used for exporting both libraries and present in more detail our methods for efficiently recording objects (types, constants and theorems) and precise dependencies in HOL4. We will refer to HOL(y)Hammer [KU14] for the details on recording objects and dependencies for HOL Light formalizations.

HOL Light and HOL4 share a common logic (higher-order logic with implicit shallow polymorphism), however their implementations differ both in terms of the programming language used (OCaml and SML respectively), data structures used to represent the terms and theorems (higher-order abstract syntax and de Bruijn indices respectively), and

the exact inference rules provided by the kernel. As HOL(y)Hammer has been initially implemented in OCaml as an extension of HOL Light, we need to export all the HOL4 data and read it back into HOL(y)Hammer, replacing its type and constant tables. The format that we chose is based on the TPTP THF0 format [SB10] used by higher-order ATPs. Since formulas contains polymorphic constants which is not supported by the THF0 format, we will present an experimental extension of this format where the type arguments of polymorphic constants are given explicitly.

Example 3.2. (Experimental template)

```
tt(name, role, formula)
```

The field name is the object's name. The field role is "ty" if the object is a constant or a type, and "ax" if the object is a theorem. The field formula is an experimental THF0 formula.

Example 3.3. (Object export from HOL4 to an experimental format)

- Type

```
(list,1) → tt(list, ty, $t > $t).
```

- Constant

```
(HD,‘‘:’a list -> :’a‘‘) →
  tt(HD, ty, ![A:$t]: (list A > A).
(CONS,‘‘:’a -> :’a list -> :’a list‘‘) →
  tt(CONS ,ty, ![A:$t]: (A > list A > list A).
```

- Theorem

```
(HD,‘‘∀ n:int t:list[int]. HD (CONS n t) = n‘‘) →
  tt(HD0, ax, (![n:int, t:(list int)]:
    ((HD int) ((CONS int) n t) = n).
```

In this example, `$t` is the type of all basic types.

All names of objects are prefixed by a namespace identifier, that allow identifying the prover and theory they have been defined in. For readability, the namespace prefixes have been omitted in all examples in this paper.

3.2.1 Creation of a HOL4 Theory

In HOL4, types and constants can be created and deleted during the development of a theory. These objects are named at the moment they are created. A theorem is a SML value of type *thm* and can be derived from a set of basic rules, which is an instance of a typed higher-order classical logic. To distinguish between important lemmas and theorems created by each small steps, the user can name and delete theorems (erase the name). Each named object still present at the end of the development is saved and thus can be called in future theories.

There are two ways in which an object can be lost in a theory: either it is deleted or overwritten. As proof dependencies for machine learning get more accurate when more intermediate steps are available, we decided to record all created objects, which results in the creation of slightly bigger theories. As the originally saved objects can be called from other theories, their names are preserved by our transformation. Each lost object whose given name conflicts with the name of a saved object of the same type is renamed.

Deleted Objects The possibility of deleting an object or even a theory is mainly here to hide internal steps or to make the theory look nicer. We chose to remove this possibility by canceling the effects of the deleting functions. This is the only user-visible feature that behaves differently in our dependency recording kernel.

Overwritten Objects An object may be overwritten in the development. As we prevent objects from being deleted, the likelihood of this happening is increased. This typically happens when a generalized version of a theorem is proved and is given the same name as the initial theorem. In the case of types and constants, the internal HOL4 mechanism already renames overwritten objects. Conversely, theorems are really erased. To avoid dependencies to theorems that have been overwritten, we automatically rename the theorems that are about to be overwritten.

3.2.2 Recording Dependencies

Dependencies are an essential part of machine learning for theorem proving, as they provide the examples on which predictors can be trained. We focus on recording dependencies between named theorems, since they are directly accessible to a user. The time mark of our method slows down the application of any rules by a negligible amount.

Since the statements of 951 HOL4 theorems are conjunctions, sometimes consisting of many toplevel conjuncts, we have refined our method to record dependencies between the toplevel conjuncts of named theorems.

Example 3.4. (Dependencies between conjunctions)

```
ADD_CLAUSES: 0 + m = m ∧ m + 0 = m ∧
SUC m + n = SUC (m + n) ∧ m + SUC n = SUC (m + n)

ADD_ASSOC depends on:
```

```
ADD_CLAUSES_c1: 0 + m = m
ADD_CLAUSES_c3: SUC m + n = SUC (m + n)
...
```

The conjunct identifiers of a named theorem `T` are noted `T_c1`, ..., `T_cN`.

In certain theorems, a toplevel universal quantifier shares a number of conjuncts. We will also split the conjunctions in such cases recursively. This type of theorem is less frequent in the standard library (203 theorems).

Example 3.5. (Conjunctions under quantifier)

`MIN_0: ∀ n. (MIN n 0 = 0) ∧ (MIN 0 n = 0)`

By splitting conjunctions we expect to make the dependencies used as training examples for machine learning more precise in two directions. First, even if a theorem is too hard to prove for the ATPs, some of its conjuncts might be provable. Second, if a theorem depends on a big conjunction, it typically depends only on some of its conjuncts. Even if the precise conjuncts are not clear from the human-proof, the re-proving methods can often minimize the used conjuncts. Furthermore, reducing the number of conjuncts should ease the reconstruction.

3.2.3 Implementation of the Recording

The HOL4 type of theorems *thm* includes a tag field in order to remember which oracles and axioms were necessary to prove a theorem. Each call to an oracle or axiom creates a theorem with the associated tag. When applying a rule, all oracles and axioms from the tag of the parents are respectively merged, and given to the conclusion of the rule. To record the dependencies, we added a third field to the tag, which consists of a dependency identifier and its dependencies.

Example 3.6. (Modified tag type)

```
type tag = ((dependency_id, dependencies),
            oracles, axioms)
type thm = (tag, hypotheses, conclusion)
```

Since the name of a theorem may change when it is overwritten, we create unmodifiable unique identifiers at the moment a theorem is named.

It consists of the name of the current theory and the number of previously named theorems in this theory. As a side effect, this enables us to know the order in which theorems are named which is compatible by construction with the pre-order given by the dependencies. Every variable of type *thm* which is not named is given the identifier *unnamed*. Only identifiers of named theorems will appear in the dependencies.

We have implemented two versions of the dependency recording algorithm, one that tracks the dependencies between named theorems, other one tracking dependencies

between their conjuncts. For the named theorems, the dependencies are a set of identified theorems used to prove the theorem. The recording is done by specifying how each rule creates the tag of the conclusion from the tag of its premises. The dependencies of the conclusion are the union of the dependencies of the unnamed premises with its named premises.

This is achieved by a simple modification of the `Tag.merge` function already applied to the tags of the premises in each rule.

When a theorem $\vdash A \wedge B$ is derived from the theorems $\vdash A$ and $\vdash B$, the previously described algorithm would make the dependencies of this theorem the union of the dependencies of the two. If later other theorems refer to it, they will get the union as their dependencies, even if only one conjunct contributes to the proof. In this subsection we define some heuristics that allow more precise tracking of dependencies of the conjuncts of the theorems.

In order to record the dependencies between the conjuncts, we do not record the conjuncts of named theorems, but only store their dependencies in the tags. The dependencies are represented as a tree, in which each leaf is a set of conjunct identifiers (identifier and the conjunct's address). Each leaf of the tree represents the respective conjunct c_i in the theorem tree and each conjunct identifier represents a conjunct of a named goal to prove c_i.

Example 3.7. (An example of a theorem and its dependencies)

```
Th0 (named theorem): A ∧ B
Th1: C ∧ (D ∧ E)
     with dependency tree Tree([Th0],[Th0_c2])

This encodes the fact that:
  C depends on Th0.
  D ∧ E depends Th0_c2 which is B.
```

Dependencies are combined at each inference rule application and dependencies will contain only conjunct identifiers. If not specified, a premise will pass on its identifier if it is a named conjunct (conjunct of a named theorem) and its dependency tree otherwise. We call such trees passed dependencies. The idea is that the dependencies of a named conjunct should not transmit its dependencies to its children but itself. Indeed, we want to record the direct dependencies and not the transitive ones.

For rules that do not preserve the structure of conjunctions, we flatten the dependencies, i.e. we return a root tree containing the set of all (conjunct) identifiers in the passed dependencies. We additionally treat specially the rules used for the top level organization of conjunctions: `CONJ`, `CONJUNCT1`, `CONJUNCT2`, `GEN`, `SPEC`, and `SUBST`.

- `CONJ`: It returns a tree with two branches, consisting of the passed dependencies of its first and second premise.

- `CONJUNCT1` (`CONJUNCT2`): If its premise is named, then the conjunct is given a conjunct identifier. Otherwise, the first (second) branch of the dependency tree of its premise become the dependencies of its conclusion.

- `GEN` and `SPEC`: The tags are unchanged by the application of those rules as they do not change the structure of conjunctions. Although we have to be careful when using `SPEC` on named theorems as it may create unwanted conjunctions. These virtual conjunctions are not harmful as the right level of splitting is restored during the next phase.

  Example 3.8. (Creation of a virtual conjunction from a named theorem)

  $\forall$ `x.x` $\vdash$ $\forall$ `x.x`

  `SPEC [A` $\wedge$ `B]`

  $\forall$ `x.x` $\vdash$ `A` $\wedge$ `B`

  `CONJUNCT1`

  $\forall$ `x.x` $\vdash$ `A`

- `SUBST`: Its premises consist of a theorem, a list of substitution theorems of the form $(A = B)$ and a template that tells where each substitution should be applied. When `SUBST` preserves the structure of conjuncts, the set of all identifiers in the passed dependencies of the substitution theorems is distributed over each leaf of the tree given by the passed dependencies of the substituted theorems. When it is not the case the dependency should be flattened. Since the substitution of sub-terms below the top formula level does not affect the structure of conjunctions, it is sufficient (although not necessary) to check that no variables in the template is a predicate (is a boolean or returns a boolean).

The heuristics presented above try to preserve the dependencies associated with single conjuncts whenever possible. It is of course possible to find more advanced heuristics, that would give more precise human-proof dependencies. However, performing more advanced operations (even pattern matching) may slow down the proof system too much; so we decided to restrict to the above heuristics.

Before exporting the theorems, we split them by recursively distributing quantifiers and splitting conjunctions. This gives rise to conflicting degree of splitting, as for instance, a theorem with many conjunctions may have been used as a whole during a proof. Given a theorem and its dependency tree, each of its conjunctions is given the set of identifiers of its closest parent in this tree. Then, each of these identifiers is also split maximally. In case of a virtual conjunction (see the `SPEC` rule above), the corresponding node does not exist in the theorem tree, so we take the conjunct corresponding to its closest parent. Finally, for each conjunct, we obtain a set of dependencies by taking the union of the split identifiers.

Example 3.9. (Recovering dependencies from the named theorem Th1)

```
Th0 (named theorem): A ∧ B
Th1 (named theorem): C ∧ (D ∧ E)
     with dependency tree Tree([Th0],[Th0_c1])

  Recovering dependencies of each conjunct
Th1_c0: Th0
Th1_c1: Th0_c1
Th1_c2: Th0_c1
  Splitting the dependencies
Th1_c0: Th0_c1 Th0_c2
Th1_c1: Th0_c1
Th1_c2: Th0_c1
```

3.3 Evaluation

In this section we describe the setting used in the experiments: the ATPs, the transformation from HOL to the formats of the ATPs, the dependencies accessible in the different experiments, and the features used for machine learning.

3.3.1 ATPs and Problem Transformation

HOL(y)Hammer supports the translation to the formats of various TPTP ATPs: FOF, TFF1, THF0, and two experimental TPTP extensions. In this paper we restrict ourselves to the first order monomorphic logic, as these ATPs have been the most powerful so far and integrating them in HOL4 already poses an interesting challenge. The transformation that HOL(y)Hammer uses is heavily influenced by previous work by Paulson [PS07] and harrison [Har96]. It is described in detail in [KU14], here we remind only the crucial points. Abstractions are removed by β-reduction followed by λ-lifting, predicates as arguments are removed by introducing existentially quantified variables and the apply functor is used to reduce all applications to first-order. By default HOL(y)Hammer uses the tagged polymorphic encoding [BBPS13]: a special tag taking two arguments is introduced, and applied to all variable instances and certain applications. The first argument is the first-order flattened representation of the type, with variables functioning as type variables and the second argument is the value itself.

The initially used provers, their versions and default numbers of premises are presented in Table 3.1. The HOL Light experiments [KU14] showed, that different provers perform best with different given numbers of premises. This is particularly visible for the ATP provers that already include the relevance filter SInE [HV11], therefore we preselect a number of predictions used with each prover. Similarly, the strategies that the ATP provers implement are often tailored for best performance on the TPTP library, for the annual CASC competition [Sut14]. For ITP originating problems, especially for E prover different strategies are often better, so we run it under the alternate scheduler `Epar` [Urb15].

Prover	Version	Premises
Vampire	2.6	96
E prover	1.8	128
z3	4.32	32
CVC4	1.3	128
Spass	3.5	32
IProver	1.0	128
Metis	2.3	32

Table 3.1: ATP provers, their versions and arguments

3.3.2 Accessible Facts

As HOL(y)Hammer has initially been designed for HOL Light, it treats accessible facts in the same way as the accessibility relation defined there: any fact that is present in a theory loaded chronologically before the current one is available. In HOL4 there are explicit theory dependencies, and as such a different accessibility relation is more natural. The facts present in the same theory before the current one, and all the facts in the theories that the current one depends on (possibly in a transitive way) are accessible. In this subsection we discuss the four different accessible sets of lemmas, which we will use to test the performance of HOL(y)Hammer on.

Exact Dependencies (re-proving) They are the closest named ancestors of a theorem in the proof tree. It tests how much HOL(y)Hammer could re-prove if it had perfect predictions. In this settings no relevance filtering is done, as the number of dependencies is small.

Transitive Dependencies They are all the named ancestors of a theorem in the proof tree. It simulates proving a theorem in a perfect environment, where all recorded theorems are a necessary step to prove the conjecture. This corresponds to a proof assistant library that has been refactored into little theories [FGT92].

Loaded Theorems All theorems present in the loaded theories are provided together with all the theorems previously built in the current theory. This is the setting used when proving theorems in HOL4, so it is the one we use in our interactive version presented and evaluated in Section 3.4.5.

Linear Order For this experiment, we additionally recorded the order in which the HOL4 theories were built, so that we could order all the theorems of the standard library in a similar way as HOL Light theorems are ordered. All previously derived theorems are provided.

3.3.3 Features

Machine learning algorithms typically use features to define the similarity of objects. In the large theory automated reasoning setting features need to be assigned to each theorem, based on the syntactic and semantic properties of the statement of the theorem and its attributes.

HOL(y)Hammer represents features by strings and characterizes theorems using lists of strings. Features originate from the names of the type constructors, type variables, names of constants and printed subterms present in the conclusion. An important notion is the normalization of the features: for subterms, their variables and type variables need to be normalized. Various scenarios for this can be considered:

- All variables are replaced by one common variable.
- Variables are replaced by their de Bruijn index numbers [USPV08].
- Variables are replaced by their (variable-normalized) types [KU14].

The union of the features coming from the three above normalizations has been the most successful in the HOL Light experiments, and it is used here as well.

3.3.4 Predictors

In all our experiments we have used the modified k-NN algorithm [KU13b]. This algorithm produces the most precise results in the HOL(y)Hammer experiments for HOL Light [KU14]. Given a fixed number (k), the k-nearest neighbours learning algorithm finds k premises that are closest to the conjecture, and uses their weighted dependencies to find the predicted relevance of all available facts. All the facts and the conjecture are interpreted as vectors in the n-dimensional feature space, where n is the number of all features. The distance between a fact and the conjecture is computed using the Euclidean distance. In order to find the neighbours of the conjecture efficiently, we store an association list mapping features to theorems that have those features. This allows skipping the theorems that have no features in common with the conjecture completely.

Having found the neighbours, the relevance of each available fact is computed by summing the weights of the neighbours that use the fact as a dependency, counting each neighbour also as its own dependency

3.4 Experiments

In this section, we present the results of several experiments and discuss the quality of the advice system based on these results. The hardware used during the re-proving and accessibility experiments is a 48-core server (AMD Opteron 6174 2.2 GHz. CPUs, 320 GB RAM, and 0.5 MB L2 cache per CPU). In these experiments, each ATPs is run on a single core for each problem with a time limit of 30 seconds. The reconstruction and interactive experiments were run on a laptop with a Intel Core processor (i5-3230M 4 x 2.60GHz with 3.6 GB RAM).

3.4.1 Re-proving

We first try to re-prove all the 9434 theorems in the HOL4 libraries with the dependencies extracted from the proofs. This number is lower than the number of exported theorems because definitions are discarded. Table 3.2 presents the success rates for re-proving using the dependencies recorded without splitting. In this experiment we also compare many provers and their versions. For E prover [Sch13b], we also compare its different scheduling strategies [Urb15]. The results are used to choose the best versions or strategies for the selected few provers. Apart from the success rates, the unique number of problems is presented (proofs found by this ATP only), and CVC4 [BCD+11] seems to perform best in this respect. The translation used by default by HOL(y)Hammer is an incomplete one (it gives significantly better results than complete ones), so some of the problems are counter-satisfiable.

From this point on, experiments will be performed only with the best versions of three provers: E prover, Vampire [KV13], and z3 [DMB08]. They have a high success rate combined with an easy way of retrieving the unsatisfiable core. The same ones have been used in the HOL(y)Hammer experiments for HOL Light.

In Table 3.3, we try to re-prove conjuncts of these theorems with the different recording methods described in Section 3.2.3. First, we notice that only z3 benefits from the tracking of more accurate dependencies. More, removing the unnecessary conjuncts worsen the results of E prover and Vampire. One reason is that E prover and Vampire do well with large number of lemmas and although a conjunct was not used in the original

Prover	Version	Theorem(%)	Unique	CounterSat
E prover	Epar 3	44.45	3	0
E prover	Epar 1	44.15	9	0
E prover	Epar 2	43.95	9	0
E prover	Epar 0	43.52	2	0
CVC4	1.3	42.71	44	0
z3	4.32	41.96	8	5
z3	4.40	41.65	1	6
E prover	1.8	41.37	14	0
Vampire	2.6	41.10	14	0
Vampire	1.8	38.34	6	0
z3	4.40q	35.19	11	5
Vampire	3.0	34.82	0	0
Spass	3.5	31.67	0	0
Metis	2.3	29.98	0	0
IProver	1.0	25.52	2	35
total		50.96		38

Table 3.2: Re-proving experiment on the 9434 unsplit theorems of the standard libary

	Basic	Optimized	Basic*	Optimized*
E prover	42.43	42.41	46.23	45.91
Vampire	39.79	39.32	43.24	42.41
z3	39.59	40.63	43.78	44.18
total	46.74	46.76	50.97	50.55

Table 3.3: Success rates of re-proving (%) on the 13910 conjuncts of the standard library with different dependency tracking mechanism.

proof it may well be useful to these provers.Suprisingly, the percentage of re-proved facts did not increase compared to Table 3.2, as this was the case for HOL Light experiments. By looking closely at the data, we notice the presence of the `quantHeuristics` theory, where 85 theorems are divided into 1538 conjuncts. As the percentage of re-proving in this theory is lower than the average (16%), the overall percentage gets smaller given the increased weight of this theory. Therefore, we have removed the quantHeuristic theory in the Basic* and Optimized* experiments for a fairer comparison with the previous experiments. Finally, if we compare the Optimized experiment with the similar HOL Light re-proving experiment on 14185 Flyspeck problems [KU14], we notice that we can re-prove three percent more theorems in HOL4. This is mostly due to a 10 percent increase in the performance of z3 on HOL4 problems.

In Table 3.4 we have compared the success rates of re-proving in different theories, as this may represent a relative difficulty of each theory and also the relative performance of each prover. We observe that z3 performs best on the theories `measure` and `probability`, `list` and `finite_map`, whereas E prover and Vampire have a higher success rate on the theories `arithmetic`, `real`, `complex` and `sort`. Overall, the high success rate in the `arithmetic` and `real` theories confirms that HOL(y)Hammer can already tackle this type of theorems. Nonetheless, it would still benefit from integrating more SMT-solvers' functionalities on advanced theories based on `real` and `arithmetic`.

3.4.2 With Different Accessible Sets

In Table 3.5 we compare the quality of the predictions in different proving environments. We recall that only the transitive dependencies, loaded theories and linear order settings are using predictions and that the number of these predictions is adapted to the ability of each provers. The exact dependencies setting (re-proving), is copied from Table 3.3 for easier comparison.

We first notice the lower success rate in the transitive dependencies setting. There may be two justifications. First, the transitive dependencies provide a poor training set for the predictors; the set of samples is quite small and the available lemmas are all related to the conjecture. Second, it is very unlikely that a lemma in this set will be better than a lemma in the exact dependencies, so we cannot hope to perform better than in the re-proving experiment.

We now focus on the loaded theories and linear order settings, which are the two scenarios that correspond to the regular usage of a "hammer" system in a development: given all the previously known facts try to prove the conjecture. The results are surprisingly better than in the re-proving experiment. First, this indicates that the training data coming from a larger sample is better. Second, this shows that the HOL4 library is dense and that closer dependencies than the exact one may be found by the predictors. It is quite common that large-theory automated reasoning techniques find alternate proofs. Third, if we look at each ATP separately, we see a one percent increase for E prover, a one percent decrease for Vampire, and 9 percent decrease for z3. This correlates with the number of selected premises. Indeed, it is easy to see that if a prover performs well with a large number of selected premises, it has more chance to find the relevant lemmas. Finally, we see that each of the provers enhanced the results by solving different problems.

We can summarize the results by inferring that predictors combined with ATPs are most effective in large and dense developments.

The linear order experiments was also designed to make a valid comparison with a similar experiment where 39% of Flyspeck theorems were proved by combining 14 methods This number was later raised to 47% by improving the machine learning algorithm. Comparatively, the current 3 methods can prove 50% of the HOL4 theorems. This may be since the machine learning methods have improved, since the ATPs are stronger now or even because the Flyspeck theories contain a more linear (less dense) development than the HOL4 libraries, which makes it harder for automated reasoning techniques.

	arith	real	compl	meas
E prover	61.29	72.97	91.22	27.01
Vampire	59.74	69.57	77.19	20.85
z3	51.42	64.46	86.84	31.27
total	63.63	75.31	92.10	32.70
	proba	list	sort	f_map
E prover	42.16	23.56	34.54	33.07
Vampire	37.34	21.96	32.72	27.16
z3	54.21	25.62	25.45	43.70
total	55.42	26.77	40.00	45.27

Table 3.4: Percentage (%) of re-proved theorems in the theories `arithmetic`, `real`, `complex`, `measure`, `probability`, `list`, `sorting` and `finte_map`.

	ED	TD	LT	LO
E prover	42.41	33.10	43.58	43.64
Vampire	39.32	29.56	38.46	38.54
z3	40.63	24.66	31.22	31.20
total	46.76	37.54	50.54	50.68

Table 3.5: Percentage (%) of proofs found using different accessible sets: exact dependencies (ED), transitive dependencies (TD), loaded theories (LT), and linear order (LO)

3.4.3 Reconstruction

Until now all the ATP proved theorems could only be used as oracles inside HOL4. This defeats the main aim of the ITP which is to guarantee the soundness of the proofs. The provers that we use in the experiments can return the unsatisfiable core: a small set of premises used during the proof. The HOL representation of these facts can be given to Metis in order to re-prove the theorem with soundness guaranteed by its construction. We investigate reconstructing proofs found by Vampire on the loaded theories experiments (used in our interactive version of HOL(y)Hammer). We found that Metis could re-prove, with a one second time limit, 95.6% of these theorems. This result is encouraging for two reasons: First, we have not shown the soundness of our transformations, and this shows that the found premises indeed lead to a valid proof in HOL. Second, the high reconstruction rate suggest that the system can be useful in practice.

3.4.4 Case Study

Finally, we present two sets of lemmas found by E prover advised on the loaded libraries. We discuss the difference with the lemmas used in the original proof.

The theorem `EULER_FORMULE` states that any complex number can be represented as a combination of its norm and argument. In the human-written proof script ten theorems are provided to a rewriting tactic. The user is mostly hindered by the fact that she could not use the commutativity of multiplication as the tactic would not terminate. Free of these constraints, the advice system returns only three lemmas: the commutativity of multiplication, the polar representation `COMPLEX_TRIANGLE`, and the Euler's formula `EXP_IMAGINARY`.

Example 3.10. (In theory `complex`)

```
Original proof:
val EULER_FORMULE = store_thm("EULER_FORMULE",
  ‘‘!z:complex. modu z * exp (i * arg z) = z‘‘,
  REWRITE_TAC[complex_exp, i, complex_scalar_rmul,
  RE, IM, REAL_MUL_LZERO, REAL_MUL_LID, EXP_0,
  COMPLEX_SCALAR_LMUL_ONE, COMPLEX_TRIANGLE]);
```

```
Discovered lemmas:
COMPLEX_SCALAR_MUL_COMM COMPLEX_TRIANGLE
EXP_IMAGINARY
```

The theorem `LCM_LEAST` states that any number below the least common multiple is not a common multiple. This seems trivial but actually the least common multiple (lcm) of two natural numbers is defined as their product divided by their greatest common divisor. The user has proved the contraposition which requires two Metis calls. The discovered lemmas seem to indicate a similar proof, but it requires more lemmas, namely `FALSITY` and `IMP_F_EQ_F` as the false constant is considered as any other constant in HOL(y)Hammer and uses the combination of `LCM_COMM` and `NOT_LT_DIVIDES` instead of `DIVIDES_LE`.

Example 3.11. (In theory `gcd`)

```
Original proof:
val LCM_LEAST = store_thm("LCM_LEAST",
  ``0 < m ∧ 0 < n ==> !p. 0 < p ∧ p < lcm m n
  ==> ~(divides m p) ∨ ~(divides n p)``,
  REPEAT STRIP_TAC THEN SPOSE_NOT_THEN
  STRIP_ASSUME_TAC THEN `divides (lcm m n) p`
  by METIS_TAC [LCM_IS_LEAST_COMMON_MULTIPLE]
  THEN `lcm m n <= p` by METIS_TAC [DIVIDES_LE]
  THEN DECIDE_TAC);

Discovered lemmas:
LCM_IS_LEAST_COMMON_MULTIPLE LCM_COMM
NOT_LT_DIVIDES FALSITY IMP_F_EQ_F
```

3.4.5 Interactive Version

In our previous experiments, all the different steps (export, learning/predictions, translation, ATPs) were performed separately, and simultaneously for all the theorems. Here, we compose all this steps to produce one HOL4 step, that given a conjecture proves it, usable in any HOL4 development in an interactive advice loop. It proceeds as follows: The conjecture is exported along with the currently loaded theories. Features for the theorems and the conjecture are computed, and dependencies are used for learning and selecting the theorems relevant to the conjecture. HOL(y)Hammer translates the problem to the formats of the ATPs and uses them to prove the resulting problems. If successful, the discovered unsatisfiable core, consisting of the HOL4 theorems used in the ATP proof, is then read back to HOL4, returned as a proof advice, and replayed by Metis.

In the last experiment, we evaluate the time taken by each steps on two conjectures, which are not already proved in the HOL4 libraries. The first tested goal C_1 is $gcd\ (gcd\ a\ a)\ (b + a) = (gcd\ b\ a)$, where $gcd\ n\ m$ is the greatest common divisor of n

and m. It can be automatically proved from three lemmas about *gcd*. The second goal is C_2 is $Im(i * i) = 0$, where Im the imaginary part of a complex number. It can be automatically proved from 12 lemmas in the theories `real`, `transc` and `complex`.

In Table 3.6, the time taken by the export and import phase linearly depends on the number of theorems in the loaded libraries (given in parenthesis), as expected by the knowledge of our data and the complexity analysis of our code.

The time shown in the fourth column ("Predict") includes the time to extract features, to learn from the dependencies and to find 96 relevant theorems. The time needed for machine learning is relatively short. The time taken by Vampire shows that the second conjecture is harder. This is backed by the fact that we could not tell in advance what would be the necessary lemmas to prove this conjecture. The overall column presents the time between the interactive call and the display of advised lemmas. The low running times support the fact that our tool is fast enough for interactive use.

	Export	Import	Predict	Vampire	Total
C_1 (2224)	0.38	0.20	0.29	0.01	0.97
C_2 (4056)	0.67	0.43	0.59	1.58	3.42

Table 3.6: Time (in seconds) taken by each step of the advice loop

3.5 Conclusion

In this paper we present an adaptation of the HOL(y)Hammer system for HOL4, which allows for general purpose learning-assisted automated reasoning. As HOL(y)Hammer uses machine learning for relevance filtering, we need to compute the dependencies, define the accessibility relation for theorems and adapt the feature extraction mechanism to HOL4. Further, as we export all the proof assistant data (types, constants, named theorems) to a common format, we define the namespaces to cover both HOL Light and HOL4.

We have evaluated the resulting system on the HOL4 standard library toplevel goals: for about 50% of them a sufficient set of dependencies can be found automatically. We compare the success rates depending on the accessibility relation and on the treatment of theorems whose statements are conjunctions. We provide a HOL4 command that translates the current goal, runs premise selection and the ATP, and if a proof has been found, it returns a Metis call needed to solve the goal. The resulting system is available at `https://github.com/HOL-Theorem-Prover/HOL/src/holyhammer`.

3.5.1 Future Work

The libraries of HOL Light and HOL4 are currently processed completely independently. We have however made sure that all data is exported in the same format, so that same concepts and theorems about them can be discovered automatically [GK14]. By combining the data, one might get goals in one system solved with the help of theorems

from the other, which can then be turned into lemmas in the new system. A first challenge might be to define a combined accessibility relation in order to evaluate such a combined proof assistant library.

The format that we use for the interchange of HOL4 and HOL Light data is heavily influenced by the TPTP formats for monomorphic higher-order logic [SB10] and polymorphic first-order logic [BP13]. It is however slightly different from that used by Sledgehammer's `fullthf`. By completely standardizing the format, it would be possible to interchange problems between Sledgehammer and HOL(y)Hammer.

In HOL4, theorems include the information about the theory they originate from and other attributes. It would be interesting to evaluate the impact of such additional attributes used as features for machine learning on the success rate of the proofs. Finally, most HOL(y)Hammer users call its web interface [KU15b], rather than locally install the necessary prover modifications, proof translation and the ATP provers. It would be natural to extend the web interface to support HOL4.

Acknowledgement

We would like to thank Josef Urban and Michael Färber for their comments on the previous version of this paper. This work has been supported by the Austrian Science Fund (FWF): P26201.

Chapter 4

Beagle as an External ATP Method

Abstract

This paper presents BEAGLE_TAC, a HOL4 tactic for using Beagle as an external ATP for discharging HOL4 goals. We implement a translation of the higher-order goals to the TFA format of TPTP and add trace output to Beagle to reconstruct the intermediate steps derived by the ATP in HOL4. Our translation combines the characteristics of existing successful translations from HOL to FOL and SMT-LIB; however, we needed to adapt certain stages of the translation in order to benefit form the expressiveness of the TFA format and the power of Beagle. In our initial experiments, we demonstrate that our system can prove, without any arithmetic lemmas, 81% of the goals solved by Metis.

4.1 Introduction

Interactive theorem provers (ITPs) help researchers certify large or complex proofs, such as the proof of the Kepler conjecture, or modeling complex algorithms and systems. Currently, their main drawback is that they need a lot of human guidance. To solve this issue, for a number of common tasks automation is provided, in particular in the context of higher-order logic internal automated theorem provers (ATPs) based on model elimination (MESON [Har96]), tableau (Blast [Pau99]) and resolution (Metis [Hur05]). The internal implementation may be limited, by the need to interact with the ITP for every proved step.

By contrast, external ATPs can easily be optimized for faster proof search. For this reason, many ITPs exploit their performance by transforming problems to the syntax of various ATPs and calling them on the translated formulas. In this setting, internal ATPs provide a complementary role as they are helping to reconstruct the proof. Such use of external ATPs for premise selection has been successfully used by Isabelle/HOL [PB10], HOL Light [KU14] Mizar [Urb08], and Coq [AFG+11].

In this paper we investigate the use of the TFA (TPTP) format as an interface between ITPs and ATPs. In order to do so we define and implement a translation from the higher-

[†]NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

order logic prover HOL4 to the many-sorted first-order logic with arithmetic defined by the TFA format [SSCB12]. this allows using Beagle [BW13] — an automated theorem prover for first-order logic with equality over linear integer and rational arithmetic — to discharge HOL4 goals. As Beagle is a recent project, we provide a first evaluation of its performance on translated higher-order goals. In order to verify (and possibly reconstruct) the found proofs, we implemented basic proof traces in Beagle. The code described in this paper is available at `https://github.com/barakeel/HOLtoTFF`.

Recently, there has been the emergence of using bridges to ATPs for premise selection. In particular Sledgehammer [PB10] has became an almost indispensable tool for many Isabelle/HOL users, often simply employed instead of library search. Similar tools have been developed for HOL Light [KU14] and Mizar [KU15d]. In the context of HOL4, an export to SMT-solvers Yices and z3 is provided [BW10, Web11]. We are not aware of any HOL4 translation to a superposition based ATP, of any translation from an ITP to a superposition based ATP that use modulo theory decision procedures, such as done by SPASS+T [PW08] and Beagle. SPASS+T uses external SMT-solvers whereas the decision procedures are built inside Beagle, which should make proof reconstruction easier.

Another approach is to implement ATPs inside ITPs, like the implementation of a SMT solver inside Coq [Les11] or the Metis [Hur05] prover. This method gives a prover which is correct by construction. However, experiments [AFG+11] show that you get a better efficiency for complex provers when using external ATPs and checking their answers.

Our translation builds on the ideas of Hurd [Hur03] further enhanced by Meng and Paulson [MP08]. The most related translation is the one done in Isabelle/HOL towards the SMT-solver z3 together with the proof reconstruction are fully described in the PhD thesis of Böhme [BÏ2]. The procedure is now integrated as a part of premise selection in Sledgehammer together with the corresponding proof reconstruction method `smt` [BBP13]. In this work we adapt the translation in a few ways to support Beagle and linear integer arithmetic. Namely the handling of nested predicates, polymorphic axioms and the mapping of natural numbers need to be performed differently to efficiently use our targeted ATP.

The rest of this paper is organized as follows. In Section 4.2 we describe our translation from HOL4 to the TFA. In Section 4.3 we present our experiments with the translation. Our experiments with Beagle proof traces and an initial investigation of proof reconstruction from such traces is explained in Section 4.4. Finally we conclude in Section 4.5 and present an outlook on the future work.

4.2 Translation

In this section, we present the details of our translation, explaining the choices and differences from the existing ones. The HOL4 implementation follows the order presented in this section except for the mapping of monomorphic types and arithmetic constants which are both performed during the printing phase.

4.2.1 TFA Format

The TFA format is part of the TPTP family. It is used to express typed first-order problems with arithmetics and is an extension of the existing FOF format for untyped first-order logic. It contains the predefined basic types: $\$o$ reserved for the return type of predicates, $\$i$ for individuals (used by default if no type is declared), $\$int$, $\$rat$ and $\$real$ for interpreted arithmetics. The equality $\$equal$ is a polymorphic predicate over the basic types. Arithmetic predicates and functions are supported by interpreted overloaded symbols such as $\$less$, $\$uminus$ or $\$sum$.

A TFA problem consists of three parts: basic types (alphanumerical strings interpreted as disjoint sets), function and predicate symbols with their types, and formulas with their role (either axiom or conjecture) and statement. The type of a function or predicate symbol of arity $n \geq 0$ is represented by $(t_1 * \ldots * t_n) > t_{n+1}$ where $t_1 \ldots t_{n+1}$ are basic types. The basic type of a bound variable is given in a formula, at the quantifier position. The TFA format does not support currying or subtyping and overloading and polymorphism are restricted to the defined predicates or functions.

4.2.2 Polymorphic Types

There are many ways of encoding polymorphic types into first-order logic [BBPS13]. Most of them are limited by the fact that first-order provers mainly support one-sorted logic. As a consequence, it is necessary to modify the formula (for instance, by adding predicates or function symbols), thus making the problem harder to solve by the ATPs. These encodings preserve soundness and completeness but compromise efficiency.

Since we want to preserve the arithmetic types of TFA, it is not possible to add predicates that express polymorphism. Therefore, we use an approach similar to a hard type encoding that recently showed good results in an experiment combining SPASS and Isabelle/HOL [BPWW12]. As Beagle handles natively the many-sorted logic of the TFA format, we directly map the HOL4 type of monomorphic variables and constants. For instance, a constant c used with arity 2 having the type $list[int] \to int \to int \to int$ is translated to $(list\_int * \$int) > int\_F\_int$. The type $\$int$ is the TFA reserved type for integers. The types $list\_int$ and int_F_int are created basic types meant to represent lists of integers and functions from integers to integers used as arguments (see defunctionalization, section 4.2.5).

In order to achieve a complete translation, an instantiation of the polymorphic problem needs to be performed. It has been shown by Bobot and Paskevich [BP11] that the problem of computing a finite set of ground instances of the polymorphic formulas such that the resulting ground formulas are equivalent to the original polymorphic formulas is undecidable. Nevertheless, heuristic finite monomorphization using an iterative procedure usually preserves provability. In our implementation, for each constant c in a theorem thm, we search for a constant in the whole problem of the same name with a less general type. The derived substitution is used to create an instantiated copy of the theorem thm and added to the problem. We repeat this process for every theorems a maximum of 3 times, with a limited number of instantiations. These parameters are good enough for

small problems, as in the practical experiments performed in Section 4.3 we reach a fixed point in 73 percent of the cases.

We can derive the maximum number of instantiations by looking at possible instantiations not only in the conjecture but also in other provided theorems. Our instantiation algorithm is similar to the ones used by MESON [Har96] or by Sledgehammer [BBP13], however we use a different recursion scheme and we limit the explosion of the instantiation process by introducing different kinds of bounds.

4.2.3 λ-abstractions

There are two commonly used ways of removing λ-abstractions from higher-order terms in order to translate them to first-order logic: λ-lifting and combinator encoding. The encoding using combinators is a complete one, which means that using the definitions of the combinators a first-order prover can construct an equivalent of every λ-abstraction. Even though this encoding has been shown [PB10] to be reasonable for pure atps, it is not as efficient as λ-lifting for smt-solvers [BBP13]. therefore most systems use λ-lifting. An additional advantage of λ-lifting is that it transforms formulas into ones that are close to the originals. Therefore we chose to use the incomplete λ-lifting making the TFA output more readable.

In higher-order logic, a formula is a term of type *bool* and naturally sub-formulas are sub-terms of type *bool*. Let $abs = \lambda x_1 \dots x_n.\ t[x_1 \dots x_n, v_1 \dots v_m]$ be the most top left lambda-abstraction in a formula f, where t is a term with variables $x_1,\dots x_n$ bound in abs and $v_1,\dots,v_m$ free in abs. Let $P[abs]$ be the smallest sub-formula containing abs. Keeping in mind that free variables may be captured, this sub-formula is locally rewritten in f to:

$$P[abs := Abs'] \wedge \forall v1 \dots v_m x_1 \dots x_n.\ Abs'\ x_1 \dots x_n = t[x_1 \dots x_n, v_1 \dots v_m]$$

where $Abs' = Abs\ v_1 \dots v_m$ and Abs is a fresh symbol in the whole problem. There a few variations of this approach: instead of abstracting a term on the subformula level, it can be done on the whole formula level [Bla12] or even on the whole problem level [Urb08]. This allows for sharing copies of the same abstraction, which is very useful in bigger problems for premise selection. As our problems so far have been rather small, we have refrained from such optimizations.

Example 4.1. (Free variable captured)

$$\forall v.\ P\ (\lambda x.\ x + v) \ \rightsquigarrow\ \forall v.\ P\ (Abs\ v) \wedge \forall x\ v.\ (Abs\ v)\ x = x + v$$

4.2.4 Nested Formulas

In higher-order logic predicates are identified with terms of type *bool* and can appear as arguments of functions and other predicates. This is not possible in the first-order TPTP formats and this step rewrites the formula in order to collapse all formula levels, effectively removing formulas as arguments of non logical operators.

The most common approach for transforming such formulas into FOF is to name each sub-formula with a boolean variable and substitute all occurences of this sub-formula by this variable and add the variable's definition to the problem. This is commonly used in transformations to first-order logic, as it results in smaller problems [KU14]. When translating to Beagle the isomorphic variant of $\forall x : \$o.\ x = T \ \vee \ x = F$ cannot be provided to the ATP, because it leads to a non-terminating proof search, therefore in our translation we perform disjunctive cases on sub-formulas used as arguments which is costly as it creates two copies for each of them.

Let t be the most top left sub-formula used as argument in a formula f. Let $P[t]$ be the smallest sub-formula containing the formula t. This sub-formula is locally rewritten in f to:

$$(t \Rightarrow P[t := T]) \wedge (\neg t \Rightarrow P[t := F])$$

This translation leaves only the boolean T or F as possible arguments. Since the type returned by predicates $\$o$ is reserved in TFA, we create an isomorphic type *bool* and isomorphic booleans *btrue* and *bfalse* complemented by the axiom $btrue \neq bfalse$.

4.2.5 Defunctionalization

In higher-order logic, the same function (either constant or variable) can be applied with different arities. Defunctionalization transforms such problems into equivalent problems, where each function has a fixed number of arguments. In order to minimize defunctionalization the problem is first rewritten to a clause set. This frees existentially quantified functions which reduces the number of necessary applications of defunctionalization.

The process of defunctionalization introduces an apply functor. In our implementation we use a separate functor *App* for each type. In higher-order logic each functor is equivalent to identity, making it possible to rewrite using the equation $f\ x = App\ f\ x$, so that every function symbol f is used as an argument in the translated formula. This transformation can be performed extensively, making the transformation complete (together with the extensionality principle for each type). This makes the problems quite inefficient.

A commonly used variant (Sledgehammer, HOL(y)Hammer) is to preserve the function symbol if it is free and used with its lowest arity relative to the whole problem. In order to be as close as possible to the complete version we additionally perform defunctionalization on constants which share the type with a universally quantified variable. This procedure is quite effective, however, it cannot be performed for arithmetic functions as it would prevent us from mapping them to their TFA counterparts. The way partially applied arithmetic functions could be treated, is by adding definitions of such constants in terms of the TFA ones. For the unary minus operation this would amount to the reflexive HOL equation $uminus(x) = uminus(x)$ translated to the FOL equation $App(uminus, x) = \$uminus(x)$. This is analogous to the way partially applied logical operators are translated by Sledgehammer. In Sledgehammer such partially applied logical operators occur very rarely, and adding such encoding more likely hinders the proof, therefore we did not implement such transformation for partially applied arithmetic so far.

4.2.6 Linear Integer Arithmetic

The translation of the integer arithmetic of HOL4 to the TFA format is straightforward. Indeed, the TFA predicates $\$less, \ldots$, and functions $\$sum, \ldots$ can be directly mapped to the constants $<, \ldots$ and $+, \ldots$ used with the same arity. Partial application of an arithmetic operator and non linear multiplications are left uninterpreted in this evaluation. Since natural numbers are not supported in the TFA format, we inject them into integers using these rewrite rules:

$$\forall x : num.\ F[x] \ \rightsquigarrow\ \forall x' : int.\ (x' \geq 0) \Rightarrow F[x']$$

$$F[x : num] \ \rightsquigarrow\ F[x'] \ \wedge (x' : int \geq 0)$$

$$F[f] \ \rightsquigarrow\ F[f'] \wedge \forall (x' : int)\ y.\ (x' \geq 0) \Rightarrow f'(x', y) \geq 0$$

The third rule is only presented with a function f of arity 2 having one numeral as its argument. A generalization of this rule is applied for every function returning a numeral. There is one difference between the translation used in Sledgehammer for SMT-solvers and ours. We use the second rule only for free variables and we add the third rule to complement it, making the translation more complete and compact but leaving more work for the prover. Rational and real linear arithmetic are also supported by Beagle and a similar mapping could easily be added.

4.3 Experiments

All tests were performed with Beagle version 0.7 and HOL4 repository version on a dual-core processor 2.1 GHz CPU with 3.7GB RAM. Beagle's timeout was set to 15 seconds per goal. We consider all HOL4 standard library goals that have been solved by the tactic METIS_TAC. We recall that METIS_TAC calls a first-order prover based on resolution without any theory reasoning; as a consequence, it must be fed with the theory lemmas that are needed to solve the goal. On the contrary, BEAGLE_TAC calls Beagle which does handle linear integer arithmetic, and is consequently left without any hint for this theory.

In the first experiment, we test BEAGLE_TAC on 271 of these goals without providing any arithmetic lemmas. To measure the impact of the pseudo-monomorphization procedure (sec. 4.2.2), we launch BEAGLE_TAC with and without monomorphization. During the test without monomorphization, polymorphic lemmas are left uninstantiated. Thus, polymorphic types are mapped to newly created monomorphic types. The results presented in table 4.1 demonstrate that we could solve 81% of Metis provable problems.

Since Beagle is not complete, it can give answer "Unknown" for the problems which it cannot prove or disprove. For the problems that Beagle can disprove, it answers "Satisfiable". Such problems arise, since our translation is incomplete. The Table 4.2 compares the time taken by BEAGLE_TAC (split into translation time, problem writing time, and time taken by Beagle) with the time taken by METIS_TAC. The tactic METIS_TAC is a lot faster than BEAGLE_TAC. The translation takes more time than

Without monomorphization				With monomorphization			
Unsat.	Sat.	Unknown	Timeout	Unsat.	Sat.	Unknown	Timeout
70 %	15%	7%	8%	81 %	2%	8%	9%

Table 4.1: Percentage of goals solved by BEAGLE_TAC

METIS_TAC but Beagle is the limiting factor. This is mostly due to the weakness of our translation. We have to recall that unlike Metis, Beagle has to perform the arithmetic reasoning itself (without any arithmetic axioms). We will compare the impact of arithmetic reasoning below.

BEAGLE_TAC	Translation	Writing	Beagle	METIS_TAC
4.55	0.82	0.18	3.55	0.11

Table 4.2: Mean time in seconds

We will now compare the efficiency of BEAGLE_TAC on different classes of problems, by splitting our problem set into categories and comparing the number of solved problems and the average solving time depending on the category.

Higher-order Problems In Table 4.3, the performance of BEAGLE_TAC is measured on first-order and higher-order problems. Despite BEAGLE_TAC's reasonable performance on first-order problems, it solves only half of the higher-order problems. A lot could be done to improve the supports for higher-order in our translation. The processes of defunctionalization and boolean instantiations can increase dramatically the size of the formulas and λ-lifting lacks completeness.

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
first-order	91.9%	2.71	2.48	0.13
higher-order	55.8%	11.07	7.36	0.04

Table 4.3: Higher-order efficiency and run-time in seconds

Polymorphic Problems In Table 4.4, we evaluate the effectiveness of the pseudo-monomorphization procedure. The proportion of problems solved are similar for monomorphic and polymorphic problems and the time taken by our translation does not change, which means that our heuristic for instantiating polymorphic types is efficient and well-suited to Beagle's capabilities.

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
Non-arith.	81.6%	5.91	4.15	0.08
Arithmetic	79.6%	3.62	3.15	0.13

Table 4.5: Arithmetic efficiency and run-time in seconds

	Proportion	BEAGLE_TAC	Beagle	METIS_TAC
Monomorphic	81.1%	5.83	4.28	0.12
Polymorphic	79.9%	3.32	2.85	0.09

Table 4.4: Monomorphization efficiency and run-time in seconds

The results discussed above in Table 4.1 have already been split in two categories: with and without monomorphization. The monomorphization step enable BEAGLE_TAC to solve 11% more goals. However, we have to note that this step increases the size of the problems, which results in more "Timeout"s. Concerning the algorithm itself, a fixed point has been found in 102 out of 139 (73%) polymorphic problems.

Arithmetic Problems In Table 4.5, we separate the problems containing at least one arithmetic constant. Again, the proportion of problems solved are similar, which shows that Beagle handles TFA arithmetic well. We can further note that the time taken by BEAGLE_TAC is lower on arithmetic problems whereas the time taken by METIS_TAC increases, which indicates that Beagle's built-in arithmetic decision procedures are efficient.

Named Theorems We can reprove 258 out of 270 (95%) HOL4 named theorems involving only arithmetics and/or pure higher-order terms (containing no uninterpreted constants). Some proofs fail because the original problems did not include extensionality, so that this property was not carried out to defunctionalized functions in the TFA translated version. Other proofs fail, since we do not declare the finiteness of the boolean type. This can be expressed in first-order logic, and in fact a HOL4 theorem that expresses this property could be added to the problems.

4.4 Reconstruction

An important characteristic of interactive theorem proof systems is the fact that every derived theorem is completely certified. An unverified call to an external prover compromises this property. It could be, that Beagle or the printing phase of our translation may happen to be unsound. However, reconstructing a proof that involves external theories is a challenging task for a number of reasons. The ATP may not output which small steps it actually performs. Replaying each of the small steps may be complicated as every single step may require a lot of specialized code and proofs to be simulated. Nevertheless,

there are a number of successful examples of checking SMT-solver proofs: the integration of veriT into Coq [AFG+11] and reconstruction of z3 proofs in Isabelle/HOL and HOL4 [BW10].

We have investigated the first steps towards reconstructing the proof found by Beagle in HOL4. We have implemented a minimal proof trace functionality in Beagle and added a parser for trace in HOL4. The proof trace contains a list of clauses, as well as information about the usage of the split rule. The Beagle main loop maintains two sets of clauses, see Fig. 4.1. The old set contains clauses that have been used as a premise at least once, whereas the new set contains clauses that have been derived, but so far have not been used as a premise. Moreover, when the split rule is used Beagle creates derived loops in a DPLL way. In our output, we store every clause that is being added to the old set, together with the additional information about the current level of splitting, as well as tagging the clauses which were split. When Beagle successfully proves the conjecture, we can reconstruct a tree-like structure, where each vertex represents the application of a split rule and each edge stands for a list of clauses sorted in order of creation.

In order to successfully reconstruct all Beagle proofs, a lot more work is required, as all the rules need to be recorded and replayed. For instance, the simplification rule combines arithmetic with first-order logic reasoning, making the steps proved by this rule not easily automatically provable in HOL4. We have tried to reconstruct it by combining the application of the HOL4 arithmetic decision procedure COOPER_TAC [Nor03] and METIS_TAC, however this fails in some cases. As Beagle is still in development and prone to many updates, we decided to postpone implementing code for the reconstruction of such rules. It may be possible to reconstruct the proof in two stages: in the first stage using Beagle for lemma filtering with arithmetic as done in HOL(y)Hammer and Sledgehammer in their interaction with pure ATPs, and in the second stage turn off the arithmetic mapping and find also the necessary arithmetic lemmas to be given to Metis.

4.5 Conclusion

In this paper, we investigated the suitability of the TFA format as an interface between ITPs and ATPs, by implementing an interface between the interactive proof system

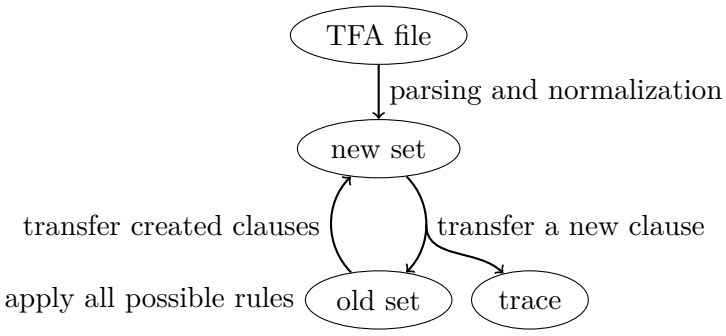


Figure 4.1: Beagle main loop augmented with the trace

HOL4 and an automated system Beagle. We tested the efficiency of Beagle on translated higher-order formulas, and showed that it could prove, without any arithmetic lemmas, 81% of the goals provable by Metis. The translation from HOL4 into a TPTP format is greatly inspired by similar translations (especially Sledgehammer), however we had to adapt certain stages of the translation to the capacities of Beagle (transformation of nested predicates, handling of polymorphic assumptions, etc.) and respecting the constraints of the TFA format. We mapped naturals and integers so that we could benefit form the linear integer arithmetic decision procedure of Beagle. Furthermore, we have introduced a minimal proof trace in Beagle, as a first step toward replaying the proof. The project helped reveal a few bugs in Beagle, which have been fixed by its authors now.

BEAGLE_TAC is more expressive than METIS_TAC, as it combines first-order logic with linear integer arithmetic, so we believe that HOL4 users could already benefit from our tactic. However many optimizations are still necessary for BEAGLE_TAC to compete with other proof methods available in HOL4. A mapping for real numbers and rationals should be provided in order to evaluate the capabilities of Beagle for these domains. We expect that like in other experiments an external ATP/SMT can outperform an internal one as we increase the number of lemmas, so a proper evaluation with a larger number of premises should be carried out. In a different line of work our translation could directly print its results to the TDFG format, so as to compare the strength of Beagle and SPASS+T. A long term future work is creating a "hammer-system" for HOL4 that would implement a relevance filter combined with ATP-based premise selection. This later project could also be combined with the bridge from HOL4 to z3.

Acknowledgement

Peter Baumgartner and Josh Bax explained us the internals of Beagle and helped us find an issue in the initial version of our translation; Alexis Saurin reviewed a previous version of this paper; Jasmin Blanchette provided us with many useful references.

Chapter 5

Aligning Concepts across Proof Assistant Libraries

Abstract

As the knowledge available in the computer understandable proof corpora grows, recognizing repeating patterns becomes a necessary requirement in order to organize, synthesize, share, and transmit ideas. In this work, we automatically discover patterns in the libraries of interactive theorem provers and thus provide the basis for such applications for proof assistants. This involves detecting close properties, inducing the presence of matching concepts, as well as dynamically evaluating the quality of matches from the similarity of the environment of each concept. We further propose a classification process, which involves a disambiguation mechanism to decide which concepts actually represent the same mathematical ideas.

We evaluate the approach on the libraries of six proof assistants based on different logical foundations: HOL4, HOL Light, and Isabelle/HOL for higher-order logic, Coq and Matita for intuitionistic type theory, and the Mizar Mathematical Library for set theory. Comparing the structures available in these libraries our algorithm automatically discovers hundreds of isomorphic concepts and thousands of highly similar ones.

5.1 Introduction

5.1.1 Context

With the diversity of interactive theorems provers [HUW14], the lack of interoperability is a growing issue. Formalized proofs originating from one prover are hardly reusable in a different one. Discovering and identifying the structures that occur in multiple libraries becomes an important step to better interoperability as the libraries of theorem provers grow.

The benefits of links between different structures have since long been known by mathematicians [Cor12]. Algebraic structures such as fields [Rot10] enable mathematicians to transport properties from real to complex numbers. Moreover the whole field of category theory has been about generalization [Awo06] with recent techniques such as

classifying a topos of a theory as very powerful transfer mechanism [Uni13]. In computer programming, oriented-object languages [Mey88] can share a method across many object instances using inheritance. Both examples shows how an interconnected structure is beneficial for better insights and faster development.

To this end, we develop an algorithm that automatically evaluates the similarity between formalized concepts (units of thought). This is achieved by inferring the mathematical properties they possess, which is a reflection of the structure they describe or belong to.

5.1.2 Challenges

Aligning libraries comes with a set of challenges. The mere fact that common mathematical structures have been (re-)formalized in each proof assistant makes this initiative conceivable.

The first difficulty is to express the mathematical properties uniformly. The multiplicity of the logics of the studied provers make this step quite complicated. Indeed, they have often different degree of support for lambda-abstractions, polymorphism, type classes, type hierarchies, algebraic hierarchies, etc. Those features produce some idiosyncratic constructions in the formal developments in each prover.

The next step is to define and recognize which mathematical concepts appear in the library. There may be for instance types, constants, subterms, formula subtrees or even proof tactics. Our goal will be to define what are the unit concepts and which ones are a combination of those concepts. Another issue is that some concepts are defined many times inside one library. Indeed different integer representations can be more suitable for some applications (like code extraction [HKKN13]). Conversely, a concept can belong to many different structures. It is especially common in the traditional set theoretic approach, where the empty set $\varnothing$ also stands for the natural number 0. This is realized by most formalizations of set theory, for example in the foundations of Mizar [GKN10] and Isabelle/ZF [Pau16].

Having delimited our notion of "concepts", we wish to derive their similarities. A uniform representation for the properties makes it easy to infer which concepts share the same properties. We would like to emphasize here that the approach is more effective and more comprehensive than looking only at their definitions. Already for minimally different definitions, recognizing that they represent the same concept is not straightforward. This becomes very hard when definitions are foundationally different, for instance the real numbers may be defined through Dedekind cuts or Cauchy sequences. Moreover, the similarity measure may indicate for example the discovery of the underlying ring structures of integers and real numbers, which would not be possible if we restrict to the discovery of perfect matches only. Furthermore, the context in which the concept is expressed can be essential. To capture its influence, we also study the interconnections between properties inside a library that allow finding similar relations between concepts in different libraries.

We hope that solving these issues will create libraries of alignments suitable for the different types of applications envisioned.

5.1.3 Applications

The principal application of our work is transferring theorems between libraries. Deep embeddings [JBDD15] are typically best-suited to check the soundness of the provers and prove meta-theorems about the system studied. Yet, the imported theorems are difficult to integrate with the current developments since they are created at different logical levels. Therefore, shallow embeddings [MD14] that can be obtained through reflection [KW10] are preferred. Even then, if no concept mapping is performed, the potential risk is to create parallel developments on the same set of concepts. And additionally to the unnecessary repetitions, the equivalence between the two sets would have to be proven, which may not be possible. From our discovered alignments, it is possible to lift the set of mappings found to theorems and proofs. This yields a possible translation that can be used to import a library into another system in a sensible manner by reusing the common concepts. Still, each translated theorem needs to be derived in the other prover and porting proofs is a difficult process due to the possible differences in definitions. In our previous work [GK15b], we relied on the matching algorithm to transfer proof knowledge between two HOL systems and evaluated how it improved the success of a machine-learning based proving framework HOL(y)Hammer [KU14].

The second set of applications arises from the fact that our alignment procedure could also be used effectively within a single library. A first practical use is the removal of duplicate constants and theorems. Another possibility is to hyper-link similar objects to create a better proving environment. Moreover, the properties shared by similar concepts can be combined into a structure and the concepts made instances of this structure. In the case of types, this could lead to a possible refactoring of the type hierarchy present in the system which could be essential to share proofs across different domains. Proof assistants attempt to maximize sharing. This idea is most visible in proof assistants based on type theory such as Lean [dMKA+15] where its automation relies on its library structure. But it has been at the basis of one of the earliest proof assistant Mizar [BBG+15]. Using fuzzy mappings inside one library, initial experiments on the possibility of producing new conjectures from analogues of theorems of a related domain were performed in [GKU16]. Finally, various proof refactoring techniques [WADG11, DWA13, Kle14] rely on similarities between concepts, such as these found here. Also refactoring may benefit from the patterns in the formalizations revealed by our algorithm.

5.1.4 Contributions

This paper is an extended version of our work presented at CICM 2014 [GK14] which introduced a simple concept matching algorithm for a single foundation (higher-order logic). In this paper we present many extensions of this work, which allow much better automatic discovery of isomorphic and similar structures in multiple formal mathematical libraries based on different foundations. Specifically the contributions of this work are:

- We design a fixpoint algorithm with various scoring functions for automatic discovery of similar concepts and theorems in and across proof assistant libraries. We find

thousands of mappings between concepts. These include one-to-many mappings where concepts are related to multiple counterparts.

- We build properties and concepts from the objects of formal libraries: theorems, constants and subterms. During this step, we experiment with various degrees of normalization and an optional conceptualization of subterms, as well as different level of type inclusion.
- We evaluate the proposed approaches on the libraries originating from 6 interactive theorem provers based on different foundations including set theory and type theory. We translate them into a common representation, manually aligning the term representations of the different logics.
- We give an interpretation of the correlation between matches used in our fixpoint algorithm, and show that it can also be a key idea to produce sensible mappings for formulas.
- We investigate the possibility of using an intermediate library as a translation between two libraries by constructing transitive matches.
- We define various degree of subjective similarities given by the mappings. The highest degree is defined as an optimal matches. It happens when the two related objects represent the same object conceptually.
- We describe a classification algorithm that decides which matches are optimal. We produce hundreds of such optimal matches for each pair of libraries.

5.1.5 General Principles of the Algorithm

Our algorithm takes as an input the objects of two proof assistant libraries. These objects are types, constants and theorems. We do not consider the proofs. The aim of the algorithm is to recognize the constants (including types), or more generally subterms, representing the same (and/or related) concept occurring in different libraries. We will use properties such as associativity or nilpotence, extracted from the term representation of theorems to evaluate the similarity of two constants. A general guideline is that the more properties two constants have in common the more similar they are. In addition to this main idea, relations between similarity pairs together with a number of heuristics help refine the accuracy of our similarity measures. This means, that we will use a self-improving mechanism, called dynamical scoring, where each similarity pair is influenced by the similarities of other pairs. For instance, the strength of a matching between $<$ and $\subset$ is correlated with the degree of similarities of the constants 0 and $\varnothing$. The result is a list of pair of constants, sorted by their similarity scores. On top of that, a procedure can be applied to decide if the best scoring match should be in the final mappings. This procedure relies on additional techniques such as disambiguation or type coherence. If the choice is not delayed, the score of the match is raised or diminished according to the decision. This in turn influences further applications of the dynamical scoring algorithm. An overview of the proposed procedure is presented in Fig. 5.1.

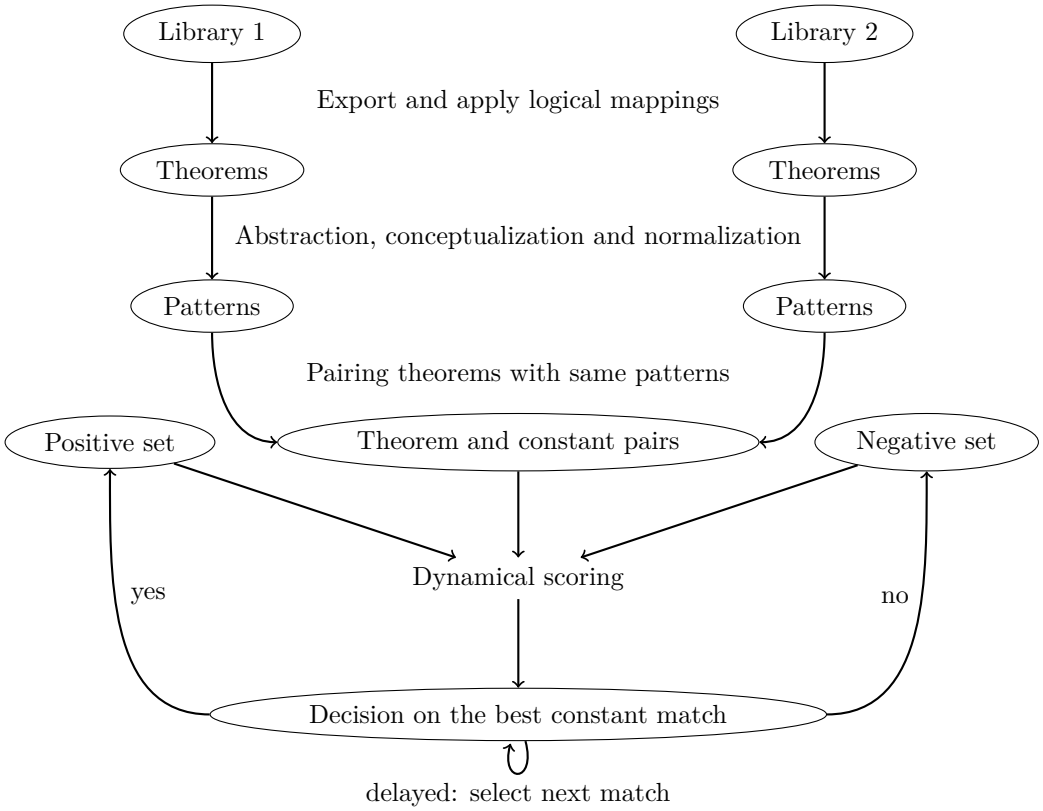


Figure 5.1: Workflow graph

5.1.6 Plan

The rest of this paper is organized as follows. In Section 5.2 we explain the process of creating concepts and properties inside one prover. We describe how we match properties and deduce similarity scores between concepts in Section 5.3. In Section 5.4 we evaluate each step of our approach. We next describe additional techniques applied on top of the scoring procedure that improve the quality of our results (Section 5.5). We discuss related work in Section 5.6. In Sections 5.7 and 5.8 we conclude and present an outlook on the future work.

5.2 Creating Properties and Concepts from Theorems

The only prerequisite of our algorithms, is a common term representation of theorems in the considered proof assistant libraries. This requirement is immediately satisfied when considering matching of concepts in different formalizations or proof libraries of one system, but it may be harder to satisfy for libraries or proof assistants based on different foundations. We focus on the term structures, or the syntactic structures rather than the the semantics of the formulas in order to work across the different logical foundations.

The term structures are exported from the internal term representation in each prover and thus contains implicit arguments and coercions that are not visible in the external syntax. This additional information makes detecting alignments more challenging but produces more precise mappings.

We believe that aligning concepts using this approach rather than providing a deep embedding is more appropriate. Indeed, the proof assistants have been meant to help proof developers and therefore their syntax is usually designed to correspond to standard mathematics. This also implies that the libraries created by the users often state theorems in a similar way. Moreover, each formal proof library is completely self-sufficient which means that basic types, such as integers, real numbers, or sets, are likely to be defined in all the proof libraries, which asserts that certain concept alignments do exist. Since the logical operators are often tied with intricacies of the logic, they usually need to be recognized and mapped manually (see Section 5.4.1). In all our experiments we will assume that the representation of the terms corresponds to a version of type theory that includes the basic predicate logic and equality. Therefore, we manually recognize the constants $\forall, \exists, \Rightarrow, \neg, \wedge, \vee, \Leftrightarrow, =$. We additionally map the the type of propositions, which we will denote as $\$o$ and the type of all types $\$t$. The names $\$o$ and $\$t$ are the TPTP [SSCB12] notations used for the two types. These mappings actually collapse the type hierarchy. This however has no consequence for the algorithm: the intent is to discover concept similarities, and only proving or disproving their equality requires a sound system. Furthermore, even if the found concept pairs are not equivalent, but only similar, such pairs are still often useful.

With a common representation of theorems, we can identify the theorems that are instances of the same property. Since two statements may represent the same property even if they are presented in different forms, we normalize the statements of the theorems. Furthermore, the found properties should not depend on the name of the constants present in the statement, therefore constants should be abstracted after normalization. From this intuition, we now give a formal definition of a property.

Definition 5.1 (Property)**.** Given a set of terms $\mathcal{T}$, and a normalization method $N : \mathcal{T} \to \mathcal{T}$, a property P of a theorem $T \in \mathcal{T}$ is defined by:

$$P =_{def} \lambda C_1, \ldots, C_n.\ N(T)$$

where $C_1, \ldots, C_n$ are the non-logical constants appearing in $N(T)$ ordered by a left outermost traversal of $N(T)$. Two properties will be said equal if they are α-equivalent.

Definition 5.2 (Derived matchings)**.** Two theorems T_1 and T_2 which share the same property P are called a matching pair of theorems. Let (T_1, T_2) be a a matching pair of theorems with normalized forms $N(T_1) = P(D_1, \ldots, D_n)$ and $N(T_2) = P(E_1, \ldots, E_n)$. The matching pairs of constants $(D_1, E_1), \ldots, (D_n, E_n)$ are induced by the pair (T_1, T_2). We will also say that two constants D and E have the same property if they occur at the same position in two equal properties. The similarity of D and E will be measured by the number and quality of these properties.

Remark 1. The distinction made in our previous work [GK14] between patterns of theorems and properties of constants is now subsumed by this single definition. Patterns of theorems are now also called properties. The distinction between different positions inside a property is now defined implicitly by the process of inducing pairs of constants. These changes lead to a much more concise description and a significant gain in memory and speed for our algorithms.

Example 5.3. Given the below theorems T_1 and T_2, their respective normalizations, and the properties extracted from their statements:

$$T_1 :\ \forall x : \mathit{num}.\ x + 0 = x \qquad T_2 :\ \forall x : \mathit{real}.\ x = x \times 1$$

$$N(T_1) :\ \forall x : \mathit{num}.\ x = x + 0 \qquad N(T_2) :\ \forall x : \mathit{real}.\ x = x \times 1$$

$$P_1 : \lambda \mathit{num}, +, 0.\ \forall x : \mathit{num}\ x = x + 0 \qquad P_2 : \lambda \mathit{real}, \times, 1.\ \forall x : \mathit{real}.\ x = x \times 1$$

The properties P_1 and P_2 are α-equivalent, therefore the theorems T_1 and T_2 form a matching pair of theorems, and the following three matching pairs of constants are derived:

$$\mathit{num} \leftrightarrow \mathit{real},\ + \leftrightarrow \times,\ 0 \leftrightarrow 1$$

The purpose of a normalization method is to maximize the number of shared properties without sacrificing the characteristics of each individual one. This is typically done by rewriting the theorems into a normal form and by extending the types of the considered concepts. We will first focus on the rewriting based methods: computing conjunctive normal forms, reordering commutative and associative-commutative connectives, and normalizing subterms. Then we will discuss different levels of typing that can be applied and how they interact with the normalization methods.

The effect of the rewriting based methods is illustrated on a running example. For clarity, type information is omitted, constants are not abstracted.

Example 5.4. (Running) The constants $\times$, *alt_pi*, s, 0, *cos* and fst respectively stand for multiplication, π, successor, zero, cosine and projection on the first argument.

$$\forall y\ x.\ x = \mathit{alt\_pi}\ \times (s\ (s\ 0)) \Rightarrow \mathit{cos}\ x = \mathrm{fst}\ 0\ y$$

5.2.1 Conjunctive Normal Forms

First, we split the theorems into separate conjuncts even when they appear under quantifiers. Each conjunct can be considered as separate theorems from this point. We then rewrite every theorem statement to conjunctive normal forms. In this normalization we assume classical logic, however if this is not desired, it is possible to consider intuitionistic clausification [Ott05]. As the focus is on proof libraries that can be expressed in type theory, we preserve the equivalences $\Leftrightarrow$ and consider them as equalities between propositions.

Example 5.5. (Running) $\forall y\ x.\ \neg(x = \mathit{alt\_pi} \times (s\ (s\ 0))) \vee \mathit{cos}\ x = \mathrm{fst}\ 0\ y$

5.2.2 Subterms

Certain concepts are declared as a constant in one proof library but left as a construction over simpler concepts in another. The number 1 can be defined as a single constant *one* (in Proofpower), by the successor of zero $S(0)$ in all the libraries that use a unary representation of numbers or by a binary representation (for examle $BIT_1(0)$). In some cases it is only clear from the context, whether a certain subterm is supposed to represent a single constant or a more complex construction. Consider the HOL4 type $prod(real, real)$. It is used to represent the complex numbers as well as pair of reals [GK15b].

Matching of whole subterms also enables us to automatically factor type arguments. Type arguments are usually the first arguments of a function, therefore thanks to curryfication we can find subterms that represent type instances of constants. This approach works even with the type classes of Isabelle/HOL. For instance, in the experiments we will discover that the subterm *zeronat* in Isabelle/HOL is similar to the constant 0 of type *num* in HOL Light.

Because it is impractical to consider all possible subterms as a concept, we will impose some practical restrictions. First, the selected subterms must be formed from function applications and constants only, in particular they do not contain any variables. Second, we only select a subterm if it appears sufficiently frequently in a proof library. Moreover, a subterm is more likely to represent a concept if it is smaller and the conceptualization of big subterms reduces the complexity of the pattern. So, a simple heuristic is to check whether the subterm appears in a number of theorems greater or equal to two times its size.

Every time a subterm substitution is applicable, we duplicate theorem statements. The substitutions replace the selected subterms by newly defined constants. A simple type inference mechanism on a subterm is used to determine the type of its defined constant, in case types at constant positions are required. We will only use maximal substitutions, where a maximal number of replacements is performed with the one with the largest subterms applied first, since they are the most likely to produce new matching pairs of theorems.

Example 5.6. (Running)
By creating a new constant definition $c =_{def} s\ (s\ 0)$, we obtain:

$$\forall y\ x.\ \neg(x = alt\_pi \times c) \vee cos\ x = \text{fst}\ 0\ y$$

Since different substitutions may lead to different pairs, it would be also interesting to consider matching large sets of terms with many possible substitutions. As we do not focus on one representation of terms, the applicability of various term indexing techniques [Gra96, RSV01, Sch13a] for arbitrary proof assistant terms is left open.

Equivalent Concepts In an effort to minimize the number of equivalent concepts inside one library, we identify constants that are in the same equivalence class of the equality relation. In practice, it requires recognizing theorems of the forms of $c_1 = c_2$ in order to extract an equality relation and replacing constants of an equivalence class by a new

constant representing them. The process of conceptualization of subterms is performed before the construction of the equality relation. Therefore constants representing subterms will also be identified with members of their equivalent class.

Example 5.7. (Running)
Relevant equalities found in the library after conceptualization: $2 = c$, $\pi = alt\_pi$.
Substitution by a unique representative of the equivalent class induced by the equalities:

$$\forall y\ x.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

5.2.3 Associativity and Commutativity

Rewriting terms modulo associativity and commutativity in the higher-order setting has been studied by Walukiewicz [Wal98]. The simplifiers of certain proof assistants, including HOL Light and Isabelle, implement procedures for normalizing terms modulo AC as part of their simplifiers. The proof checker Dedukti [DHK03] allows reasoning modulo equations [Bla03] which can include AC, therefore contains an algorithm for normalizing $\lambda\Pi$ terms modulo such equations.

The requirements of our algorithm are slightly different than of these above. The usual requirement is to reduce α-equivalent terms to the same normal forms given a number of AC rules and a total ordering $<_{ord}$ on ground terms. This order can be constructed by comparing top constructors and in case of equality recusively comparing subterms. In our context, however, the names of the non-logical constants (and variables) should not influence the term ordering. This is because the properties must be independent of the names of the used constants. Therefore the AC normalization procedure works on abstracted terms where different abstraction symbols are used for constants and types.

The names of constants are abstracted in the theorems, but they are important in the induced pairs of constants. In particular, when both arguments of a commutative constant are α-equivalent, the term ordering cannot compare the two possible orderings of the whole term. Consider for example the theorem statement $g(x) = h(x)$, where g and h are constants and x is a variable. Then in theory we would need to create both versions. As this may be explosive in case of large applications of AC connectives, we have not implemented this yet.

1. No normalization, even the order of subterms applied to the logical constants is preserved.

2. Normalization based on AC of the logical constants only. For this ordering:

$$\lambda x.\ (x = \pi \times 2) <_{ord} \lambda x.\ (\pi \times 2 = x)$$
$$\lambda x\ y.\ cos\ x = \text{fst}\ 0\ y <_{ord} \text{fst}\ 0\ y = \lambda x\ y.\ cos\ x$$
$$\lambda x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y <_{ord} \lambda x\ y.\ cos\ x = \text{fst}\ 0\ \vee \neg(x = \pi \times 2)$$
$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y <_{ord} \forall y\ x.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

We get by applying rewrites starting from the deeper subtrees (innermost):

$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ 0\ y$$

3. The set of all constants that have an associative or a commutative property in the corresponding proof library. This become imprecise in higher-order foundations, where AC properties may be stated using higher-order predicates (for example $associative(+)$). We get by applying rewrites starting from the deeper (innermost) subtrees: The constant $\times$ is commutative, but since constants are abstracted the ordering cannot distinguish between $\pi \times 2$ and $2 \times \pi$. So the running example is left unchanged.

4. Finally we add the commutativity of all constants. This means that given a function, the procedure reorders its arguments (possibly more than 2) so that the resulting term is minimal in the term ordering. If $\lambda y.\ fst\ y\ 0 <_{ord} \lambda y.\ fst\ 0\ y$ then the normalized form of the running example becomes:

$$\forall x\ y.\ \neg(x = \pi \times 2) \vee cos\ x = \text{fst}\ y\ 0$$

The last normalization may seem strange at first as it creates an inconsistent normalization. However, it does allow for matching concepts which are stated with differently ordered arguments in different libraries.

5.2.4 Typing Information

In order to find more matches, we also try to normalize the type information across the different proof libraries. We consider four levels of typing available before term normalization and we depict their effect on the following example.

Example 5.8. (Typing) In this statement, the constants *hd*, *list*, *int* and 0 respectively stand for head, type constructor for lists, integer and zero.

$$\exists\ l : list\ int.\ hd\ int\ l = 0$$

1. Type erasure. The type matches can be recovered using the types of the constants involved in a matching and applying type coherence (see Section 5.5.1).

$$\exists\ l.\ hd\ int\ l = 0$$

2. Simple types. We create a simple type (one constant) for each unique formula occurring on the left of a type judgment. This approach can be useful if we consider their types, for example when matching constants with dependent types of Coq against set theory constants typed using the Mizar soft type system. Given a new constant definition $d =_{def} list\ int$, the typing example normalizes to:

$$\exists\ l : d.\ hd\ int\ l = 0$$

3. Variable types. Including the types of all variables is enough to recover all types in simple type theory; however it is not enough to recover all types in more intricate type systems.

$$\exists\, l : \mathit{list}\ \mathit{int}.\ \mathit{hd}\ \mathit{int}\ l = 0$$

4. Constant types. This combines the previous approach with the types of constant at each positions inside the terms. In this last typing example, $t is written t for simplicity.

$$\exists\, l : (\mathit{list} : t \Rightarrow t)\ (\mathit{int}{:}t).\ (\mathit{hd} : (\forall a{:}t.\ (\mathit{list} : t \Rightarrow t)\ a \Rightarrow a))\ (\mathit{int}{:}t)\ l = (0{:}(\mathit{int}{:}t))$$

The proposed type levels of typing are recursive, which means that the types are themselves constants whose types are also included in the formula until a defined type, including the basic types of propositions $\$o$ and types $\$t$, is reached. The later typing levels are available only to the proof libraries where they are meaningful. In our case study, this implies that the fourth typing information level is not available for Mizar. In Mizar's soft type system a term does not have a unique type, and checking whether a term belongs to a type does require theorem proving. It would be possible to make use of the cluster-rounding algorithms implemented by the Mizar checker [Try07], even so with terms belonging to many types, this would not match to any of the other considered proof libraries so far.

Using the different levels of typing information increases the accuracy of patterns and allows the use of more precise settings, which limit the number of ambiguous matches. However, additional typing information increases the number of missed matches, that would require type alignments not detected by our algorithm.

5.3 Similarity

Pairs of constants have an intrinsic similarity based on the number of properties they share as well as the quality of the theorem pairs that created those properties. Various heuristics helps us value the similarity of these pairs of theorems. The most important heuristic for a pair of theorems is the quality of the induced pairs of constants. The correlations between pairs of concepts is captured by our scoring functions. Applying these functions iteratively result in a dynamical system, where the confidence on a pair of concepts evolves relative to the influence of the other pairs. The propagation of this effect stops when the system reaches a stable state, which it always does as demonstrated in Section 5.3.6. When a stable state is reached, the similarity between concepts should correspond to inherited higher scores.

5.3.1 Sets of Pairs

Before we introduce concrete pair scoring functions, we discuss how the pairs are computed in order to explain the motivation for the functions.

Given two proof libraries lib_1 and lib_2, we first compute all the properties. We next create all possible theorem pairs, by considering all the properties which appear in both libraries. For each property P we apply the same pairing mechanism. Given n_1 theorems from the library lib_1 representing this property, and n_2 theorems from the library lib_2 which represent the property P, we create $n_1 * n_2$ theorem pairs by considering the Cartesian product of the two sets. In practice, the number of theorems sharing a property is small compared to the size of the library(see Section 5.4.2). Therefore the observed time complexity is less than quadratic with respect to the number of theorems in the library.

The list of all pairs of theorems is named $(t_w)_{1\leq w\leq m}$.

We then compute the set of induced pairs of constants from each pair of theorems. The union of these sets is the set of all pairs of constants for our library pair (lib_1, lib_2). The list of all pair of constants is named $(c_v)_{1\leq v\leq n}$.

Remark 2. Notice that the variables t and c stand for pairs of theorems and constants and not single theorems and constants. We will rarely need to access the components of the pairs, and we will explicitly refer to the first and second component of the pairs in such cases.

5.3.2 Scores

We define the scores of pairs of objects recursively by:

$$score_t(t_w) = coef_t(t_w) \times \sum_{i=1}^{n} \delta(c_i, t_w) score_c(c_i)$$

$$score_c(c_v) = g(coef_c(c_v) \times \sum_{j=1}^{m} \delta(c_v, t_j) score_t(t_j))$$

where δ is the characteristic function of the relation R "is induced by" defined in Section 5.2:

$$\delta(c, t) = 1 \text{ if } c \ R \ t \text{ and } 0 \text{ otherwise,}$$

g is a normalization function from $\mathbb{R}^+$ to $[0; 1[$:

$$g(x) = \frac{x}{x+1}$$

and $coef_t(t_w)$,$coef_c(c_v)$ are coefficients based on heuristics defined and justified in Section 5.3.3.

In the following, we will use *score* and *coef* when it is clear from the context which of the scores and coefficients are meant.

The function g guarantees the convergence of the vector sequence generated by the algorithm (see Section 5.3.6). It also reduces the difference between good and very good values providing a smoothing of the scores. This function has a similar role as the sigmoid in backpropagation neural networks [Hec88].

5.3.3 Heuristics

The principal reasons for the choice of the following heuristics for coefficients is simplicity, trials and errors, and inspiration from the TF-IDF [Jon04] heuristics for finding the most relevant words in a text. Our heuristics include the coefficients for the *score* functions that scale the sum of the scores of their dependent objects (scores of theorems depends on score of constants and conversely). The coefficient for theorem pairs is composed of two parts, where the first part estimates the quality of the property represented by the pair and the second part corresponds to the recursively considered constants.

In order to estimate the quality of a property, we first compute the frequency of the property $P(t_w)$ of the pair t_w in the set of pairs of theorems, i.e. the number of pairs of theorems representing the property $P(t_w)$:

$$freq(t_w) = card\{t'_w \mid P(t_w) = P(t'_w)\}$$

Since rarer properties should get a higher score we compose this evaluation with a decreasing function.

$$inv\_freq\_property(t_w) = \frac{1}{ln(2 + freq(t_w))}$$

where the property $P(t)$ of a pair t is the property of its first component, which is also equal to the property of its second component.

The second part of the estimation is based on the constants used in the computation of the theorem pair score:

$$ind(t_w) = card\{c \mid c\ R\ t_w\}$$

$$inv\_ind(t_w) = \frac{1}{ln(2 + ind(t_w))}$$

$$average\_const\_score(t_w) = inv\_ind(t_w) \times \sum_{i=1}^{n} \delta(c_i, t_w) score_c(c_i)$$

Remark 3. If we consider scores as probabilities, it would seem natural to take a product instead of a sum of these probabilities. However, a general guiding rule is that theorem pairs should benefit from an additional good matches in its constants pairs, and be penalized by additional bad matches. There are two possibilities of enforcing such constraints. Either we can take the sum of the scores (which we chose to do) or we can allow the product of the scores to have a value greater than 1. Given the guiding rule, the simple interpretation by probabilities does not create a representative model for our scores.

The total score of a theorem can also be computed by multiplying the inverse of the property frequency by the average constant score:

$$score(t_w) = inv\_freq\_property(t_w) \times average\_const\_score(t_w)$$

This implies that the coefficient of the theorem pair t_w is:

$$coef(t_w) = inv\_freq\_propert(t_w) \times inv\_ind(t_w)$$

We will next give the coefficient for constant scores. If two constants appear in many theorems, they are more likely to have some common properties, whereas rare constants with the same amount of properties should be advantaged. Therefore we apply the following coefficient to the scores of pairs of constants.

$$freq(d) = card\{\text{theorems containing the constant } d\}$$

$$inv\_freq\_const(c_w) = coef_c((d_1, d_2)) = \frac{1}{ln(2 + freq(d_1) \times freq(d_2))}$$

where d_1 and d_2 are the components of the pairs c_w.

This coefficient is the only part of the scoring function which requires inspecting the constants composing a pair.

The *inv_freq_property* is definitely the most important of these coefficients, which together with the correlations created by the sums comprises the core of the algorithm. The other two *inv_ind* and *inv_freq_const* are not critical.

5.3.4 A Dynamical System

In this section we will discuss the relation between our algorithm and dynamical systems, which are algorithms that iteratively refine the interrelated scores. Our algorithm starts by assigning the value 1 for each pair of constants. Next, it calculates a score for each pair of theorems which in turn gives a new score to the pair of constants. This process is then repeated until a fixpoint is reached.

Our experiments will confirm the effectiveness of this approach, however first we will present a theoretical point of view, where we consider the iterative process as a dynamical system. This will reveal more about how pairs of concepts are connected. And the study of properties such as convergence and regional uniqueness, will give us the assurances about the termination of the process and its sensitivity to initial conditions. Finally some important but non-critical conjectures will be made for a global uniqueness and the rate of convergence.

In order to distinguish the different steps of the algorithm we will add the second argument to the score functions, referring to the scores at time t as $score(x, t)$. We can now express the scores of each pairs of concepts at time $t + 1$ in function of the pairs of each concepts at time t. In the following, $(X^t)_{t\in\mathbb{N}} = (x_i^t)_{1\leq i\leq n} \in \mathbb{R}^{+n}$ stands for the series of successive vectors of scores of constants. Since we can express the scores of pairs of theorems in function of these vectors, we derive the following relation between scores of constants at time $t + 1$ and t:

$$x_v^{t+1} =_{def} score_c(c_v, t + 1) = g(\sum_{j=1}^{m} l_{v,j} score_t(t_j, t))$$

where

$$l_{v,j} = coef_c(c_v)\delta(c_v, t_j)$$

Similarily

$$score(t_j, t) = \sum_{i=1}^{n} k_{j,i} x_i^t$$

where

$$k_{j,i} = coef(t_j)\delta(c_i, t_j)$$

By linearity we obtain:

$$x_v^{t+1} = g(\sum_{j=1}^{m}(\sum_{i=1}^{n} l_{v,j}k_{j,i}x_i^t)) = g(\sum_{i=1}^{n}(\sum_{j=1}^{m} l_{v,j}k_{j,i}x_i^t)) = g(\sum_{i=1}^{n}(\sum_{j=1}^{m} l_{v,j}k_{j,i})x_i^t)$$

Given the coefficients $a_{v,i} = \sum_{j=1}^{m} l_{v,j}k_{j,i}$ we have:

$$x_v^{t+1} = g(\sum_{i=1}^{n} a_{v,i}x_i^t) =_{def} f_v(x_0^t, \ldots, x_n^t)$$

Essentially, each component at time $t+1$ is given by a linear function of the components at time t combined with the normalization function g. We name this combined function function f_v. We denote by f the compound function defined by $X^{t+1} = f(X^t)$. The linear part $A = (a_{v,i})_{1 \leq v,i \leq n}$ of f is called the correlation matrix of our system.

Remark 4. We restrict our study to non-negative scores, since f is stable in $\mathbb{R}^{+n}$ and the starting value is in $\mathbb{R}^{+n}$.

Famous examples of dynamical system include the Mandelbrot fractal which reveals the complexity of determining convergence regions, the double rod pendulum and the Lorentz attractor which show extreme sensibility to the initial conditions.

The theory of dynamical systems appears naturally in many scientific domains. Various parameters that affect the stability of such system are presented by Barbarossa [Bar11] in order to understand biological mechanisms. The relationship between dynamical systems and Markov chains, hinted by the correlation matrix that resembles a transition matrix, was studied by Attal [Att10] in the context of open physical systems.

Our system was designed to be a discrete strongly-monotone multi-dimensional non-linear dynamical system. This implies that our system is non-chaotic and posseses many other general properties of strongly-monotone systems, Hirsch [Hir88].

5.3.5 Correlations

We will now show how scores interacts with each other. Two pairs of constants are correlated if they are induced by the same pair of theorems. The number and quality of those pairs of theorems decide the strength of the correlation. This is measured by the coefficients $a_{v,i}$ of the correlation matrix (see Section 5.3.4). In Fig 5.2, a graph representing a part of the correlation matrix created by aligning Isabelle/HOL with Mizar

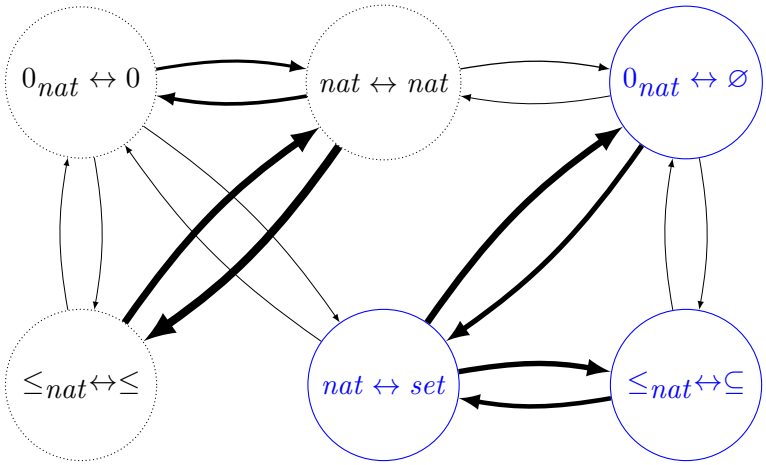


Figure 5.2: Part of the correlation graph of Isabelle/HOL-Mizar matches with stronger correlations drawn with wider arrows.

is depicted. This graph is almost symmetric. The reason for the asymmetry is that the coefficient *inv_freq_const* gives a small penalty to pairs having constants appearing in many theorems (see Section 5.3.3). Thus the influence received by a pair with rarer constants is slightly stronger. A time lapse representation of the dynamic scoring reveals how it is affected by correlations in Fig 5.3. It demonstrates that scores continuously update by taking into account changes in the other pairs through those correlations.

5.3.6 Soundness of the Algorithm

To be confident that our algorithm terminates and to determinate if the choice of the initial of the arbitrary conditions influences the result of our algorithm, we prove some properties of stability of the dynamical system namely convergence and regional uniqueness.

Proof of Convergence

Theorem 5.9 (Bounded property)**.** *The image of f is in $[0;1[^n$.*

Proof. The image of g is in $[0;1[$. Therefore the image of each f_v is in $[0;1[$ because $a_{v,i} \in \mathbb{R}^+$. The thesis follows. □

Definition 5.10 (Less or equal)**.** Let $X = (x_i)_{1 \leq i \leq n}$ and $Y = (y_i)_{1 \leq i \leq n}$ be in $\mathbb{R}^{+n}$.

$$X \leq Y \Leftrightarrow_{def} \forall i \in [\![1;n]\!].\ x_i \leq y_i$$

Remark 5. This is a partial order.

Definition 5.11 (Increasing)**.**
$X = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^{+n}$ and $Y = (y_i)_{1 \leq i \leq n} \in \mathbb{R}^{+n}$.

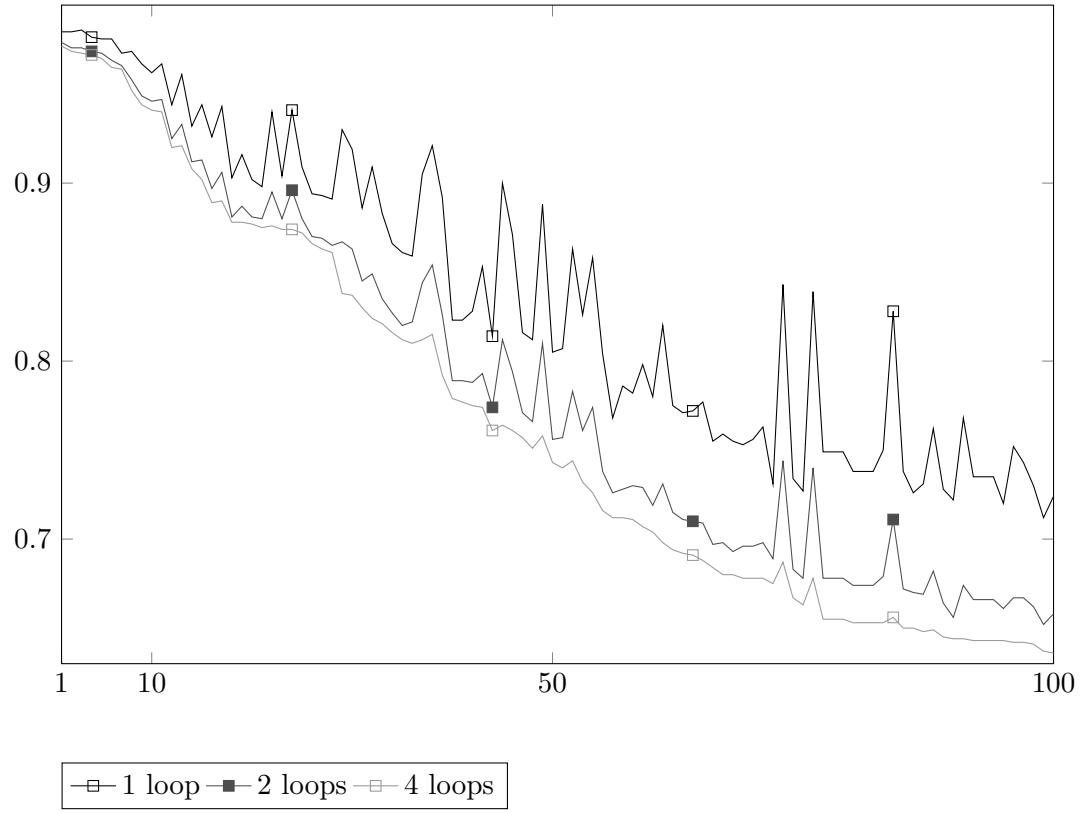


Figure 5.3: Scores of the best 100 constant pairs when matching Coq with HOL4 after 1,2,4 loops, ordered by their rank after 16 loops. The graph after 16 iterations is not presented here as it would be very close to the 4 loops curve.

A function $f : \mathbb{R}^{+n} \to \mathbb{R}^{+m}$ is increasing when:

$$X \leq Y \Rightarrow f(X) \leq f(Y)$$

Theorem 5.12 (Increasing property)**.** *f is increasing.*

Proof. The function g is increasing and $a_{v,i} \in \mathbb{R}^+$, therefore each function f_v is increasing. The thesis follows. □

Theorem 5.13 (Existence)**.**

$$(X^0 \in \mathbb{R}^{+n} \wedge X^0 \leq X^1) \Rightarrow (\exists X^{lim}.\ \lim_{t\to\infty} X^t = X^{lim})$$

Proof. By induction. We have $X^0 \leq X^1$ by assumption. Suppose $X^t \leq X^{t+1}$. Thus by the increasing property, $X^{t+1} = f(X^t) \leq f(X^{t+1}) = X^{t+2}$. Therefore, the sequence $(X^t)_{t\in\mathbb{N}}$ is increasing.

Each function f_v restricted to non-negative real numbers has its image in $[0; 1[$. By an

easy induction using the bounded property, each component of $X = (x_i)_{1 \leq i \leq n} \in \mathbb{R}^n$ and $Y = (y_i)_{1 \leq i \leq n} \in \mathbb{R}^n$ is increasing and bounded in $\mathbb{R}$ which implies that each of them converges and so does the full sequence. □

Theorem 5.14 (Existence: decreasing case)**.**

$$(X^0 \in \mathbb{R}^{+n} \wedge X^0 \geq X^1) \Rightarrow (\exists X^{lim}.\ \lim_{t \to \infty} X^t = X^{lim})$$

Proof. Analogous to the previous theorem. □

Remark 6. Although $X^0 \leq X^1$ (or $X^0 \geq X^1$) is only a sufficient condition, there exist sequences that do not converge if this assumption is omitted. 2-cycles are examples of such sequences.

Corollary 5.15. *The point $I^0 = (1, \ldots, 1)$ is converging.*

Proof. The image I^1 of I^0 is in $[0; 1[^n$ thus $I^1 < I^0$. □

Theorem 5.16 (Regional uniqueness)**.** *If there exists a sequence $(X^t)_{t \in \mathbb{N}}$ such that*

$$\lim_{t \to \infty} X^t = X^{lim}$$

then

$$\forall Y.\ X^{lim} \leq Y \leq X^0 \Rightarrow \lim_{t \to \infty} f^t(Y) = X^{lim}.$$

Proof. Since f^t is increasing and X^{lim} is a fixpoint for f:

$$f^t(X^{lim}) \leq f^t(Y^0) \leq f^t(X^0)$$

$$X^{lim} \leq Y^t \leq X^t$$

By the squeeze theorem on each component $\lim_{t \to \infty} Y^t = X^{lim}$. □

Corollary 5.17. *All points in the higher-dimensional rectangle defined by I^{lim} and I^0 converge to the same limit:*

$$\forall Y.\ (I^{lim} \leq Y \leq I^0 \Rightarrow Y^{lim} = \lim_{t \to \infty} Y^t = I^{lim})$$

Remark 7. All previous results still hold if we replace f by a function f' provided that it is defined on $\mathbb{R}^{+n}$, increasing and its image is in $[0; 1[^n$.

Remark 8. There are at most 2^n fixpoints because a simple substitution of variables creates a polynomial in one variable of degree at most 2^n. This occurs for example when all components are independent of each others. Indeed, each equation can have 2 solutions which implies 2^n fixpoints.

The uniqueness of the attracting point is important, because it ensures that for almost all starting values in $\mathbb{R}^{+*n}$, the final scores are the same. It is easy to prove that is true for $n = 1$ and we conjecture that it is true in general.

Hypothesis 1 (Global uniqueness)**.** Almost all points in $\mathbb{R}^{+n}$ converge to a unique fixpoint in $\mathbb{R}^{+n}$.

Rate of Convergence

To estimate the rate of convergence, we need to look at the eigenvalues of the linear operator l that approximates our differentiable function f near the fixpoint reached by I_0. These eigenvalues determine the structure of the phase space and may yield very different outcomes depending on the parameters [Bar11]. Assuming the global uniqueness property, the fixpoint is locally attractive which implies the following conjecture.

Hypothesis 2 (Local stability)**.** All eigenvalues of the linear operator l have module less than 1.

A consequence of this property for our algorithm is that it converges linearly in the region R with an error bounded by r^n where r is a positive real smaller than 1. Therefore the scores computed at each iteration become more precise. In our experiments we will stop the recursion when the difference between the scores in two steps is below 0.001. The results in Table 5.5 shows that it takes about 30 iterations to reach such state.

5.3.7 Translation: Scoring Substitutions

Since translations is one of the most important application of aligning libraries, we give here a small preview of how it would work.

To illustrate this, we take the same example Fig 5.2. Let us suppose that we would like to translate a theorem containing the constants 0_{nat},$\leq_{nat}$ and nat from Isabelle/HOL to Mizar. Our objective will then be to find an counterpart for each of the constant in the theorem in Mizar. We will call such the set of such mappings a substitution. Since there are multiple possibilities for each counterpart, the number of possible substitutions grows exponentially. Therefore, we need to figure out which ones are the best.

A simple method is to just take the highest scoring pairs involving those constants. However, it can lead to an incoherent translation since each match is considered independently. In our example, it may translate *nat* to *nat* but 0 to the $\varnothing$. That is why it is important to consider the correlations between matches inside a substitution. A possible method would be to find the substitutions that have the smallest diameter in the graph, choosing the distance to be the inverse of the correlation. In our example, the top two scoring substitutions can be extracted from the dotted (black) and non-dotted (blue) sets shown in Fig 5.2. Those substitutions give us two possible translations. A fully-fledged translation mechanism would rank substitutions based on their correlations and scores. Additionally, it would also check the type of the resulting term.

5.4 Experiments

All experiments were performed on an Intel Core i5-3230M 2.60GHz laptop with 4 GB memory. The results of all experiments and the source code that produced them are available at:

`http://cl-informatik.uibk.ac.at/users/tgauthier/alignments/`

	conjuncts of theorems	constants	types	theories
Mizar	51086	6462	2710	1230
Coq	23320	3981	860	390
HOL4	16476	2188	59	126
HOL Light	16191	790	30	68
Isabelle/HOL	14814	1046	30	77
Matita	1712	339	290	101

Table 5.1: Number of objects in each library. A constant will be considered a type if it appears in the right-hand side of a type judgement. The constants column does not include constants that represent types.

We used the following versions of the provers and their libraries. For all provers each "named" theorem was split into its conjuncts as part of the export phase. Further statistics on the libraries are given in Table 5.1.

- HOL4 [SN08] version Kananaskis 10. We considered all the theories built as part of the standard build sequence.
- HOL Light [Har09] SVN version 225, including the core library built with the system, the standard library (`Library` directory) and the complex and multivariate developments [Har13].
- Isabelle [WPN08] 2016. We used the Complex_Main theory including all the imported theories down to the object logic HOL.
- Coq [HH14] version 8.5. Recording of the Coq library was performed via a plugin that allowed us to extract the kernel representation of the objects. We considered all theories in the distribution standard library.
- Matita [ARC14] version 0.5.3-1. We used the last released version of Matita that was able to output XML object representation, again considering only the standard library distributed with the system.
- Mizar [BBG+15] version 8.1.03 including the whole Mizar Mathematical Library 5.29.1227. We used the representation processed using the MPTP XML export [Urb06b].

5.4.1 Logical Mappings

Even if the mathematical formulas come from provers with very different foundations, the properties which they represent are similar from a high-level perspective. However,

the specifics of each logic make the internal representations (proof assistant kernel representation) quite different. In order to restore the similarity, we transform the logical constructs of each prover to a common representation. The term structure is inspired by simply-typed lambda calculus, where terms are lambda-terms and all logical constructors are constants. We also allow terms to appear on the level of types which enable us to capture various extensions of the logic, including such common ones as polymorphism (for higher-order logic) and dependent types. We perform the following logical mappings adapted to the logic of each proof assistant.

For Coq and Matita which are based on the calculus of construction we perform the following mappings:

- The dependent product construction where the bound variable is not actually used in the body is replaced by an implication, and the used one is identified with universal quantification. For instance, the typing judgment $f : (\forall x : int.\ num)$ can be translated to $(f : int \Rightarrow num)$ because num does not contain x.
- The type hierarchy is collapsed to a single type $\$t$ including *Set*.
- The type of propositions *Prop* is represented by $\$o$.
- De Bruijn indexes are replaced by variable names.
- Each of the logical constructs starting with *Case*, *Cast*, *Fix*, *Cofix* and *Letin* are replaced by a fixed logical constant. This makes it impossible to match them with their counterpart in other logics. However, user named theorems do not typically contains these constructs. Indeed, they appear in 397 theorem statements in Coq and 28 theorem statements in Matita.
- A compatible order for the declaration of constants is re-inferred (this is necessary, as we do not record it in the kernel based export).

In HOL4, HOL Light and Isabelle/HOL which are based on higher-order logic, minimal structural modifications are necessary:

- Polymorphic constants are given explicit types arguments, including type variables when they are not instantiated. For example, taking the head of a list $hd\ l$ is rewritten $hd\ a\ l$ where a is the type of the elements of l.
- Explicit types are given to HOL types based on their number of arguments. For instance, the type constructor *pair* that constructs the cartesian product of two types has two arguments so it is given the type $\$t \Rightarrow \$t \Rightarrow \$t$.
- The boolean type which is also the predicate type is mapped to $\$o$.
- The function type operator $>$ is mapped to $\Rightarrow$.

The first-order set theory of Mizar also requires structural changes

	Import	S	Normalization time			Number of properties		
Mizar	15.05	1218	11.22	36.67	50.22	31692	32725	41311
Coq	4.82	414	3.77	8.85	15.30	5517	6717	8686
HOL4	8.07	427	6.45	10.63	23.59	9265	10522	14064
HOL Light	13.80	448	4.69	10.77	25.81	11450	12182	21948
Isabelle/HOL	4.38	444	2.98	5.52	10.84	10521	10953	14729
Matita	0.40	38	0.27	0.51	1.27	1092	1263	1712

Table 5.2: Effects of different normalization on the number of properties. The two first columns present the import time and number of frequent subterms (S) for each prover. The following columns show the time taken and number of properties produced by different normalization. The levels of normalizations are in order of inclusion: CNF + AC of logical constants, additional type information, additional subterm substitutions.

- All functions are curried to match their standard higher-order representation. For example: $f(x, y)$ is rewritten as $(f\ x)\ y$.
- A element of type *true* in Mizar is a set as defined by the axioms of set theory. Therefore we map the type *true* to the type $\$t$.

Remark 9. The logical constants that have been manually mapped are only allowed to match themselves in the algorithm. This restriction is loosened for Mizar where the type $\$t$ is allowed to match any type.

5.4.2 Most Frequent Properties

For most provers we normalize formulas by transforming them to CNF and rewriting them modulo AC logical operators. Furthermore, types are included at all constant positions. Only in the case of Mizar, variable types need to be used and more complex types are mapped to unit types represented by a single constant.

The relative size of all libraries can be estimated from the the numbers of theorems presented in Table 5.1, and the import time shown in Table 5.2, which corresponds to the total size of the theorems. The average theorem size is small in Coq and Isabelle/HOL, average in HOL4 and large in HOL Light. We also observe that normalization time depends linearly on the theorem size.

The evolution of the number of properties relative to the different normalization is shown in Table 5.2. The type information increases the precision of the theorems, by splitting equivalent classes of theorems into smaller classes. Conversely, subterm conceptualization relaxes the matching constraints by creating at most one variant for each theorem. Still, we observe that the number of properties grows as new theorems are added to the library.

The most frequent properties for each prover involving one constant are presented in Table 5.3. Unsurprisingly, commutativity, associativity, and transitivity are very common. In Isabelle/HOL, the ubiquity of type classes reduces the repetition of such properties. On the contrary, many properties are repeated in Coq, with some of them originating from isomorphic structures. These structures are often not identical: some structures lead to more efficient (or relevant) computations whereas others are more convenient for proving. A simple example is the distinction between binary naturals and unary naturals. From a mathematical perspective these two represent the same concept but their algorithmic properties are different.

We also present the most common properties involving two constants in the libraries in Table 5.4. We manually named the automatically derived properties appearing in Tables 5.3 and 5.4 from their term representation. But we could have also rely on a learning-based automated method for naming properties developed by the second author [AK16].

5.4.3 Matching Algorithm

In Table 5.5 we show the performance of the pairing mechanism and dynamical scoring loop included in the matching algorithm. The presented approach is fast and scales across many formal libraries, as opposed to the previously considered approaches [GK14]. One reason is the efficiency of the pairing step. The other is the limited number of pairs obtained after the pairing step. The scoring loop can then only recurse over these pairs. Assuming, that the numbers of constants inside a property and the number pair of theorems related to a pair of constants are bounded, one iteration of the algorithm is linear in the total number of pairs. This is confirmed by the relatively fast scoring times. Our algorithm converges below the 0.001 threshold in about 30 steps in each pair of provers with no observed dependence on the size of the considered libraries.

The results also give some hints about the similarities between provers. First, the number of theorem pairs can be used as a weak indicator of the degree of similarity between two provers. However it is largely skewed by the size of the libraries. Looking at the number of common properties is slightly more convincing, as shown by the 1457 common properties shared by HOL Light and HOL4.

To give better estimates, we further base our analysis on the scores of the pairs of constants. Each of the subsequent graphs reproduce the scores of the best matches, when aligning two libraries.

5.4.4 Effect of Normalization

According to the heuristics presented in Section 5.3.3, there are two main ways normalization can improve the score of a match. First, by creating more theorems pairs that induces this match. This will in practice imply that the constants involved in the match will share more properties. Second, by decreasing the frequency of the theorem pairs as their precision is increased. Each normalization step has a positive and a negative effect on the score. Subterm conceptualization, CNF normalization, and AC rewriting

increase the numbers of common properties but diminish their accuracy. The reverse is true for typing information. The combination of those effects is presented in Fig 5.4. Aligning Coq and Mizar, the positive influence prevails in all cases. The addition of type information contributes to the largest improvement. Aligning HOL Light and Isabelle/HOL, subterm conceptualization is the game changer. Indeed, automatic factorization of type arguments is made possible through subterm conceptualization. This is essential for aligning other provers with Isabelle/HOL because type arguments occurs naturally during the instantiation of type classes. For the two considered pairs of provers, the application of commutativity to all operators improves the scores minimally. It is not clear whether this minimal score gain translates into more accurate matches, therefore we will not include this normalization by default.

5.4.5 Evaluation of the Best Scoring Pairs

The matching algorithm was run on all pairs of provers and the scores of the first thousand best matches are depicted in Fig 5.5. The HOL4 and HOL Light provers share many similar concepts. The alignment of Matita is significantly better with Coq than with any other provers. Since the library of Coq contains a lot of basic mathematical and algebraic concepts it can be aligned well with a variety of provers.

The presented scores only give an estimate of the quality of the matches. For efficiency reasons, our algorithm only focuses on a syntactic analysis of the similarities. Semantic measures are needed to capture the full complexity of a concept. More, repetition inside one library may artificially increase the number of concepts with strong matches. Therefore, a manual inspection of the pairs is necessary to evaluate the final quality of the matches.

To reduce the number of manual inspections to a reasonable level, we will focus our human analysis on the following pairs of provers by order of subjective difficulties: HOL Light-HOL4, HOL Light-Isabelle/HOL, HOL4-Isabelle/HOL, Coq-Matita Coq-HOL4, Isabelle/HOL-Mizar, and Coq-Mizar. To further simplify our analysis, we make a distinction between three classes of matches. Because it is a subjective judgment, there is no strict limit between each of the following classes.

A pair of constant will be called:

- an optimal match, if the two concepts were intended to represent the same mathematical object, although their definitions may differ. A match between a polymorphic constant with an implicit type argument and one of its instantiations will also be considered optimal.
  Examples of such optimal matches include: reals numbers defined differently, binary and unary natural numbers, specific instance of addition and addition ($+_{int}$ vs $+$).
- an approximate match, if its components are the carriers or the operators of a known morphism between large mathematical structures.
  Example: sets vs list of lists, reals vs integers, $<_{real}$ vs $<_{int}$, union vs intersection.
- a singular match if it comes from a smaller morphism appearing inside a structure.
  Example: division vs less than, $+_{real}$ vs $*_{int}$.

Depending on the application, the range of interesting matches may differ. For a translation between libraries recognition of optimal matches is necessary, whereas to take inspiration from proofs in other domains approximate and singular matches are also of high value [GK15b]. The addition of high-scoring singular and approximate pairs will be useful to learn patterns in the use of these concepts and in their relation to proving.

Overall, our matching algorithm tries to evaluate the quality of the mapping in the context of the two whole libraries. Therefore optimal matches are expected to have higher scores, followed by approximates ones and finally singular ones. The degree to which our algorithm is capable of ordering the matches correctly is presented in Table 5.4.5.

Among the provers of the HOL family the first non-optimal match occurs quite early but is always preceded by a conflicting optimal match. As there are six different definitions of natural numbers in Coq, our algorithm has to find that these six competing versions match the HOL Light natural numbers. Moreover the relation between algebraic structures is lost. Indeed, the reals from Coq are constructed from only one specific version of the numerals.

Similar issues occur when aligning Mizar with any of the other provers. The issues are further amplified by the fact that many constants in Mizar are implicitly polymorphic. Combining the two effects, + in Mizar matches to the different version of + for integers in Coq but also + for integers, complex numbers and reals. Since most of the the first hundred matches are of this kind, we have to look further for matches about more involved concepts, such as lists and trigonometry. This means that many approximate matches and a few singular matches are mixed with the optimal ones. We will therefore define additional methods to separate them in Section 5.5.

5.4.6 Transitive Matches

When experimenting with more than two libraries, it is possible to consider one library as a translation step between two others. Indeed, concepts can be mapped from the initial library to the intermediate library and then to the targeted library. We will give a transitive score *transitive_score* to such matches defined as the product of the intermediate scores on the path linking the two constants. In the following experiment we will look at the transitive matches produced when our initial and final libraries are Isabelle/HOL and Mizar (the order is irrelevant) and the other libraries are used as intermediates. We only map concepts through one intermediate library at a time in order to measure their performance. This approach can easily be generalized so that the mapping travels through multiple translators.

In Fig 5.6, the two best intermediates seem to be Coq and HOL Light followed by HOL4. However, because of the multiplication of similar structures in Coq, most interesting matches will be found in HOL Light. Compared to direct scores, transitive matches may have an unfair advantage as they can generate a large number of one-to-many mappings.

In order to evaluate the gain obtained with the help of transitive matches, we compare the transitive scores to the scores (*direct_score*) that were attributed during a direct

match. We additionally define two other measures to evaluate the novelty of our matches:

$$dif = transitive\_score - direct\_score$$
$$w\_dif = direct\_score * (transitive\_score - direct\_score)$$

Comparing all intermediate libraries, we discover that Coq is actually giving optimal matches of lower quality. In Table 5.7, the third top scoring match is singular and already the first one uses an approximate intermediate. The matches with best transitive scores through HOL4 and HOL Light are more accurate. To measure the novelty of those matches, we will rely on the other two scores to order them. In Table 5.9 transitive matches between Isabelle/HOL and Mizar are discovered through HOL Light and ordered by their difference scores *dif*. We first observe that all of the considered matches were not discovered by the direct approach and that the first three are optimal. It means that they were concepts that did not share any property directly but had both properties in common with HOL Light. Using HOL4 (see Table 5.8), new optimal matches were a bit less frequent. In order to achieve a compromise between novelty and quality we use the scoring function *w_dif*.

Overall, this experiment demonstrates that the use of a transitive method can discover new matches and reinforce the confidence on existing matches. To maximize optimal matches in future applications, a combined score given by the sum of the transitive and direct scores may be considered.

Remark 10. The transitive type matches were excluded from the tables because the Mizar type system makes the distinction between approximate and optimal type matches fuzzy.

5.5 Strategies

In order to further distinguish optimal matches from non-optimal ones, we provide a number of matching strategies. These iterative procedures divide matches into a positive and a negative set. The strategies aim to maximize the number of optimal matches and minimizing the number of singular or approximate matches in the positive set. These two sets are build incrementally. When a pair is chosen to be a member of the positive (respectively negative) category, it receives a positive (respectively negative) reinforcement. The purpose of these reinforcements is to increase or decrease the strength of the influence of all pairs beyond the scores that were attributed in the scoring loop. We first define the terminology used to describe the strategies.

Definition 5.18 (Positive, negative and undecided matches)**.** A *positive match* is an element of the positive set. A *negative match* is an element of the negative set. An *undecided match* is neither an element of the positive set nor an element of the negative set. It will become positive or negative as the two sets grow.

Definition 5.19 (Positive reinforcement)**.** A positive reinforcement is a modification applied to the score of a positive match and amounts to:

- Fixing its score to 3.

This number has been experimentally determined and roughly says that we are more than 3 times more confident that a selected match is optimal than for an undecided match.

Definition 5.20 (Negative reinforcement)**.** From a negative match N, a negative reinforcement performs two modifications:

- Fixing the score of N to 0.
- Fixing the score of each pair of theorems that induces N to 0.

The second modification is justified in Section 5.5.1 by the constant coherence constraint. This modification would be implicit if we had used a product instead of a sum in the scoring function for pair of theorems. Scores based on products are however not consistent with other constraints as shown in Section 5.3.3.

All presented strategies first run the dynamical scoring algorithm. Next, a pair (or a set of pairs) of constants is chosen based on their scores and additional heuristics to decide if it (they) should be classified as a positive or as a negative match. A positive (respectively negative) reinforcement is applied to each element of the positive (respectively negative) set. Dynamical scoring is then rerun to account for the newly updated scores. A new selection is performed, and the whole process is repeated as long as there exist undecided pairs. Pairs with zero scores are always put in the negative set. If not stated otherwise by a strategy, the pair with the highest score will be assigned to the positive set.

Remark 11. Convergence is guaranteed by the fact that the dynamical scoring algorithm is restarted from fixed initial values greater or equal to 1 after each decision. Reinforcement scores are fixed and non-negative and thus can be treated as coefficients. Reinforcement scores should not be modified by dynamical scoring. Otherwise, the algorithm would always converge to the fixpoint found with no reinforcement by uniqueness of the limit.

Beside a strengthening of the influence of the top matches, another advantage of the two level approach is that we separate algorithms for scoring the degree of isomorphism of matches from the ones deciding which matches are indeed optimal. Therefore, it is possible to use different scoring techniques for each of them, which will be exploited by the disambiguation in Section 5.5.3. Furthermore, some of the following techniques work globally on optimal matches (and are not helpful for searching for more approximate and singular matches, like it is the case for conjecturing [GKU16]).

We present three options which can be combined to form a strategy. First, we propose natural coherence constraints on the two set of matches. Second, we consider greedy matching. Third, we discuss disambiguation, which aims at resolving conflicts created by a multiple counterpart mappings. These three options can be used independently or combined and even used together with some human advice. The strongest combinations of the three automatic options is evaluated through the quality of the positive matches they produce.

5.5.1 Coherence Constraints

After correctly identifying a match, simple coherence constraints based on the logical relation between the different objects (types, constants and theorems) can be applied: type coherence and constant coherence.

Definition 5.21 (Type coherence)**.** A set of constant matches is type coherent if for every constant pair c of a set s, the type matches induced by c are also in the set s.

A weak form of type coherence was already induced by the correlations: Higher scoring constants cause higher scores in the types. However, this influence may not raise the scores of the types enough for it to become a positive match. In some cases other factors of the algorithm might give such types negative advice. To enforce type coherence on the positive set, as soon as we add a match to the positive set, we also add their induced type matches to the positive set.

Remark 12. Type coherence is recursive, since types have themselves types.

Although we classify constants rather than theorems (this distinction is less pronounced in intuitionistic type theory [KMU14]), we assume in the following definition that the theorems are also assigned to a positive and negative set of theorems. After the strategy terminates, a theorem will be said to be in the positive set if it has a score greater than 0 and in the negative set if its score is 0. We can then state a similar constraint relation between theorems and constants that we call constant coherence.

Definition 5.22 (Constant coherence)**.** For every theorem pair t of the positive set of theorems, the constant matches induced by t are also in the positive set of constants.

Constant coherence is enforced by zeroing the scores of theorems that contain negative pairs. This constraint also implies that if a type match is in the negative set then the constant pairs that induce it will in the next iteration be in the negative set. Indeed, theorem pairs that include a constant pair will also include its induced type pairs. Therefore if one of its type pairs is given a score of 0, all of the theorem pairs than induce the constant pair are given a score of 0. Hence the total score of the constant will be 0. This consequence can be seen as the contrapositive of type coherence satisfied by the negative set.

5.5.2 Greedy Method

In our dynamical scoring experiments we observe that a match with the highest score involving one constant is most often an optimal one, and the subsequent ones involving the same constant are approximate ones. For example, a match of integers with integers may be followed by a match of integers with reals. In order to remove those undesirable competing matches, as soon as a pair p is categorized as positive, the greedy strategy puts all pairs that have a common constant with p in the negative set. The first advantage of the strategy is a drastic reduction of the number approximate matches. Also, if applied with type coherence and without subterm conceptualization, there will be at most one

possible translation of a formula which will type-check by construction. In this particular case, the selection of coherent substitution is not needed (see Section 5.3.5).

There are however two downsides. First, it makes the algorithm more brittle. The presence of an approximate match early in the procedure leads to a series of approximate matches derived from the first one. For instance, matching integers with naturals leads to all operations about integers being matched to operation about naturals. Second, the algorithm does not allow for one-to-many mappings in the positive set. If there are multiple structures defining the same objects, as it is often the case for example for computational reasons, the greedy algorithm will only map one.

5.5.3 Disambiguation

Often an approximate match and an optimal match about the same constant are in the wrong order, the scores differing only by a small margin. The greedy method fails to recognize the ambiguity and assumes that the first match is correct.

To solve this problem, we propose a method that delay the selection of a positive match by measuring its ambiguity. To this end, we will penalize a pair if its constants have multiple possible counterparts in the other library. The hope is that the selection of less ambiguous matches will help the classification process.

Technically, we define the ambiguity score of a match $c = \{d_1, d_2\}$ by first defining its ambiguity sets, which are consists of the matches that shares a constant with c.

$$E_1 = \{c' = \{d, e\} \mid d = d_1 \vee e = d_1\}$$

$$E_2 = \{c' = \{d, e\} \mid d = d_2 \vee e = d_2\}$$

And then its ambiguity scores.

$$ambiguity_1(c) = \sum_{c' \in E_1} score(c')$$

$$ambiguity_2(c) = \sum_{c' \in E_2} score(c')$$

$$ambiguity(c) = ln(10 + (1 + ambiguity_1(c)) * (1 + ambiguity_2(c)))$$

$$ambiguity\_score(c) = \frac{1}{ambiguity(c)} \times score(c)$$

Remark 13. Different ambiguity scores were also tried. Most of them improved the results in a similar manner. Therefore, the quality of matches is not very sensitive to small changes in the scoring functions, and the application of disambiguation is more important than its particular implementation.

It is easy to note that for pairs with similar scores the higher an ambiguity score is the less ambiguous it is. We will then use these ambiguity scores instead for the selection of our positive matches. These new scores will affect our algorithm only by changing the order of the selected positive matches. They will not replace the reinforced

scores or the scores of the undecided matches. Thus they do not change the correlation between matches. From our experiments, preventing the ambiguity of one pair from being transmitted to other pairs seems necessary to preserve the stability of dynamical scoring.

5.5.4 Human Advice

Automated techniques may not be sufficiently precise or the user may desire to have more control over the alignment of the two libraries. That is why we also provide ways to combine those techniques with additional human advice.

In this setting, the user is shown the 40 (modifiable) undecided best matches. She can select a few of them and decide which sets they belong to. After the negative and positive reinforcements are applied and scores updated accordingly, the user can repeat the selection on the new 40 undecided best matches. To facilitate the procedure, commands are available to the user that allow him to undo a decision, show the positive and negative sets, display the undecided matches according to different orders and stop the iteration. By default, the order used is defined by the scores. But we also propose to regroup undecided matches by the constants they share (to perform manual disambiguation) or by similarity of the names of their constants. As a compromise, human advice can also be restricted to type matches. In this case, constant pairs (that are not a type pairs) with a score greater than the best undecided type pair are classified automatically.

Those human advice scenarios were not fully evaluated as it was more efficient for us to improve the automation than having us manually compensate for the algorithm's failures. Moreover, since we do not have expert knowledge in all the libraries our decisions were also error-prone. Therefore, these combined methods were mainly useful as debugging tools.

5.5.5 Results

Experiments are performed on the studied pair of provers. We run two different strategies. Both have type coherence and disambiguation enabled, but one of them applies the greedy restriction where the other one does not. We will refer to them as the greedy and non-greedy strategy.

Non-optimal Matches

We measure the effectiveness of the classification made by the greedy strategy by finding the first non-optimal matches in the positive set. Table 5.10 shows their rank. It also presents the size of the section which is the total number of matches in the positive set. The small size of each section compared to the total number of matches shows that the strategy is quite effective at eliminating non-optimal matches.

A manual inspection of the definitions in Table 5.12 reveals that the constants *Rev* and *rev_append* are actually the same. Table 5.11 also shows how this pair of constants was discovered from their similar properties. It would have been harder to recognize from their definitions. Indeed the first one is constructed by a match statement and the second

one by a conjunction. After checking more definitions, we realized that the component of the pairs (*measure*, *gtof*), (*ALL*, *pred_list*) and (*EVERY*, *pred_list*) also represent the exact same concepts even if their names indicate otherwise. Those misjudged matches could be used to create more consistent naming schemes across formalizations.

The similarity between HOL Light and HOL4 is striking. There exist 790 constants in HOL Light and a little less than 300 of them can be map to a constant in HOL4 that have exactly the same meaning. More, this number does not include the one-to-many mappings discovered by the non-greedy version. Aligning HOL Light and HOL4 to Isabelle/HOL is a bit more challenging. But the algorithm is still effective and the greedy strategy discovered more than 100 optimal matches in each case.

Disambiguation was an essential component for aligning correctly Coq with HOL4. We find that 188 concepts have a counterpart in the other library. These include only a few non-optimal matches, less than 10 in the first hundred matches. Even some of those non-optimal mappings can be interesting. At rank 98 (not shown in the table), the Coq constant *Ensemble*(set in French) is matched with the HOL4 constant *llist*(lazy lists). This match may even appear to be better than an optimal one involving the HOL4 constant *set*, after studying their usage in both provers. In this alignment, the booleans of Coq could not match the related boolean type in HOL4, since it is the same as the reserved type $*o* which is restricted to match only to itself. This issue could be solved easily by mapping the boolean type of Coq to $*o* during the application of the logical mappings.

The results are more modest for alignments with Mizar, although the size of the positive set is comparable. The percentage of optimal matches decrease rapidly and there are almost no optimal matches after the 50th. This is mainly due to the wrong choices made early on. Therefore, to align a prover with Mizar, it is better to run the non-greedy strategy. Another option, before better logical mappings, normalization or strategies are implemented for Mizar, would be to run the matching algorithm with some human advice. But with a low frequency of correct matches, it would be a tedious manual work and may defeat the purpose of our project.

Optimal matches

We inspect the optimal matches found by the greedy strategy to judge their value. We also compare it to the non-greedy strategy to find which optimal matches the greedy strategy missed. Table 5.13 presents interesting optimal matches found by the greedy and non-greedy strategies. The selected optimal matches in this table illustrate different achievements of our approach.

Subterm conceptualization renders possible to match pair of reals with complex numbers, and they are indeed used in that way in HOL4. The greedy startegy may however not recognize this similarity. Indeed, the concept of a pair of reals may exist in both prover, yielding the match of the two isomorphic concepts. Therefore it would prevent any further matches involving those concepts. Another effect of conceptualization is the automatic factorization of type arguments. Thus, the subterm *power real* can be identified with the constant *real_pow*. The advantage of regrouping constants using their reflexive transitive

closure modulo equality appears when the two constants *PI* and *ALT_PI* automatically match the same counterpart *P_t*. The concepts *relation*, *decidable*, *transitive* shared by Matita and Coq show the different degree of abstractions of each library. We can also illustrate the effectiveness of type coherence when aligning Coq and HOL4. A match between the constants *length* and *LENGTH* (representing the length of a list) directly implies a match between the types *nat* and *num* (representing natural numbers).

All in all, the matching algorithm works across a variety of different theories (list, complex, probability, ...). This approach performs well on any kind of theories as long as developers of formal libraries state properties of the mathematical objects in a relatively similar manner. Still, distinguishing between an isomorphic construction and a structure sharing similar properties remains a challenge. We observe that running the algorithm in non-greedy mode enable us to obtain one-to-many mappings but those may also contain some non-optimal matches too. As an intermediate between the greedy method and the non-greedy strategy, a dynamic evaluation of the level of greediness (number of allowed counterparts of a constant) deepening the ideas used to implement the disambiguation option could be implemented.

Complexity and Convergence

Applying a strategy does not come for free. The repetitive application of dynamical scoring is the most costly operation. To our advantage, the scoring updates takes less and less time to converge after each iteration. Indeed, the number of undecided matches diminishes. And the number of loops needed to reach a fixpoint is minimized since we restart the algorithm from the previous fixpoint. Let us take for example the process of aligning Coq and HOL4. The first scoring loop takes 7.37 seconds. But the greedy method with type coherence and disambiguation gives a total of 188 positive matches in 154 seconds.

This method of restarting from the previous fixpoint has experimentally always terminated. However, to guarantee convergence we would have to reset the scores after each update or only allow updates that do not mix positive and negative reinforcements. By construction, a positive (or negative) reinforcement will increase (respectively decrease) the current scores of all pairs. Therefore, updating scores after positive and negative reinforcement separately is enough to make the first step of the dynamical scoring monotonic. And this condition implies convergence as proved in Section 5.3.6.

5.6 Related Work

Our approach is in certain ways similar to latent semantic analysis [LFL98] used to find synonyms in multiple documents or text fragments. The relation is even more obvious if we consider our properties to be documents and concepts to be words. Similar pieces of text should contain words similar in meaning in the same way that similar properties contains similar concepts. However, our approach is able to use the structure of the properties, which cannot be done for informal documents.

A number of translations between formal mathematical libraries are able to use given matched concept. For this, usually matching concepts have been found manually. The first translation that introduced maps between concepts was the one of Obua and Skalberg [OS06]. There, two commands for mapping constructs were introduced: `type-maps` and `const-maps` which allow a user to map HOL Light and HOL4 concepts to corresponding ones in Isabelle/HOL. Given a type (or constant) in the maps, during the import of a theorem all occurrences of this type in the source system are replaced by the given type of the target system. In order for this construction to work, the basic properties of the concepts must already exist in the target system, and their translation must be avoided. Due to the complexity of finding such existing concepts and specifying the theorems which do not need to be translated, Obua and Skalberg were able to map only small number of concepts like booleans and natural numbers, leaving integers or real numbers as future work.

The translation of Keller and Werner [KW10] was the first one, which was able to map concepts between systems based on different foundations. The translation from HOL Light to Coq proceeds in two phases. First, the HOL proofs are imported as a defined structure. Second, using the *reflection* mechanism, native Coq properties are built. It is the second phase that allows mapping the HOL concepts like natural numbers to the Coq standard library type `N`.

The translation that maps so far the biggest number of concepts has been done by the second author [KK13]. The translation process consists of three phases, an exporting phase, offline processing and an import phase. The offline processing provides a verification of the (manually defined) set of maps and checks that all the needed theorems will be either skipped or mapped. This allows to quickly add mappings without the expensive step of performing the actual proof translation, and in turn allows for mapping 70 HOL Light concepts to their corresponding Isabelle/HOL counterparts. All these concept maps have been found and provided manually.

Bortin et al. [BJL06] implemented the AWE framework which allows the reuse of Isabelle/HOL formalization recorded as a proof trace multiple times for different concepts. Theory morphisms and parametrization are added to a theorem prover creating objects with similar properties. The use of theory morphisms together with concept mappings is one of the basic features of the MMT framework [Rab13]. This allows for mapping concepts and theorems between theories also in different logics. So far all the mappings have been done completely manually.

Hurd's OpenTheory [Hur11] aims to share specifications and proofs between different HOL systems by defining small theory packages. In order to write and read such theory packages by theorem prover implementations a fixed set of concepts is defined that each prover can map to. This provides highest quality standard among the HOL systems, however since the procedure requires manual modifications to the sources and inspection of the libraries in order to find the mappings, so far only a small number of constants and types could be mapped. Similar aims are shared by semi-formal standardizations of mathematics, for example in the OpenMath content dictionaries. For a translation between semi-formal mathematical representation again concept lookup tables are constructed manually [SW06, CDD+01].

The Dedukti proof checker [DHK03], based on the $\lambda\Pi$-modulo calculus, can import and verify developments from Coq and HOL systems. An example Coq proof has been shown to be translatable to Dedukti and to be instantiated with HOL natural numbers [AC15]. One of the main challenges was to match the different typing levels of Coq and HOL into a common structure in the logic of Dedukti.

The proof advice systems for interactive theorem proving have studied similar concepts using various similarity measures. The methods have so far been mostly restricted to similarity of theorems and definitions. They have also been limited to single prover libraries. Heras and Komendantskaya in the proof pattern work [HK14] try to find similar Coq/SSReflect definitions using machine learning. Hashing of definitions in order to discover constants with same definitions in Flyspeck has been done in [KU15b]. Searching for similar lemmas in order to find interesting properties has been tried for Mizar using the MoMM system [Urb06a] as well as for HOL Light intermediate lemmas [KU15c].

5.7 Conclusion

We have developed a methodology for matching concepts across formal mathematical libraries. Our approach relies on measuring the similarity of the properties of those concepts complemented by a dynamical process that iterates through their structural interrelation. Additional techniques such as subterm conceptualization and disambiguation appear to be highly beneficial to the quality of the matches and in some cases essential to the matching process.

Our experiments on multiple proof assistant libraries lead to the discovery of thousands of similar concept pairs. The full method performs particularly well between provers based on higher-order logic and variants of type theory. Aligning set-theoretical and type-theoretical provers automatically gives a smaller number of perfect matches.

5.8 Future Works

We have focused on heuristic ways to match concepts avoiding the use of metadata, such as the names of the theories, theorems, and constants. Such metadata, as well as scoring heuristics refined by an unsupervised machine learning process could be used in practical applications of matches.

Furthermore, it would be interesting to test the quality of the found matches in the various applications. Sharing proof knowledge [GK15b] could be performed across the studied libraries that have learning-assisted reasoning support [BKPU16]. An early experiment with conjecturing [GKU16] through analogies created from concept matches indicates some success. But more approaches [KUV15b] to transfer and create properties using knowledge from different mathematical domains could be tried. We would also like to provide a database of concept matches to create the possibility for external users to exploit the data for their own applications.

Acknowledgement

We thank Pierre Boutillier, Pierre-Marie Pédrot, Enrico Tassi and Yves Bertot for their help with creating a Coq plugin to export Coq formulas at the first "Coq coding sprint". We thank Josef Urban for his export of the Mizar library which we rely on. We valued the discussions with Dennis Müller, Florian Rabe, and Michael Kohlhase on the theoretical concept of theory morphism and their applications. We appreciated Chad Brown for his comments on the evaluation results. This research was supported by ERC starting grant no. 714034 *SMART*.

Coq		HOL4		HOL Light	
Property	Thms	Property	Thms	Property	Thms
Commutativity	157	Injectivity Eq	72	Commutativity	34
Associativity	143	Commutativity	48	Associativity	30
Transitivity	94	Injectivity Eq TA	31	Injectivity Eq	25
Nilpotence	75	Associativity	29	Nilpotence TA	15
Injectivity	63	Transitivity	22	Transitivity	15

Isabelle/HOL		Matita		Mizar	
Property	Thms	Property	Thms	Property	Thms
Injectivity Eq	23	Inductive def	69	Truth 2A	123
Injectivity Eq 2TA	9	Truth	13	Transitivity TA	67
Injectivity Eq 3TA	6	Commutativity	9	Truth 3A	64
Equality def	6	Nilpotence	7	Injectivity	43
Identity def TA	6	Associativity	6	Associativity	41

Property	Pattern
Commutativity	(C0 V0) V1 = (C0 V1) V0
Associativity	(C0 V2) ((C0 V1) V0) = (C0 ((C0 V2) V1)) V0
Transitivity	(C0 V1) V2 ∨ ¬ ((C0 V0) V2 ∨ ¬ ((C0 V1) V0))
Nilpotence	V0 = (C0 V0) V0
Injectivity Eq	(V1 = V0) = (C0 V1 = C0 V0)
Injectivity	¬ (C0 V1 = C0 V0) ∨ (V1 = V0)
Inductive def	(V4 V1) ∨ (∃ V1 V0, (¬ (V4 ((((C0 V3) V2) V1) V0))))
	(V2 V3) ∨ (∃ V1 V0, (¬ (V2 ((C0 V1) V0))))
	(V4 V5) ∨ (∃ V1 V0, (¬ (V4 ((((C0 V3) V2) V1) V0))))
	(V2 V3) ∨ (∃ V1 V0, (¬ (V2 ((C0 V1) V0))))
Equality def	((C0 V2) V2) (λ V1 V0, (V1 = V0))
Identity def	C0 = (λ V0, V0)
Truth	C0
Truth 2A	C0 V1 V0

Table 5.3: Most frequent properties involving one constant in number of theorems. The suffix "xA" precises the number of arguments "x" of the constant. The suffix "xTA" precises the number of silent arguments (often type arguments) "x" of the constant. The property "Inductive def" actually regroups four different properties that are abstracted from inductive definitions.

Coq		HOL4		HOL Light	
Property	Thms	Property	Thms	Property	Thms
Left distributivity	256	Inverse F	58	Implication 2A	95
Right distributivity	169	Linearity	43	Property on	33
Left neutral	133	Different	41	Monotonicity	28

Isabelle/HOL		Matita		Mizar	
Property	Thms	Property	Thms	Property	Thms
Inverse F	19	Implication	18	Right neutral F	101
Implication 3A	18	Inverse F	10	Left distributivity	80
Implication 2A	11	Structure of	9	Inverse F	74

Property	Pattern
Left distributivity	(C0 ((C1 V2) V1)) V0 = (C1 ((C0 V2) V0)) ((C0 V1) V0)
Right distributivity	(C0 V1) ((C1 V2) V0) = (C1 ((C0 V1) V2)) ((C0 V1) V0)
Left neutral	V0 = (C1 V0) C0
Right neutral F	C1 V0 = C1 (C0 V0)
Inverse F	V0 = C1 (C0 V0)
Linearity	C0 ((C1 V1) V0) = (C1 (C0 V1)) (C0 V0)
Monotonicity	(C1 V1) V0 = (C1 (C0 V1)) (C0 V0)
Implication	C1 $\vee$ $\neg$ C0
Implication 2A	(C1 V1) V0 $\vee$ $\neg$ (C0 V1) V0
Implication 3A	((C1 V2) V1) V0 $\vee$ $\neg$ ((C0 V2) V1) V0
Different	$\neg$ (C1 = C0)
Property on	C0 (λ V3, ((C1 (V2 V3)) (V1 V3))) V0 $\vee$ $\neg$ ((C0 V2) V0) $\vee$ $\neg$ ((C0 V1) V0)
Structure of	C1 | $\neg$ (C0 V0)

Table 5.4: Most frequent properties involving two constants in number of theorems. The suffix "xA" precises the number of arguments "x" of each constant. The suffix "F" precises that the property should be understood at the function level.

	Pairing	Props	T pairs	C pairs	Scoring	Loops
Coq - Mizar	1.74	833	46113	34772	13.43	38
HOL4 - Mizar	1.48	814	36918	26433	10.9	36
Coq - HOL4	0.92	500	32127	15365	7.37	33
HOL Light - Mizar	1.34	679	27395	20085	6.65	29
Coq - HOL Light	0.52	379	24600	11189	5.78	34
Isabelle/HOL - Mizar	0.92	558	20363	13838	5.99	35
Coq - Isabelle/HOL	0.38	349	19758	8273	2.35	19
HOL4 - HOL Light	0.44	1457	18296	5861	2.68	23
Coq - Matita	0.19	250	13365	5147	2.65	34
HOL4 - Isabelle/HOL	0.21	427	10074	4562	2.06	32
Matita - Mizar	0.32	221	9401	4549	1.48	21
HOL Light - Isabelle/HOL	0.20	392	7552	3208	1.41	30
HOL4 - Matita	0.10	158	3469	1434	0.70	29
HOL Light - Matita	0.14	120	2335	1043	0.50	28
Isabelle/HOL - Matita	0.08	117	1540	962	0.50	32

Table 5.5: Statistics for each pair of provers gathered during pairing and dynamical scoring of pairs of constants and theorems, ordered by their number of pairs of theorems. Presented form left to right in this table: pairing time (in seconds), number of common properties, number of theorem pairs, number of constant pairs, scoring time (in seconds) and number of loops necessary to reach a fixpoint.

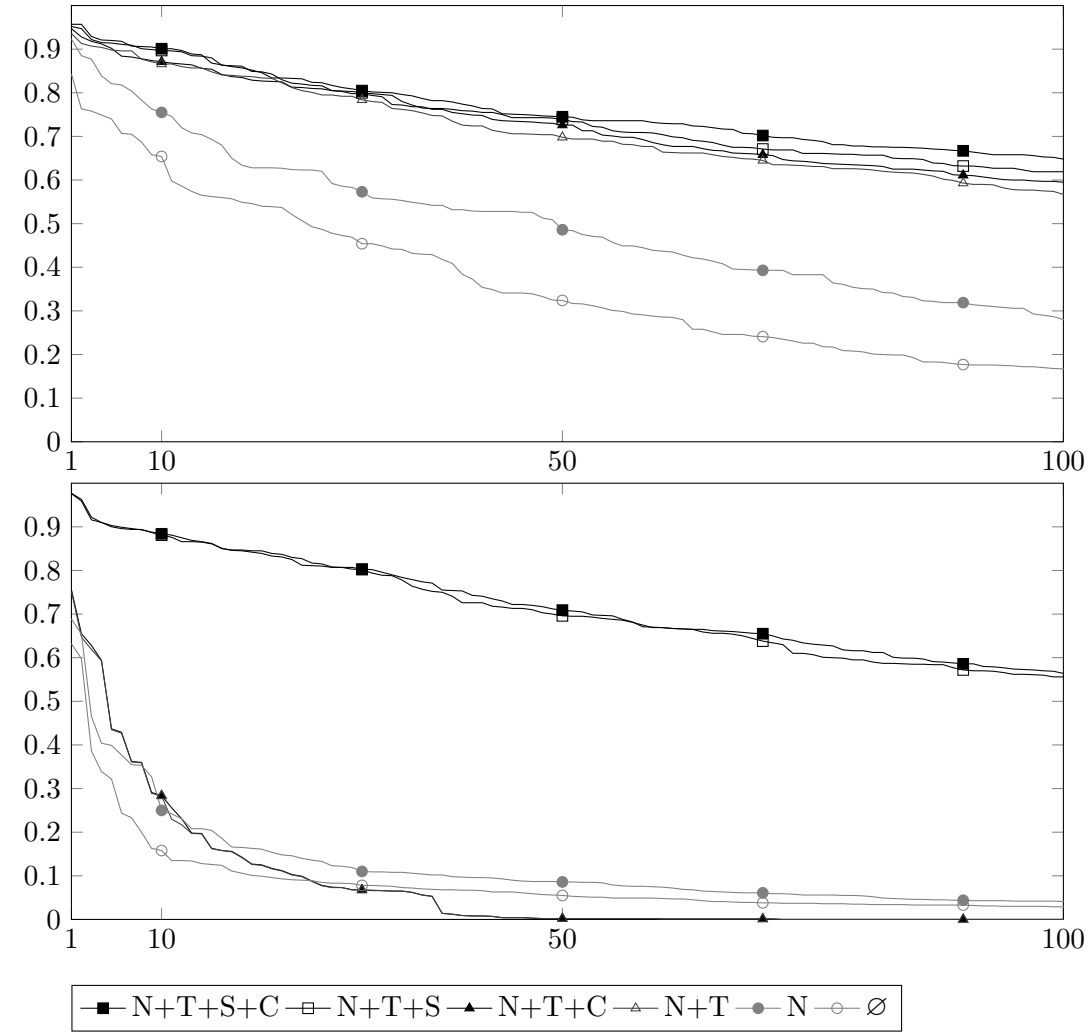


Figure 5.4: Effect of normalization when aligning Coq with Mizar (top figure) and HOL Light with Isabelle/HOL (bottom figure). "N" stands for CNF normalization + logical AC. "T" stands for default typing formation. "S" stands for subterm conceptualization. "C" stands for the fourth level of AC normalization which includes permutation of arguments in non-commutative constants.

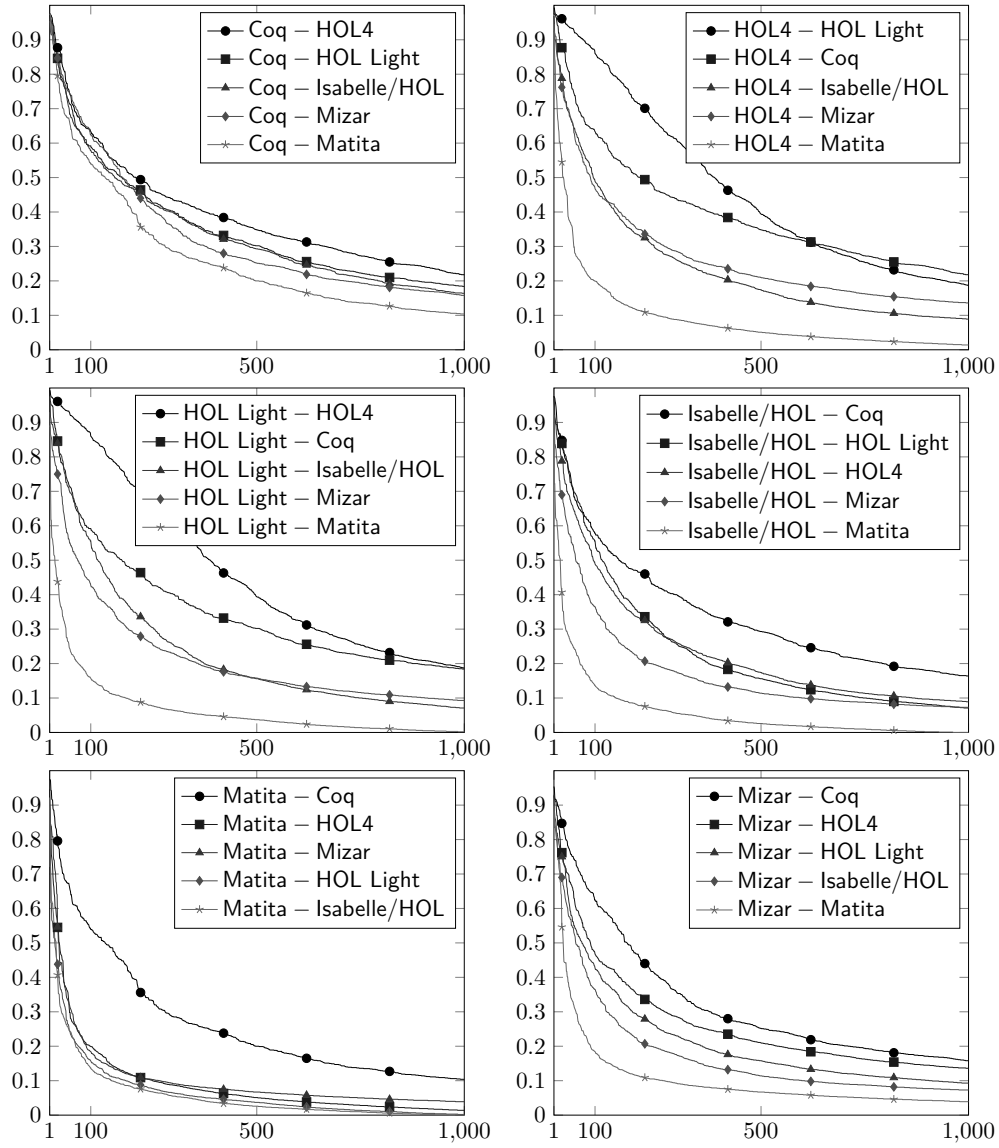


Figure 5.5: Scores of the best thousand pairs of constants with default settings.

Prover 1	Prover 2	Constant 1	Constant 2	rank	score
HOL4	HOL Light	**num**	**real**	25	0.96
		num	*num*	2	0.99
		real	*real*	1	0.99
HOL4	Isabelle/HOL	**real**	**nat**	4	0.89
		real	*real*	2	0.95
		num	*nat*	1	0.97
HOL Light	Isabelle/HOL	**real**	**int**	10	0.88
		real	*real*	1	0.97
		int	*int*	24	0.81
Coq	Matita	**Z**	**nat**	1	0.97
		Z	*Z*	12	0.87
		nat	*nat*	2	0.97
Coq	HOL4	**Z**	**real**	2	0.97
		Z	*num*	3	0.97
		R	*real*	10	0.94
Isabelle/HOL	Mizar	**dvd nat**	$\leq$	6	0.82
		dvd nat	*divides*	35	0.58
		less_eq nat	$\leq$	9	0.78
Coq	Mizar	**Z**	**real**	2	0.94
		Z	*integer*	17	0.87
		R	*real*	8	0.91

Table 5.6: First non-optimal match in each studied pair of provers (in bold), followed by the first optimal matches for each constant

Isabelle/HOL	Mizar	through Coq	Direct	Transitive
zero int	0	*BinNums_N_0*	0.83	0.82
zero int	0	*BinNums_Z_0*	0.50	0.78
dvd nat	$\leq$	*le*	0.81	0.77
less_eq nat	$\leq$	*le*	0.78	0.77

Table 5.7: First four transitive matches through Coq (excluding types) with best transitive scores.

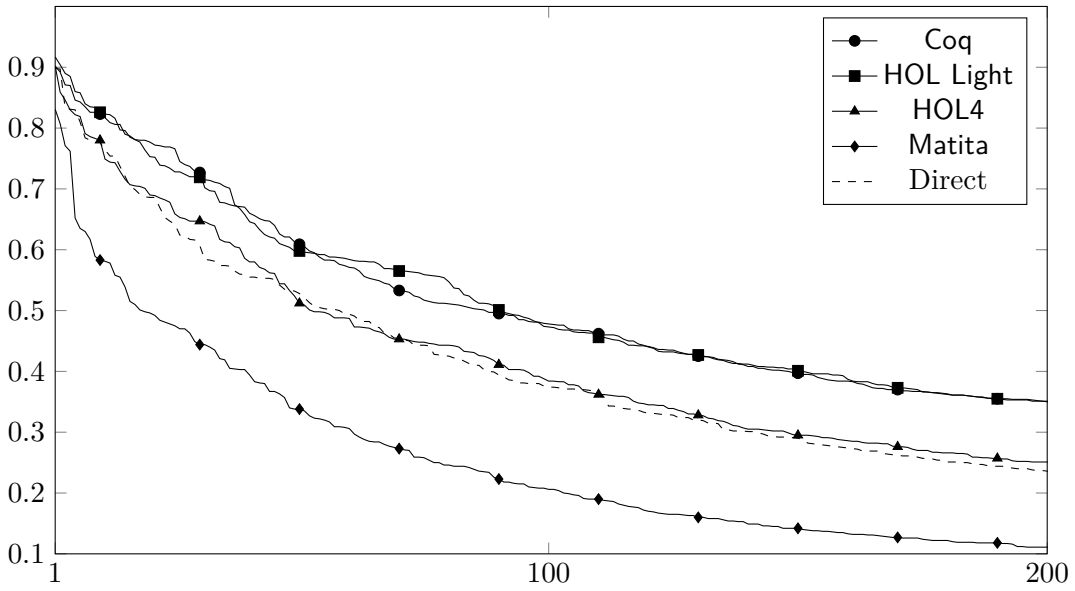


Figure 5.6: Best transitive matches between Isabelle/HOL and Mizar.

Isabelle/HOL	Mizar	through HOL4	Direct	Transitive	*w_dif*
(*less_eq real*)	$\leq$ 0	*real_lte*	0.40	0.64	0.10
(*zero real*)		(*real_of_num* 0)			
cos real	*cos*	*cos*	0.45	0.58	0.06
sin real	*sin*	*sin*	0.45	0.57	0.05
real	*real*	*real*	0.75	0.81	0.04

Table 5.8: First four transitive matches through HOL4 (excluding types) with best *w_dif* scores.

Isabelle/HOL	Mizar	through HOL Light	Direct	Transitive	*dif*
(*less real*)	$<$ 0	*real_le*(*real_of_num*	0	0.50	0.50
(*zero real*)		(*NUMERAL* 0))			
power real	|^	*real_pow*	0	0.44	0.44
times complex	$*$	*complex_mul*	0	0.40	0.40
pred	*carrier*	*cart real*	0	0.40	0.40

Table 5.9: First four transitive matches through HOL Light (excluding types) with best *dif* scores.

Prover 1	Prover 2	Sect	Constant 1	Constant 2	rank
HOL4	HOL Light	303	*extreal*	*complex*	227
			extreal_pow	*complex_pow*	228
			extreal_mul	*complex_mul*	229
HOL4	Isabelle/HOL	159	*modu*	*real_norm complex*	39
			prod	*complex*	40
			EVERY	*pred_list*	90
			nub	*remdups*	106
			SNOC	*insert*	107
HOL Light	Isabelle/HOL	123	*FCONS*	*case_nat*	78
			ALL	*pred_list*	79
			DIV	*binomial*	92
			rational	*positive*	108
Coq	Matita	84	*N*	*Z*	13
			$= 0_N$	*Zle*	34
			N_mul	*Ztimes*	46
			transitive	*symmetric*	48
Coq	HOL4	188	*rev_append*	*REV*	7
			BinNums_N	*ext_real*	45
			0_BinNums_N	*extreal_of_num* 0	46
			bool	*rat*	49
			constr_bool_2	*rat_1*(*rat_of_num*(*NUMERAL*(*BIT1 ZERO*))	61
			measure	*gtof*	71
Isabelle/HOL	Mizar	137	*list*(*X*)	*Element* (*QC-WFF*(*X*))	2
			sup	\/	3
			sqrt	$\_^2$	24
Coq	Mizar	168	*RIneq_Rsqr*	*min*	9
			sqrt	$\_^2$	10

Table 5.10: First suspected non-optimal matches in the positive set. The column Sect shows the total number of elements in the positive set.

`rev_append`	`REV`
`∀ l, rev l = rev_append l [].`	`∀ L. REVERSE L = REV L []`
`∀ l l', rev_append l l' = rev l ++ l'.`	`∀ L1 L2. REV L1 L2 = REVERSE L1 ++ L2`

Table 5.11: Properties shared by the Coq constant `rev_append` and the HOL4 constant `REV`.

rev_append	REV

`Fixpoint rev_append (l l': list A) : list A :=`
`match l with`
`| [] => l'`
`| a::l => rev_append l (a::l')`
`end.`
	`(∀ acc. REV [] acc = acc) ∧`
	`∀ h t acc. REV (h::t) acc =`
	`REV t (h::acc)`

Table 5.12: Defintions of the Coq constant `rev_append` and the HOL4 constant `REV`.

Prover 1	Prover 2	Mode	Constant 1	Constant 2	rank
HOL4	HOL Light	G	*prod*	*prod*	1
			sum	*psum*	3
			RTC	*RTC*	269
		NG	(*prod real*) *real*	*complex*	251
HOL4	Isabelle/HOL	G	$\frac{\pi}{2}$	$\frac{\pi}{2}$	7
		NG	(*prod real*) *real*	*complex*	36
HOL Light	Isabelle/HOL	G	*real_pow*	*power real*	1
			ITLIST	*foldr*	95
Coq	Matita	G	*relation*	*relation*	1
			decidable	*decidable*	3
		NG	*Transitive N*	*transitive nat*	53
Coq	HOL4	G	*length*	*LENGTH*	3
			nat	*num*	4
			0_Z	*int_0*(*int_of_num* 0)	30
			Z	*int*	31
		NG	*BinNums_positive*	*num*	21
			BinNums_N	*num*	47
			BigN	*num*	48
Isabelle/HOL	Mizar	G	*pi*	*P_t*	7
			arccos	*arcos*	35
		NG	(*fold nat*) *nat***	$\longrightarrow$**	21
			member nat	*in*	3
Coq	Mizar	G	*PI*	*P_t*	90
			ALT_PI		
			Rlist	*FinSequence REAL*	106

Table 5.13: Interesting optimal matches found by running a strategy with disambiguation and type coherence in greedy mode (G) and non-greedy mode (NG). The presented non-greedy matches are not found by the greedy algorithm. The match (**) may not be an optimal one.

Chapter 6

Sharing HOL Proof Knowledge

Abstract

New proof assistant developments often involve concepts similar to already formalized ones. When proving their properties, a human can often take inspiration from the existing formalized proofs available in other provers or libraries. In this paper we propose and evaluate a number of methods, which strengthen proof automation by learning from proof libraries of different provers. Certain conjectures can be proved directly from the dependencies induced by similar proofs in the other library. Even if exact correspondences are not found, learning-reasoning systems can make use of the association between proved theorems and their characteristics to predict the relevant premises. Such external help can be further combined with internal advice. We evaluate the proposed knowledge-sharing methods by reproving the HOL Light and HOL4 standard libraries. The learning-reasoning system HOL(y)Hammer, whose single best strategy could automatically find proofs for 30% of the HOL Light problems, can prove 40% with the knowledge from HOL4.

6.1 Introduction

As *Interactive Theorem Prover* (ITP) libraries were developed for decades, today their size can often be measured in tens of thousands of facts [BHMN15, MML]. The theorem provers typically differ in their logical foundations, interfaces, functionality, and the available formalized knowledge. Even if the logic and the interface of the chosen prover are convenient for a user's purpose, its library often lacks some formalizations already present in other provers' libraries. Her only option is then to manually repeat the proofs inside her prover. She will then take ideas from the previous proofs and adapt them to the specifics of her prover. This means that in order to formalize the desired theory, the user needs to combine the knowledge already present in the library of her prover, with the knowledge present in the other formalization.

We propose an approach to automate this time-consuming process: It consists of overlaying the two libraries using concept matching and using learning-assisted automated reasoning methods [KU14], modified to learn from multiple libraries and able to predict advice based on multiple libraries. In this research we will focus on sharing

proof knowledge between libraries of proof assistants based on higher-order logic, in particular HOL4 [SN08] and HOL Light [Har09]. Extending the approach to learning from developments in provers that do not share the same logic lies beyond the scope of this paper.

Once a sufficient number of matching concepts is discovered, theorems and proofs about these concepts can be found in both libraries, and we can start to implement methods for using the combined knowledge in future proofs. To this end, we will use the AI-ATP system HOL(y)Hammer [KU14]. We will propose various scenarios augmenting the learning and prediction phases of HOL(y)Hammer to make use of the combined proof library. In order to evaluate the approach, we will simulate incrementally reproving a prover's library given the knowledge of the library of the other prover. The use of the combined knowledge significantly improves the proof advice quality provided by HOL(y)Hammer. Our description of the approach focuses on HOL Light and HOL4, but the method can be applied to any pair of provers for which a mapping between the logics is known.

6.1.1 Related Work

As reuse of mathematical knowledge formalizations is an important problem, it has already been tackled in a number of ways. In the context of higher-order logic, OpenTheory [Hur11] provides cross-prover packages, which allow theory sharing and simplify development. These packages provide a high-quality standard library, but need to be developed manually. The *Common HOL Platform* [Ada15a] provides a way to re-use the proof infrastructure across HOL provers.

Theory morphisms provide a versatile way to prove properties of objects of the same structure. The idea has been tried across Isabelle formalizations in the AWE framework by Bortin et al. [BJL06]. It also serves as a basis for the MMT (Module system for Mathematical Theories) framework [Rab13].

With our method, this principle was developed in both directions. We first search for similar properties of structures to find possible morphism between different fields. We then use these conjectured morphisms to translate the properties between the two fields. Our main idea is that we don't prove the isomorphism which is often a complex problem but we learn from the knowledge gained from the derived properties. Moreover, even when the two fields are not completely isomorphic, the method often gives good advice. Indeed, suppose the set of reals in one library were incorrectly matched to the set of rationals in the other, we can still rely on properties of rationals that are also true for reals.

A direct approach is to create translations between formal libraries. This can only be applied when the defined concepts have the same or equivalent definitions. The HOL/Import translation from HOL4 and HOL Light to Isabelle/HOL implemented by Obua and Skalberg [OS06] already mapped a number of concepts. This was further extended by the second author [KK13] to map 70 concepts, including differently defined real numbers. HOL Light has also been translated into Coq by Keller and Werner [KW10]. It is the first translation between systems based on significantly different logics. In each of these

imports, the mapping of the concepts has been done manually.

Compared with manually defined translations, our approach can find the mappings and the knowledge that is shared automatically. It can also be used to prove statements that are slightly different and in some cases even more general. Additionally, the proof can use preexisting theorems in the target library. On the other hand, when a correct translation is found by hand, it is guaranteed to succeed, while our approach relies on AI-ATP methods which fail for some goals. The possibility of combining the two approaches is left open.

Overview

The rest of this paper is organized as follows. In Section 6.2, we introduce the AI-ATP system HOL(y)Hammer and describe automatic recognition of similar concepts in different formal proof developments. In Section 6.3, we propose a number of scenarios for combining the knowledge of multiple provers. In Section 6.4, we evaluate the ability to reprove the HOL4 and HOL Light libraries using the combined knowledge. In Section 6.5 we conclude and present an outlook on the future work.

6.2 Preliminaries

6.2.1 HOL(y)Hammer

HOL(y)Hammer [KU15b] is an AI-ATP proof advice system for HOL Light and HOL4. Given a user conjecture, it uses machine learning to select a subset of the accessible facts in the library, that are likely to prove the conjecture. It then translates the conjecture together with the selected facts to the input language of one of the available ATP systems to find the exact dependencies necessary to prove the theorem in higher-order logic. This method is also followed by the system Sledgehammer [PB10].

In this section we shortly describe how HOL(y)Hammer processes conjectures, as we will augment some of these steps in Section 6.3. First, we describe how libraries are exported. Then, we explain how the exported objects and dependencies are processed to find suitable lemmas. Finally, we briefly show how the conjecture can be proven from these lemmas. More detailed descriptions of these steps are presented in [KU14, GK15a].

Export

We will associate each ITP library with the set of constants and theorems that it contains. In particular, the type constructors will also be regarded as constants in this paper. As a first step, we define a format for representing formulas in type theory, as we aim to support formulas from various provers. A subset of this format is chosen to represent the higher-order logic statements in HOL Light and HOL4. Each object is exported in this format with additional information about the theory where it was created. The theory information will let us export incompatible developments (i.e. ones that can not be loaded into the same ITP session or even originate from different ITPs) into

HOL(y)Hammer [KR14]. Additionally, we can fully preserve the names of the original constants in the export. Finally, the dependencies of each theorem (i.e the set of theorems which were directly used to proved it) are extracted. This last step is achieved by patching the kernels of HOL4 and HOL Light.

Premise Selection

The premise selection algorithm takes as input an (often large) set of accessible theorems, a conjecture, and the information about previous successful proofs. It returns a subset of the theorems that is likely to prove the conjecture. It involves three phases: feature extraction, learning, and prediction.

The features of a formula are a set of characteristics of the theorem, which we represent by strings. Depending on the choice of characterization, it can simply be the list of the constants and types present in the formula, or the string representation of the normalized sub-terms of the formula, or even features based on formula semantics [KUV15a]. The *feature extraction* algorithm takes a formula as input and computes this set.

A relation between the features of conjectures and their dependencies is inferred from the features of all proved theorems and their dependencies by the *learning* algorithm. This step effectively finds a function that given conjecture characteristics finds the premises that are likely to be useful to prove this conjecture. *Prediction* refers to the evaluation of this function on a given conjecture.

These phases will be influenced by the concept matching (see Section 6.2.2) and differentiated in each of the scenarios (see Section 6.3).

Translation and Reconstruction

A fixed number of most relevant predicted lemmas (all the experiments in this paper fix this number to 128, as it has given best results for HOL in combination with E-prover [GK15a]) are translated together with the conjecture to an ATP problem. If an ATP prover is able to find a proof, various reconstruction methods are attempted. The most basic reconstruction method is to inspect the ATP proof for the premises that were necessary to prove the conjecture. This set is usually sufficiently small, so that certified ITP proof methods (such as MESON [Har96] or Metis [Hur03]) can prove the higher-order counterpart of the statement and obtain an ITP theorem.

6.2.2 Concept Matching

Concept matching [GK14] allows the automatic discovery of concepts from one proof library or proof assistant in another. An AI-ATP method can benefit from the library combination only when some of the concepts in the two libraries are related: Without such mappings the sets of features of the theorems in each library are disjoint and premise selection can only return lemmas from the library the conjecture was stated in. As more similar concepts are matched (for example we conjecture that the type of integers in HOL4 `h4/int` and the type of integers in HOL Light `hl/int` describe the same type), the feature extraction mechanism will characterize theorems talking about the matched

concepts by the same features. As a consequence, we will also get predicted lemmas from the other library. We will discuss how such theorems from a different library can be used without sacrificing soundness in Section 6.3.

For a step by step of the concept matching algorithm, we will refer to our previous work [GK14] and only present here a short summary and the changes that improve the matching for the scenarios proposed in this paper. Our algorithm is implemented for HOL4 and HOL Light, but we believe the procedure can work for any pair of provers based on similar logics such as Coq [HH14] and Matita [ARC14].

Summary

Our matching algorithm is based on the properties (such as associativity, commutativity, nilpotence, ...) of the objects of our logic (constants and types). If two objects from two libraries share a large enough number of relevant properties, they will eventually be matched, even though they may have been defined or represented differently. In the description of the procedure, we will consider every type as a constant. Initially, the set of matched constants contains only logical constants. First, we give a highest weight for rare properties with a lot of already matched constants. Second, we look at all possible pairs of constants and find their shared properties. The final score for a pair of constant is the sum of their weights amortised by the total number of properties of each constant. The two constants with the highest similarity score are matched. The previous two steps are repeated until there are no more shared properties between unmatched constants.

Improvements and Limitations

The similarity scoring heuristic can be evaluated more efficiently than the ones presented in [GK14] and is able to map more constants correctly: Thanks to a better representation of the data the time taken to run our implementation of the matching algorithm on the standard library of HOL Light (including complex and multivariate) and the standard library of HOL4 was decreased from 1 hour to 5 minutes. By computing only the initial property frequencies and using them together with the proportion of matched constants to influence the weight of each property in the iterative part the time can be further decreased to 2 minutes. The algorithm now returns 220 correct matches instead of the 178 previously obtained and 15 false positives (pairs that are matched but do not represent the same concept) instead of 32. The better results are a consequence of the inclusion of types in the properties and the updated scoring function.

The proposed approach can only match objects that have the same structure. In the case of the two proof assistants we focus on, it can successfully match the types of natural numbers, integers or real number, however it is not able to match the dedicated HOL Light type `hl/complex` to the complex numbers of HOL4 represented by pairs of real numbers `h4/pair`(`h4/real`,`h4/real`). This issue could be partially solved by the introduction of a matching between sub-terms combined with a directed matching. The type `hl/complex` could then be considered as pair of reals in HOL4. For the reverse direction, we would need to know if the pair of reals was intended to represent a pair

of reals or a complex. One idea to solve this problem could be to create a matching substitution that also depends on the theorems. These general ideas could form a basis for a future extension of the matching algorithm.

6.3 Scenarios

In this section we propose four ways an AI-ATP system can benefit from the knowledge contained in a library of a different prover. We will call these methods "scenarios" and we will call the library of a different prover "external". All four scenarios require the base libraries to already be matched. This means, that we have already computed a matching substitution from the theorems of both libraries and in all the already available facts in the libraries, the matched constants are replaced by their common representatives.

Throughout our scenarios, we will rely on the notion of equivalent theorems to map lemmas from one library to the other. This notion is defined below, as well as some useful notations.

Definition 6.1 (Equivalent theorems)**.** Two theorems are considered equivalent if their conjunctive normal forms are equal modulo the order of conjuncts, disjuncts, and symmetry of equality. Given a theorem t, the set of the theorems equivalent to it in the library lib will be noted $E(lib, t)$.

Remark 14. This definition only makes sense if the two libraries can be represented in the same logic. This is straightforward if the two share the same logic.

Definition 6.2 (Notations)**.** Given a library, we define the following notations:

- $Dep(t)$ stands for the set of lemmas from which a theorem t was proved. We call them the dependencies of t. This definition is not recursive, i.e. the set does not include theorems used to prove these lemmas.

- The function $Learn()$ infers a relation between conjectures and sets of relevant lemmas from the relation between theorems and their dependencies.

- $Pred(c, L)$ is the set of lemmas related to a conjecture c predicted by the relation L.

In each scenario, each library plays an asymmetric role. In the following, the library where we want to prove the conjecture, is called the internal or the initial library. In contrast, the library from which we get extra advice from, is called the external library. In this context, using HOL(y)Hammer alone without any knowledge sharing is our default scenario, naturally named "internal predictions". We illustrate each selection method by giving an example of a theorem that could only be reproved by its strategy. These examples are extracted from our experiments described in Section 6.4.

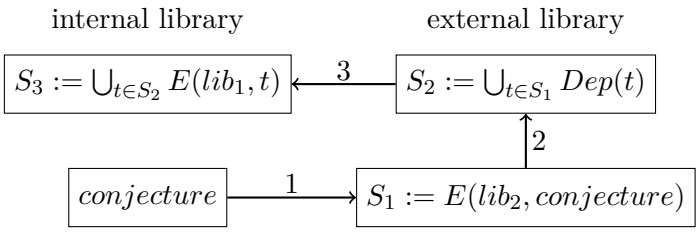


Figure 6.1: Finding lemmas from dependencies in the external library.

Scenario 1: External Dependencies

The first scenario assumes that the proof libraries are almost identical. We compute the set of theorems equivalent to the conjecture in the external library. For all of their dependencies, we return the lemmas in the library equivalent to these dependencies. The scenario is presented in Fig 6.1. This scenario would work very well, if the corresponding theorem is present in the external library and a sufficient corresponding subset of its dependencies is already present in the initial library. As this is often not the case (see Section 6.4), we will use an AI-ATP method next.

Example 6.3. The theorem `REAL_SUP_UBOUND` in HOL4 asserts that each element of a bounded subset of reals is less than its supremum. The equivalent theorem in HOL Light has 3 dependencies: the relation between $<$ and $\leq$ `REAL_NOT_LT`, the antisymmetry of $<$ `REAL_LT_REFL` and the definition of supremum `REAL_SUP`. Each of them have one equivalent in HOL4. The resulting problem was translated and solved by an ATP and the 3 lemmas appeared in the proof.

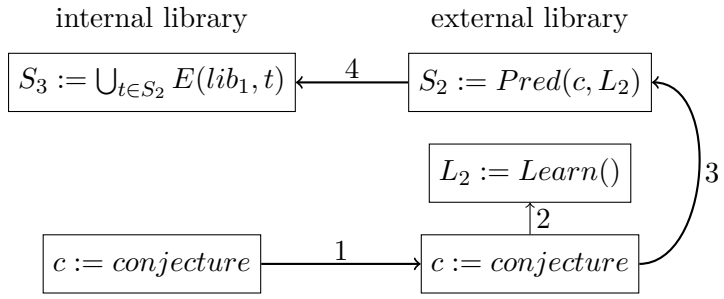


Figure 6.2: Learning and predicting lemmas in the external library

Scenario 2: External Predictions

The next scenario is depicted in Fig 6.2. The steps are as follows: We translate the conjecture to the external library (step 1). We predict the relevant lemmas in the external library (steps 2 and 3). We map the predicted lemmas back to the initial library using

their equivalents (step 4). To sum up, this scenario proposes an automatic way of proving a conjecture providing that the external library contains relevant lemmas that have equivalents in the internal library. One advantage of this scenario over the standard "internal predictions" is that the relation between features and dependencies is fully developed in the external library, yielding better predictions.

In our experiments, the translation step is not needed because the matching is already applied and the logic of our provers are the same.

Example 6.4. The theorem `LENGTH_FRONT` from the HOL4 theory `rich_list` states that the length of a non-empty list without its last element is equal to its length minus one. The subset of predicted lemmas used by the ATP were 6 theorems about natural numbers and 6 theorems about list. These theorems are HOL4 equivalents of selected HOL Light lemmas.

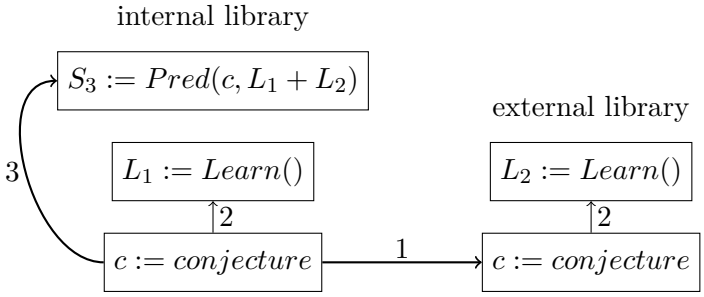


Figure 6.3: Learning in both libraries and predicting lemmas in the internal library.

Scenario 3: Combined Learning

In this and the next scenario we will combine the knowledge from the external library with the information already present in the internal library. The scenario is presented in Fig 6.3. First, the conjecture is translated to the external prover. Second, the features suitable for proving the conjecture are learned from the dependencies between the theorems in both systems. Third, lemmas from the original library containing these features are predicted. In a nutshell, this scenario defines an automatic method, that enhances the standard "internal predictions" by including advice from the external library about the relevance of each feature.

Example 6.5. This example and the next one are using advice from HOL4 in HOL Light which means that the roles of the two provers are reversed compared to the first two examples. The HOL Light theorem `SQRT_DIV` asserts that the square root of the quotient of two non-negative reals is equal to the quotient of their square roots. In this scenario no external theorems are translated but learning form the HOL4 proofs still improved the predictions directly made in HOL Light. The proof found for this theorem is based on the dual theorems for multiplication `SQRT_MUL` and inversion `SQRT_INV` and basic

properties of division `real_div`, multiplication `REAL_MUL_SYM`, inversion `REAL_LE_INV_EQ` and absolute value `REAL_ABS_REFL`.

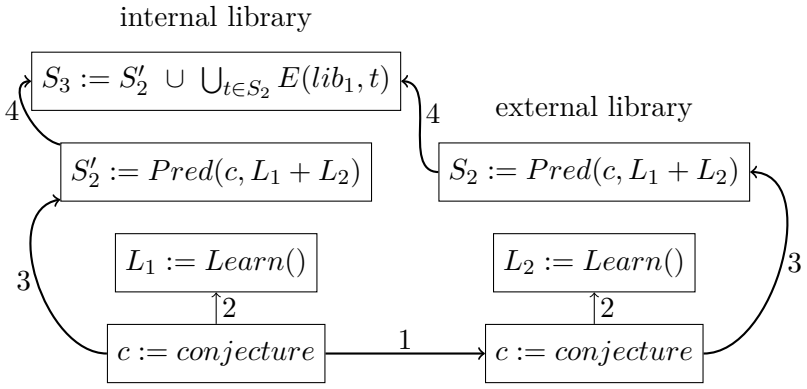


Figure 6.4: Learning and predicting lemmas from both libraries.

Scenario 4: Combined Predictions

The last and most developed scenario, shown in Fig 6.4, associate the strategies from the two preceding scenarios, effectively learning and predicting lemmas from both libraries. The first and second steps are the same as in "combined learning". The third step predicts lemmas in both libraries from the whole learned data. Finally, we map back the external predictions and return them together with the internal predictions.

Example 6.6. Let n, m, p be natural numbers.
The HOL Light theorem `HAS_SIZE_DIFF` declares that if a set A has n elements and B is a subset of A that has m elements then the difference $B \setminus A$ has $n - m$ elements. The first two lemmas necessary for the proof were directly found in HOL Light. One is the definition of the constant `HAS_SIZE` which asserts that a set has size p if and only if it is finite and has cardinality p. The other `CARD_DIFF` is almost the same as the theorem to be proved but stated for the cardinality of finite sets. The missing piece `FINITE_DIFF` is predicted inside the HOL4 library. Its equivalent in HOL Light declares that the difference of two finite sets is a finite set, which allows the ATP to conclude.

6.3.1 Unchecked Scenarios

In each of the previous scenarios, the final predicted lemmas come from the initial library. This means that our approach is sound with respect to the internal prover. The application of the matching substitution on one library renames the constants in all theorems injectively because no non-trivial matching is performed between two constants of the same library.

We will now consider the possibility of returning matched lemmas from the external library even if they do not have an equivalent in the internal one. This means giving

advice to the user in the form: "your conjecture can be proved using the theorems th_1 and th_2 that you already have and an additional hypothesis with the given statement which you should be able to prove." To verify that these scenarios are well-founded, a user would need to prove the proposed hypotheses. That could be achieved by either importing the theorems or applying the approach recursively. If a constant contained in these lemmas is matched inconsistently then each method would fail to reprove the lemmas, preserving the coherence of the internal library. We do not yet have an import mechanism from HOL4 to HOL Light (and conversely) or a recursive mechanism for our scenarios. In this recursive approaches, the predicted facts in the external library should be restricted to those proved before the conjecture when it has an equivalent in the external library. Otherwise, a loop in the recursive algorithm may be created.

We will still evaluate the "unchecked" scenarios to see what is the maximum added value such mechanisms could generate.

6.4 Evaluation

We perform all the experiments on a subset of the standard libraries of HOL Light and HOL4. The HOL4 dataset includes 15 type constructors, 509 constants, and 3935 theorems. The HOL Light dataset contains 21 type constructors, 359 constants and 4213 theorems. The subsets were chosen to include a variety of fields ranging from list to real analysis. The most similar pairs of theories are listed by their number of common equivalent classes of theorems in Table 6.1. The number of theorems in each theory is indicated in parenthesis.

HOL4 theory	HOL Light theory	common theorems
`pred_set`(434)	`sets`(490)	128
`real`(469)	`real`(291)	81
`poly`(87)	`poly`(142)	72
`bool`(177)	`theorems`(90)	61
`transc`(229)	`transc`(355)	58
`arithmetic`(385)	`arith`(245)	57
`integral`(83)	`transc`(355)	48

Table 6.1: The seven most similar pairs of theories by their number of common equivalent classes of theorems according to our matching

The matching, predictions, and the preparation of the ATP problems have been done on a laptop with 4 Intel Core i5-3230M 2.60GHz processors and 3.6 GB RAM. All ATP problems are evaluated on a server with 48 AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM and 0.5 MB L2 cache per CPU. A single core is assigned to each ATP problem. The ATP used is E prover version 1.8 running in the automatic mode with a time limit of 30 seconds.

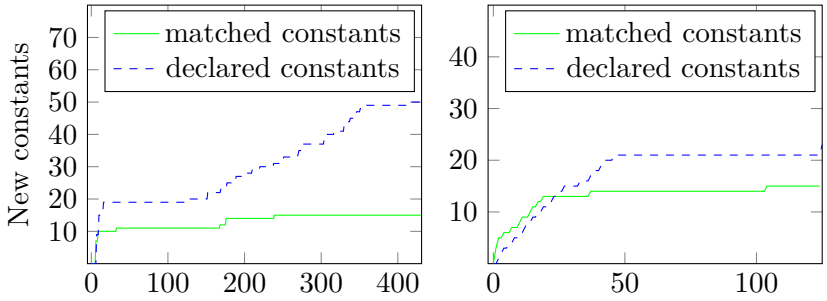


Figure 6.5: Evolution of the number of matched constants in the HOL4 theory `list` and in the HOL Light theory `lists`

Simulation

We will try to prove each theorem in an environment, where information is restricted to the one that was available when this theorem was proved. This amounts to:

- forgetting that it is a theorem and the knowledge of its dependencies,
- finding the subset of facts in the library that are accessible from this theorem,
- computing the matching with the other library based on this subset only,
- predicting lemmas from this subset (plus the other library in the "unchecked" scenarios).

For the purpose of our simulation, the external library is always completely known, as we suppose that it was created previously. In reality, the two libraries were developed in parallel, with many HOL4 theories available before similar formalizations in HOL Light have been performed.

In Fig. 6.5, we show the evolution of the number of matched constants and compare it to the number of declared constants in the theory during the incremental reproving of two theories. The first graph shows that the number of matched constants stagnate whereas the declared constants continue to increase in the second half of the theory. This suggests that theories formalizing the same concepts may be developed in different directions for each prover. The second graph indicates a better coverage of the HOL Light theory `lists`. In the beginning, the number of matched constants grows even more rapidly than the number of declared constants because new matches are found for constants defined in previous theories.

Results

Scenario	checked(%)	unchecked(%)
empty	4.19	
external dependencies	5.06 (23.50)	10.75 (49.94)
external predictions	17.49	34.42
external any	18.07	34.74
internal predictions	**43.57**	
combined learning	44.03	
combined predictions	44.59	53.46
any	**50.06**	55.73
any checked or unchecked	62.80	

Table 6.2: Percentage of reproved theorems in the HOL4 library (internal) with the knowledge from the HOL Light library (external).

Scenario	checked(%)	unchecked(%)
empty	3.14	
external dependencies	6.08 (29.22)	10.11 (48.63)
external predictions	12.74	33.94
external any	13.55	34.32
internal predictions	**30.92**	
combined learning	35.13	
combined predictions	35.56	44.06
any	**40.19**	47.07
any checked or unchecked	54.71	

Table 6.3: Percentage of reproved theorems in the HOL Light library (internal) with the knowledge from the HOL4 library (external).

Table 6.4: *
In the first column, scenarios are listed based on their predicted lemmas.
empty: no lemmas
external dependencies: dependencies of equivalent external theorems
external predictions: external lemmas from external advice
external any: problems solved by any of the two previous scenarios
internal predictions: internal lemmas from internal advice
combined learning: internal lemmas from external and internal advice
combined predictions: external and internal lemmas from external and internal advice
any: problems solved by at least one scenario of the same column
In the second column, we restrict ourself from using external theorems that do not have an internal equivalent, where as we allow it in the third column. The last line combines all the problems solved by at least one checked or unchecked scenario.

The success rates for each scenario and each proof assistant are compiled in Tables 6.2 and 6.3. The scenario "empty" gives the number of facts provable without lemmas and is fully subsumed by the other methods.

The external dependencies scenario is the only one that is not directly comparable to the others, as it was performed only on the theorems that have an equivalent in the other library (876 in HOL Light and 847 in HOL4). The percentage of theorems proved by this strategy relative to its experimental subset is shown in parentheses. This strategy is quite efficient on its subset but contributes weakly to the overall improvement. These results are combined with the "external predictions" scenario to evaluate what can be reproved with external help only. In HOL4, the combined learning and predictions increases the number of problems solved over the initial "internal predictions" approach only by one percent. The improvement is sharper in HOL Light. It suggests that HOL4 provides a better set for the learning algorithm. The improvement provided by all scenarios can be combined to yield a significant gain compared to the performance of HOL(y)Hammer

alone, namely additional 6.5% of all HOL4 and 9.3% of all HOL Light theorems. Another 10–15% could be added by the "unchecked" scenarios.

Results by Theory

In Table 6.5, we investigate the performance of the "external dependencies" scenario on the largest theories in our dataset. Some theories only minimally benefit from the external help. This is the case for `rich_list` and `iterate`, where only few correct mappings could be found. We can see asymmetric results in pairs of similar theories. For example, the `real` theory in HOL Light can be 72.16% reproved from HOL4 theories whereas the similar theory in HOL4 does not benefit as much. This suggest that the `real` theory HOL4 is more dense than its counterpart. A similar effect is observed for the `transc` formalization. The theories `pred_set` and `sets` seem to be comparably dense.

Scenario	`real`	`pred_set`	`list`	`arithmetic`	`rich_list`	`transc`
external dependencies	30.91	24.65	10.23	18.18	1.52	5.24

Scenario	`sets`	`analysis`	`transc`	`int`	`iterate`	`real`
external dependencies	25.51	27.1	25.91	52.61	5.47	72.16

Table 6.5: Reproving success rate in the six largest theories in HOL4 using HOL Light and the "checked external dependencies" scenario, as well as in the six largest HOL Light theories using HOL4.

6.5 Conclusion

We proposed several methods for combining the knowledge of two ITP systems in order to prove more theorems automatically. The methods adapt the premise selection and proof advice components of the HOL(y)Hammer system to include the knowledge of an external prover. In order to do it, the concepts defined in both libraries are related through an improved matching algorithm. As the constants in two libraries become related, so are the statements of the theorems. Machine learning algorithms can combine the information about the dependencies in each library to predict useful dependencies more accurately.

We evaluated the influence of an external library on the quality of advice, by reproving all the theorems in a large subset of the HOL4 and HOL Light standard libraries. External knowledge can improve the success from 43% to 50% in HOL4 and from 30% to 40% in the number of HOL Light solved goals. This number could reach 54% for HOL4 and 62% for HOL Light if we include the "unchecked" scenarios, where the user is not only suggested known theorems, but also hypotheses left to prove. Proving such proposed lemmas, either with the help of a translation or by calling an AI-ATP method with shared knowledge is left as future work.

The proposed approach evaluated the influence of an external proof assistant library for the quality of learning and prediction. An extension of the approach could be used inside a single library: mappings of concepts inside a single library, such as those the work of Autexier and Hutter [AH15], could provide additional knowledge for a learning-reasoning system.

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF): P26201.

Chapter 7

Statistical Conjecturing

Abstract

A critical part of mathematicians' work is the process of conjecture-making. This involves observing patterns in and between mathematical objects, and transforming such patterns into conjectures that describe or better explain the behavior of the objects. Computer scientists have since long tried to reproduce this process automatically, but most of the methods were typically restricted to specific domains or based on brute-force enumeration methods. In this work we propose and implement methods for generating conjectures by using statistical analogies extracted from large formal libraries, and provide their initial evaluation.

7.1 Introduction

In the past decade, there has been considerable progress in proving conjectures over large formal corpora such as Flyspeck [Hal12], isabelle/hol [NPW02], the Mizar Mathematical Library (MML) [GKN10] and others. This has been achieved by combining high-level learning or heuristic fact-selection mechanisms with a number of methods and strategies for guiding the strongest (typically superposition-based) automated theorem provers (ATPs). While this approach has not reached its limits [BKPU16], and its results are already very useful, it still seems quite far from the way humans do mathematics. In particular, even with very precise premise (fact) selection, today's ATPs have trouble finding many longer Mizar and Flyspeck proofs of the toplevel lemmas in their libraries. In fact, only a few of such lemmas would be called a "theorem" by a mathematician. Often the "theorem" would be just the final proposition in a formal article, and the toplevel lemmas preceding the theorem would be classified as simple technical steps that sometimes are not even mentioned in informal proofs.

An important inductive method that mathematicians use for proving hard problems is *conjecturing*, i.e., proposing plausible lemmas that could be useful for proving a hard problem.[1] There are likely a number of complicated inductive-deductive feedback loops

[1] A famous example is the Taniyama-Shimura conjecture whose proof by Wiles finished the proof of Fermat's Last Theorem.

involved in this process that will need to be explored, however it seems that a major inductive method is *analogy*. In a very high-level way this could be stated as: *"Problems and theories with similar properties are likely to have similar proof ideas and lemmas."*

Again, analogies can likely be very abstract and advanced. Two analogue objects may be related through a morphism. They can also be two instances of the same algebraic structure. The organization of such structures was recently addressed by the MMT framework [Rab13] in order to represent different logics in a single consistent library. In this work we start experimenting with the analogies provided by *statistical concept matching* [GK14]. This method has been recently developed in order to align formal libraries of different systems and to transfer lemmas between them [GK15b]. Here we use statistical concept matching to find the *most similar sets of concepts* inside one large library. The discovered sets of matching concepts can then be used to translate a hard conjecture C into a "related" conjecture C', whose (possible) proof might provide a further guidance for proving C. The remaining components that we use for this first experiment are standard large-theory reasoning components, such as fast machine-learners that learn from the thousands of proofs and propose the most plausible lemmas for proving the related conjectures, and first-order ATPs – in this case we use Vampire 4.0 [KV13].

7.2 Matching Concepts

In order to apply some analogies, we first need to discover what they are by finding similar concepts. For our initial evaluation, our similarity matching will be limited to concepts represented by constants and ground sub-terms. Later this can be extended to more complex term structures. We describe here a general concept matching algorithm inspired and improved upon [GK14] and discuss how this algorithm can be adapted to find analogies within one library.

7.2.1 Matching Concepts between Two Libraries

Given two theorem libraries, the first step is to normalize all statements, as this increases the likelihood of finding similar theorem shapes. If two theorems have the same shape (that we will call such abstract shapes a property), then the concrete constants appearing in this two theorems at the same positions are more likely to be similar. We will say that such pairs of constants are derived from the theorem pair.

Example 7.1. Given the theorems T_1 and T_2 with the statements, their respective normalizations, and the properties extracted from their statements:

$$T_1 : \ \forall x : num.\ x = x + 0 \qquad T_2 : \ \forall x : real.\ x = x \times 1$$

$$P_1 : \lambda num, +, 0.\ \forall x : num\ x = x + 0 \qquad P_2 : \lambda real, \times, 1.\ \forall x : real.\ x = x \times 1$$

The properties P_1 and P_2 are α-equivalent, therefore the theorems T_1 and T_2 form a matching pair of theorems, and the following three matching pairs of constants are derived:

$$num \leftrightarrow real,\ + \leftrightarrow \times,\ 0 \leftrightarrow 1$$

We further observe that the matchings $(+, \times)$ and $(0, 1)$ are in relation through the theorem pair (T_1, T_2). The strength of this relation – *correlation* – is given by the number and accuracy of the theorem pairs these matchings are jointly derived from. We will call the graph representing the correlations between different matchings the *dependency graph*. The *similarity score* of each matching (i.e., pair of concepts) is then initialized with a fixed starting value and computed by a dynamical process that iterates over the dependency graph until a fixpoint is reached.

The principal advantage of this method is that the algorithm is not greedy, allowing a concept to match multiple other concepts in the other libraries. This is a desirable property, since we want to be able to create multiple analogues for one theorem. Matchings are then combined into *substitutions*, which are in turn applied to the existing theorem statements yielding plausible conjectures.

Very few adjustments are needed to adapt this method to a single library. We create a duplicate of the library and match it with its copy. Since we are not interested in identity substitutions, we prevent theorems from matching with their copies. However, we keep self matches between constants in the dependency graph since good analogies can be often found by using partial substitutions.

7.3 Context-dependent Substitutions

Constant matchings by themselves create special one-element substitutions that transport the properties of one constant to another. In general, substitutions are created from translating more than one constant. Suppose, that we know that addition is similar to union and multiplication to intersection. We can hope to transport the distributivity of multiplication over addition to the distributivity of intersection over union. Therefore, the correlations between the matchings are crucial for building the substitutions.

We now present two methods for creating substitutions from a theorem. These methods are based on the correlations between the concept pairs and the similarity score of each concept pair.

We want to construct substitutions that are most relevant in the local context, i.e., for the symbols that are present in the theorem we want to translate (*target theorem*).

The first method starts by reducing the dependency graph to the subgraph of all concept pairs whose first element is contained in the target theorem. We first select a starting node in this subgraph which is not an identity matching, and recursively find nodes (possibly identities) that are connected by the strongest edges of the subgraph, under the constraint that no two selected nodes have the same first element. The algorithm stops when no new node can be added. The final set of nodes obtained in this way forms a partial substitution. We run this algorithm either for all starting nodes, or for those with similarity scores above a certain threshold. This produces a set of substitutions, which are effectively the most relevant substitutions for the target theorem. This process seems to produce many substitutions, however in practice, many of them are identical, which limits their total number.

The second method is a brute-force approach where we first find the set of concepts

($Matched(T)$) that match at least one concept in the set of concepts ($Concepts(T)$) present in the target theorem T. We would like to create all possible substitutions between $Concepts(T)$ and $Matched(T)$ and rank them, however this would often blow up the generation phase. To limit the number of possible substitutions, we remove possible matches by iterating the following process until the number of substitutions is below a chosen threshold (1000): We select the constant C in T with most remaining matchings, remove its worst match, recompute the number of remaining matches for all constants in T, and check if the number of substitutions is already below the threshold. If so, the process terminates, otherwise we continue the iteration.

Next, we select the 200 substitutions with the highest *combined score*. The combined score of a substitution S is computed by multiplying the average correlation and similarity in S, i.e. formally as follows:

$$CombinedScore(S) = AverageCorrelation(S) \times AverageSimilarity(S)$$

$$AverageCorrelation(S) = \frac{1}{|S|^2} \times \sum_{M \in S, M' \in S} Correlation(M, M')$$
$$AverageSimilarity(S) = \frac{1}{|S|} \times \sum_{M \in S} SimilarityScore(M)$$

On top of these combined scores, the diversity of substitutions can be maximized by the following shuffling. The process iteratively chooses a (not yet selected) substitution with the best *diversity score* and increases the set of selected substitutions $\mathfrak{T}$. These scores are then updated to penalize the substitutions that have more matchings in common with the already selected ones.

$$CommonMatchings(S, T) = |S \cap T|$$
$$DiversityScore(S) = \frac{CombinedScore(S)}{(1 + \sum_{T \in \mathfrak{T}} CommonMatchings(S, T))^3}$$

These substitutions will eventually be applied to the initial theorem to create new conjectures. We hope that if such conjectures can be proved, they will improve the AI/ATP methods by enriching the theory. We consider in Section 7.4 two possible scenarios where the combination of conjecturing by analogies and premise selection could be useful.

7.4 Scenarios

We will consider two scenarios for the use of conjecturing: without and with a user given goal.

In the first scenario no goal set by the user, and our system should decide what is the next step in the development of the whole library. In this context, our algorithm

produces the most likely conjectures by applying the first substitution generation method on all theorems. We then evaluate the conjectures by trying to prove them. We estimate the difficulty of a proved conjecture by the number of lemmas that were used in its proof. We estimate the relevance of a conjecture by how much adding this conjecture as a new theorem in the library helps to automatically prove formalized statements appearing after it in the library.

In the second scenario, a goal is given and the system should figure a way how to prove it. The typical AI-ATP method is to search through all lemmas (in an evaluation this means the lemmas which occur before the goal), and select the most relevant ones using machine learning techniques. If however an important lemma was not proven or proven after the goal, the premise selection fails to produce it. Therefore, we propose a way to produce some of such relevant missing lemmas. This method is depicted in Fig 7.1. We first select the 20 best scoring substitutions for this goal, which creates 20 new goals. We then try to prove them using the AI-ATP system. If some of them are successfully proved, we obtain small sets of lemmas which were sufficient to prove the translated goals. These lemmas are then translated back to the original concepts. We run our AI-ATP system again and remove those it cannot prove. The final step is to try to prove the user goal from those additional lemmas. This strategy simulates the following thought process a human could have: "Can I prove this conjecture for a similar concept"?, "If so, what where the necessary lemmas?" "If some of this lemmas holds for my original concept, they would likely help me in my original proof".

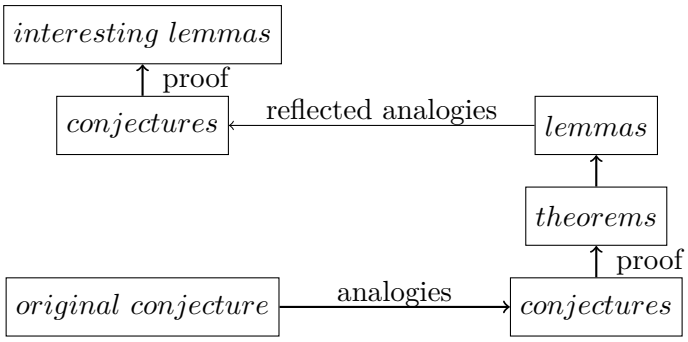


Figure 7.1: Creating additional relevant lemmas for a conjecture through analogies.

7.5 Experiments

7.5.1 Untargeted Conjecture Generation

In the first scenario, we apply[2] the 20 "most plausible" substitutions to all MML theorems, and attempt to prove a small part (73535) of those, each time using the whole MML. We want to see how complicated and interesting the conjectures can be. After premise

[2]The experimental version of our conjecturer used here is available at:
`http://147.32.69.25/~mptp/conj/conjecturing_standalone.tar.gz` .

selection Vampire can prove 10% (7346) of them in 10 s, which is reduced to 4464 after pseudo-minimization [KU14] and removing tautologies and simple consequences of single lemmas. An example of a reasonably interesting conjecture (and analogy) with a new nontrivial proof is the convexity of empty subsets of real linear spaces, induced from a similar claim about them being "circled".[3]

```
registration
  let X be non empty RLSStruct;
  cluster empty -> circled for Element of bool the carrier of X;
```

Here "circled" is defined as[4]

```
definition
  let X be non empty RLSStruct; let A be Subset of X;
  attr A is circled means :Def6: :: RLTOPSP1:def 6
  for r being Real st abs r <= 1 holds r * A c= A;
```

and "convex" as[5]

```
definition
  let V be non empty RLSStruct; let M be Subset of V;
  attr M is convex means :Def2: :: CONVEX1:def 2
  for u, v being VECTOR of V
    for r being Real st 0 < r & r < 1 & u in M & v in M holds
      (r * u) + ((1 - r) * v) in M;
```

For example the following properties of circled[6] and convex[7] subsets are quite similar, leading the conjecturer into conjecturing further properties like the one stated above.

```
registration
  let X be RealLinearSpace;
  let A, B be circled Subset of X;
  cluster A + B -> circled ;

theorem :: CONVEX1:2
 for V being non empty Abelian add-associative vector-distributive
   scalar-distributive scalar-associative scalar-unital RLSStruct
 for M, N being Subset of V st M is convex & N is convex holds
   M + N is convex
```

[3] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/rltopsp1.html#CC1
[4] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/rltopsp1.html#V3
[5] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/convex1.html#V1
[6] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/rltopsp1.html#FC3
[7] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/convex1.html#T2

7.5.2 Targeted Conjecture Generation

In the second experiment, we have used as our target the set of 22069 *ATP-hard* Mizar toplevel lemmas (theorems). These are the theorems that could not be proved in any way (neither with human-advised premises, nor with learned premise selection) in our recent extensive experiments with state-of-the-art AI/ATP methods over the MML [KU15d]. For the current experiment, those experiments are very thorough. They used high ATP time limits, many ATPs, their targeted strategies, a number of learning methods and their combinations, and a number of iterations of the learning/proving loop, approaching in total a month of a server time. Proving these problems with a low time limit and a single AI/ATP method is thus quite unlikely.

To each such hard theorem T we apply the 20 best (according to the diversity score) substitutions. Such substitutions are additionally constrained to target only the part of the MML that existed before T was stated. This gives rise to 441242 conjectures, which we attempt to prove – again only with the use of the MML that precedes T. Because of resource limits, we use only one AI/ATP method: k-NN with 128 best premises, followed by Vampire using 8 s. This takes about 14 hours on a 64-CPU server, proving 9747 (i.e. 2.2%) of the conjectures. We do two rounds of pseudo-minimization and remove tautologies and simple consequences of single lemmas. This leaves 3414 proved conjectures, originating from 1650 hard theorems, i.e., each such conjecture C is a translation of some hard theorem T under some plausible substitution σ ($C = \sigma(T)$). We translate the MML lemmas L_C^i needed for the proof of C "back" into the "terminology of T" by applying to them the reverse substitution σ^{-1}.

This results in 26770 back-translated conjectures. For each of them we hope that (i) it will be provable from the MML preceding T, and (ii) it will be useful for proving its T, since its image under σ was useful for proving $\sigma(T)$. We use again only 8 s to select those that satisfy (i), yielding after the minimization and removal of trivial ones 2170 proved back-translated lemmas, originating from 500 hard theorems. For each of these 500 theorems T we do standard premise selection (using slices of size 128 and 64) over the preceding MML, and add these lemmas (obviously only those that "belong" to T) to the premises of T. Then we run Vampire for 30 s on the standard and lemma-augmented problems. While there is no difference when using 128 lemmas, for 64 lemmas we obtain (in addition to the 6 proofs that both the standard and augmented methods find) an interesting new proof of the theorem `MATHMORP:25`[8], which cannot be found by Vampire using the standard method even with a time limit of 3600 s.

```
theorem :: MATHMORP:25
for T being non empty right_complementable Abelian
            add-associative right_zeroed RLSStruct
for X, Y, Z being Subset of T
  holds X (+) (Y (-) Z) c= (X (+) Y) (-) Z
```

To find this proof, our concept matcher first used the statistical analogy between addition[9]

[8] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/mathmorp.html#T25
[9] http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/rusub_4.html#K6

and subtraction[10] in additive structures (`addMagma`). By doing that, it inadvertently constructed as a conjecture the theorem `MATHMORP:26`[11], that actually immediately succeeds `MATHMORP:25` in MML. This alone is remarkable, because this theorem was not known at the point when `MATHMORP:25` was being proved. Using premise selection and Vampire, `MATHMORP:26` was proved in 4 s, and a particular back-translated lemma from its proof turned out to be provable and crucial for proving `MATHMORP:25` automatically. This lemma is actually "trivial" for Mizar, since it follows by climbing Mizar's extensive type hierarchy [GKN10] from an existing typing of the `"(-)"` function. However, as mentioned above, we were not able to get this proof without this lemma even with much higher time limits.

7.6 Conclusion and Future Work

We have investigated the application of a concept matching algorithm to formulate conjectures through analogies. We have described a way to combine it with premise selection methods and ATPs. This was designed to create potential intermediate lemmas that help an ATP to find complex proofs.

While these are just first experiments, it seems that statistical concept matching can occasionally already come up with plausible conjectures without resorting to the (in large libraries rather impossible) brute-force term enumeration methods. So far we do not even use any of the manually invented heuristic methods such as those pioneered by Lenat [Len76] and fajtlowicz [Faj88], and rather rely on a data-driven approach. Such heuristics and other methods could be combined with the statistical ones.

We can likely improve the matching algorithm by allowing the concepts to be represented by more complex term structures [VSU10]. This may help us to connect concepts from more different domains. In the same direction, we could also relax our concept of properties to allow matching with errors. A more generic solution would be to try different shapes of theorems using substitutions trees or genetic programming, but this might need efficient implementation.

We can also modify how the matching algorithm and the AI-ATP system are combined. A simple approach is to enhance the premise selection algorithm of the AI-ATP system with the discovered similarities between concepts. In our experiments we also observe an increasing number of conjectures given by the number of possible substitutions. A heuristic semantic evaluation could complement the substitutions scores to estimate the likely of a conjecture to be true.

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF) grant P26201 and by the European Research Council (ERC) grant AI4REASON.

[10] `http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/mathmorp.html#K3`
[11] `http://mizar.cs.ualberta.ca/~mptp/7.13.01_4.181.1147/html/mathmorp.html#T26`

Chapter 8

Learning to Reason with Tactics

Abstract

Techniques combining machine learning with translation to automated reasoning have recently become an important component of formal proof assistants. Such "hammer" techniques complement traditional proof assistant automation as implemented by tactics and decision procedures. In this paper we present a unified proof assistant automation approach which attempts to automate the selection of appropriate tactics and tactic-sequences combined with an optimized small-scale hammering approach. We implement the technique as a tactic-level automation for HOL4: TacticToe. It implements a modified A*-algorithm directly in HOL4 that explores different tactic-level proof paths, guiding their selection by learning from a large number of previous tactic-level proofs. Unlike the existing hammer methods, TacticToe avoids translation to FOL, working directly on the HOL level. By combining tactic prediction and premise selection, TacticToe is able to re-prove 39% of 7902 HOL4 theorems in 5 seconds whereas the best single HOL(y)Hammer strategy solves 32% in the same amount of time.

8.1 Introduction

Example 8.1. Proof automatically generated by TacticToe for the user given goal

```
Goal: ‘‘∀l. FOLDL (λxs x. SNOC x xs) [] l = l‘‘
Proof:
  SNOC_INDUCT_TAC THENL
  [ REWRITE_TAC [APPEND_NIL, FOLDL],
    ASM_REWRITE_TAC [APPEND_SNOC, FOLDL_SNOC]
      THEN CONV_TAC (DEPTH_CONV BETA_CONV)
      THEN ASM_REWRITE_TAC [APPEND_SNOC, FOLDL_SNOC] ]
```

Many of the state-of-the-art interactive theorem provers (ITPs) such as HOL4 [SN08], HOL Light [Har09], Isabelle [WPN08] and Coq [BC04] provide high-level parameterizable tactics for constructing the proofs. Such tactics typically analyze the current goal state and assumptions, apply nontrivial proof transformations, which get expanded into possibly many basic kernel-level inferences or significant parts of the proof term. In this work we

develop a tactic-level automation procedure for the HOL4 ITP which guides selection of the tactics by learning from previous proofs. Instead of relying on translation to first-order automated theorem provers (ATPs) as done by the hammer systems [BKPU16, GK15a], the technique directly searches for sequences of tactic applications that lead to the ITP proof, thus avoiding the translation and proof-reconstruction phases needed by the hammers.

To do this, we *extract and record* tactic invocations from the ITP proofs (Section 8.2) and *build efficient machine learning classifiers* based on such training examples (Section 8.3). The learned data serves as a guidance for our *modified A*-algorithm* that explores the different proof paths (Section 8.4). The result, if successful, is a certified human-level proof composed of HOL4 tactics. The system is evaluated on a large set of theorems originating from HOL4 (Section 8.5), and we show that the performance of the single best TacticToe strategy exceeds the performance of a hammer system used with a single strategy and a single efficient external prover.

Related Work There are several essential components of our work that are comparable to previous approaches: tactic-level proof recording, tactic selection through machine learning techniques and automatic tactic-based proof search. Our work is also related to previous approaches that use machine learning to select premises for the ATP systems and guide ATP proof search internally.

For HOL Light, the Tactician tool [Ada15b] can transform a packed tactical proof into a series of interactive tactic calls. Its principal application was so far refactoring the library and teaching common proof techniques to new ITP users. In our work, the splitting of a proof into a sequence of tactics is essential for the tactic recording procedure, used to train our tactic prediction module.

The system ML4PG [KHG12, HK14] groups related proofs thanks to its clustering algorithms. It allows Coq users to inspire themselves from similar proofs and notice duplicated proofs. Our predictions comes from a much more detailed description of the open goal. However, we simply create a single label for each tactic call whereas each of its arguments is treated independently in ML4PG. Our choice is motivated by the k-NN algorithm already used in HOL(y)Hammer for the selection of theorems.

SEPIA [GWR15] is a powerful system able to generate proof scripts from previous Coq proof examples. Its strength lies in its ability to produce likely sequences of tactics for solving domain specific goals. It operates by creating a model for common sequences of tactics for a specific library. This means that in order to propose the following tactic, only the previously called tactics are considered. Our algorithm, on the other hand, relies mainly on the characteristics of the current goal to decide which tactics to apply next. In this way, our learning mechanism has to rediscover why each tactic was applied for the current subgoals. It may lack some useful bias for common sequences of tactics, but is more reactive to subtle changes. Indeed, it can be trained on a large library and only tactics relevant to the current subgoal will be selected. Concerning the proof search, SEPIA's breadth-first search is replaced by an A*-algorithm which allows for heuristic guidance in the exploration of the search tree. Finally, SEPIA was evaluated on three

chosen parts (totaling 2382 theorems) of the Coq library demonstrating that it globally outperforms individual Coq tactics. In contrast, we demonstrate the competitiveness of our system against the successful general-purpose hammers on the HOL4 standard library (7902 theorems).

Machine learning has also been used to advise the best library lemmas for new ITP goals. This can be done either in an interactive way, when the user completes the proof based on the recommended lemmas, as in the Mizar Proof Advisor [Urb04], or attempted fully automatically, where such lemma selection is handed over to the atp component of a *hammer* system [BKPU16, GK15a, KU14, BGK+16, KU15d].

Internal learning-based selection of tactical steps inside an ITP is analogous to internal learning-based selection of clausal steps inside ATPs such as MaLeCoP [UVŠ11] and femalecop [KU15a]. These systems use the naive Bayes classifier to select clauses for the extension steps in tableaux proof search based on many previous proofs. Satallax [Bro13] can guide its search internally [FB16] using a command classifier, which can estimate the priority of the 11 kinds of commands in the priority queue based on positive and negative examples.

8.2 Recording Tactic Calls

Existing proof recording for HOL4 [Won95, KH12] relies on manual modification of all primitive inference rules in the kernel. Adapting this approach to record tactics would require the manual modification of the 750 declared HOL4 tactics. Instead, we developed an automatic transformation on the actual proofs. Our process singles out tactic invocations and introduces calls to general purpose recording in the proofs. The main benefit of our approach is an easy access to the string representation of the tactic and its arguments which is essential to automatically construct a human-level proof script. As in the LCF-style theorem prover users may introduce new tactics or arguments with the `let` construction inline, special care needs to be taken so that the tactics can be called in any other context. The precision of the recorded information will influence the quality of the selected tactics in later searches. The actual implementation details of the recording are explained in Section 8.6.

8.3 Predicting Tactics

The learning-based selection of relevant lemmas significantly improves the automation for hammers [BGK+16]. Therefore we propose to adapt one of the strongest hammer lemma selection methods to predict tactics in our setting: the modified distance-weighted *k nearest-neighbour* (k-NN) classifier [KU13b, Dud76]. Premise selection usually only prunes the initial set of formulas given to the ATPs, which then try to solve the pruned problems on their own. Here we will use the prediction of relevant tactics to actively guide the proof search algorithm (described in Section 8.4).

Given a goal g, the classifier selects a set of previously solved goals similar to g, and considers the tactics that were used to solve these goals as relevant for g. As the predictor

bases its relevance estimation on frequent similar goals, it is crucial to estimate the distance between the goals in a mathematically relevant way. We will next discuss the extraction of the features from the goals and the actual prediction. Both have been integrated in the SML proof search.

8.3.1 Features

We start by extracting the syntactic features that have been successfully used in premise selection from the goal:

- names of constants, including the logical operators,
- type constructors present in the types of constants and variables,
- first-order subterms (fully applied) with all variables replaced by a single place holder V.

We additionally extract the following features:

- names of the variables,
- the top-level logical structure with atoms substituted by a single place holder A and all its substructures.

We found that the names of variables present in the goal are particularly important for tactics such as case splitting on a variable (`Cases_on var`) or instantiation of a variable (`SPEC_TAC var term`). Determining the presence of top-level logical operators (i.e implication) is essential to assess if a "logical" tactic should be applied. For example, the presence of an implication may lead to the application of the tactic `DISCH_TAC` that moves the precondition to the assumptions. Top-level logical structure gives a more detailed view of the relationship between those logical components. Finally, we also experiment with some general features because they are natural in higher-order logic:

- (higher-order) subterms with all variables unified, including partial function applications.

8.3.2 Scoring

In all proofs, we record each tactic invocation and link (associate) the currently open goal with the tactic's name in our database. Given a new open goal g, the *score of a tactic T wrt. g* is defined to be the score (similarity) of T's associated goal which is most similar to g. The idea is that tactics with high scores will be more likely to solve the open goal g, since they were able to solve similar goals before.

We estimate the similarity (or co-distance) between an open goal g_o and a previously recorded goal g_p using their respective feature sets f_o and f_p. The co-distance $tactic\_score_1$ computed by the k-NN algorithm is analogous to the one used in the premise selection task [KU13b]. The main idea is to find the features shared by the

two goals and estimate the rarity of those features calculated via the TF-IDF [Jon04] heuristics. In a second co-distance $tactic\_score_2$, we additionally take into account the total number of features to reduce the seemingly unfair advantage of big feature sets in the first scoring function.

$$tactic\_score_1(f_o, f_p) = \sum\nolimits_{f \in f_o \cap f_p} \text{tfidf}(f)^{\tau_1}$$

$$tactic\_score_2(f_o, f_p) = \frac{tactic\_score_1(f_o, f_p)}{(1 + ln\ (1 + card\ f_o))}$$

Moreover, we would like to compare the distance of a recorded goal to different goals opened at different moment of the search. That is why we normalize the scores by dividing them by the similarity of the open goal with itself. As a result, every score will lie in the interval $[0, 1]$ where 1 is the highest degree of similarity (i.e. the shortest distance). We respectively refer to those normalized scores later as $tactic\_norm\_score_1$ and $tactic\_norm\_score_2$.

8.3.3 Preselection

Since the efficiency of the predictions will be crucial during the proof search, we preselect 500 tactics before the search. A sensible approach here is to preselect a tactic based on the distance between the statement of the conjecture to be proven and the statement(s) for which the tactic is part of the proof. During the proof, when an open goal is created, only the scores of the 500 preselected tactics will be recalculated and the tactics will be reordered according to these scores.

8.3.4 Orthogonalization

Different tactics may transform a single goal in the same way. Exploring such equivalent paths is undesirable, as it leads to inefficiency in automated proof search. To solve this problem, we do not directly assign a goal to the associated tactic, but organize a competition on the closest feature vectors (tactic string together with the features of an associated goal). The winner is the tactic appearing in the most feature vectors provided that it has the same effect as the original tactic. We associate this tactic with the features of the targeted goal instead of the original in our feature database. As a result, already successful tactics are preferred, and new tactics are considered only if they provide a different contribution.

8.3.5 Self-learning

If the search algorithm finds a proof, we record both the human and computer-generated proof in the feature database. Since recording and re-proving are intertwined, the additional data is available for the next proof search. The hope is that it will be easier for TacticToe to learn from its own discovered proofs than from the human proof scripts [Urb07].

8.4 Proof Search Algorithm

Despite the best efforts of the prediction algorithms, the selected tactic may not solve the current goal, proceed in the wrong direction or even loop. For that reason, the prediction needs to be accompanied by a proof search mechanism that allows for backtracking and can choose which proof tree to extend next and in which direction.

Our search algorithm takes inspiration from the A*-algorithm [HNR68] which uses a cost function and heuristics to estimate the total distance to the destination and choose the shortest route. The first necessary adaptation of the algorithm stems from the fact that a proof is in general not a path but a tree. This means that our search space has two branching factors: the choice of a tactic, and the number of goals produced by tactics. The proof is not finished when the current tactic solves its goal because it often leaves new pending open goals along the current path.

Algorithm Description In the following, we assume that we already know the distance function (it will be defined in 8.4.1) and describe how the A*-algorithm is transformed into a proof search algorithm. In order to help visualizing the proof steps, references to a proof search example depicted in Figure 8.1 will be made throughout the description of the algorithm. To minimize the width of the trees in our example the branching factor is limited to two tactics $tactic_1$ and $tactic_2$ but a typical search relies on 500 preselected tactics.

Our search algorithm starts by creating a root node containing the conjecture as an open goal. A list of 500 potential tactics is attached to this node. A score for each of those tactics is given by the tactic selection algorithm. The tactic with the best score ($tactic_2$ in our example) is applied to the conjecture. If no error occurs, it produces a new node containing a list of goals to be solved. The first of these goals ($goal_1$) is the open goal for the node, other goals ($goal_2$) are pending goals waiting for the first goal to be proved. From now, we have more than one node that can be extended, and the selection process has two steps: First, we select the best unused tactic for each open goal ($tactic_1$ for $goal_1$, $tactic_1$ for the $conjecture$). Next, we chose the node ($goal_1$) with the highest co-distance (see next paragraph) which is supposed to be the closest to finish the proof. The algorithm goes on creating new nodes with new open goals ($goal_3$) until a tactic ($tactic_2$) proves a goal ($goal_1$). This is the case when a tactic returns an empty list of goals or if all the goals directly produced by the tactic have already been proven. At this point, all branches originating from the node of the solved goal are deleted and the tactic that led to the proof is saved for a later reconstruction (see Section 8.4.2).

The whole process can stop in three different ways. The conjecture is proven if all goals created by a tactic applied to the conjecture are closed. The search saturates if no predicted tactics are applicable to any open goals. The process times out if it runs longer than the fixed time limit (5 seconds in our experiments).

Optimizations A number of constraints are used to speed up the proof search algorithm. We forbid the creation of nodes that contain a parent goal in order to avoid loops. We minimize parallel search by imposing that two sibling nodes must not contain the same

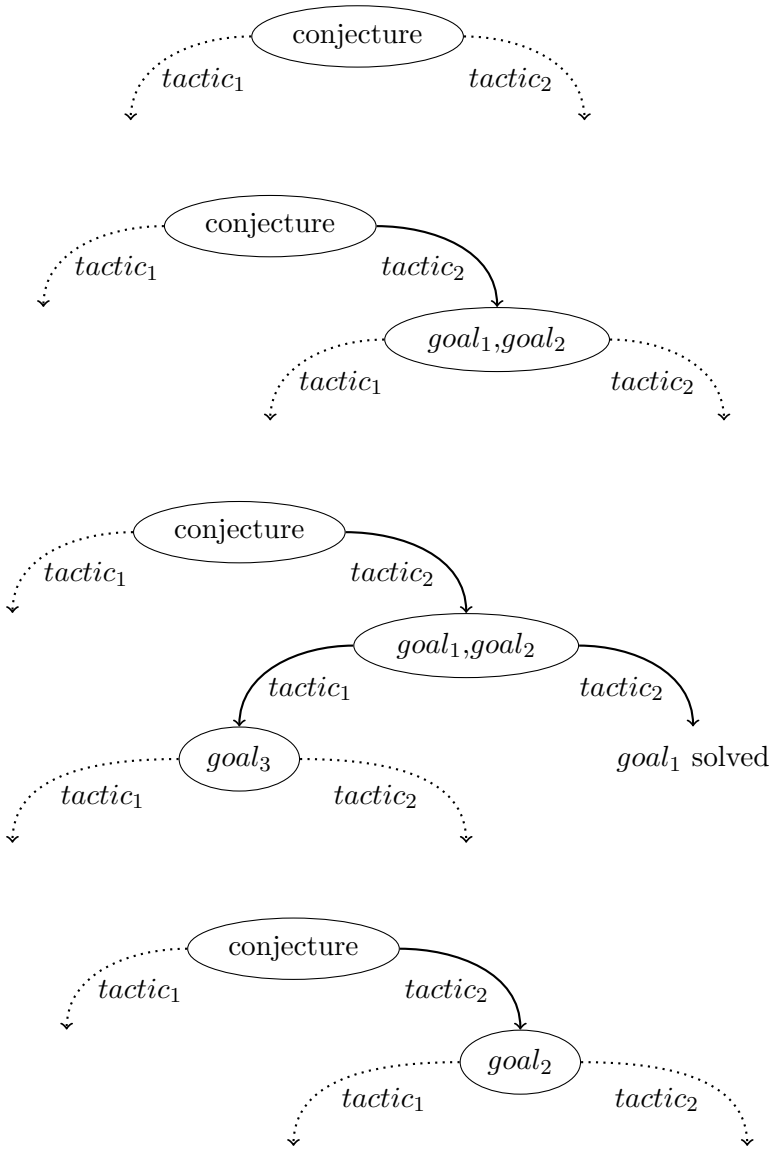


Figure 8.1: 4 successive snapshots of a proof attempt showing the essential steps of the algorithm: node creation, node extension and node deletion.

set of goals. We cache the tactic applications and the predictions so that they can be replayed quickly if the same open goals reappears anywhere in the search. Tactics are restricted to a very small time limit, the default being 0.02 seconds in our experiment. Indeed, a tactic may loop or take a large amount of time, which would hinder the whole search process. Finally, we reorder the goals in each node so that the hardest goals according to the selection heuristic are considered first.

8.4.1 Heuristics for Node Extension

It is crucial to define a good distance function for the (modified) A*-algorithm. This distance (or co-distance) should estimate for the edges of each node, how close it is to complete the proof. For the heuristic part, we rely on the score of the best tactic not yet applied to the node's first goal. Effectively, the prediction scores evaluate a co-distance to a provable goal, with which we approximate the co-distance to a theorem. A more precise distance estimation could be obtained by recording the provable subgoals that have already been tried [KU15c], however this is too costly in our setting. We design the cost function, which represents the length of the path already taken as a coefficient applied to the heuristics. By changing the parameters, we create and experiment with 5 possible co-distance functions:

$$\begin{aligned} codist_1 &= tactic\_norm\_score_1 \\ codist_2 &= tactic\_norm\_score_2 \\ codist_3(k_1) &= k_1^d * tactic\_norm\_score_1 \\ codist_4(k_1, k_2) &= k_1^d * k_2^w * tactic\_norm\_score_1 \\ codist_5(k_1, k_2) &= k_1^d * k_2^w \end{aligned}$$

where d is the depth of the considered node, w is the number of tactics previously applied to the same goal and k_1, k_2 are coefficients in $]0, 1[$.

Remark 15. If $k_1 = k_2$, the fifth co-distance has the same effect as the distance $d + w$.

Admissibility of the Heuristic and Completeness of the Algorithm An important property of the A*-algorithm is the admissibility of its heuristic. A heuristic is admissible if it does not overestimate the distance to the goal. The fifth co-distance has no heuristic, so it is admissible. As a consequence, proof searches based on this co-distance will find optimal solutions relative to its cost function. For the third and fourth co-distances, we can only guarantee a weak form of completeness. If there exists a proof involving the 500 preselected tactics, the algorithm will find one in a finite amount of time. It is sufficient to prove that eventually the search will find all proofs at depth $\leq k$. Indeed, there exists a natural number n, such that proofs of depth greater than n have a cost coefficient smaller than the smallest co-distance at depth $\leq k$. Searches based on the first two co-distances are only guided by their heuristic and therefore incomplete. This allows them to explore the suggested branches much deeper.

Remark 16. The completeness result holds only if the co-distance is positive, which happens when top-level logical structures are considered.

In the future, we consider implementing the UCT-method [BPW+12] commonly used as a selection strategy in Monte-Carlo tree search. This method would most likely find a better balance between completeness and exploration.

8.4.2 Reconstruction

When a proof search succeeds (there are no more pending goals at the root) we need to reconstruct a HOL4 human-style proof. The saved nodes consist of a set of trees where each edge is a tactic and the proof tree is the one starting at the root. In order to obtain a single HOL4 proof, we need to combine the tactics gathered in the trees using tacticals. By the design of the search, a single tactic combinator, `THENL`, is sufficient. It combines a tactic with a list of subsequent ones, in such a way that after the parent tactic is called, for each created goal a respective tactic from the list is called. The proof tree is transformed into a final single proof script by the following recursive function P taking a tree node t and returning a string:

$$P(t) = \begin{cases} P(c) & \text{if } t \text{ is a root,} \\ tac & \text{if } t \text{ is a leaf,} \\ tac \texttt{ THENL } [P(c_0), \ldots, P(c_n)] & \text{otherwise.} \end{cases}$$

where tac is the tactic that produced the node, c is the only successful child of the root and $c_0, \ldots, c_n$ are the children of the node produced by the successful tactic.

The readability of the created proof scripts is improved, by replacing replacing `THENL` by `THEN` when the list has length 1. Further post-processing such as removing unnecessary tactics and theorems has yet to be developed but would improve the user experience greatly [Ada15b].

8.4.3 Small "hammer" Approach

General-purpose proof automation mechanisms which combine proof translation to ATPs with machine learning ("hammers") have become quite successful in enhancing the automation level in proof assistants [BKPU16]. As external automated reasoning techniques often outperform the combined power of the tactics, we would like to combine the TacticToe search with HOL(y)Hammer for HOL4 [GK15a]. Moreover our approach can only use previously called tactics, so if a theorem is essential for the current proof but has never been used as an argument of a tactic, the current approach would fail.

Unfortunately external calls to HOL(y)Hammer at the proof search nodes are too computationally expensive. We therefore create a "small hammer" comprised of a faster premise selection algorithm combined with a short call to the internal prover Metis [Hur03]. First, before the proof search, we preselect 500 theorems for the whole proof search tree using the usual premise selection algorithm with the dependencies. At each node a simpler selection process will select a small subset of the 500 to be given to Metis using

a fast similarity heuristic (8 or 16 in our experiment). The preselection relies on the theorem dependencies, which usually benefits hammers, however for the final selection we only compute the syntactic feature distance works better.

During the proof search, when a new goal is created or a fresh pending goal is considered, the "small hammer" is always called first. Its call is associated with a tactic string for a flawless integration in the final proof script.

8.5 Experimental Evaluation

The results of all experiments are available at:

`http://cl-informatik.uibk.ac.at/users/tgauthier/tactictoe/`

8.5.1 Methodology and Fairness

The evaluation imitates the construction of the library: For each theorem only the previous human proofs are known. These are used as the learning base for the predictions. To achieve this scenario we re-prove all theorems during a modified build of HOL4. As theorems are proved, they are recorded and included in the training examples. For each theorem we first attempt to run the TacticToe search with a time limit of 5 seconds, before processing the original proof script. In this way, the fairness of the experiments is guaranteed by construction. Only previously declared SML variables (essentially tactics, theorems and simpsets) are accessible. And for each theorem to be re-proven TacticToe is only trained on previous proofs.

Although the training process in each strategy on its own is fair, the selection of the best strategy in Section 8.5.2 should also be considered as a learning process. To ensure the global fairness, the final experiments in Section 8.5.3 runs the best strategy on the full dataset which is about 10 times larger. The performance is minimally better on this validation set.

8.5.2 Choice of the Parameters

In order to efficiently determine the contribution of each parameter, we design a series of small-scale experiments where each evaluated strategy is run on every tenth goal in each theory. A smaller dataset (training set) of 860 theorems allows testing the combinations of various parameters. To compare them, we propose three successive experiments that attempt to optimize their respective parameters. To facilitate this process further, every strategy will differ from a default one by a single parameter. The results will show in addition to the success rate, the number of goals solved by a strategy not solved by another strategy X. This number is called $U(X)$.

The first experiment concerns the choice of the right kind of features and feature scoring mechanism. The results are presented in Table 8.1. We observe that the higher-order features and the feature of the top logical structure increase minimally the number of problems solved. It is worth noting that using only first-order features leads to 18 proofs

ID	Learning parameter	Solved	$U(D_1)$
D_0	$codist_1$ (length penalty)	172 (20.0%)	5
D_1	$codist_0$ (default)	179 (20.8%)	0
D_2	no top features	175 (20.3%)	18
D_3	no higher-order features	178 (20.7%)	8

Table 8.1: Success rate of strategies with different learning parameters on the training set.

not found by relying on additional higher-order features. The attempted length penalty on the total number of features is actually slightly harmful.

In the next experiment shown in Table 8.2, we focus our attention on the search parameters. To ease comparison, we reuse the strategy D_1 from Table 8.1 as the default strategy. We first try to change the tactic timeout, as certain tactics may require more time to complete. It seems that the initial choice of 0.02 seconds per tactic inspired by hammer experiments [GKKN15] involving Metis is a good compromise. Indeed, increasing the timeout leaves less time for other tactics to be tried, whereas decreasing it too much may prevent a tactic from succeeding. Until now, we trusted the distance heuristics completely not only to order the tactics but also to choose the next extension step in our search. We will add coefficients that reduce the scores of nodes deep in the search. From D_6 to D_8, we steadily increase the strength of the coefficients, giving the cost function of the A*-algorithm more and more influence on the search. The success rate increases accordingly, which means that using the current heuristics is a poor selection method for extending nodes. A possible solution may be to try to learn node selection independently from tactic selection. So it is not surprising that the strategy D_9 only relying on the cost function performs the best. As a minor consolation, the last column shows that the heuristic-based proof D_1 can prove 10 theorems that D_9 cannot prove. Nevertheless, we believe that the possibility of using a heuristic as a guide for the proof search is nice asset of TacticToe.

The third experiment, presented in Table 8.3, evaluates the effect of integrating the "small hammer" in the TacticToe search. At a first glance, the increased success rate is significant for all tested parameters. Further analysis reveals that increasing the number of premises from 8 to 16 with a timeout of 0.02 seconds is detrimental. The D_{19} experiment demonstrates that 0.1 seconds is a better time limit for reasoning with 16 premises. And the D_{18} experiment reveals the disadvantage of unnecessarily increasing the timeout of Metis. This reduces the time available for the rest of the proof search, which makes the success rate drop.

The best strategy which does not rely on the "small hammer" approach D_9 will be called TacticToe (NH) (for no "small hammer") in the remaining part of the paper, and the best strategy relying on the approach, D_{19}, will be referred to as TacticToe (SH) ("small hammer").

ID	Searching parameter	Solved	$U(D_9)$
D_1	$codist_0$ (default)	179 (20.8%)	10
D_4	tactic timeout 0.004 sec	175 (20.3%)	9
D_5	tactic timeout 0.1 sec	178 (20.7%)	10
D_6	$codist_3(0.8)$	192 (22.3%)	10
D_7	$codist_4(0.8, 0.8)$	199 (23.1%)	7
D_8	$codist_4(0.4, 0.4)$	205 (23.8%)	3
D_9	$codist_5(0.8, 0.8)$	211 (24.5%)	0

Table 8.2: Success rate of strategies with different search parameters on the training set.

ID	"small hammer" parameter	Solved	$U(D_{19})$
D_9	$codist_5(0.8, 0.8)$ (default: no small hammer)	211 (24.5%)	19
D_{16}	8 premises + timeout 0.02 sec	281 (32.7%)	21
D_{17}	16 premises + timeout 0.02 sec	270 (31.4%)	18
D_{18}	8 premises + timeout 0.1 sec	280 (32.6%)	11
D_{19}	16 premises + timeout 0.1 sec	289 (33.6%)	0

Table 8.3: Success rate of strategies with different parameters of "small hammer" on the training set.

8.5.3 Full-scale Experiments

We evaluate the two best TacticToe strategies on a bigger data set. Essentially, we try to re-prove every theorem for which a tactic proof script was provided. The majority of theorems in the HOL4 standard library (7954 out of 10229) have been proved this way. The other theorems were created by forward rules and almost all of those proofs are bookkeeping operations such as instantiating a theorem or splitting a conjunction.

In addition we evaluate the two proposed more advanced strategies: self-learning and orthogonalization.

We will also compare the performance of TacticToe with the HOL(y)Hammer system HOL4 [GK15a], which has so far provided the most successful general purpose proof automation. Although HOL(y)Hammer has already been thoroughly evaluated, we reevaluate its best single strategy to match the conditions of the TacticToe experiments. Therefore, this experiment is run on the current version of HOL4 with a time limit of 5 seconds. The current best strategy for HOL(y)Hammer in HOL4 is using E prover [Sch02, Sch13b] with the *new_mzt_small* strategy discovered by `BliStr` [Urb15]. To provide a baseline the less powerful *auto* strategy for E prover was also tested.

The evaluation of TacticToe is performed as part of the HOL4 build process, whereas HOL(y)Hammer is evaluated after the complete build because of its export process.

ID	Parameter	Solved	$U(\mathsf{TacticToe}(SH))$
TacticToe (NH)	default	2349 (29.73%)	173
TacticToe (SH)	"small hammer"	3115 (39.42%)	$U(blistr) : 1335$
TacticToe (E_2)	self-learn	2343 (29.66%)	187
TacticToe (E_3)	self-learn + ortho	2411 (30.51%)	227
HOL(y)Hammer (auto)	E knn 128 auto	1965 (24.87%)	525
HOL(y)Hammer (blistr)	E knn 128 blistr	2556 (32.35%)	776

Table 8.4: Full-scale experiments and comparison of the different strategies on a common dataset of 7902 theorems

The consequence is that overwritten theorems are not accessible to HOL(y)Hammer. Conversely, each theorem which was proved directly through forward rules was not considered by TacticToe. To estimate the relative strength of the two provers in a fair manner we decided to compute all subsequent statistics on the common part of the dataset. This common part consists of 7902 theorems from 134 theories.

Table 8.4 gathers the results of 4 TacticToe strategies and 2 HOL(y)Hammer strategies. Combining the advantages of tactic selection done by TacticToe with premise selection gives best results. Indeed, the combined method TacticToe (SH) solves 39.42% on the common dataset whereas the best HOL(y)Hammer single-strategy only solves 32.35% of the goals. Surprisingly, the effect of self learning was a little negative. This may be caused by the fact that recording both the human proof and the computer-generated script may cause duplication of the feature vectors which happens when the proofs are similar. The effect of this duplication is mitigated by the orthogonalization method which proves 62 more theorems than the default strategy. We believe that testing even stronger learning schemes is one of the most crucial steps in improving proof automation nowadays.

Table 8.5 compares the success rates of re-proving for different HOL4 theories. TacticToe (SH) outperforms TacticToe (NH) on every considered theory. Even if a lot weaker due to the missing premise selection component, TacticToe (NH) is hugely better than HOL(y)Hammer (blistr) in 4 theories: `measure`, `list`, `sorting` and `finite_map`. The main reason is that those theories are equipped with specialized tactics, performing complex transformation such as structural induction and TacticToe (NH) can reuse them. Conversely, HOL(y)Hammer (blistr) is more suited to deal with dense theories such as `real` or `complex` where a lot of related theorems are available and most proofs are usually completed by rewriting tactics.

8.5.4 Reconstruction

96% of the HOL(y)Hammer (auto) (E prover with auto strategy) proofs can be reconstructed by Metis in 2 seconds using only the dependencies returned by the prover. In comparison, all the TacticToe successful proofs could be reconstructed and resulted in

proof scripts that were readable by HOL4 and solved their goals.

Furthermore, the generated proof returned by TacticToe is often more readable and informative than the single Metis call returned by HOL(y)Hammer. Since each tactic calls had a time limit of 0.02 seconds, the reconstructed proof is guaranteed to prove the goal in a very short amount of time. Those considerations indicate that often TacticToe generated scripts can contribute to the development of formal libraries in a smoother manner.

8.5.5 Time and Space Complexity

Here, we try to gain some insight by measuring different proof search variables. We will keep track of the total number of nodes in the proof state, the time it took to get a successful proof and the size of the final proof script. In Table 8.6, the number of nodes is computed over failing searches whereas the average time and the proof size is evaluated over successful searches. We estimate the total search space explored by the number of nodes. It is 4 times larger in the TacticToe (NH) version because of the lack of time consuming Metis calls. The average proof size (number of tactics in the final script) is around 3 even without explicit Metis invocations. A detailed analysis of the proofs by their size in Fig. 8.2 confirms the fact that most proofs are short. One of the proofs happens to be 39 steps long, but it is not a common case. This indicates the need to focus on high-quality predictions. Since the currently recorded tactics may not cover enough space, a way to generate new tactic calls may be necessary. The depiction of the numbers of problems solved in a certain amount of time in Fig. 8.3 shows that it is increasingly harder to solve new goals. Nevertheless, it seems that our strongest strategy TacticToe (SH) can benefit most from an increased time limit.

The total time of a search is split into 5 parts: predictions, tactic application, node creation, node selection, and node deletion. Usually in a failing search, the total prediction time takes less than a second, the tactic applications consume one to two seconds and the rest is used by the node processing parts. A simple improvement would be to reduce the bookkeeping part in a future version of TacticToe.

	arith	real	compl	meas
TacticToe (NH)	37.3	19.7	42.6	19.6
TacticToe (SH)	60.1	46.1	63.7	22.1
HOL(y)Hammer (blistr)	51.9	66.8	72.3	13.1
	proba	list	sort	f_map
TacticToe (NH)	25.3	48.1	32.7	53.4
TacticToe (SH)	25.3	51.9	34.7	55.5
HOL(y)Hammer (blistr)	25.3	23.3	16.4	18.1

Table 8.5: Percentage (%) of re-proved theorems in the theories `arithmetic`, `real`, `complex`, `measure`, `probability`, `list`, `sorting` and `finite_map`.

ID	nodes		proof size		time
	average	max	average	max	average
TacticToe (NH)	94.66	421	3.34	39	0.66
TacticToe (SH)	25.27	407	2.34	34	0.83

Table 8.6: search statistics

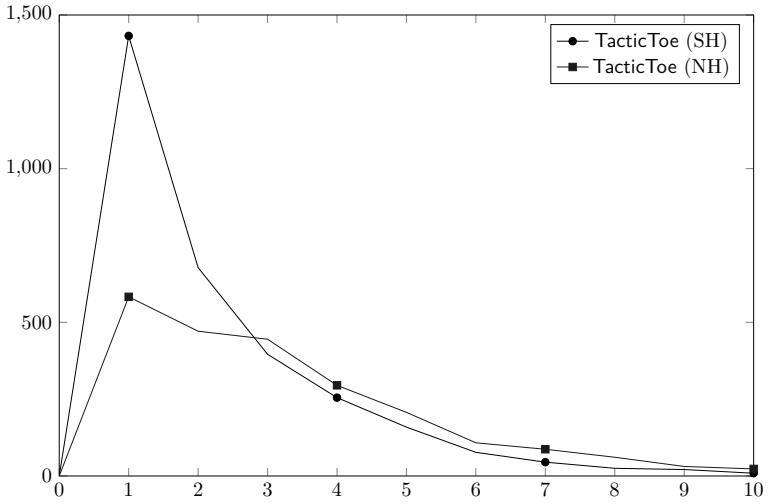


Figure 8.2: Number of searches (y axis) that result in a proof of size exactly x (x axis).

8.5.6 Case Study

Investigating further the different qualities of TacticToe, we study its generated proof scripts on an example in `list` theory (see Example 8.2). The theorem to be proven states the equivalence between the fact that a number n is greater than the length of a list ls with the fact that dropping n elements from this list returns an empty list.

The human proof proceeds by induction on n followed by solving both goals using rewrite steps combined with an arithmetic decision procedure. Both TacticToe proofs (NH and SH) follow the general idea of reasoning by induction but solve the base case and the inductive case in a different way. The base case only needs rewriting using the global simpset in the TacticToe (NH) proof, which is simulated by a call to Metis in the (SH) proof. The inductive case should require an arithmetic decision procedure as hinted by the human proof. This is achieved by rewriting using an arithmetic simpset in the second proof. In the first proof however, a rewriting step and case splitting step were used to arrive at a point where Metis calls succeed. The tactic proof produced by TacticToe (NH) often looks better than the one discovered by TacticToe (SH) in that it does not involve Metis calls with a large numbers of premises.

Example 8.2. (In theory `list`)

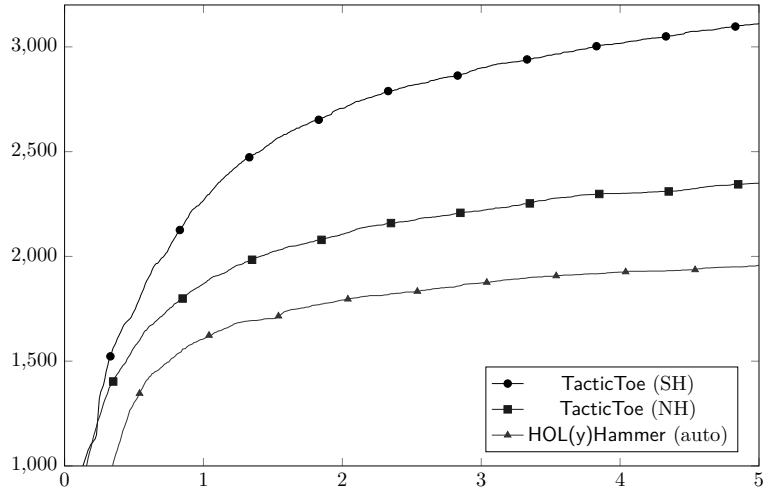


Figure 8.3: Number of problems solved in less than t seconds.

```
Goal: ‘‘∀ls n. (DROP n ls = []) ⇔ n ≥ LENGTH ls‘‘

Human proof: LIST_INDUCT_TAC THEN SRW_TAC [] [] THEN DECIDE_TAC

TacticToe (NH) proof: LIST_INDUCT_TAC THENL [SRW_TAC [] [], SRW_TAC [ARITH_ss] []]

TacticToe (SH) proof:
  LIST_INDUCT_TAC THENL
  [ METIS_TAC [...],
    NTAC 2 GEN_TAC THEN SIMP_TAC (srw_ss ()) [] THEN
    Cases_on ‘n‘ THENL [METIS_TAC [...], METIS_TAC [...]] ]
```

8.6 Recording Tactic Calls

We present the implementation details of recording tactic calls from the LCF-style proof scripts of HOL4. We first discuss parsing the proofs and identifying individual tactic calls. We next show how tactics calls are recorded together with their corresponding goals using a modified proof script. Finally, the recorded data is organized as feature vectors with tactics as labels and characteristics of their associated goals as features. These feature vectors constitute the training set for our selection algorithm.

8.6.1 Extracting Proofs

Our goal is to do a first-pass parsing algorithm to extract the proofs and give them to a prerecording function for further processing. For that purpose, we create a custom theory rebuilder that parses the string representation of the theory files and modifies them. The proofs are extracted and stored as a list of global SML declarations. The rebuilder then

inserts a call to the prerecorder before each proof with the following arguments: the proof string, the string representation of each declaration occurring before the proof, the theorem to be proven and its name. After each theory file has been modified, a build of the HOL4 library is triggered and each call of the prerecording function will perform the following steps: identifying tactics, globalizing tactics and registering tactic calls. The effects of those steps will be depicted on a running example taken from a proof in the list theory.

Example 8.3. Running example (original call)

```
val MAP_APPEND = store_thm ("MAP_APPEND",
  --‘!(f:’a->’b).!l1 l2. MAP f (APPEND l1 l2) = APPEND (MAP f l1) (MAP f l2)‘--,
    STRIP_TAC THEN LIST_INDUCT_TAC THEN ASM_REWRITE_TAC [MAP, APPEND]);
```

Example 8.4. Running example (extracted proof)

```
   "STRIP_TAC THEN LIST_INDUCT_TAC THEN ASM_REWRITE_TAC [MAP, APPEND]"
```

8.6.2 Identifying Tactics in a Proof

Parsing proofs is a more complex task than extracting them due to the presence of infix operators with different precedences. For this reason, in this phase we rely on the Poly/ML interpreter to extract tactics instead of building a custom parser. In theory, recording nested tactics is possible, but we decided to restrict ourselves to the outermost tactics, excluding those constructed by a tactical (see list in Example 8.5). The choice of the recording level was made to reduce the complexity of predicting the right tactic and minimizing the number of branches in the proof search. In particular, we do not consider `REPEAT` to be a tactical and record `REPEAT X` instead of repeated calls to `X`.

Example 8.5. `THEN ORELSE THEN1 THENL REVERSE VALID by suffices_by`

Example 8.6. Running example (identified tactics)

```
"STRIP_TAC" "LIST_INDUCT_TAC" "ASM_REWRITE_TAC [MAP, APPEND]"
```

8.6.3 Globalizing Tactics

The globalization process attempts to modify a tactic string so that it is understood by the compiler in the same way anywhere during the build of HOL4. In that manner, the TacticToe proof search will be able to reuse previously called tactics in future searches. A first reason why a tactic may become inaccessible is that the module where it was declared is not open in the current theory. Therefore, during the prerecording we call the Poly/ML compiler again to obtain the module name (signature in SML) of the tactic tokens. This prefixing also avoids conflicts, where different tactics with the name appears in different module. There are however some special cases where the module of a token is not declared in a previous module. If the token is a string, already prefixed, or a SML reserved token then we do not need to perform any modifications. If a value is declared in the current theory (which is also a module), we replace the current value (or

function) by its previous declaration in the file. This is done recursively to globalize the values. Theorems are treated in a special manner. Thanks to the HOL(y)Hammer tagging system [KU14], they can be in most cases fetched from the HOL4 database. Terms are reprinted with their types to avoid misinterpretation of overloaded constants.

Since certain values in HOL4 are stateful (mostly references), we cannot guarantee that the application of a tactic will have exactly the same effect in a different context. This is not a common issue, as a fully functional style is preferred, however there is one important stateful structure that we need to address: the simplification set is stored globally and the simplification procedures rely on the latest version available at the moment of the proof.

Example 8.7. Running example (globalized tactics)

The tactic `LIST_INDUCT_TAC` is not defined in the signature of the *list* theory. That is why, to be accessible in other theories its definition appears in its globalization.

```
"Tactic.STRIP_TAC"
"let val LIST_INDUCT_TAC = Prim_rec.INDUCT_THEN
( DB.fetch \"list\" \"list_INDUCT\" ) Tactic.ASSUME_TAC in LIST_INDUCT_TAC
end"
"Rewrite.ASM_REWRITE_TAC
[( DB.fetch \"list\" \"MAP\" ) , ( DB.fetch \"list\" \"APPEND\" ) ]"
```

8.6.4 Registering Tactic Calls

To judge the effectiveness of a tactic on proposed goals, we record how it performed previously in similar situations. For that, we modify the proofs to record and associate the globalized tactic with the goal which the original tactic received. Each original tactic is automatically modified to perform this recording as a side effect. The code of the record function `R` is defined below in Example 8.8. The first line checks if the globalized tactic `gtac` produces the same open goals as the original tactic. In the second line we save the globalized tactic and the features of the goal to a file. Storing features instead of goals was preferred in order to avoid unnecessary recomputation. It is also more convenient since features can be stored as a list of strings. In the running example only constant features are presented (the complete set of extracted features was discussed in Section 8.3.1). Finally, the original tactic is called to continue the proof.

Example 8.8. Pseudo-code of the recording function

```
fun R (tac,gtac) goal =
  (test_same_effect gtac tac goal; save (gtac, features_of goal); tac goal)
```

Example 8.9. Running example (recording proof string) R is the recording function

```
( ( R ( STRIP_TAC , "Tactic.STRIP_TAC" ) ) ) THEN
( ( R ( LIST_INDUCT_TAC , "( let val LIST_INDUCT_TAC = Prim_rec.INDUCT_THEN
( DB.fetch \"list\" \"INDUCT\") Tactic.ASSUME_TAC in LIST_INDUCT_TAC end )" )
) ) THEN
```

```
( R ( ASM_REWRITE_TAC [ MAP , APPEND ] , "Rewrite.ASM_REWRITE_TAC
[( DB.fetch \"list\" \"MAP\" ) , ( DB.fetch \"list\" \"APPEND\" ) ]" ) )
```

The application of the recording function `R` and its subcalls will only take place during a second HOL4 build where the proofs have been replaced by their recording variants. This replacement will be performed by a modified version of the rebuilder that extracted the proofs. It will also create a call to our search algorithm before the recording proof and a call to a post-recorder after it. The post-recorder will create feature vectors consisting of the name of the current theorem, its features and every globalized tactic in the proof. This second set is used to preselect the tactics before trying to re-prove a theorem (see Section 8.3.3).

8.7 Conclusion

We proposed a new proof assistant automation technique which combines tactic-based proof search, with machine learning tactic prediction and a "small hammer" approach. Its implementation, TacticToe, achieves an overall performance of 39% theorems on the HOL4 standard library surpassing HOL(y)Hammer best single-strategy and proving 1335 additional theorems. Its effectiveness is especially visible on theories which use inductive data structures, specialized decision procedures, and custom built simplification sets. Thanks to the learning abilities of TacticToe, the generated proof scripts usually reveal the high-level structure of the proof. We therefore believe that predicting ITP tactics based on the current goal features is a very reasonable approach to automatically guiding proof search, and that accurate predictions can be obtained by learning from the knowledge available in today's large formal proof corpora.

There is plenty of future work in the directions opened here. To improve the quality of the predicted tactics, we would like to predict their arguments independently. To be even more precise, the relation between the tactic arguments and their respective goals could be used. Additionally, we could aim for a tighter combination with the ATP-based hammer systems. This would perhaps make TacticToe slower, but it might allow finding proofs that are so far both beyond the ATPs and TacticToe's powers. The idea of reusing high-level blocks of reasoning and then learning their selection could also be explored further in various contexts. Larger frequent blocks of (instantiated) tactics in ITPs as well as blocks of inference patterns in ATPs could be detected automatically, their usefulness in particular proof situations learned from the large corpora of ITP and ATP proofs, and reused in high-level proof search.

Acknowledgments

This work has been supported by the ERC Consolidator grant no. 649043 *AI4REASON* and ERC starting grant no. 714034 *SMART*.

Chapter 9

Conclusion

To conclude, I first give a summary of the main ideas presented during the course of this thesis and the effect they have through their implementation on the state of proof automation in Section 9.1. Then, I discuss a natural continuation to this work following the research directions explored in this thesis in Section 9.2.

9.1 Summary

In this section, I recapitulate the various approaches developed in this thesis that led to this increase in proof automation assistance for ITP users. My main angle of attack was the use of statistical machine learning methods backed by an understanding of mathematical logic. These methods automatically recognize similarities and patterns occurring in formulas. The abstract knowledge, that ensues from this learning, guides the proof search mechanisms of HOL(y)Hammer and TacticToe.

In the first part of the thesis, I concentrated on extending the support for proof automation by increasing the capabilities of HOL(y)Hammer. For this, I upgraded HOL(y)Hammer into a platform where ITPs and ATPs could connect. Practically, I extracted the machine learning data from HOL4 to connect it with HOL(y)Hammer and contributed to the adaptation of the interface to HOL Light.

Then, to expand the mathematical knowledge available to machine learning reasoners, I investigated how to transfer this knowledge between theorem provers. Thanks to the expertise of my supervisor Kaliszyk, I realized that the most important and challenging requirement for such transfer was to recognize which concepts are similar in the different theorem provers. I evaluated how this shared knowledge influences the premise selection of HOL(y)Hammer on the library pair HOL4-HOL Light. And I exploited approximate alignments in the concept matching algorithm to create conjectures in Mizar.

Finally, I created the tactical theorem prover TacticToe on top of HOL4. In this work, I took advantage of human mathematical knowledge that is not present in the syntactical representation theorems such as specialized algorithms (e.g. arithmetic simplifications, equation solvers). By recording each tactic (algorithm in HOL4) and the context in which they are used, TacticToe was able to choose suitable tactics for a new goal. This resulted in a large improvement over general strategies provided by HOL(y)Hammer which only rely on theorem selection for proof guidance.

Each new proving method was evaluated against a set of ITP theorems. I tested many important parameters for TacticToe and HOL(y)Hammer: external ATPs, number of

predictions, generated features, accessible set of lemmas and different timeouts. Similarly, many algorithms for the alignment system were designed and tested: normalization, conceptualization, disambiguation.

Overall, this thesis proposed and tested new proof automation techniques that go beyond the hammer technology. My work on the interoperability between ITPs, which was conceived with the goal of creating more knowledge for learning-assisted reasoning systems in mind showed promising results. The main beneficiaries are HOL4 users that now have two new learning-assisted reasoning systems at their disposition: HOL(y)Hammer and TacticToe. Effort was made to make the installation and use of each system as user-friendly as possible. Consequently, proving theorems in HOL4 is becoming more attractive. Many of the minor uninteresting steps that developers struggle with can be discharged by HOL(y)Hammer and TacticToe.

9.2 Vision

As a way forward, I propose to explore in more depth two important ideas that could lead to breakthroughs in automated reasoning: learning and creativity. Many of the created AI methods could benefit from stronger models and a larger database. Moreover, there is some synergy that could be exploited by re-targeting different algorithms. For instance, the conjecturing method that is employed to synthesize terms could be re-purposed for tactic synthesis. Next, I would like to extend the possibility of our system to be able to prove more and more complex goals. I believe that this can only be achieved when our system is able to define its own concepts and design its own tactics. Finally, to avoid heading in the wrong research direction and give relevant feedback, the evaluation framework should be updated to capture the effect of newly developed reasoning techniques.

9.2.1 Machine Learning Models

In order to guide more accurately the proof search, a logical step is to improve the accuracy and generalization power of our predictors. Therefore, I consider replacing the weighted k-nearest neighbor by more general machine learning predictors such as random forests or neural networks. The main issue is that these models so far provided limited improvements. One possible reason is that the data currently retrieved from theorem provers is not extensive enough. I envision three ways in which this problem can be solved: gathering mathematical knowledge from different systems, more detailed proof recording and through self-learning (also known as reinforcement learning [SB98]).

Knowledge Sharing In this thesis, we only evaluated the effect of sharing knowledge between two provers on proof automation. Yet, we have matched concepts between six ITPs. These libraries contain together more than 100,000 top-level theorems which could be enough for more data-heavy predictors. If not, we will need to find how to merge this knowledge with informal databases such as Wikipedia.

Deeper Recording A straightforward way to extract more data from proofs is to record proofs at three different levels: the top-level theorems and their dependencies, the sequence of tactics and the trace of kernel inference rules. In general, there is a trade-off between the granularity of the recording and the difficulty to learn and exploit effectively the recorded data. With a more precise recording such as the kernel trace, much more training data is generated, so stronger models and longer training are necessary. Moreover, since kernel steps are small, the proof search needs to explore further the proof tree. So, the computational complexity will explode unless predictions become close to perfect. However, the fundamentals learned by the fine-grained model are also an advantage. By learning how to combine small steps into bigger ones, the action space becomes much larger. With this a future system might be able to find proofs (or shorter proofs) that are not obvious from a high-level perspective.

Reinforcement Learning In order to increase the database of learned fact, our system can rely on its own previous proof attempts to increase its knowledge in the next run. So far, I have tested only one iteration of this process on TacticToe and saw a minimal improvement over the supervised learning method of the original system. With stronger models and by increasing the number of learning iterations, TacticToe (or any other machine-learning based proof automation) might be able to learn more from its successes and failures. This approach has been successful in the domain of perfect information games such as Go [SSS+17].

9.2.2 Emulating Creativity in Mathematics

In a second avenue of research, I would like to discuss ways of reasoning that go beyond the current proof automation where a step by step reasoning algorithm is used to derive theorems from previous ones. My goal is to create a set of innovative methods that is closer to the way mathematicians do mathematics. I think that being able to synthesize proof objects without formally deriving them should be a corner stone for advanced reasoning software. Of course, being able to reason in a vague way may also lead to inconsistencies. That is why it will be important to back them by fully-verified provers such as HOL(y)Hammer and TacticToe.

We present here how three kind of objects (terms, tactics and concepts) could be synthesized. They respectively correspond to three inductive methods: conjecturing, algorithm invention and theory development.

Term Synthesis (Conjecturing) One of the most important inductive method in theorem proving among mathematicians is conjecturing. Mathematicians propose conjectures in two related situations. On the one hand a conjecture can be an open ended question that would deepen the knowledge of the field if it were solved. On the other hand, mathematicians conjecture crucial lemmas that would pave the way for a proof of a difficult theorem. Either way, it is hard for intelligent systems to reproduce the conjecturing process by which humans come up with new potential facts. My proposal is to test our substitution based conjecturing procedure further as it is the most promising method for

now. Eventually, more general generative models will have to be thought through. And probably most important of all is to be able to organize all the generated conjectures in the proving space without doing expensive reasoning. One idea toward a solution is to rely on free associative reasoning [BCC87].

Tactic Synthesis (Algorithm Invention) Sometimes, a mathematician (or a programmer if we consider a more general case) may need to invent a proof method (called tactic in HOL4) such as the Euclid algorithm for solving the greatest common divisor problem. I think this step may be easier than conjecturing as programs are compositional which is not the case for formulas. The hope is that our system will be able to learn how to program larger and larger proof methods incrementally. A simple example is when a sequence of tactics is very frequently applied during proof search, then it could be beneficial for our system to create a new tactic by composing this sequence of tactics. This would reduce the branching factor and lead to deeper proofs.

Concept Synthesis (Theory Construction) Often, searching for a proof of a theorem directly is not the right approach even with experience in the domain. It is preferable to build a theory and construct an understanding of the relation between mathematical concepts before a proof attempt. This understanding can be used to set milestones via the conjecturing method. After some theory exploration [BCJ+06, CJRS13], it can be useful to refactor the knowledge discovered by recognizing important concepts that have not been explicitly defined yet. Tidying up the theory would simplify the learning process by giving a better understanding and intuition about the domain.

9.2.3 Evaluation of the Progress

For many experimental projects in computer science, a feedback loop is essential. The reason is that in general programmers have little intuition for the right approach to solve a difficult and wide-ranging set of problems. By analyzing the results of a prototype, a developer can discard unfeasible ideas and improve promising ones. This analysis consists of understanding the settings of the experiments and their implication on the results. Since the relation between the different parameter settings is non-trivial, we try to simplify the analysis of our results by providing a single objective function that can evaluate how successful the tested algorithm is. The number of re-proven theorems in a library is an objective function that has been used throughout this thesis. It is relatively easy to measure and interpret. However, in practice programmers benefit from a more detailed feedback that the one given by the objective function. By analyzing the reason of the successes and failures in each proof attempt, a developer can create a fix for a particular problems that generalizes well over the set of all problems. Although based on weaker generalization principles, this self-improving methodology can be used by the program on its own with no human assistance as it has been demonstrated by reinforcement learning programs on perfect information games.

The principal drawback of the current evaluation process is that it is heavily dependent on the set of theorems tested and the knowledge available. The issue is that the difficulty

of the problem sets should be just right to allow for a gradual improvement of our system. This may not be the case for the set of top-level theorems in the ITP libraries used during evaluation in this thesis. To create this smooth learning curve for our program, I propose a guided approach where the details of the proofs are removed little by little as the program gets better. In the long run, I can even imagine that a "teacher" program will replace the hand-crafted guide by learning how to suggest problems adapted to the strength of the prover and increasing its mathematical understanding.

Bibliography

[AC15] Ali Assaf and Raphaël Cauderlier. Mixing HOL and Coq in Dedukti (extended abstract). In Cezary Kaliszyk and Andrei Paskevich, editors, *Workshop on Proof eXchange for Theorem Proving (PxTP)*, volume 186 of *EPTCS*, pages 89–96. EasyChair, 2015. URL `https://doi.org/10.4204/EPTCS.186.9`.

[Ada15a] Mark Adams. The common HOL platform. In Cezary Kaliszyk and Andrei Paskevich, editors, *Workshop on Proof eXchange for Theorem Proving (PxTP)*, volume 186 of *EPTCS*, pages 42–56. EasyChair, 2015. URL `https://doi.org/10.4204/EPTCS.186.6`.

[Ada15b] Mark Adams. Refactoring proofs with Tactician. In Domenico Bianculli, Radu Calinescu, and Bernhard Rumpe, editors, *Human-Oriented Formal Methods (HOFM)*, volume 9509 of *LNCS*, pages 53–67. Springer, 2015. URL `http://dx.doi.org/10.1007/978-3-662-49224-6_6`.

[AFG+11] Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Théry, and Benjamin Werner. A modular integration of SAT/SMT solvers to Coq through proof witnesses. In Jean-Pierre Jouannaud and Zhong Shao, editors, *Conference on Certified Programs and Proofs (CPP)*, volume 7086 of *LNCS*, pages 135–150. Springer, 2011. URL `http://dx.doi.org/10.1007/978-3-642-25379-9_12`.

[AH15] Serge Autexier and Dieter Hutter. Structure formation in large theories. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 9150 of *LNCS*, pages 155–170. Springer, 2015. URL `https://doi.org/10.1007/978-3-319-20615-8_10`.

[AK16] David Aspinall and Cezary Kaliszyk. What's in a theorem name? (rough diamond). In Jasmin C. Blanchette and Stephan Merz, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 9807 of *LNCS*, pages 459–465. Springer, 2016. URL `http://doi.org/10.1007/978-3-319-43144-4_28`.

[ARC14] Andrea Asperti, Wilmer Ricciotti, and Claudio Sacerdoti Coen. Matita tutorial. *Journal of Formalized Reasoning*, 7(2):91–199, 2014. URL `http://doi.org/10.6092/issn.1972-5787/4651`.

[Att10] Stéphane Attal. Markov chains and dynamical systems: The open system point of view. *Communications on Stochastic Analysis*, 4:523–540, 2010.

[Awo06] Steve Awodey. Category theory, volume 49 of Oxford Logic Guides, 2006.

[BÏ12] Sascha Böhme. *Proving Theorems of Higher-Order Logic with SMT Solvers*. PhD thesis, Technische Universität München, 2012. URL `http://mediatum.ub.tum.de/doc/1084525/1084525.pdf`.

[Bar11] Maria Barbarossa. Stability of discrete dynamical systems. *Matrix*, 21:22, 2011.

[BBF+16] Jasmin C. Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. Semi-intelligible Isar proofs from machine-generated proofs. *Journal of Automated Reasoning*, 56(2):155–200, 2016. URL `https://doi.org/10.1007/s10817-015-9335-3`.

[BBG+15] Grzegorz Bancerek, Czeslaw Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol Pąk, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 9150 of *LNCS*, pages 261–279. Springer, 2015. URL `http://doi.org/10.1007/978-3-319-20615-8_17`.

[BBP13] Jasmin C. Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *Journal of Automated Reasoning*, 51(1):109–128, 2013. URL `http://dx.doi.org/10.1007/s10817-013-9278-5`.

[BBPS13] Jasmin C. Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. In Nir Piterman and Scott A. Smolka, editors, *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013. URL `https://doi.org/10.1007/978-3-642-36742-7_34`.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer, 2004.

[BCC87] Daniela Battaglia, Corrado Cavallero, and Piercarla Cicogna. Temporal reference of the mnemonic sources of dreams. *Perceptual and Motor Skills*, 64(3):979–983E, 1987. URL `http://journals.sagepub.com/doi/pdf/10.2466/pms.1987.64.3.683`.

[BCD+11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Conference on Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011. URL `https://doi.org/10.1007/978-3-642-22110-1_14`.

[BCJ+06] Bruno Buchberger, Adrian Craciun, 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. URL https://doi.org/10.1016/j.jal.2005.10.006.

[Ber08] Yves Bertot. A short presentation of Coq. In Otmane Ait Mohamed, César Muñoz, and Sofiène Tahar, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *LNCS*, pages 12–16. Springer, 2008. URL http://doi.org/10.1007/978-3-540-71067-7_3.

[BGK+16] Jasmin C. Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *Journal of Automated Reasoning*, 57(3):219–244, 2016. URL http://dx.doi.org/10.1007/s10817-016-9362-8.

[BHMN15] Jasmin C. Blanchette, Maximilian Haslbeck, Daniel Matichuk, and Tobias Nipkow. Mining the archive of formal proofs. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 9150 of *LNCS*, pages 3–17. Springer, 2015. URL https://doi.org/10.1007/978-3-319-20615-8_1.

[BJL06] Maksym Bortin, Einar Broch Johnsen, and Christoph Lüth. Structured formal development in Isabelle. *Nordic Journal of Computing*, 13(1-2):2–21, 2006.

[BKPU16] Jasmin C. Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1):101–148, 2016. URL https://doi.org/10.6092/issn.1972-5787/4593.

[Bla03] Frédéric Blanqui. Rewriting modulo in deduction modulo. In Robert Nieuwenhuis, editor, *Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *LNCS*, pages 395–409. Springer, 2003. URL https://doi.org/10.1007/3-540-44881-0_28.

[Bla12] Jasmin C. Blanchette. *Automatic Proofs and Refutations for Higher-order Logic*. PhD thesis, Technische Universität München, 2012. URL http://mediatum.ub.tum.de/doc/1097834/1097834.pdf.

[BP11] François Bobot and Andrey Paskevich. Expressing polymorphic types in a many-sorted language. In Cesare Tinelli and Viorica Sofronie-Stokkermans, editors, *Symposium on Frontiers of Combining Systems (FroCoS)*, volume 6989 of *LNCS*, pages 87–102. Springer, 2011. URL http://dx.doi.org/10.1007/978-3-642-24364-6_7.

[BP13] Jasmin C. Blanchette and Andrei Paskevich. TFF1: The TPTP typed first-order form with rank-1 polymorphism. In Maria Paola Bonacina, editor, *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 414–420. Springer, 2013. URL http://dx.doi.org/10.1007/978-3-642-38574-2_29.

[BPW+12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. URL https://doi.org/10.1109/TCIAIG.2012.2186810.

[BPWW12] Jasmin C. Blanchette, Andrei Popescu, Daniel Wand, and Christoph Weidenbach. More SPASS with Isabelle - Superposition with hard sorts and configurable simplification. In Lennart Beringer and Amy P. Felty, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 7406 of *LNCS*, pages 345–360. Springer, 2012. URL http://dx.doi.org/10.1007/978-3-642-32347-8_24.

[Bro13] Chad E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. *Journal of Automated Reasoning*, 51(1):57–77, 2013. URL https://doi.org/10.1007/s10817-013-9283-8.

[BW10] Sascha Böhme and Tjark Weber. Fast LCF-style proof reconstruction for Z3. In Matt Kaufmann and Lawrence C. Paulson, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 6172 of *LNCS*, pages 179–194. Springer, 2010. URL http://dx.doi.org/10.1007/978-3-642-14052-5_14.

[BW13] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In Maria Paola Bonacina, editor, *Conference on Automated Deduction (CADE)*, volume 7898 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2013. URL http://dx.doi.org/10.1007/978-3-642-38574-2_3.

[CDD+01] David Carlisle, James Davenport, Mike Dewar, Namhyun Hur, and William Naylor. Conversion between MathML and OpenMath. Technical Report 24.969, The OpenMath Society, 2001.

[CJRS13] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. Automating inductive proofs using theory exploration. In Maria Paola Bonacina, editor, *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 392–406. Springer, 2013. URL https://doi.org/10.1007/978-3-642-38574-2_27.

[CK18] Łukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. *Journal of Automated Reasoning*, 61(1):423–453, 2018. URL https://doi.org/10.1007/s10817-018-9458-4.

[Cor12] Leo Corry. *Modern algebra and the rise of mathematical structures*. Birkhäuser, 2012.

[DHK03] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003. URL http://dx.doi.org/10.1023/A:1027357912519.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008. URL https://doi.org/10.1007/978-3-540-78800-3_24.

[dMKA+15] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Conference on Automated Deduction (CADE)*, volume 9195 of *LNCS*, pages 378–388. Springer, 2015. URL https://doi.org/10.1007/978-3-319-21401-6_26.

[Dud76] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(4):325–327, 1976. URL http://dx.doi.org/10.1109/TSMC.1976.5408784.

[DWA13] Dominik Dietrich, Iain Whiteside, and David Aspinall. Polar: A framework for proof refactoring. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 776–791. Springer, 2013. URL http://doi.org/10.1007/978-3-642-45221-5_52.

[Faj88] Siemion Fajtlowicz. On conjectures of Graffiti. *Discrete Mathematics*, 72(1-3):113–118, 1988. URL https://doi.org/10.1016/0012-365X(88)90199-9.

[FB16] Michael Färber and Chad E. Brown. Internal guidance for Satallax. In Nicola Olivetti and Ashish Tiwari, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 9706 of *LNCS*, pages 349–361. Springer, 2016. URL http://dx.doi.org/10.1007/978-3-319-40229-1_24.

[FGT92] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. Little theories. In Deepak Kapur, editor, *Conference on Automated Deduction (CADE)*, volume 607 of *LNCS*, pages 567–581. Springer, 1992. URL https://doi.org/10.1007/3-540-55602-8_192.

[FP13] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Symposium on Programming Languages and Systems (APLAS)*, volume 7792 of *LNCS*, pages 125–128. Springer, 2013. URL https://doi.org/10.1007/978-3-642-37036-6_8.

[GK14] Thibault Gauthier and Cezary Kaliszyk. Matching concepts across HOL libraries. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 8543 of *LNCS*, pages 267–281. Springer, 2014. URL http://doi.org/10.1007/978-3-319-08434-3_20.

[GK15a] Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In Xavier Leroy and Alwen Tiu, editors, *Conference on Certified Programs and Proofs (CPP)*, pages 49–57. ACM, 2015. URL http://doi.org/10.1145/2676724.2693173.

[GK15b] Thibault Gauthier and Cezary Kaliszyk. Sharing HOL4 and HOL Light proof knowledge. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 9450 of *LNCS*, pages 372–386. Springer, 2015. URL http://doi.org/10.1007/978-3-662-48899-7_26.

[GK19] Thibault Gauthier and Cezary Kaliszyk. Aligning concepts across proof assistant libraries. *Journal of Symbolic Computation*, 90:89–123, 2019. URL https://doi.org/10.1016/j.jsc.2018.04.005.

[GKKN15] Thibault Gauthier, Cezary Kaliszyk, Chantal Keller, and Michael Norrish. Beagle as a HOL4 external ATP method. In Stephan Schulz, Leonardo De Moura, and Boris Konev, editors, *Workshop on Practical Aspects of Automated Reasoning (PAAR)*, volume 31 of *EPiC*, pages 50–59. EasyChair, 2015. URL http://doi.org/10.29007/8xbv.

[GKN10] Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *Journal of Formalized Reasoning*, 3(2):153–245, 2010. URL https://doi.org/10.6092/issn.1972-5787/1980.

[GKU16] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. Initial experiments with statistical conjecturing over large formal corpora. In Andrea Kohlhase, Paul Libbrecht, Bruce R. Miller, Adam Naumowicz, Walther Neuper, Pedro Quaresma, Frank Wm. Tompa, and Martin Suda, editors, *Work in Progress at the Conference on Intelligent Computer Mathematics (CICM-WiP)*, volume 1785, pages 219–228. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1785/W23.pdf.

[GKU17] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David Sands, editors,

Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), volume 46 of *EPiC*, pages 125–143. EasyChair, 2017. URL http://www.easychair.org/publications/paper/340355.

[GKU+18] Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Learning to prove with tactics. *CoRR*, 2018. URL http://arxiv.org/abs/1804.00596.

[Gon08] Georges Gonthier. Formal proof the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008. URL http://www.ams.org/notices/200811/tx081101382p.pdf.

[Gra96] Peter Graf. *Term Indexing*, volume 1053 of *LNCS*. Springer, 1996. URL http://dx.doi.org/10.1007/3-540-61040-5.

[GWR15] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: search for proofs using inferred automata. In Amy P. Felty and Aart Middeldorp, editors, *Conference on Automated Deduction (CADE)*, volume 9195 of *LNCS*, pages 246–255. Springer, 2015. URL http://dx.doi.org/10.1007/978-3-319-21401-6_16.

[Hal12] Thomas Hales. *Dense Sphere Packings: A Blueprint for Formal Proofs*, volume 400 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2012.

[Har96] John Harrison. Optimizing proof search in model elimination. In M. McRobbie and J.K. Slaney, editors, *Conference on Automated Deduction (CADE)*, number 1104 in LNAI, pages 313–327. Springer, 1996. URL https://doi.org/10.1007/3-540-61511-3_97.

[Har09] John Harrison. HOL Light: An overview. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5674 of *LNCS*, pages 60–66. Springer, 2009. URL http://dx.doi.org/10.1007/978-3-642-03359-9_4.

[Har13] John Harrison. The HOL Light theory of Euclidean space. *Journal of Automated Reasoning*, 50(2):173–190, 2013. URL http://dx.doi.org/10.1007/s10817-012-9250-9.

[Hec88] Robert Hecht-Nielsen. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988. URL http://doi.org/10.1016/0893-6080(88)90469-8.

[HH14] Gérard Huet and Hugo Herbelin. 30 years of research and development around Coq. In Suresh Jagannathan and Peter Sewell, editors, *Symposium on Principles of Programming Languages (POPL)*, pages 249–250. ACM, 2014. URL http://doi.acm.org/10.1145/2535838.2537848.

[HHM+10] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller. A revision of the proof of the Kepler conjecture. *Discrete & Computational Geometry*, 44(1):1–34, 2010. URL `http://dx.doi.org/10.1007/s00454-009-9148-4`.

[Hir88] Morris W. Hirsch. Stability and convergence in strongly monotone dynamical systems. *Journal für die reine und angewandte Mathematik*, 383:1–53, 1988. URL `http://eudml.org/doc/152991`.

[HK14] Jónathan Heras and Ekaterina Komendantskaya. Recycling proof patterns in Coq: Case studies. *Mathematics in Computer Science*, 8(1):99–116, 2014. URL `http://dx.doi.org/10.1007/s11786-014-0173-1`.

[HKKN13] Florian Haftmann, Alexander Krauss, Ondřej Kunčar, and Tobias Nipkow. Data refinement in Isabelle/HOL. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 100–115. Springer, 2013. URL `https://doi.org/10.1007/978-3-642-39634-2_10`.

[HNR68] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems Science and Cybernetics*, 4(2):100–107, 1968. URL `http://dx.doi.org/10.1109/TSSC.1968.300136`.

[Hur03] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68, 2003.

[Hur05] Joe Hurd. System description: The Metis proof tactic. In Carsten Schuermann Christoph Benzmueller, John Harrison, editor, *Workshop on Empirically Successful Automated Reasoning in Higher-Order Logic (ESHOL)*, pages 103–104, 2005. URL `https://arxiv.org/pdf/cs/0601042`.

[Hur11] Joe Hurd. The OpenTheory standard theory library. In Mihaela Gheorghiu Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, volume 6617 of *LNCS*, pages 177–191. Springer, 2011. URL `http://dx.doi.org/10.1007/978-3-642-20398-5_14`.

[HUW14] John Harrison, Josef Urban, and Freek Wiedijk. History of interactive theorem proving. In Jörg H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014. URL `https://doi.org/10.1016/B978-0-444-51624-4.50004-6`.

[HV11] Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Conference on Automated Deduction (CADE)*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011. URL `https://doi.org/10.1007/978-3-642-22438-6_23`.

[JBDD15] Mélanie Jacquel, Karim Berkani, David Delahaye, and Catherine Dubois. Verifying B proof rules using deep embedding and automated theorem proving. *Software and System Modeling*, 14(1):101–119, 2015. URL http://dx.doi.org/10.1007/s10270-013-0322-z.

[Jon04] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5):493–502, 2004. URL https://doi.org/10.1108/00220410410560573.

[KAE+10] Gerwin Klein, June Andronick, Kevin Elphinstone, Gernot Heiser, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an operating-system kernel. *Communications of the ACM*, 53(6):107–115, 2010. URL http://doi.acm.org/10.1145/1743546.1743574.

[KBKU13] Daniel Kühlwein, Jasmin C. Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine learning for Sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013. URL http://dx.doi.org/10.1007/978-3-642-39634-2_6.

[KH12] Ramana Kumar and Joe Hurd. Standalone tactics using OpenTheory. In Lennart Beringer and Amy P. Felty, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 7406 of *LNCS*, pages 405–411. Springer, 2012. URL http://doi.org/10.1007/978-3-642-32347-8_28.

[KHG12] Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in Proof General: Interfacing interfaces. In Cezary Kaliszyk and Christoph Lüth, editors, *Workshop On User Interfaces for Theorem Provers (UITP)*, volume 118 of *EPTCS*, pages 15–41. EasyChair, 2012. URL http://dx.doi.org/10.4204/EPTCS.118.2.

[KK13] Cezary Kaliszyk and Alexander Krauss. Scalable LCF-style proof translation. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 51–66. Springer, 2013. URL http://dx.doi.org/10.1007/978-3-642-39634-2_7.

[Kle14] Gerwin Klein. Proof engineering considered essential. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *Symposium on Formal Methods (FM)*, volume 8442 of *LNCS*, pages 16–21. Springer, 2014. URL https://doi.org/10.1007/978-3-319-06410-9_2.

[KMNO14] Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens. CakeML: A verified implementation of ML. In Suresh Jagannathan and Peter Sewell, editors, *Symposium on Principles of Programming Languages*

(POPL), volume 49, pages 179–191. ACM, 2014. URL http://doi.acm.org/10.1145/2535838.2535841.

[KMU14] 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)*, volume 30 of *EasyChair Proceedings in Computing*, pages 27–34. EasyChair, 2014. URL http://www.easychair.org/publications/paper/196774.

[KR14] Cezary Kaliszyk and Florian Rabe. Towards knowledge management for HOL Light. In Stephen Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 8543 of *LNCS*, pages 357–372. Springer, 2014. URL http://doi.org/10.1007/978-3-319-08434-3_26.

[KS10] Alexander Krauss and Andreas Schropp. A mechanized translation from higher-order logic to set theory. In Matt Kaufmann and Lawrence C. Paulson, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 6172 of *LNCS*, pages 323–338. Springer, 2010. URL https://doi.org/10.1007/978-3-642-14052-5_23.

[KSR16] Cezary Kaliszyk, Geoff Sutcliffe, and Florian Rabe. TH1: the TPTP typed higher-order form with rank-1 polymorphism. In Pascal Fontaine, Stephan Schulz, and Josef Urban, editors, *Workshop on Practical Aspects of Automated Reasoning (PAAR)*, volume 1635, pages 41–55. CEUR-WS.org, 2016. URL http://ceur-ws.org/Vol-1635/paper-05.pdf.

[KU13a] Cezary Kaliszyk and Josef Urban. PRocH: Proof reconstruction for HOL Light. In Maria Paola Bonacina, editor, *Conference on Automated Deduction (CADE)*, volume 7898 of *LNCS*, pages 267–274. Springer, 2013. URL https://doi.org/10.1007/978-3-642-38574-2_18.

[KU13b] Cezary Kaliszyk and Josef Urban. Stronger automation for Flyspeck by feature weighting and strategy evolution. In Jasmin C. Blanchette and Josef Urban, editors, *Workshop on Proof eXchange for Theorem Proving (PxTP)*, volume 14 of *EPiC*, pages 87–95. EasyChair, 2013. URL http://www.easychair.org/publications/paper/141245.

[KU14] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014. URL https://doi.org/10.1007/s10817-014-9303-3.

[KU15a] Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly Efficient Machine Learning Connection Prover. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 9450

of *LNCS*, pages 88–96. Springer, 2015. URL http://dx.doi.org/10.1007/978-3-662-48899-7_7.

[KU15b] Cezary Kaliszyk and Josef Urban. HOL(y)Hammer: Online ATP service for HOL Light. *Mathematics in Computer Science*, 9(1):5–22, 2015. URL https://doi.org/10.1007/s11786-014-0182-0.

[KU15c] Cezary Kaliszyk and Josef Urban. Learning-assisted theorem proving with millions of lemmas. *Journal of Symbolic Computation*, 69:109–128, 2015. URL http://doi.org/10.1016/j.jsc.2014.09.032.

[KU15d] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *Journal of Automated Reasoning*, 55(3):245–256, 2015. URL http://dx.doi.org/10.1007/s10817-015-9330-8.

[KUV15a] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Efficient semantic features for automated reasoning over large theories. In Qiang Yang and Michael Wooldridge, editors, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3084–3090. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/435.

[KUV15b] Cezary Kaliszyk, Josef Urban, and Jiří Vyskočil. Learning to parse on aligned corpora. In Christian Urban and Xingyuan Zhang, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 9236 of *LNCS*, pages 227–233. Springer, 2015. URL https://doi.org/10.1007/978-3-319-22102-1_15.

[KV13] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Conference on Computer Aided Verification (CAV)*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013. URL https://doi.org/10.1007/978-3-642-39799-8_1.

[KW10] Chantal Keller and Benjamin Werner. Importing HOL Light into Coq. In Matt Kaufmann and Lawrence C. Paulson, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 6172 of *LNCS*, pages 307–322. Springer, 2010. URL http://dx.doi.org/10.1007/978-3-642-14052-5_22.

[Len76] Douglas Lenat. *An Artificial Intelligence Approach to Discovery in Mathematics*. PhD thesis, Stanford University, 1976.

[Les11] Stéphane Lescuyer. *Formalizing and Implementing a Reflexive Tactic for Automated Deduction in Coq*. PhD thesis, Université Paris-Sud, 2011. URL https://tel.archives-ouvertes.fr/tel-00713668.

[LFL98] Thomas K. Landauer, Peter W. Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998. URL http://doi.org/10.1080/01638539809545028.

[MD14] Magnus O. Myreen and Jared Davis. The reflective Milawa theorem prover is sound - (down to the machine code that runs it). In Gerwin Klein and Ruben Gamboa, editors, *Conference on Interactive Theorem Proving (ITP)*, volume 8558 of *LNCS*, pages 421–436. Springer, 2014. URL `http://dx.doi.org/10.1007/978-3-319-08970-6`.

[Mey88] Bertrand Meyer. *Object-oriented software construction*, volume 2. Prentice hall New York, 1988.

[MGK+17] Dennis Müller, Thibault Gauthier, Cezary Kaliszyk, Michael Kohlhase, and Florian Rabe. Classification of alignments between concepts of formal mathematical systems. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 10383 of *LNCS*, pages 83–98. Springer, 2017. URL `http://doi.org/10.1007/978-3-319-62075-6_7`.

[MML] The Mizar Mathematical Library. URL `http://mizar.org/`.

[MP08] Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008. URL `http://dx.doi.org/10.1007/s10817-007-9085-y`.

[Nor03] Michael Norrish. Complete integer decision procedures as derived rules in HOL. In David A. Basin and Burkhart Wolff, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 2758 of *LNCS*, pages 71–86. Springer, 2003. URL `https://doi.org/10.1007/10930755_5`.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[OS06] Steven Obua and Sebastian Skalberg. Importing HOL into Isabelle/HOL. In Ulrich Furbach and Natarajan Shankar, editors, *International Joint Conference on Automated Reasoning (IJCAR)*, volume 4130 of *LNCS*, pages 298–302. Springer, 2006. URL `http://dx.doi.org/10.1007/11814771_27`.

[Ott05] Jens Otten. Clausal connection-based theorem proving in intuitionistic first-order logic. In Bernhard Beckert, editor, *Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2005)*, volume 3702 of *LNCS*, pages 245–261. Springer, 2005. URL `https://doi.org/10.1007/11554554_19`.

[Pau99] Lawrence C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–87, 1999. URL `http://www.jucs.org/jucs_5_3/a_generic_tableau_prover`.

[Pau16] Lawrence C. Paulson. Isabelle's logics: FOL and ZF, 2016.

[PB10] Lawrence C. Paulson and Jasmin C. Blanchette. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *Workshop on the Implementation of Logics (IWIL)*, volume 2 of *EPiC*, pages 1–11. EasyChair, 2010. URL `http://www.easychair.org/publications/paper/62805`. Invited talk.

[PS07] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007. URL `http://doi.org/10.1007/978-3-540-74591-4_18`.

[PW08] Virgile Prevosto and Uwe Waldmann. SPASS+T. In Geoff Sutcliffe, Renate Schmidt, and Stephan Schulz, editors, *Workshop on Empirically Successful Computerized Reasoning (ESCoR)*, volume 192, pages 18–33. CEUR-WS.org, 2008. URL `http://ceur-ws.org/Vol-192/paper02.pdf`.

[Rab13] Florian Rabe. The MMT API: A generic MKM system. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 7961 of *LNCS*, pages 339–343. Springer, 2013. URL `https://doi.org/10.1007/978-3-642-39320-4_25`.

[Rot10] Joseph J Rotman. *Advanced modern algebra*, volume 114. American Mathematical Society, 2010.

[RSV01] I. V. Ramakrishnan, R. C. Sekar, and Andrei Voronkov. Term indexing. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1853–1964. Elsevier and MIT Press, 2001.

[SB98] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1st edition, 1998. ISBN 0262193981.

[SB10] Geoff Sutcliffe and Christoph Benzmüller. Automated reasoning in higher-order logic using the TPTP THF infrastructure. *Journal of Formalized Reasoning*, 3(1):1–27, 2010. URL `http://dx.doi.org/10.6092/issn.1972-5787/1710`.

[Sch02] Stephan Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2-3):111–126, 2002. URL `http://iospress.metapress.com/content/n908n94nmvk59v3c/`.

[Sch13a] Stephan Schulz. Simple and efficient clause subsumption with feature vector indexing. In Maria Paola Bonacina and Mark E. Stickel, editors,

Automated Reasoning and Mathematics - Essays in Memory of William W. McCune, volume 7788 of *LNCS*, pages 45–67. Springer, 2013. URL https://doi.org/10.1007/978-3-642-36675-8_3.

[Sch13b] Stephan Schulz. System description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013. URL https://doi.org/10.1007/978-3-642-45221-5_49.

[SN08] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008. URL http://dx.doi.org/10.1007/978-3-540-71067-7_6.

[SSCB12] Geoff Sutcliffe, Stephan Schulz, Koen Claessen, and Peter Baumgartner. The TPTP typed first-order form with arithmetic. In Nikolaj Bjørner and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 7180 of *Lecture Notes in Computer Science*, pages 406–419. Springer, 2012. URL http://dx.doi.org/10.1007/978-3-642-28717-6_32.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–, 2017. URL http://dx.doi.org/10.1038/nature24270.

[Sut09] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009. URL http://dx.doi.org/10.1007/s10817-009-9143-8.

[Sut14] Geoff Sutcliffe. The CADE-24 automated theorem proving system competition - CASC-24. *AI Communications*, 27(4):405–416, 2014. URL https://doi.org/10.3233/AIC-140606.

[SW06] Clare M. So and Stephen M. Watt. On the conversion between content MathML and OpenMath. In J.M. Borwein, E.M. Rocha, and J.F.Rodrigues, editors, *Conference on the Communicating Mathematics in the Digital Era (CMDE)*, pages 169–182, 2006.

[Try07] Andrzej Trybulec. Checker. Compiled by Freek Wiedijk, 2007. URL http://www.cs.ru.nl/F.Wiedijk/mizar/by.pdf.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL `https://homotopytypetheory.org/book`.

[Urb04] Josef Urban. MPTP - Motivation, Implementation, First Experiments. *Journal of Automated Reasoning*, 33(3-4):319–339, 2004. URL `http://dx.doi.org/10.1007/s10817-004-6245-1`.

[Urb06a] Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *International Journal on Artificial Intelligence Tools*, 15(1):109–130, 2006. URL `http://ktiml.mff.cuni.cz/~urban/MoMM/momm.ps`.

[Urb06b] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning*, 37(1-2):21–43, 2006. URL `http://doi.org/10.1007/s10817-006-9032-3`.

[Urb07] Josef Urban. Malarea: a metasystem for automated reasoning in large theories. In Geoff Sutcliffe, Josef Urban, and Stephan Schulz, editors, *Workshop on Empirically Successful Automated Reasoning in Large Theories (ESLART)*, volume 257. CEUR-WS.org, 2007. URL `http://ceur-ws.org/Vol-257/05_Urban.pdf`.

[Urb08] Josef Urban. Automated reasoning for Mizar: Artificial intelligence through knowledge exchange. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *LPAR Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and International Workshop on the Implementation of Logics (IWIL)*, volume 418. CEUR-WS.org, 2008. URL `http://ceur-ws.org/Vol-418/paper1.pdf`.

[Urb15] Josef Urban. BliStr: The Blind Strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence (GCAI)*, volume 36 of *EPiC*, pages 312–319. EasyChair, 2015. URL `http://www.easychair.org/publications/paper/BliStr_The_Blind_Strategymaker`.

[USPV08] 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)*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008. URL `http://dx.doi.org/10.1007/978-3-540-71070-7_37`.

[UVŠ11] Josef Urban, Jiří Vyskočil, and Petr Štěpánek. MaLeCoP Machine Learning Connection Prover. In Kai Brünnler and George Metcalfe, editors, *Conference on Automated Reasoning with Analytic Tableaux and Related Methods*

(TABLEAUX), volume 6793 of *LNCS*, pages 263–277. Springer, 2011. URL http://dx.doi.org/10.1007/978-3-642-22119-4_21.

[VSU10] Jiří Vyskočil, David Stanovský, and Josef Urban. Automated proof compression by invention of new definitions. In Edmund M. Clarke and Andrei Voronkov, editors, *Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 6355 of *LNCS*, pages 447–462. Springer, 2010. URL https://doi.org/10.1007/978-3-642-17511-4_25.

[WADG11] Iain Whiteside, David Aspinall, Lucas Dixon, and Gudmund Grov. Towards formal proof script refactoring. In James H. Davenport, William M. Farmer, Josef Urban, and Florian Rabe, editors, *Conference on Intelligent Computer Mathematics (CICM)*, volume 6824 of *LNCS*, pages 260–275. Springer, 2011. URL http://doi.org/10.1007/978-3-642-22673-1_18.

[Wal98] Daria Walukiewicz. A total AC-compatible reduction ordering on higher-order terms. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1443 of *LNCS*, pages 530–542. Springer, 1998. URL https://doi.org/10.1007/BFb0055081.

[Web11] Tjark Weber. SMT solvers: new oracles for the HOL theorem prover. *International Journal on Software Tools for Technology Transfer*, 13(5):419–429, 2011. URL http://dx.doi.org/10.1007/s10009-011-0188-8.

[Wie06] Freek Wiedijk, editor. *The Seventeen Provers of the World, Foreword by Dana S. Scott*, volume 3600 of *LNCS*. Springer, 2006. URL https://doi.org/10.1007/11542384.

[Won95] Wai Wong. Recording and checking HOL proofs. In E. Thomas Schubert, Phillip J. Windley, and Jim Alves-Foss, editors, *Workshop on Higher Order Logic Theorem Proving and Its Applications*, volume 971 of *LNCS*, pages 353–368. Springer, 1995. URL https://doi.org/10.1007/3-540-60275-5_76.

[WPN08] Makarius Wenzel, Lawrence C. Paulson, and Tobias Nipkow. The Isabelle framework. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *LNCS*, pages 33–38. Springer, 2008. URL https://doi.org/10.1007/978-3-540-71067-7_7.

[WR27] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. Cambridge University Press, 1927.