Simon Colton   Volker Sorge (eds.)

Second Workshop on the Role of Automated Deduction in Mathematics

In conjunction with CADE-18
July 31st 2002, Copenhagen, Denmark

Preface

An original goal of automated theorem proving was its application to mathematics, whether by proving established results, enhancing calculation techniques or facilitating discovery of new results. There is still much scope for the use of automated deduction to add to mathematics and we hope to explore these possibilities in this workshop.

Contributions which detail the employment of automated deduction techniques in any area of mathematical research were encouraged. This includes looking at the interaction between automated deduction programs and other computational systems which have been developed over recent years to automate different areas of mathematical activity. Such systems include computer algebra packages, tutoring programs, systems developed to help explore a mathematical theory, and those developed to help present and archive mathematical theories.

The proceedings of the workshop therefore contain papers on a variety of different aspects of the application of automated reasoning in a mathematical context. Cairns and Gow argue that the acceptance of deduction systems amongst mathematicians could be enhanced if these systems were more designed to further the actual goals of their human users. Chen, Kobayashi, Murao and Suzuki present a formalisation of an algebraic proof method, induction on the number of sets, in Isabelle. Colton, McCasland, Bundy and Walsh discuss a role of automated theory formation (which includes automated deduction) in the production of mathematics exercises. Kerber and Pollet are concerned with the representation of mathematical data, where the objects are enriched with some of their mathematical properties, and which can therefore lead to more human-oriented proofs. Similarly, Schwarzweller is concerned with the representation of mathematical objects in knowledge bases. He presents a properties-based representation for mathematical theorems and domains that facilitates checking whether a general theorem holds in a particular domain.

Simon Colton and Volker Sorge
17th June 2002

Table of Contents

Automated Deduction Systems for Real Mathematicians

Paul Cairns and Jeremy Gow

UCL Interaction Centre, University College London,
26, Bedford Way, London WC1H 0AP, UK
{p.cairns, j.gow}@ucl.ac.uk
WWW home page: www.uclic.ucl.ac.uk

Abstract. Automated deduction systems currently have a low uptake (or even recognition) amongst mathematicians. This paper takes a human computer interaction perspective on the role of automated deduction in mathematics. We first dismiss the fallacy that making systems easy to use will make them used by heuristically comparing Microsoft Word and LaTeX systems in mathematical authoring. Through considering the goals of mathematicians, we propose ways to develop automated deduction systems that mathematicians actually want. There are clearly considerable technological and even philosophical barriers but it is hoped that these can be surmounted in time.

1 Introduction

Computer algebra systems have made considerable impact on the working practices of mathematicians, particularly applied mathematicians. Using systems such as Mathematica, a mathematician can perform many symbolic manipulations on complicated models and at the right moment convert these to numerical results for a concrete problem. This has radically changed not only the working practices of mathematicians but also those of physicists and engineers who no longer need rely on expert mathematicians to help them with their problems.

Given the wide variety of mature automated deduction systems (ADS), it would be hoped that at least one system might have already had a comparable impact on the working practices of mathematicians. Yet, with one or two notable exceptions [16], mathematicians have barely even heard of automated deduction systems let alone used them. This is not that the ADS developers have not tried to make such systems available to mathematicians: the UITP conference [2] was expressly concerned with producing good user interfaces to theorem provers; the proof transformation and presentation workshop has similar goals [21]; and this workshop too is clearly hoping to make in-roads in this area. There is clearly then some greater gap between mathematicians and ADS than can be explained by difficult user interfaces.

In this paper, we take a human-computer interaction perspective on the role of automated deduction in mathematics. Rather than considering ADS as the solutions to mathematicians' problems, we consider what problems mathematicians might currently have and whether ADS can help out. First, we discuss why making systems easy to use does not make them fit for use by comparing the easy to use word processor Microsoft Word with the document preparation system LaTeX [14]. Next, we take a high-level look at the needs of mathematicians as individuals and as a community and see how ADS might meet those needs. Finally, the paper proposes some ways forward to bring a user-oriented perspective to the development of ADS for mathematicians.

2 Easy to use $\not\Rightarrow$ Useful

It is a common selling point of many software systems that they are easy to use. This is all well and good but ease of use in itself does not constitute a useful system. If the software is not useful in the sense of helping users to achieve their goals then no amount of easy to use interfaces is going to make users want to use the system. With mathematicians, this can be seen most clearly in the uptake of LaTeX above other easy to use systems such as Microsoft Word.

LaTeX is a document preparation system, that is, it allows the user to type in the content of the document and LaTeX will format it, typeset it and generally make a good attempt to make it look good on the printed page. The user must enter content made up of the words that the user wishes to convey together with instructions for displaying symbolic equations and further instructions for the usual typographical aids such as section headings, paragraphs, footnotes, lists, references and the like. The content is then processed separately to produce a printable version of the document.

This contrasts markedly with the WYSIWYG ("What You See Is What You Get") approach of most popular word processors. With these, the user types the words and they appear on the screen as they would on the printed document — content and presentation are inextricably linked. Toolbars and menu allow the user to change the formatting of the document and those changes appear immediately on the screen. This means the user never needs to stop entering content in order to process the document to see what it looks like and thus their work has a seamless flow from starting entering content to printing out the final document.

In terms of ease of use, WYSIWYG systems, as epitomised by Microsoft Word, are the most easy to use. The user starts the application and is immediately producing the document in its final form. LaTeX is far from easy to use: the user must not only enter the content but must instruct the system on how to present equations, what sort of document is being produced and what presentational features to include. None of these instructions are made available to the user via the system as LaTeX is separate from any text editor that might be used to prepare the documents. Novice users must type with the manual to hand and even expert users consult the manual to achieve difficult or unusual typesetting effects.

At face value then mathematicians, like many other professionals, should prefer using Word over LaTeX. But this is clearly not the case. Many journals are produced exclusively in LaTeX to the point that they even produce style sheets so that the authors can achieve the journal style on their submissions [24]. Publishing companies, such as Springer, produce their mathematical texts entirely in LaTeX. And the mathematical community produces most of its work using LaTeX to the point that in talks, when introducing a new symbol, the speaker will sometimes give the LaTeX code for the symbol so that the listeners know what it is.

The reason for this apparent irrationality in an otherwise logical community must be that whilst Word is easy to use it is not easy to do what mathematicians want to do. For instance, consider the following anecdotal evidence.

Mathematicians commonly want to put equations into their text so, for example, to include the following sort of a equation (taken from [6]) should be a routine task.

$$C = \bigcup_{f\in 2^{\omega}} \bigcap_{n\in \omega} \overline{U_{f|_{n}}}$$

In LaTeX, this is done with the somewhat complicated instructions:

```
$C = \bigcup_{f\in 2^{\omega}} \bigcap _{n\in \omega}
\overline{U_{f|_{n}}$
```

Only an expert user or a novice user sitting with the manual would know how to do this. In Word, though, it is enough to go to the menu `Insert|Object` and select the equation editor. The user is then taken from the Word document into the special editor and presented with a palette of buttons. The icons on the buttons clearly correspond to different types of symbol that could be used in equations and it is fairly easy to find the exact symbol that is required and so build up the equation. However, already the WYSIWYG paradigm has been broken, the user has been taken from the flow of producing a document to go to a special new process for equations. Whereas, in LaTeX, typing the equation was done in the same way that other typesetting is set up — by typing instructions. Even so, for the novice, the task was reasonably painless in Word and probably a lot less difficult than LaTeX.

However, the important point in this scenario is that this is what mathematicians *commonly* do. Very quickly, whichever system is being used, the mathematician user is going to get used to entering equations and producing certain symbols that they use frequently. Yet with Word, the only way to enter equations is through the editor and the only way to produce symbols is through the palette of buttons. There are no shortcuts to getting to, say, $\bigcup$ quickly. Expert users must go at the same pace as novices, violating a well-known usability principle that there must be ways for experts to improve their performance [18]. And for every equation, the user must interrupt their current mode of use to enter the special editor but, for mathematicians, entering equations should be the normal mode of use. In LaTeX, there is a single mode of use and the codes for symbols become quicker to recall with practice. As time goes by, expert users will automatically speed up and come to write equations as rapidly as they write ordinary sentences.

This is only one aspect of preparing mathematical documents and the analysis is only heuristic rather than empirically demonstrated. However, LaTeX is undoubtedly the most successful tool in this area. There may be many reasons why this is so, including historical serendipity, but the lesson stands that ease of use in itself will not persuade mathematicians to use a tool. With regards to the role of ADS in mathematics, simply making ADS easier to use will not necessarily change their current value to mathematicians nor will it suddenly make mathematicians realise what they are missing out on. Either ADS must offer something new that mathematicians really want or they must support what mathematicians currently do with tangible benefits over the existing ways of doing things. In both cases, this requires a deep understanding of the activities and needs of mathematicians.

3 What do mathematicians want?

It is impossible to define what mathematicians want without extensive research into a broad cross-section of the mathematical community. And even then, it may be impossible to come up with general indications that do not exclude the majority of mathematicians. However, there are some broad, high-level goals that it is easy to assert that mathematicians really do want to achieve. Additionally, it is worth making a distinction between what individuals want and what the community as a whole wants.

As individuals, some simple goals of mathematicians are:

- Do (good) maths
- Publish results/achieve recognition
- Don't feel stupid

Mathematicians by definition want to do maths. It is not even clear that they want to do maths that would be recognised universally as good maths but at the very least they want to do maths that is good for them. Also, at some point, the majority of mathematicians want to receive recognition for their work. The usual channel for this is the traditional academic route of papers in journals or conferences but other approaches will do.

The third goal of not feeling stupid is one made emphatically by Cooper, and it is not particular to mathematicians but to everyone [8]. None of us want to look stupid and this is such a personal goal that it is almost never made explicit. Yet it can greatly clarify and motivate a lot of human activity. For mathematicians, not feeling stupid could mean producing correct proofs, or at least not obviously wrong ones, having mastery of a discipline or even just being able to ask interesting questions.

Notice that there is no high-level goal of mathematicians that they want to learn about automated deduction or that they want to use ADS. In fact, giving mathematicians a complicated system that they find difficult to understand and cannot see the benefit of will almost certainly make them feel stupid [8].

As a community, mathematicians also have high-level goals:

- Disseminate results
- Ensure the quality of published work
- Find existing results

These goals can be seen in the activity of most of the mathematical societies [15, 1]. They provide edited journals which ensure that good quality results are disseminated. Also, the American Mathematical Society publishes the Mathematical Reviews, synopses of every paper published in a mathematical journal in paper, CD ROM and web-based versions. These are so that there is a single resource for mathematicians to consult when they want to find existing mathematics. Again, ADS do not feature explicitly.

To bring automated deduction into mathematics is going to require more than just producing good ADS. Killer applications will be ones that meet at least one goal of a significant number of mathematicians.

3.1 Automated deduction in education

We have deliberately avoided the issue of mathematics education. This is mainly because it is not obviously a goal of individual mathematicians. However, it almost certainly is a goal of a large group of mathematicians or even the mathematicial community. Melis has has some success in using proof planning methods to teach mathematicians generic proving skills [17]. Once the educational value of ADS has been recognised, it may be that mathematicians will be more generally open to ADS and their goals might begin to include specific aspects of ADS.

4 Ways Forward

The two sets of goals outlined above are very general. It may be that in analysing these goals in more depth it will become apparent how ADS could fit in.

In addressing the mathematicians' goal to do mathematics, it is not clear how ADS could best offer support. This is mostly because there is no real understanding of where mathematicians could use support and whether ADS are the right tools to provide it. We are about to start a project to investigate how computer-based tools might support mathematicians. The first step is to examine thoroughly what it is that mathematicians actually do when they do maths. The investigation will not only interview and observe mathematicians about their work but also look at the products of mathematical work such as text books, journals, talks and classes to understand how mathematicians communicate their ideas to each other. Also, the project will look at what sorts of interactions might support mathematicians. For example, would a tool that helps mathematicians correct proofs as they work be useful? Currently with ADS we seem to be a long way from this goal however it would be easy to simulate such a system using a "Wizard of Oz" type set up [20]. Users may think that they are interacting with a piece of software where in reality they are interacting with a human across a network in a constrained manner. Simply performing this simulation will offer valuable insights into whether such software would be useful or degenerate into the merely irritating as with the Microsoft Word "Paper Clip". Additionaly, such simulations could suggest other ways in which ADS could be used that have not been fully explored.

As well as, going to the target users, we can consider psychological models of how people reason. There are some well-established models with good empirical evidence [12, 22]. These models are able to specify the kinds of mistake that people make generally when performing reasoning tasks. This could add to the user experience if, when working with software, the system not only affirmed correct proofs but identified and corrected common mistakes.

And there could be other auxiliary systems which would make a proof support tool more attractive such as support for finding particular results. These functions not traditionally considered an important part of automated deduction but could feasibly complement it well when integrated as part of a general support tool. Indeed, once thinking along these lines, it may be possible to come up with a myriad different tools that mathematicians might find useful. Which is why resorting back to an empirical knowledge of what mathematicians actually need is essential.

Within computer science, there is a growing recognition that mathematics is a burgeoning discipline with no real standards for communication and exploitation of results. This has lead to the area of mathematical knowledge management that aims to investigate ways in which mathematics can be organised for better retrieval [5]. This will require setting standards for the representation of mathematics and applications that can convert the standard representations into forms suitable for mathematicians to use. In addition, ensuring the quality of all of this new mathematics is difficult. Here then is an ideal area in which ADS could offer support — they could help mathematicians to check existing mathematics and to ensure that conversions between different representations do not introduce errors.

Though the issues of mathematicial knowledge management are not being driven by mathematicians, they address two of the goals identified earlier, namely that of disseminating results and achieving recognition for them. As such, mathematical knowledge management holds promise for developing new tools that mathematicians will want to use.

Just as mathematicians goal is to do maths, the goal of a researcher in automated deduction is to advance knowledge of automated deduction. Our sort of user investigation requires stepping back from the systems and becoming more of an HCI researcher. Is there something more immediate that HCI can offer for developers of ADS? We propose that Cooper's notion of personas (personæ) may have potential. He advocates not designing a system for a generic user, in this case a generic mathematician, but rather to develop an idealised but realistic mathematician persona. By targeting this persona, you are likely to produce a system that this sort of mathematician will definitely want rather than a generic system that no particular mathematician wants.

Personæ could be developed based on only a few interviews with mathematicians — Beyer and Holtzblatt [3] advocate that only fifteen users are sufficient to identify the major user activities. The key thing is to keep the persona realistic (even idiosyncratic) so that the system is more likely to address real goals of mathematicians rather than amorphous goals of an idealised user. Indeed, it may be possible to include personal anecdotes or even characteristics of famous mathematicians in the personæ. User goals are then implicitly embodied and whilst the resultant system may not suit all, or even the majority, of potential users,it has a high chance of being exactly right for some users.

5 Barriers between Mathematicians and ADS

Promoting the role of automated deduction in mathematics is not without barriers. A commonly occurring concern is that of formalisation. ADS naturally work at a very formal level of logic with manipulations at a symbolic level. This is undoubtedly a strength of ADS because it means that there can be a great deal of confidence in a proof that can be checked (albeit not generated) by a simple mechanistic system. Mathematicians though work at a very informal level which facilitates communication between humans but is very prone to errors. An analysis of journal articles reveals numerous small errors and even some quite substantial ones [11]. Similarly, standard textbooks suffer from mistakes [13], or if not actual mistakes, such large leaps of logic that it is almost impossible to reconstruct the original thinking [19].

To get humans and ADS to work together will requiring translating (and possibly correcting) the informal proofs of humans into the formal language of ADS and filling in the logical gaps. If this fails, the ADS will have to communicate back a breakdown in logic that the user may not even be aware of as relevant. There are some systems [26, 9] that propose ways of making the translation to formal proofs and others which offer ways of translating back [10, 25]. But it remains to be seen if they can be combined into a single system that can help a user correct a proof as they write it.

Formalisation, as currently done by ADS, may not be rich enough to meet the needs of mathematicians. From our own work in presenting mathematical proofs, there is also the issue of the role of examples in formal mathematics [7]. Like many formalisations of mathematics, our Polya-Lamport framework divides mathematical statements into definitions and theorems. However, examples do not fall so simply into these categories. In some sense, examples can be irrelevant to the development of a theory — illustrative but simply instantiations of existing results. In other situations, examples are crucial as providing stereotypical exemplars. They provide instances of theorems and motivations for new definitions. And in themselves they need to be defined but often have things proven about them so do not fall naturally into either definition or theorem. Dismissing them as irrelevant to theory completely overlooks their central role to mathematicians.

The simple solution to the formal/informal mismatch would be to exhort mathematicians to become more formal [4]. But this is not the goal of mathematicians so is unlikely to happen. It may be that by revealing some of the benefits of mathematical knowledge management, mathematicians might decide it is in their interest to become more formal (much as it is in their interest to learn LaTeX). But for the moment, any system which aims to bridge the gap between mathematicians and machines must make some attempt to bridge between the formal and the informal.

A further issue with bringing computer-based support to mathematicians is that of creativity. Mathematicians, like all researchers, are involved in a creative process. Sometimes that process feels instantaneous, "a flash of inspiration," and no amount of structured support is going to be available in all such circumstances. However, a lot of creativity arises a result of sustained, intentional effort and hence computers could feasibly be useful. Even in this case, a traditional HCI view of developing software to support user tasks does not really apply — there is no clear task involved in being creative, at least not one that HCI is currently able to address.

Shneiderman has developed the Genex framework that describes how computer systems might support creativity [23]. However, at the actual create step of the framework, the particular systems that might promote innovation need to be tailored for the type of creativity. It may be that ADS could play a useful role here, say, supporting the "what if" possibilities around a proof or a definition. The Genex framework does not point to ways forward in this area so, once again, it will be essential to rely on empirical investigations to find out what really supports creating new mathematics.

Even allowing for the almost philosophical issues raised by bringing automated deduction into mainstream mathematics, there are more mundane problems such as: many systems are not easily ported to a new platform; there is not a great deal of technical support for some of the systems; and there is no standard way to communicate between systems. There is going to have to be a lot of routine standardisation and development before automated deduction systems are made available to ordinary mathematicians.

6 Conclusions

We have considered how making mathematicians use automated deduction systems is far more than just making the systems easier to use. ADS must support mathematicians' goals, be they personal or communal goals, if ADS are to have a broad-based uptake in the mathematical community. Mathematical knowledge management seems to be employing ADS in a goal-directed way and so has a lot of promise. However, helping mathematicians to do mathematics requires a deeper understanding of what mathematicians do than is currently available.

Also, ADS may be more be appropriate support tools only as part of a larger system that can identify mistakes or find existing results. This broader sort of proof support can only be justified through a real understanding of what mathematicians would actually want from such a system.

7 Acknowledgments

Many thanks to Prof. Harold Thimbleby for his comments on this paper. This work was supported through EPSRC grant GR/N29280.

References

1. American Mathematical Society, `www.ams.org`
2. R. Backhouse (ed.), *Proceedings of User Interfaces for Theorem Provers*, Technical University of Eindhoven Computer Science Report 98/08, 1998
3. H. Beyer & K. Holtzblatt, *Contextual Design: Defining Customer-centered Systems*, Morgan Kaufmann, 1998
4. B. Buchberger, 'Mathematical knowledge management in Theorema,' *First International Conference on Mathematical Knowledge Management*, 2001
5. B. Buchberger & O. Caprotti, *First International Workshop on Mathematical Knowledge Management*, 2001 `www.risc.uni-linz.ac.at/institute/conferences/MKM2001`
6. P. A. Cairns, *Boundary Properties and Construction Techniques in General Topology*, DPhil Thesis, University of Oxford, 1995
7. P. A. Cairns & J. Gow, 'On Dynamically Presenting a Topology Course,' *First International Conference on Mathematical Knowledge Management*, 2001
8. A. Cooper, *The Inmates are Running the Asylum*, SAMS, 1999
9. I. Dahn & G. Schwabe, 'Personalizing Textbooks with Slicing Technologies — Concept, Tools, Architecture, Collaborative Use', HICSS, 2001
10. A. Fiedler, 'P. rex: An Interactive Proof Explainer,' in R. Goré, A. Leitsch & T. Nipkow (eds.) *Automated Reasoning – 1st International Joint Conference*, LNAI 2083, Springer Verlag, p416-420, 2001
11. J. Harrison, 'Formalized Mathematics,' *Math. Universalis*, 2, 1996
12. P. Johnson-Laird & R. Byrne, *Deduction*, Psychology Press, 1991
13. L. Lamport, 'How to write a proof,' *American Mathematical Monthly*, 102(7) p600-608, 1994
14. L. Lamport, *LaTeX: A Document Preparation System, 2nd edn*, Addison-Wesley, 1994
15. London Mathematical Society, `www.lms.ac.uk`
16. W. McCune, 'Solution of the Robbins problem,' *J. Automated Reasoning*, 19(3) p263-276, 1997
17. E. Melis, C. Glasmacher, C. Ullrich & P. Gerjets, 'Automated Proof Planning for instructional design,' in *Annual Conference of the Cognitive Science Society*, p633-638, 2001
18. J. Nielsen, *Usability Engineering*, Morgan Kaufmann, 1993
19. L. C. Paulson & K. Grabczewski, 'Mechanizing Set Theory,' *J. of Automated Reasoning*, 17 p291-323, 1996
20. J. Preece, Y. Rogers & H. Sharp, *Interaction Design*, John Wiley & Sons, 2002

21. *Proof Transformation & Presentation, PTP-01*, IJCAR, 2001, `www.ags.uni-sb.de/~ptp-01`
22. L. J. Rips, *The Psychology of Proof*, MIT Press, 1994
23. B. Shneiderman, "Creating creativity: user interfaces for supporting innovation," in J. M. Carrol (ed.) *Human-Computer Interaction in the New Millenium*, Addison Wesley, 2002
24. *Topology & Its Applications*, `www.elsevier.com/locate/latex`
25. M. Wenzel, 'Isar — a Generic Interpretative Approach to Readable Formal Proof Documents,' in *Proceedings of TPHOLs'99*, Springer, 1999
26. C. Zinn, 'Towards the Mechanical Verification of Textbook Proofs,' *7th Workshop on Logic, Language, Information and Computation (WOLLIC-2000)*, 2000

Notes on Formalizing Induction on the Number of Sets — “Ideal $\subseteq \bigcup_i$ PrimeIdeal$_i$ $\Rightarrow \exists i.$ Ideal $\subseteq$ PrimeIdeal$_i$” —

(Extended Abstract)

CHEN Lingjun[1], Hidetsune KOBAYASHI[1],
Hirokazu MURAO[2*], and Hideo SUZUKI[3]

[1] Department of Mathematics, Nihon University
[2] Department of Computer Science, University of Electro-Communications
[3] Tokyo Institute, Polytechnic University

1 Introduction

This paper presents an application of inductive theorem proving in ideal theory, appeared in the first steps of our effort for formalizing abstract ring theory. Induction is one of the most important and useful strategies for mathematical proof, and widely used also in mechanical theorem proving [KMM00,ACL], [Wal92,INK], [BvHHS91,vH89]. It is often used to prove equations in elementary arithmetics. More advanced use is to prove that some fact holds for arbitrary number of objects, usually a consecutive sequence of linearly ordered objects. In the early stage of our on-going project, we faced a proposition of this kind, and in investigating its appropriate formalization, we noticed that the statement of the fact in terms of natural language implies the fact in meta-logic and the inductive proof uses this implied fact. To prove some fact that holds for a set of some objects, inductive hypothesis assumes that it holds for its arbitrary subset. In this extended abstract, we describe this particular problem. We clarify the subtle meaning of the statement in natural language and we present how to formalize the fact and the proof. The formalization is done with Isabelle/Isar [Wen], a generic theorem prover with higher-order logic.

Software systems for automated reasoning and for mathematical proof assistant has been well developed to date, such as Coq [Coq], HOL [GM93], Isabelle/Isar, Nuprl [Jac94], TPS [TPS], and has been being applied to practical problems. They are used mainly to certify the correctness of behaviors of hardware, software or algorithms. Mathematics itself also often be their target. There are some softwares dedicated solely to mathematics, such as Geometry Expert [CGZ96], Theorema [The]. Also, most of the existing software systems give definitions of basic mathematical concepts and structures, such as group, ring and

* Supported in part by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (C), 14580365, 2002.

so on, as part of library or simple sample problems, and TPTP [TPT] includes many related problems. Most of those examples treat merely the properties of concrete calculations and operations of domain elements, rather than those of the concepts and the structures, namely mathematical theory itself. On the other hand, there are a limited number of experiments on theories of abstract algebra. Set theory has been extensively studied [PG96], [Far01] as well as group theory has been [KP99], [Yu90]. In his PhD thesis work [Jac95], Paul Jackson explored, with the plan of introducing rigor into computations performed in computer algebra systems, computational abstract algebra required by the plan. State-of-the-art in this direction will the implementation of Buchberger's algorithm in Coq by Laurent Thery [The01]. The MIZAR project [Miz] has been developing and maintaining a big archive of formal descriptions of mathematical theories with computer-assisted formal proofs. Also in the Mizar library [RST01], formalization of abstract theory of rings and ideals can be found, which further accomplished the formalization of Hilbert basis theorem.

It is our recognition that abstract ring theory has been being developed and evolved in pure abstraction and will be easily adapted for formalization, and that the techniques used for proof are rather limited and the applicable area of every technique is almost distinct and therefore, the automation will not be very difficult. Motivated by this recognition and inspired by the work by Chou *et. al.* [CGZ96], we started a project to mechanize the theory of commutative rings. The aim of our project is systematic formalization with automated mechanical theorem proving of the ring theory, namely, transfer of the theory in a textbook to a machine. The project is initiated by defining and describing the notions and the facts in a textbook by Atiyah and Macdonald [AM69] using Isabelle/Isar. In this initial attempt, we give step-by-step specification of proof manually for every fact and describe how it should be proved, especailly in the basic part of the theory for efficiency, and to invistigate how the whole theory is structured and ought to be reorganized and to design appropriate schemes for future automation.

The target in this short paper is about the set theoretic property of an ideal and an arbitrary number of prime ideals, as is given in the title. In the proof, induction is applied to a set of ideals, and we notice that the hypothesis makes assumptions not merely on the number of set elements but on a set of arbitrary subsets of the set, which will attract us to developing a generic method for formalization of the induction on the number of target objects. The main goal is a concise formal description of the fact and a clean formalization of the inductive proof and the techniques and tools used in our effort.

The next section describes our target mathematical fact and a inductive proof. Preparing some basic mathematical notions in Sect. 3, we formalize the fact and the proof in Sect. 4. In the formalization, we point out the fact implied in the statement by natural language in Sect. 4.1 and for treating this, we introduce a mathematical concept in Sect. 4.2, which is generic to the induction on the number of objects.

2 Proposition to Formalize and Prove

The main topic in this paper lies in the following basic mathematical fact. We quote the statement of the fact and the proof from the textbook [AM69, p.8].

Proposition 1.11 1 (i) Let $\mathfrak{p}_1, ..., \mathfrak{p}_n$ be prime ideals and let $\mathfrak{a}$ be an ideal contained in $\bigcup_{i \le n} \mathfrak{p}_i$. Then $\mathfrak{a} \subseteq \mathfrak{p}_i$ for some i.
Proof. (i) is proved by induction on n in the form

$$\mathfrak{a} \not\subseteq \mathfrak{p}_i \; (1 \le {}^{\forall} i \le n) \Rightarrow \mathfrak{a} \not\subseteq \bigcup_{i \le n} \mathfrak{p}_i. \tag{1}$$

It is certainly true for $n = 1$. If $n > 1$ and the result is true for $n - 1$, then for each i there exists $x_i \in \mathfrak{a}$ such that $x_i \notin \mathfrak{p}_j$ whenver $j \neq i$. If for some i we have $x_i \notin \mathfrak{p}_i$, we are through. If not, then $x_i \in \mathfrak{p}_i$ for all i. Consider the element

$$y = \sum_{i=1}^{n} x_1 x_2 \cdots x_{i-1} x_{i+1} x_{i+2} \cdots x_n;$$

we have $y \in \mathfrak{a}$ and $y \notin \mathfrak{p}_i$ $(0 \le i \le n)$. Hence $\mathfrak{a} \not\subseteq \bigcup_{i \le n} \mathfrak{p}_i$. ■

Remarks. We rewrite the newly introduced element y as e_n:

$$e_n = \sum_{i=1}^{n} x_1 x_2 \cdots x_{i-1} x_{i+1} x_{i+2} \cdots x_n,$$

and instead of the above symmetric definition, we consider the inductive definition:

$$e_n = e_{n-1} x_n + x_1 x_2 \cdots x_{n-1}.$$

Note here that $e_{n-1} x_n$ represents such an element of $\mathfrak{a}$ that is $\in \mathfrak{p}_n$ but $\notin \mathfrak{p}_i$ for any $i < n$, while $x_1 x_2 \cdots x_{n-1} \in \mathfrak{a}$ is $\notin \mathfrak{p}_n$ but $\in \mathfrak{p}_i$ for any $i < n$. Especially, e_{n-1} is assumed $\notin \mathfrak{p}_i$ for any $i < n$, whose existence in $\mathfrak{a}$ is certified by the induction hypothesis. Assuming that $e_{n-1} \notin \mathfrak{p}_n$ directly leads to the conclusion. In the other case that $e_{n-1} \in \mathfrak{p}_n$, the multiplication by x_n, which guarantees becoming $\in \mathfrak{p}_n$ without violating the assumed property of e_{n-1}, is redundant. Therefore, in this context, the only requirement is that for all $i < n$, there exists x_i such that $x_i \in \mathfrak{p}_i$ but $x_i \notin \mathfrak{p}_n$.

3 Formalization of Basic Mathematical Concepts

This section describes the basic mathematical concepts required for formalizing the fact described in the previous section, and explain how they can be formalized. Their concrete formal descriptions in Isabelle/Isar are given in Appendix 5 and more details may be found in our progress report [CKMS02]. The concepts of our major concern introduced in this section are *ideal*s, *prime ideal*s, and a method (function) to represent the product of indexed expressions.

An *ideal* $\mathcal{I}$ of a *ring* $\mathcal{R}$ (described later) is such a non-empty *subgroup* (closed under the additive operation, described later) of $\mathcal{R}$ that contain all the elements of $\mathcal{R}$ multiplied by an element of $\mathcal{I}$:

$$\forall x \in \mathcal{R}. \forall y \in \mathcal{I}. (\texttt{mul}_{\mathcal{R}}\, x\, y) \in \mathcal{I}.$$

The multiplication operation (denoted as $\texttt{mul}_{\mathcal{R}}$ above) is defined in $\mathcal{R}$ as a component of a ring. Commutative *ring* is defined as such an extension of (additive) group that incorporates commutative multiplication operation, as well as a unit, termed one, of multiplication. A commutative ring $\mathcal{R}$ is represented by a record with two components for those extensions added to a record of a *group*. An (additive) *group* is a set (termed carrier) with addition operation, its inverse operation (usually called negation) and a unit (termed zero), and is represented as a record of those four components. Group is closed under addition and negation, and a subset of some set, usually a group, is called a *subgroup* of the set if it is closed under these operations and consequently includes zero. Addition and multiplication are defined to be commutative, associative, and further distributive in their combination as usual, and also the units are defined in conjunction with these operations. Ring inherits the property of group, and is closed under multiplication.

An ideal $\mathcal{I}$ is called *prime* if it never contains the product of non-members, *i.e.*, if $\texttt{mul}_{\mathcal{R}}\, x\, y \in \mathcal{I}$ implies $x \in \mathcal{I}$ or $y \in \mathcal{I}$. Such an ideal is called a *prime ideal.*

To express such terms as $x_1 x_2 \cdots x_n$, we usually use such a notation as $\prod_{i=1}^{n} x_i$. We prepare a similar function in Isabelle. Formally, as in Isabelle, indexed variables are treated as a function of integer values of an index, and represented by the function symbol, *i.e.*, x_i's are represented by x. We define $\texttt{nmul0}_R(n, f)$ as

$$\texttt{nmul0}_R(n, f) = \prod_{i=1}^{n} f(i),$$

where each $f(i)$ is $\in \mathcal{R}$ and the multiplication is performed over $\mathcal{R}$. This function will be formally defined by primitive recursion as

$$\texttt{nmul0}_{\mathcal{R}}(n, f) = \begin{cases} \texttt{one}_{\mathcal{R}} & \text{for } n = 0, \\ \texttt{mul}_{\mathcal{R}}\, f(\texttt{Suc}\ n)\, \texttt{nmul0}_{\mathcal{R}}(n, f) & \text{for } n > 0. \end{cases}$$

4 Formalizing the Proof: Induction on the Number of Ideals

4.1 Overview – Problem and Resolution

Using the definitions of the previous section, we may translate our target proposition into Isabelle/Isar as

$$\begin{aligned} &[\!|\ \forall i \le n.\, \texttt{PrimeIdeal}_{\mathcal{R}}(\texttt{P}\ i);\ \texttt{Ideal}_{\mathcal{R}}\mathfrak{a};\ \mathfrak{a} \subseteq \bigcup\{..n\text{::}\ \texttt{nat}\}\ \texttt{P}\ |\!] \\ &\qquad \Longrightarrow \exists i \le n.\, \mathfrak{a} \subseteq (\texttt{P}\ i); \end{aligned}$$

where "$\{..n{::}\,\texttt{nat}\}$" represents a set of natural numbers $\leq n$, "$\bigcup s$ P" does the union of the sets (P x) for $x \in s$, *i.e.*, $\{\, y.\, \exists x \in s.\, y \in (\texttt{P}\ x) \,\}$, and "P i" stands for an expression with subscript i, $\mathfrak{p}_i$. In the inductive proof (of the case of n from $n-1$) above, we have assumed the existence of x_i for all $i \leq n$ such that $x_i \in \mathfrak{a}$, $x_i \in \mathfrak{p}_i$, and $x_i \notin \mathfrak{p}_j$ for all $j \neq i$ and $j \leq n$:

$$\forall i \leq n.\ (\exists x_i.\, x_i \in \mathfrak{a} \,\&\, x_i \in (\texttt{P}\ i) \,\&\, \forall j \leq n.\, j \neq i \longrightarrow x_i \in (\texttt{P}\ j))\ .$$

Literally, the inductive hypothesis when $(n-1)$ in the form (1) insists only that there exists and element of $\mathfrak{a}$ not included in any of $\mathfrak{p}_1, ..., \mathfrak{p}_{n-1}$:

$$\exists e \in \mathfrak{a}.\, \forall i \leq n-1.\, e \notin \mathfrak{p}_i$$

and tells nothing about the relation with $\mathfrak{p}_n$ or $\mathfrak{p}_n$ itself. In contrast, the natural language version of the proof above introduces and uses the assumption $x_i \notin \mathfrak{p}_n$ with no special assumption. What is the gap between the formalization and the proof in English, or what is implied by the assumptions in natural language? In the formalization process above, we have considered $\mathfrak{p}_n$ a new comer to a fixed set of ideals, $\mathfrak{p}_1, ..., \mathfrak{p}_{n-1}$, and try to explore the formal proof. On the other hand, what is required for an inductive hypothesis is the fact that the conclusion holds for an aribtray choice of $(n-1)$ ideals from $\mathfrak{p}_1, ..., \mathfrak{p}_n$. The addition of $\mathfrak{p}_n$ ought to be regarded simply as an increment of the number of elements in the target set of prime ideals, and never prompts us a special treatment for $\mathfrak{p}_n$. Although not clearly stated in the proposition, the claim holds for an arbitrary set of prime ideals, and this fact is substantial for completing the induction. Formal proof requires clear indication of this fact, and formal description of the proposition must include the specification($\forall$) for it. The specification of $\mathfrak{p}_i$ in the form of (P i) supports further clarification in our effort for formalization. We regard P as a map from an integer to an element of some ordered set of prime ideals (P: $\mathcal{N} \rightarrow \{\, \texttt{P}\ i \,\}$, where $\mathcal{N}$ represents the set of natural numbers). The proposition should state P, the set of $\mathfrak{p}_i$'s, can be arbitrary. Now, the formal description of the proposition (i) in the form (1) ought to be

$$\begin{aligned} &\Lambda \texttt{P}.\, \Lambda n.\, [\!|\, \forall i \leq n.\, \texttt{PrimeIdeal}_{\mathcal{R}}(\texttt{P}\ i);\ \forall i \leq n.\, \mathfrak{a} \not\subseteq (\texttt{P}\ i);\ \texttt{Ideal}_{\mathcal{R}}\mathfrak{a} \,|\!] \\ &\qquad \Longrightarrow \mathfrak{a} \not\subseteq \bigcup \{..n{::}\,\texttt{nat}\}\ \texttt{P}; \end{aligned}$$

The next thing to consider is how to represent a choice of $(n-1)$ ideals from $\mathfrak{p}_1, ..., \mathfrak{p}_n$. Here notice again that with respect to the two ordered sets of n and of $n-1$ elements, the element of their difference is of no significance, but we regard them as two ordered sets happen to have equal elements. Our idea to clarify this situation is the use of a function which establishes an elementwise correspondence between the two ordered sets, more concretely, a function for indices (subscripts) of elements. Let $S_n = \{\, \sigma_i,\ 1 \leq i \leq n \,\}$ and $S^k_{n-1} = \{\, \tau_i,\ 1 \leq i \leq n-1 \,\}$ be ordered sets of n and of $n-1$ elements respectively. We assume that $S_n - S^k_{n-1} = \{\, \sigma_k \,\}$ and that there is a correspondence between the elements as follows:

$$\tau_i = \begin{cases} \sigma_i & \text{for } i < k, \\ \sigma_{i+1} & \text{for } i > k. \end{cases}$$

We define a function for subscripts as follows:

$$\boldsymbol{iskip}(k,i) \overset{\text{def}}{=} \text{if } i < k \text{ then } i \text{ else } i+1.$$

Then, the above correspondence can be rewritten simply as $\tau_i = \sigma_{\boldsymbol{iskip}(k,i)}$. A set of $(n-1)$ chosen elements from $\mathfrak{p}_1, ..., \mathfrak{p}_n$ can be represented by

$$\{\, \mathfrak{p}_{\boldsymbol{iskip}(k,i)}.\, 1 \le i \le n \,\} = \{\, \mathfrak{p}_i.\, 1 \le i \le n \,\&\, i \ne k \,\}.$$

The function form

$$\boldsymbol{iskip}(k) = (\lambda\; i.\, \text{if } i < k \text{ then } i \text{ else } i+1)$$

may be used for representing their union/intersection as

$$\bigcup \{..n{::}\ \mathtt{nat}\}\ (\mathtt{P} \circ \boldsymbol{iskip}(k)),$$

where $f \circ g$ represents a function composition such that

$$f \circ g = (\lambda\; x.\, f(g(x))).$$

4.2 Generic Tools

With Isabelle, induction is done from n to $n+1$ via applying `induct_tac`, and $n+1$ is represented as `Suc` n, the successor of n. In the following, we denote `Suc` n by $n+1$ for simplicity. We prepare some generic tools required to complete our inductive proof. Very basic one is the following order relation:

lemma *Sucnele_notlt*: $[\,|\ k+1 \ne i;\ i \le k+1\ |\,] \Longrightarrow \neg(k < i)$;

This can be proved by simply applying rewrite rules, in a backward reasoning style, as

1. $(\neg k < i) = (i \le k)$ (*linorder_not_less* from the Isabelle/HOL library),
2. $(i \le k) = (i < k+1)$ (*less_Suc_eq_le [THEN sym]*), and
3. $(i < k) = (i \le k \,\&\, i \ne k)$ (*order_less_le*).

We investigate the range of an index variable, often used for subscripts, satisfying some condition. In the following, let C, C_1 and C_2 denote arbitrary predicate conditions. Besides these, we may freely use various notations as $x(i)$, $x\ i$ or x_i to represent indexed variables, all of which are treated as functions in Isabelle. Existence of a variable inside an indexed environment (within a formula bound by some other quantifier) may be replaced by the existence of a generic variable with index:

lemma *foreachex_exforall*: $\forall i \le n.\, \exists e.\, C(e,i) \Longrightarrow \exists x.\, \forall i \le n.\, C(x\ \ i, i)$;

The following fact will be obvious and can be proved by case distinction $(i \le n+1) = (i = n+1 \lor i \le n)$ (defined as *lesuc_iff*):

lemma *range_Suc*: $((\forall i \le n.\, C(i))\,\&\, C(n+1)) = (\forall i \le n+1.\, C(i))$;

In contrast with the inclusion of an additional Suc case above, we can prove the case of elimination. The following will be clear from ***iskip***'s definition.

lemma *skipped_choice*: $\forall i \le n{+}1.\, C_1(f(i)) \Longrightarrow \forall i \le n.\, C_1(f(\boldsymbol{iskip}(j,i)))$;

Here, j can be arbitrary. This corresponds to the premise of the inductive hypothesis in our target proof. For the induction to proceed, the corresponding conclusion must be deduced. Our idea is to use the conclusion of the above lemma as a condition of modus ponens (as P of $(P \,\&\, P \longrightarrow Q) \Longrightarrow Q$), instead of $\forall i \le n.\, C_1(f(i))$, another set of n elements. We prove a fact that holds for sets of n elements inductively, by regarding the fact as holding for an arbitrary set $F_1, ..., F_n$, and applying to a set $f(\boldsymbol{iskip}(j,1)), ..., f(\boldsymbol{iskip}(j,n))$.

lemma *skipped_choice_mp*:
$[\!|\ \forall i \le n+1.\, C_1(f(i));\ \forall F.\, (\ \forall i \le n.\, C_1(F(i)) \longrightarrow \forall i \le n.\, C_2(F(i))\ )\ |\!]$
$\Longrightarrow \forall j \le n.\, (\forall i \le n.\, C_2(f(\boldsymbol{iskip}(j,i))))$;

Actually used in our proof is the following extension, which adds some condition for an element in some set A:

lemma *Plemma_skipped_choice_mp*:
$[\!|\ \forall i \le n+1.\, C_1(f(i))$;
$\forall F.\, (\ \forall i \le n.\, C_1(F(i)) \longrightarrow \exists x \in A.\, \forall i \le n.\, C_2(x, F(i))\ )\ |\!]$
$\Longrightarrow \forall j \le n.\, (\exists x \in A.\, \forall i \le n.\, C_2(x, f(\boldsymbol{iskip}(j,i))))$;

Next, we consider the cases when ***iskip*** appears in the premise. The following seems obvious from the definition of ***iskip***, but is not in Isabelle.

lemma *skipped_cond*:
$[\!|\ t \le n;\ \forall i \le n.\, C(\boldsymbol{iskip}(t,i))\ |\!] \Longrightarrow \forall j \le n+1.\, j \ne t \longrightarrow C(j)$;

In the distinct case of $j < t$, we can use the premise; letting $i = j$ where $j \le n$ deduced and the expansion of ***iskip***'s definition give the required result. In another case when $t \le j$, we consider the case $t < j$ because $j \ne t$ is assumed in the conclusion. Then, $t + 1 \le j$, which leads to $\exists k.\, j = k + 1$ (rule *Suc_le_D*) and $k \le n$. By *Sucnele_notlt*, $\neg(t < k)$. Then expansion of ***iskip***'s definition when $i = k$ in the premise gives $C(k + 1) = C(j)$. In our proof, this lemma is used with some additional condition in conjunction with *foreachex_exforall*:

lemma *Plemma_skipped_cond*:
$[\!|\ \forall i \le n.\, \exists e \in A.\, \forall j \le n.\, C(e, \boldsymbol{iskip}(i,j))\ |\!]$
$\Longrightarrow \exists x.\, \forall i \le n.\, x(i) \in A \,\&\, (\forall j \le n+1.\, j \ne i \longrightarrow C(x(i), j))$;

4.3 Membership of Ideal Elements

Any ideal of a ring $\mathcal{R}$ is a subgroup of $\mathcal{R}$, and therefore closed under additive operation of $\mathcal{R}$. This inheritance cannot be treated automatically, and to encourage automatic reasoning, we explicitly add the known fact AGroup$\mathcal{R}$ to the premise (by *frule*).

lemma *iAddClose*: $[\mid \mathtt{Ideal}_{\mathcal{R}}\mathcal{I};\ x \in \mathcal{I};\ y \in \mathcal{I}\ \mid] \Longrightarrow x +_R y \in \mathcal{I}$;

Group, say G, is closed under (additive) inverse operation, $-_{\mathcal{R}}x \in G$ for $x \in G$, and therefore, if both x and $x +_{\mathcal{R}} y$ are included in a group, so is $y = (-_{\mathcal{R}}x) +_{\mathcal{R}} (x +_{\mathcal{R}} y)$. This argument also holds for ideals (ideal inherits the properties of group), and its contraposition gives

lemma *iInclAddNeg*:
$[\mid \mathtt{Ideal}_{\mathcal{R}}\mathcal{I};\ x \in \mathcal{I};\ y \notin \mathcal{I};\ y \in \mathtt{carrier}(\mathcal{R})\ \mid] \Longrightarrow x +_{\mathcal{R}} y \notin \mathcal{I}$;

From the definition of ideal, it will be clear that the product of elements of a ring $\mathcal{R}$ is a member of an ideal of $\mathcal{R}$ if at least one of them is a member of the ideal:

lemma *nmul0CloseI*: $\mathtt{Ideal}_{\mathcal{R}}\mathcal{I} \Longrightarrow$
$((\forall i \le n.\, x_i \in \mathtt{carrier}(\mathcal{R}))\ \&\ \exists i \le n.\, x_i \in \mathcal{I}) \longrightarrow \prod_{i \le n} x_i \in \mathcal{I}$;

Formal proof requires induction on n and case distinction of x_{n+1}; $\prod_{i \le n+1} x_i = \left(\prod_{i \le n} x_i\right) x_{n+1}$ is obviously $\in \mathcal{I}$ if $x_{n+1} \in \mathcal{I}$, and otherwise, at least one of x_i, $i \le n$ must be $\in I$, which implies $\prod_{i \le n} x_i \in \mathcal{I}$ by the induction hypothesis, and leads to the required result. In contrast, the counterpart with respect to the membership requires ideal's primality:

lemma *nmul0Pmi*: $\mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{P} \Longrightarrow$
$(\forall i \le n.\, x_i \in \mathtt{carrier}(\mathcal{R})\ \&\ x_i \notin \mathfrak{P}) \longrightarrow \prod_{i \le n} x_i \notin \mathfrak{P}$;

This can be proved by induction more simply than the previous, because non-membership of $\left(\prod_{i \le n} x_i\right) x_{n+1}$ to a prime ideal requires both those of two factors, which are given directly by the induction hypothesis.

Actually used in our proof for the main goal are the combinations of the above. A simple combination of *nmul0CloseI* and *iAddClose* gives

lemma *Plemma_makeElm*:
$[\mid \mathtt{Ideal}_{\mathcal{R}}\mathcal{I};\ \forall i \le n.\, x_i \in \mathcal{I};\ e \in \mathcal{I}\ \mid] \Longrightarrow e + \prod_{i \le n} x_i \in \mathcal{I}$;

The expression of the sum corresponds to e_{n+1} introduced in the remarks of Section 2. Required facts about the expression are as follows. A combination of *nmul0CloseI* and *iInclAddNeg* gives

lemma *Plemma_ElmNotMem*:
$[\mid e \in \mathtt{carrier}(\mathcal{R});\ \forall i \le n.\, x_i \in \mathtt{carrier}(\mathcal{R});\ \mathtt{Ideal}_{\mathcal{R}}\mathfrak{P};\ e \notin \mathfrak{P};$
$j \le n;\ x_j \in \mathfrak{P}\ \mid] \Longrightarrow e + \prod_{i \le n} x_i \notin \mathfrak{P}$;

and that of *nmul0Pmi* and *iInclAddNeg* gives

lemma *Plemma_ElmNotMem1*:
$[\mid \forall i \le n.\, x_i \in \mathtt{carrier}(\mathcal{R});\ \mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{P};\ e \in \mathfrak{P};$
$\forall i \le n.\, x_i \notin \mathfrak{P}\ \mid] \Longrightarrow e + \prod_{i \le n} x_i \notin \mathfrak{P}$;

4.4 Formal Proof of the Proposition

$$\forall n \geq 0.\ \texttt{Ideal}_{\mathcal{R}}\mathfrak{a} \implies \forall \mathfrak{P}.\ \left((\forall i \leq n.\, \texttt{PrimeIdeal}_{\mathcal{R}} \mathfrak{P}_i \ \texttt{\&}\ \mathfrak{a} \not\subseteq \mathfrak{P}_i) \longrightarrow \mathfrak{a} \not\subseteq \bigcup_{i \leq n} \mathfrak{P}_i \right)$$

where $\mathfrak{P}$ is a symbol used to designate a set of prime ideals, and considered as a map $\mathfrak{P} : \mathcal{N} \to \{\mathfrak{P}_i\}$, representing an actual prime ideal by $\mathfrak{P}_i$, a value of the map.

The proposition itself is proved by induction on n (in Isabelle/Isar, *apply* (induct_tac "n")). If $n = 0$, trivial. Assuming that the formula in the big parentheses is true for $n (\geq 0)$, we show that the correctness of the formula with n being replaced by $n + 1$. The conclusion of the inner implication ($\longrightarrow$) will be treated as

$$\mathfrak{a} \not\subseteq \bigcup_{i \leq n} \mathfrak{P}_i \ \equiv\ \exists x \in \mathfrak{a}.\, \forall i \leq n.\, x \notin \mathfrak{P}_i.$$

1. Assuming $\texttt{Ideal}_{\mathcal{R}}\mathfrak{a}$, its implied result when n, and, also the premise of the inner implication when $n \leftarrow n + 1$ with $\mathfrak{P}_i = \mathfrak{p}_i$ (*i.e.*, $\forall i \leq n + 1.\, \texttt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_i \ \texttt{\&}\ \mathfrak{a} \not\subseteq \mathfrak{p}_i$), we show the conclusion when $n \leftarrow n + 1$:

$$\begin{aligned} &[|\ \texttt{Ideal}_{\mathcal{R}}\mathfrak{a};\\ &\quad \forall \mathfrak{P}.\ ((\forall i \leq n.\, \texttt{PrimeIdeal}_{\mathcal{R}}\mathfrak{P}_i \ \texttt{\&}\ \mathfrak{a} \not\subseteq \mathfrak{P}_i) \longrightarrow \exists x \in \mathfrak{a}.\, \forall i \leq n.\, x \notin \mathfrak{P}_i);\\ &\quad \forall i \leq n + 1.\, \texttt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_i \ \texttt{\&}\ \mathfrak{a} \not\subseteq \mathfrak{p}_i)\ |]\\ &\qquad \implies \exists x \in \mathfrak{a}.\, \forall i \leq n + 1.\, x \notin \mathfrak{p}_i \end{aligned}$$

2. Direct from the premise, there exists an element of $\mathfrak{a}$ not included in any of $\mathfrak{p}_i$ for $i \leq n$, and let e be such an element. If $e \notin \mathfrak{p}_{n+1}$, then the conclusion is straight, via the rule ***range_Suc*** above. So, hereafter, we assume $e \in \mathfrak{p}_{n+1}$, and add the following formulae to the premise:

$$e \in \mathfrak{a};\ \forall i \leq n.\, e \notin \mathfrak{p}_i;\ e \in \mathfrak{p}_{n+1}$$

   Notice that this e corresponds to x_{n+1}.
3. By applying ***Plemma_skipped_choice_mp*** with $f(i) = \mathfrak{p}_i$, $C_1(X) = \texttt{PrimeIdeal}_{\mathcal{R}}X \ \texttt{\&}\ \mathfrak{a} \not\subseteq X$, $F(i) = \mathfrak{P}_i$, and $C_2(X, Y) = X \notin Y$, we duduce

$$\forall j \leq n.\, (\exists z \in \mathfrak{a}.\, \forall i \leq n.\, z \notin \mathfrak{p}_{\boldsymbol{iskip}(j,i)}).$$

4. Further translation using ***Plemma_skipped_cond*** leads to the existence of elements x_i (or a map $x : \mathcal{N} \to \{x_i\}$) such that

$$\forall j \leq n.\, (x_j \in \mathfrak{a} \ \texttt{\&}\ \forall i \leq n + 1.\, i \neq j \longrightarrow x_j \notin \mathfrak{p}_i),$$

   which is to be added to the premise.
   Here, we assume $\forall j \leq n + 1.\, x_j \in \mathfrak{p}_j$; otherwise the existence of x_j being $\notin \mathfrak{p}_j$ gives the conclusion.

5. At this point, we have the following premise:

   $[|\ \mathtt{Ideal}_{\mathcal{R}}\mathfrak{a};$
   $\forall \mathfrak{P}.\ ((\forall i \leq n.\mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{P}_i\ \&\ \mathfrak{a} \not\subseteq \mathfrak{P}_i) \longrightarrow \exists x \in \mathfrak{a}. \forall i \leq n.\, x \notin \mathfrak{P}_i)\,;$
   $\forall i \leq n+1.\mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_i\ \&\ \mathfrak{a} \not\subseteq \mathfrak{p}_i;$
   $e \in \mathfrak{a};\ \forall i \leq n.\, e \notin \mathfrak{p}_i;\ e \in \mathfrak{p}_{n+1};$
   $\forall j \leq n.\, x_j \in \mathfrak{a}\,\&\,(\forall i \leq n+1.\, i \neq j \longrightarrow x_j \notin \mathfrak{p}_i);$
   $\forall j \leq n+1.\, x_j \in \mathfrak{p}_j\ |]$

   This is sufficient for showing that $e + \prod_{j \leq n} x_j$ is a member of $\mathfrak{a}$ by *Plemma_makeElm*:

   $$[|\ \mathtt{Ideal}_{\mathcal{R}}\mathfrak{a};\ \forall i \leq n.\, x_i \in \mathfrak{a};\ e \in \mathfrak{a}\ |] \Longrightarrow e + \prod_{i \leq n} x_i \in \mathfrak{a};$$

   and that it is not a member of any of $\mathfrak{p}_j$ for $j \leq n+1$ by *Plemma_ElmNotMem* for $j \leq n$:

   $$[|\ e \in \mathtt{carrier}(\mathcal{R});\ \forall i \leq n.\, x_i \in \mathtt{carrier}(\mathcal{R});\ \mathtt{Ideal}_{\mathcal{R}}\mathfrak{p}_j; \\ e \notin \mathfrak{p}_j;\ j \leq n;\ x_j \in \mathfrak{p}_j\ |] \Longrightarrow e + \textstyle\prod_{i \leq n} x_i \notin \mathfrak{p}_j;$$

   where $\mathtt{Ideal}_{\mathcal{R}}\mathfrak{p}_j$ is deduced from $\mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_j$ and the membership $\in \mathtt{carrier}(\mathcal{R})$ from the membership $\in \mathfrak{a}$, and for $j = n{+}1$ by *Plemma_ElmNotMem1*:

   $$[|\ \forall i \leq n.\, x_i \in \mathtt{carrier}(\mathcal{R});\ \mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_{n+1};\ e \in \mathfrak{p}_{n+1}; \\ \forall i \leq n.\, x_i \notin \mathfrak{p}_{n+1}\ |] \Longrightarrow e + \textstyle\prod_{i \leq n} x_i \notin \mathfrak{p}_{n+1};$$

Finally, the contraposition of the main result gives our desired fact:

lemma*Prop_1_11_casei*:
  $[|\ \forall i \leq n.\mathtt{PrimeIdeal}_{\mathcal{R}}\mathfrak{p}_i;\ \mathtt{Ideal}_{\mathcal{R}}\mathfrak{a};\ \mathfrak{a} \subseteq \bigcup\{..n{::}\ \mathtt{nat}\}\ \mathtt{P}\ |]$
    $\Longrightarrow \exists i \leq n.\, \mathfrak{a} \subseteq \mathfrak{p}_i;$

5 Concluding Remarks

We described a formal method of inductive proof to be applied to a set of objects. It is pointed out that hypothesis of the induction on the number of set elements may assume that the fact to be proved holds for its arbitrary proper subsets. With the application in this paper, we presented a formal method to represent arbitrary subsets with n elements of a set with $n+1$ elements. We prepared a mapping function of element indices which assigns a new ordinal number to each element in a subset. The method itself is generic enough to be extended to any subsets of arbitrary number of elements. Furthermore, two lemmas, *skipped_choice_mp* and *skipped_cond*, related with this function are proved and give a general scheme of this type of inductive proof. While the former is quite simple and can be proved almost automatically, the latter is one of the most complicated and required, in its proof, more than 20 steps of manual application of a rule and simplification. In our current implementation, all the proof processes are controlled manually except the transformation of logical expressions and for the facts that can be proved by simplification or assumption. All the lemmas described here and all the substeps in Sect. 4.4 required only 30 manual steps at most.

The topic described in this paper was the first major difficulty in our formalization effort, which appeared in developing ideal thoery, one of the basis theories in ring theory. The problem itself is irrelevant to the theory, and the complication is raised by the subtle statement in natural language of a logically-clear fact in algebra. However, from the viewpoint of formalization, the fact implied in natural language is not trivial and requires a treatment in higher-order logic. Our idea for resolution is quite natural and attractive for its logical cleanness, and is sufficiently generic. Our formalization effort is in progress, and it often reveals a gap between the statement in natural language and a logical structure of a mathematical fact. Other interesting topics, our treatments and techniques used there are left for future publication.

Acknowledgement

The authors would like to thank the reviewers for helpful comments and suggestions.

References

[ACL] ACL2, a computational logic applicative common lisp. `http://www.cs.utexas.edu/users/moore/acl2/`.

[AM69] M. F. Atiyah and I. Macdonald. *Introduction to Commutative Algebra*. Addison-Wesley, 1969.

[BvHHS91] A. Bundy, F. van Harmelen, J. Hesketh, and A. Smaill. Experiments with proof plans for induction. *Journal of Automated Reasoning*, 7:303–324, 1991.

[CGZ96] S.-C. Chou, X.-S. Gao, and J.-Z. Zhang. An introduction to Geometry Expert. In *Proc. CADE-13*, number 1104 in LNAI, pages 235–239. Springer, 1996.

[CKMS02] Chen Lingjun, H. Kobayashi, H. Murao, and H. Suzuki. A progress report on formalizing theory of commutative ring. *Workshop SNSC'01*, 2001.

[Coq] The Coq proof assistant. `http://coq.inria.fr/`.

[Far01] W.M. Farmer. STMM: a set theory for mechanized mathematics. *Journal of Automated Reasoning*, 26:269–289, 2001.

[GM93] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993. see also `http://www.dcs.gla.ac.uk/ tfm/fmt/hol.html`.

[vH89] F. van Harmelen. The CLAM proof planner, user manual and programmer manual. Technical Report TP-4, Department of Artificial Intelligence, The University of Edinburgh, 1989.

[INK] The inductive theorem prover INKA. `http://www.dfki.de/vse/systems/inka/`.

[Jac94] P. Jackson. The Nuprl proof development system, version 4.1 reference manual and users's guide. Technical report, Cornell University, Department of Computer Science, 1994.
See also `http://www.cs.cornell.edu/Info/Projects/NuPrl/`.

[Jac95] P. Jackson. *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra.* PhD thesis, Cornell University, 1995.

[KMM00] M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
See also `http://www.cs.utexas.edu/users/moore/acl2/`.

[KP99] F. Kammüller and L. C. Paulson. A formal proof of Sylow's theorem. *Journal of Automated Reasoning*, 23, 1999.

[Miz] Mizar project. `http://www.mizar.org`.

[PG96] L. C. Paulson and K. Grabczewski. Mechanizing set theory. *Journal of Automated Reasoning*, 17:291–323, 1996.

[RST01] P. Rudnicki, C. Schwarzweller, and A. Trybulec. Commutative algebra in the Mizar system. *Journal of Symbolic Computation*, 32(1/2):143–169, 2001.

[The] Theorema project. `http://www.theorema.org`.

[The01] L. Thery. A machine-checked implementation of Buchberger's algorithm. *Journal of Automated Reasoning*, 26(2):107–137, 2001.

[TPS] Theorem proving system. `http://gtps.math.cmu.edu/tps.html`.

[TPT] The TPTP problem library for automated theorem proving.
`http://www.cs.miami.edu/ tptp/`.

[Wal92] C. Walther. *Mechanizing Mathematical Induction.* Handbook of Logic in Artificial Intelligence and Logic Programming. Oxford University Press, 1992.

[Wen] M. Wenzel. Isabelle/Isar reference manual.
`http://isabelle.in.tum.de/doc/isar-ref.pdf`.

[Yu90] Y. Yu. Computer proofs in group theory. *Journal of Automated Reasoning*, 6:251–286, 1990.

A Definition of Mathematical Concepts in Isabelle/Isar

group : record $'a$ groupSig =

```
        carrier :: "'a set"
        add :: "['a, 'a] ⇒ 'a"
        inverse :: "'a ⇒ 'a"
        zero :: "'a"
  constdefs AGroup :: "('a, 'more) groupSig_scheme ⇒ bool"
    "AGroup G == add_G ∈ carrier(G) → carrier(G) → carrier(G)
        & inverse_G ∈ carrier(G) → carrier(G) & zero_G ∈ carrier(G)
        & (∀x ∈ carrier(G). ∀y ∈ carrier(G). ∀z ∈ carrier(G).
            (add_G (inverse_G x) x = zero_G) & (add_G (zero_G) x = x)
            & (add_G x y = add_G y x)
            & (add_G (add_G x y) z = add_G x (add_G y z)) )"
```

subgroup :

```
  constdefs
    Subgroup :: "('a, 'more) groupSig_scheme ⇒ 'a set ⇒ bool"
    "Subgroup G H == H ⊆ carrier(G) & zero_G ∈ H
        & (∀x ∈ H. ∀y ∈ H. (add_G x y ∈ H) & (inverse_G x ∈ H) )"
```

ring : Ring$\mathcal{R}$ == $\mathcal{R}$ is a ring.

```
  record 'a ringSig = "'a groupSig" +
    mul :: "['a, 'a] → 'a"
    one :: 'a

  constdefs Ring :: "'a ringSig ⇒ bool"
    "Ring R == AGroup R & (one_R) ∈ carrier(R)
        & (mul_R ) ∈ carrier(R) → carrier(R) → carrier(R)
        & (∀x ∈ carrier(R). ∀y ∈ carrier(R). ∀z ∈ carrier(R).
            (mul_R (mul_R x y) z = mul_R x (mul_R y z))
            & (mul_R (add_R x y) z = add_R (mul_R x z) (mul_R y z))
            & (mul_R x (add_R y z) = add_R (mul_R x y) (mul_R x z))
            & (mul_R (one_R) x = x) & (mul_R x (one_R) = x)
            & (mul_R x y = mul_R y x)    )"
```

ideal : Ideal$_{\mathcal{R}}A$ == A is an ideal of a ring $\mathcal{R}$.

```
  constdefs Ideal :: "'a ringSig ⇒ 'a set ⇒ bool"
    "Ideal R I == RingR & Subgroup_R I & I ≠ {}
        & (∀x ∈ carrier(R). ∀y ∈ I. mul_R x y ∈ I)"
```

prime ideal : PrimeIdeal$_{\mathcal{R}}p$ == p is a prime ideal of a ring $\mathcal{R}$.

```
  constdefs PrimeIdeal :: "'a ringSig ⇒ 'a set ⇒ bool"
    "PrimeIdeal R I == Ideal_R I & I ≠ carrier(R)
        & (∀x ∈ carrier(R). ∀y ∈ carrier(R).
            (mul_R x y ∈ I) ⟶ (x ∈ I ∨ y ∈ I))"
```

product : nmul0$_{\mathcal{R}}(f,n)$ == $\prod_{i\leq n} f_i$, where the multiplication is performed over a ring $\mathcal{R}$.

```
  consts  nmul0 :: "['a, (nat ringSig ⇒ 'a), nat] ⇒ 'a"
  primrec nmul0_0:   "nmul0_R(f, 0) = one_R"
          nmul0_suc: "nmul0_R(f, n + 1) = mul_R (f  (n + 1)) nmul0_R(f, n)"
```

Automated Theory Formation for Tutoring Tasks in Pure Mathematics

Simon Colton,[1] Roy McCasland[1], Alan Bundy[1] and Toby Walsh[2]

[1]Division of Informatics
University of Edinburgh
United Kingdom
simonco@dai.ed.ac.uk, rmccasla@dai.ed.ac.uk
bundy@dai.ed.ac.uk

[2]Cork Constraint Computation Centre
University College Cork
Ireland
tw@4c.ucc.ie

Abstract. The HR program forms mathematical theories from as little information as the axioms of a domain. The theories include concepts with examples and definitions, conjectures, theorems and proofs. Moreover, HR uses third party mathematical software including automated theorem provers and model generators. We suggest that a potential role for theory formation systems such as HR is as an aid to mathematics lecturers. We discuss an application of HR to the generation of a set of group theory exercises. This forms part of a project using HR to make discoveries in Zariski spaces, which is also detailed.

1 Introduction

At primary and secondary school levels, there are programs which generate exercise sheets automatically as an aid to mathematics teachers and students. In these cases, the emphasis is on asking the student to perform computations. At the university level, more emphasis is placed on using deduction in exercises. Hence, there is much potential for automated reasoning and other tools to be used in mathematics education, in particular for generating tutorial exercises.

We discuss here a project which couples a research mathematician with various pieces of mathematical software. In particular, the project aims to employ the HR automated theory formation system [3] to produce novel results in the domain of Zariski spaces [12]. This project, while still in its early stages, has produced both some promising preliminary results and an insight into the usage of HR by a research mathematician. In §3 we discuss this project and the preliminary results from it, and in §4 we explain the improvements to HR which have been implemented in response to the user's suggestions. These improvements have enabled an application of HR not to a discovery task, but rather to a tutoring task: the aiding of a mathematician in producing a set of exercises in group theory. This application is discussed in §5.

2 The HR Program

The HR program (named after mathematicians Hardy and Ramanujan) performs automated theory formation in domains of pure mathematics such as number theory, graph theory, and finite algebras, such as group theory and ring theory. The initial information about a domain supplied to HR includes the axioms of the domain and optionally some initial concepts (e.g., multiplication and addition in number theory) and some objects of interest (e.g., some integers in number theory, groups in group theory, etc.). The concepts are supplied with both a definition and some examples (e.g., triples of integers related by multiplication). In finite algebraic domains, HR can start with just the axioms of the theory, as it extracts initial concepts from these, e.g., given the identity axiom in group theory, HR extracts the concept of identity elements.

HR operates by performing a theory formation step that attempts to invent a new concept from one (or two) old ones. Concept formation is facilitated by a set of general production rules that generate both a definition and set of examples for the new concept, from the definition and examples of the old concept(s) respectively. The *complexity* of a concept is measured as the number of production rule steps which were used to construct the concept. Similarly, the *applicability* of a concept is measured as the proportion of objects of interest in the theory for which the concept has a non-empty set of examples.

The production rules are described in detail in [3], [6] and [7]. In summary, these rules perform the following constructions:

- Compose rule: uses conjunction to join the definitions of two previous concepts
- Disjunct rule: uses disjunction to join the definitions of two previous concepts
- Equals rule: imposes equality in a concept's definition
- Exists rule: introduces existential quantification
- Forall rule: introduces universal quantification
- Match rule: equates variables in a concept's definition
- Negate rule: negates predicates in a concept's definition
- Size rule: counts subobjects
- Split rule: instantiates variables in a concept's definition

Figure 1 portrays how HR constructs the concept of central elements in group theory using the compose, exists and forall production rules.

A theory formation step will lead to either: (a) a concept that has no examples, (b) a concept that has exactly the same examples as a previous concept, or (c) a concept that has non-trivial examples that differ from those of all previously existing concepts. In the first case, there may be no examples for the concept because of the lack of data given to HR, or it may be because the definition of the concept is inconsistent with the axioms of the domain. Hence, HR makes a conjecture that no examples of the concept exist. In the second case, HR makes an if-and-only-if conjecture, stating that the definitions of the new concept and

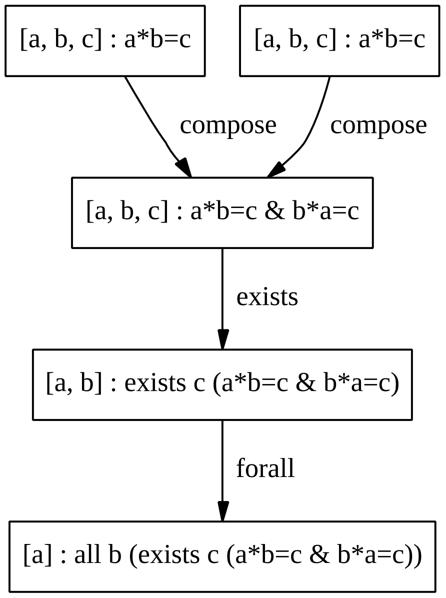

[a, b, c] : a*b=c
[a, b, c] : a*b=c
compose
compose
[a, b, c] : a*b=c & b*a=c
exists
[a, b] : exists c (a*b=c & b*a=c)
forall
[a] : all b (exists c (a*b=c & b*a=c))

Fig. 1. Construction of the concept of central elements in group theory

the previous one are equivalent. In the last case, the concept is simply added to the theory.

When HR makes conjectures in finite algebraic domains, it can invoke the Otter theorem prover [13] to attempt to prove the conjecture. If Otter fails, HR invokes the MACE model generator [14] to attempt to find a counterexample. If neither Otter nor MACE are successful, then the conjecture remains open. In cases where Otter proves an equivalence theorem, HR breaks this into a set of implication theorems, where a set of premise predicates imply a single goal predicate. Furthermore, HR extracts prime implicates from each implication theorem, i.e., it takes ever-larger subsets of the premises from the implication theorem and sees whether Otter can prove that they imply the goal.

For instance, if it made the conjecture that

$$A \wedge B \wedge C \leftrightarrow D \wedge E \wedge F,$$

then HR would extract implicates such as these:

$$A \wedge B \wedge C \rightarrow D$$

$$D \wedge E \wedge F \rightarrow A$$

etc. From the first of these, HR would attempt to extract prime implicates in this order:

$$A \rightarrow D,\ \ B \rightarrow D,\ \ C \rightarrow D$$
$$A \wedge B \rightarrow D,\ \ A \wedge C \rightarrow D,\ \ B \wedge C \rightarrow D$$
$$A \wedge B \wedge C \rightarrow D$$

stopping when it found one that Otter could prove. For more details of how HR makes conjectures, see [4].

3 The Zariski Spaces Project

Zariski Spaces were introduced in 1998 [12]. In order to understand these spaces, one needs to first understand Zariski Topologies. In a broad sense, these topologies are rather like prime factorizations. For example, the Zariski topology associated with the ring of integers consists of sets (called *varieties*) of prime ideals, one set for each integer. In particular, the variety of the integer 12 would be the set of ideals generated by 2 and by 3, respectively, since 2 and 3 are the prime factors of 12. Note that since 2 and 3 both divide 12, then the ideal generated by 2 and the ideal generated by 3 both contain the ideal generated by 12.

In the general case, let R be a commutative ring with unity, and let *specR* denote the collection of prime ideals of R. Now for each (possibly empty) subset A of R, let the *variety* of A be given by: $V(A) = \{P \in specR : A \subseteq P\}$. It is easily shown that the collection of all such varieties constitutes (the closed sets of) a topology, called the Zariski Topology on R, which we denote by $\zeta(R)$. It turns out that every topology is a semiring, if one takes the operations of addition and multiplication to be set-theoretic intersection and union respectively. Now let M be an R-module, and repeat the above process. That is to say, let *specM* denote the collection of all prime submodules of M, and for each subset B of M, let the variety of B be given by: $V(B) = \{P \in specM : B \subseteq P\}$. Finally, let $\zeta(M)$ represent the collection of all varieties of subsets of M. Then one can show that while $\zeta(M)$ seldom forms a topology, it does form a semimodule over the semiring $\zeta(R)$, where the operation in $\zeta(M)$ is taken to be intersection, and scalar multiplication is given by $V(A)V(B) = V(RAB)$.

There are reasons to suppose that Zariski Spaces might turn out to be of considerable importance in mathematics. For instance, Zariski Topologies and the study of varieties have played an important role in the development of Algebraic Geometry, and in particular, the Hilbert Nullstellensatz, which is one of the fundamental results in Algebraic Geometry. It is certainly possible that Zariski Spaces could have a similar impact on some branch of mathematics. Furthermore, some preliminary results suggest a possible connection between a certain concept in Zariski Spaces, called subtractive bases, and direct sum decompositions within a large class of modules. The search for direct sum decompositions has been a major undertaking in mathematics for some time, and has so far proven quite intractable, except in special cases. The study of semimodules in general has already yielded many applications to computer science [10], and since Zariski Spaces are first and foremost semimodules, it is possible that their study will promote further advances in theoretical computer science.

3.1 Proposed Discovery Methods

At present, HR is mostly autonomous, i.e., the user sets some parameters for the search it will perform, then HR builds a theory and the user employs various tools to extract information about the theory. We propose to extend the theory of theory formation upon which HR is based by enabling any of the functionality of HR to be replaced on occasion by human intervention. That is, we intend to make HR semi-automated by allowing the user to provide proofs and counterexamples to conjectures HR makes and to specify related concepts and conjectures to base the theory formation around. Alongside the development of HR's functionality, we also intend to develop the application to Zariski spaces. Zariski spaces represent a higher level of complexity than the domains in which HR has so far been applied, and the new application will require an incremental approach whereby (i) HR is enabled to form theories about increasingly complicated domains related to Zariski spaces (ii) testing is performed to see if HR invents various concepts and conjectures required for it to proceed and (iii) theory formation is centred around the important concepts and the results analysed for any discoveries. The proposed route to Zariski spaces is via: semigroups, semirings and semimodules, followed by groups, rings and modules and finally, using a cross domain approach, Zariski spaces.

It is unclear at the moment how HR will interact with the user and with third party pieces of mathematical software on this project. It seems likely that HR will be used for an initial exploration of each domain, to: (a) invent core concepts (b) prove some fundamental theorems using a theorem prover (HR has access to Otter, E, Spass and Bliksem through the MathWeb Software Bus [9]), and (c) generate some examples of the concepts using a model generator or computer algebra system (HR uses MACE and Maple, also possible through MathWeb). Following the initial investigation, the user will both prune uninteresting concepts and specify which concepts should be emphasised in the next theory formation session. The user will be more involved in that session, choosing to prove theorems, provide counterexamples and direct the search where appropriate.

3.2 Preliminary Results

Our approach is to employ HR in successively more complex domains, eventually arriving at Zariski spaces. Along the way, we may find interesting results in the less complex domains. A particularly interesting result which was previously unknown to us occurred in the semigroup domain, which are algebras with a single associative operation. HR produced the conjecture, which Otter then proved, that: $c * c = c \;\&\; \exists\; d\; (c * d = b) \rightarrow c * b = b$. Paraphrased, this states that, given an idempotent element c, then if any element b appears in any column in the "c" row of the Cayley table of the given semigroup, then that same element b must also appear in the "c" row *in its own column.* Admittedly, this result appeared to us at first glance to be untrue, but upon reflection, the proof was actually fairly obvious. This result is of some interest to us, because Zariski Spaces are a rather complicated algebraic structure, but at their simplest level, they are a special kind of semigroup. Moreover, every element in these spaces is an idempotent element, and therefore, this theorem is applicable to every element.

4 Additions to HR Arising from the Project

Each new application of HR necessitates some additional functionality, and the application to Zariski spaces is no exception. Firstly, we have implemented a new production rule so that HR can check whether particular concepts exhibit certain algebraic properties. Secondly, we have introduced a reaction mechanism so that we can tailor HR's search to react to a particular event, such as the introduction of a concept with a particular property. Thirdly, we have enabled HR to use Knuth-Bendix completion so that it can identify (and discard) a range of conjectures which are uninteresting. These additions are described in more detail below.

4.1 The Embed-Algebra Production Rule

All of HR's conjectures discuss concepts which it has invented, and HR uses only a small number of production rules to generate new concepts. Hence the implementation of a new production rule represents a major addition to HR's functionality. Each production rule must:

• Determine the ways in which it can be used to produce new concepts from a given set of old concepts, in terms of the parameterisations of the rule possible for the old concept(s).

• Produce a complete set of examples for the new concept for each object of interest (group, graph, integer, etc.) in the theory.

• Produce a definition for the concept which is consistent with the examples.

Drawing on work by Graham Steel [16] [17] into cross domain theory formation, we implemented the 'embed- algebra' production rule, which aims to find algebraic structures embedded in old concepts. Any concept which is a predicate of arity 4, where the first entry is an object of interest and the following three entries are of the same type, can be tested for various algebraic structures. For instance, if HR invented a concept with a predicate definition $P(a, b, c, d)$ such that b, c and d were subobjects of a of the same type, then the embed-algebra rule could be invoked to check whether the function $f(b, c) = d$ had particular algebraic properties, such as commutativity, associativity, an identity, etc.

This production rule differs from all the others in that it relies on a third party program, MACE, to generate the examples of the new concept. Given an old concept C of arity 4, MACE is used to check whether the set of triples for each object of interest that C applies to satisfies a set of axioms chosen by the user. For example, suppose that concept C had the datatable as in table 1. In this case, the embed-algebra rule takes each object of interest o in turn and collects all ground instances of C as examples of the operation upon which the axioms are to act. For instance, HR takes object o_2 and collects the following four instances of the operation upon which the axioms are going to be checked:

$$t * t = t, \quad t * u = u, \quad u * t = u, \quad u * u = t$$

HR first normalises the elements so that they are represented by the numbers 0, 1, 2, etc. It then checks that it has not seen this (possible) algebra before, and if it has, uses the results of the previous calculation.

object of interest	subobject 1	subobject 2	subobject 3
o_1	s	s	s
o_2	t	t	t
o_2	t	u	u
o_2	u	t	u
o_2	u	u	t
o_3	a	a	b
o_3	a	b	b
o_3	b	a	b
o_3	b	b	b
o_4	x	z	z
o_4	x	y	x
o_4	y	x	y

Table 1. Example datatable input to the embed-algebra production rule

If the set is new, HR then checks whether or not this operation is closed, by checking that every element which appears as the product of an operation also appears as both the left hand and right hand element in a product and that the examples constitute a Cayley table. If the operation passes this check, then MACE is invoked to check whether the operation satisfies the axioms chosen by the user. To use MACE in this manner, both the axioms and all the ground instances of the operation are passed to MACE. Hence, for o_2 above, if the user chose the group theory axioms, MACE would be passed the following file:

```
set(auto).
formula_list(usable).
0*0=0.
0*1=1.
1*0=1.
1*1=0.
all a (a*id = a & id*a = a).            \\ identity
all a (a*inv(a) = id & inv(a)*a = id).  \\ inverse
all a b c ((a*b)*c = a*(b*c)).          \\ associativity
end_of_list.
```

If MACE responds that it has found a model, then this effectively states that the model HR has supplied satisfies the axioms chosen. Hence, the operation (and thus the original concept) embeds an algebra for the particular object of interest under consideration.

In the current example, if the user had chosen the axioms for group theory, then the operation for the elements of o_1 and o_2 would both be closed and satisfy the axioms, but the operation for o_4 is not closed (z does not appear in the product anywhere) and the operation for o_3 does not have an identity element, which is required for the group theory axioms. To construct the datatable for the new concept, for each object of interest for which the operation does embed an algebra, HR first checks whether any algebra it has already is an isomorphism of the embedded algebra, and substitutes the previous one if so. Then the datatable is constructed with the objects of interest in the first column and the algebra embedded in the second column. If the embedded operation does not satisfy the axioms of the algebra, then an empty set is put in the second column.

If the concept which was input to this production rule had concept number F, and the algebra chosen to check for embedding was Alg, then the definition for the new concept would be generated as:

$$[a, b] : \text{concept } F \text{ forms } Alg\ b \text{ for } a$$

for instance:

$$[a, b] : \text{concept 17 forms group } b \text{ for } a$$

Note that the user can specify that HR checks for various algebras using this rule. These choices comprise the different parameterisations of the embed-algebra production rule that HR will carry out.

4.2 Reaction Mechanism

A motivating example for the embed-algebra production rule was for HR to find the centre of groups, not just as a subset of elements, but also as a subgroup. However, as we see from figure 1 above, HR invents the concept of a central element as an element type, not as a concept of arity four in which an operation could be embedded. It is not controversial to state that, if a mathematician invents a new type of element in an algebra like group theory, one of the first things they might try is to see whether the set of elements forms a subgroup for all groups. We wanted to model this kind of reaction to a new element type in HR, and – to cover the more general case – we implemented a 'reactive' search in HR. Such a search is similar to the way in which certain "demons" are invoked when particular slots are filled in frame representations of concepts [2].

The reactive search in HR is determined beforehand by the user, who supplies pieces of pseudo-java code to HR that specify what HR should do if certain things occur during its normal theory formation. Using Java's reflection mechanism, HR translates these pieces of code (which we call 'reactions') into Java code at runtime, and the conditions for them are checked whenever a new theory formation step is completed; a new concept is introduced; a new conjecture is made or a new counterexample is found.

For instance, the reaction script in figure 2 reacts to a new concept being introduced. With this reaction activated, for each new concept, HR first checks whether the applicability[1] of the new concept is greater than 0.8. If so, HR then adds two steps to the agenda, the first of which performs a step using the exists production rule on the new concept, and the second of which performs a step using the size production rule on the concept produced by the first step (n1). Note that if any step results in a previous concept, this is substituted in further steps added to the agenda. Finally, this reaction tells HR to mark the concept produced by the second step (n2) as interesting, so that the user can find all such concepts in HR's output.

```
react_to(concept)
condition(concept, applicability > 0.8)
condition(concept, arity == 3)
action(add_to_agenda, n1 = concept exists [2])
action(add_to_agenda, n2 = n1 size [1])
action(mark_concept, n2, interesting)
```

Fig. 2. Sample reaction script

HR's reaction mechanism is still under development. As discussed in §5 below, we use the reaction mechanism to force HR to check immediately whether a new type of element forms a subgroup in a group theory session. This is a generalisation of work done with Alison Pease [15] on getting HR to react in ways prescribed by Lakatos in [11].

4.3 Knuth-Bendix Completion to Discard Conjectures

A problem we have come across throughout the development of HR is that it produces too many conjectures of a trivial nature, as lamented in chapter 11 of [3]. We have previously relied on Otter's proof length statistic to prune conjectures which Otter found easy to prove, but we have recently found this approach can discard fairly interesting conjectures, and so is not appropriate.

Instead, to reduce the number of trivial results produced by HR, one can use a set of rewrite rules as a filter. Otter can, in some cases, be used to produce a set of rewrite rules (demodulators) for certain collections of axioms. In particular, we can use the Knuth-Bendix option in Otter to derive a list of equations from the axioms of an algebra, and these equations form a complete set of rewrite rules for the given axioms. To be more specific, in group theory, we give Otter

[1] See [8] for a description of HR's measures of interestingness, including the applicability of a concept.

the group axioms of left identity, left inverse and associativity, and Otter outputs the following list of 10 equations:

```
id*x=x.            inv(x)*x=id.     (x*y)*z=x*y*z.
inv(x)*x*y=y.      x*id=x.          inv(id)=id.
inv(inv(x))=x.     x*inv(x)*y=y.    x*inv(x)=id.
inv(x*y)=inv(y)*inv(x).
```

On inspection, this list includes the right-hand versions of identity and inverse, as well as these facts: the identity is its own inverse, the inverse composed with itself is the identity function, and the inverse of a product is the product of the inverses in reverse order. Having obtained such a list of equations, HR take each prime implicate it produces and sends the premises of that conjecture to Otter, which is told to work in Knuth-Bendix mode. Otter then uses its demodulator and paramodulator functions to simplify the premises. Sometimes, this rewriting process includes in its output the goal of the original conjecture. In these cases, HR discards the conjecture as it is considered trivial.

As an example, HR produced the following conjecture in a group theory session: $b * c = d \& b = id \rightarrow b * d = c$. This is a true statement in group theory. When we gave the premises of this conjecture to Otter in Knuth-Bendix mode, it produced several conclusions, including the one conjectured here. It should be noted that a human would typically arrive at the conclusion by first observing that since $b = id$, the first premise yields $c = d$. At this point, the conclusion would be obvious. While one can perhaps argue that this thought process is not an entirely straightforward one, and that therefore the conjecture has some merit, the result is not sufficiently important to be included in the theory, and therefore a human mathematician would probably discard it. Thus, HR's decision to discard the conjecture matches that of human mathematicians.

To test this new method, we ran HR for 1000 theory formation steps and it produced 281 prime implicates, of which 40 (14%) were discarded because Knuth Bendix completion found the goal from the premises. For example, HR discarded the conjecture $b*c = d \& c*b = d \& d*b = c \rightarrow b*d = c$, because, using the demodulators extracted from the axioms of group theory, Otter's Knuth-Bendix mode managed to re-write the premises to include the goal. We are currently working to get Otter to discard more conjectures of a similarly trivial nature. This is similar to our efforts described in [5], where we ask HR to discard conjectures about Maple [1] functions if Otter is able to prove them, as these will most likely be trivially true.

5 Application to Setting Exercises

The second author of this paper has much experience in mathematics education at the university level, and it was our aim to see if HR alongside Otter and MACE could aid him in the setting of tutorial questions for an undergraduate group theory course. Based on the observation that many such tutorial questions ask the student to show that a particular set of elements form a subgroup, we decided that HR's embed algebra production rule should be used to identify subgroup types. To facilitate this, we wrote a reaction script which ensured that, whenever HR invented a new type of element, it formed the subgroup (where possible) of those elements, and flagged those to the user for which: (a) the subset always formed a subgroup and (b) the subgroup generated was neither the trivial group with one element nor the entire parent group for at least one of the groups in HR's database. The reaction script in figure 3 achieved this functionality.

```
react_to(concept)
condition(concept,types.toString() == [group, element])
action(add_to_agenda,n1 = group003 concept compose [1,2,0,0])
action(add_to_agenda,n2 = n1 concept compose [1,0,2,0])
action(add_to_agenda,n3 = n2 embed_algebra [group])
condition(n3,applicability == 1)
action(add_to_agenda,n4 = n3 match [0,0])
action(add_to_agenda,n5 = n3 split [[1],[0]])
condition(n4,applicability < 1)
condition(n5,applicability < 1)
action(mark_concept,n3,good_subgroup)
action(mark_concept,concept,good_element_type)
```

Fig. 3. Reaction script used for subgroup types

We gave HR the groups up to order 8 to use as data for the empirical conjectures that it made, and formed a theory using 10,000 theory formation steps. We used the compose, exists, match and equals production rules in a breadth first manner, and controlled the use of the forall production rule via another reaction script. This forced HR to use the forall rule only with concepts x which relate two elements, to invent the concept of elements for which all the other elements are related to x. We have found that normal usage of the forall production rule often leads to many complicated – but generally uninteresting – concepts in group theory, so we controlled it with a reaction script instead.

The user was allowed to write any tutorial questions, as long as they were inspired in some way by something from HR's theory. On reflection of how HR's subgroup concepts were being used to generate tutorial questions, we identified the following possibilities:

• For those element types, C, which the user can prove (possibly using Otter) always form a subgroup, the teacher may set the exercise: "Prove that elements with property C form a subgroup".

• For those element types, C, for which HR has found groups where they don't form a subgroup, and, indeed, for those where such a group is later found by MACE, the teacher could ask the following question: "Characterise those groups for which the set of elements with property C form a subgroup." Note, however, that characterisation problems can often be very difficult, and sometimes of little interest to mathematics.

• For those element types where the smallest group for which the elements do not form a subgroup is fairly large (i.e., beyond the reach of simple computation), the teacher could ask the following: “Find the smallest group for which the set of elements with property C do not form a subgroup”. As the answer is large, the student will be forced to use their knowledge of group theory to answer the question, because a calculation would not easily yield the answer. Note that questions of this type may also be difficult to answer.

In addition, the mathematician may choose to simplify the questions by restricting the range of groups to which a question applies. For example, they may ask: “Show that, for Abelian/cyclic/dihedral groups, the set of elements with property C form a subgroup”. Finally, other questions may arise as part of this process, and HR may be able to inspire such questions, with MACE and Otter possibly providing answers to such questions.

5.1 Results

We ran HR on a 2.0 Ghz. Pentium 4, for 10000 theory formation steps, giving as input the datatables for all groups of order 8 or less. The run took 301 seconds and produced a total of 330 concepts. Of these concepts, 17 were element types which produced subgroups for at least some of the initial groups. Of these 17, 10 concepts produced subgroups for all of the initial groups. Of these last 10, 8 concepts produced at least one non-trivial subgroup; that is, a subgroup which was neither the identity subgroup, nor the group itself. The concept definitions of this last type were as follows:

```
g43. [a, b] : exists c (c*c=b)
g93. [a, b] : all c (exists d (d*c=b & c*d=b))
g102. [a, b] : exists c d (b*c=d & d*b=c)
g110. [a, b] : all c (b*(c*b)=c)
g124. [a, b] : exists c (c*(c*b)=b & inv(c*b)=(c*b))
g224. [a, b] : inv(b)=b & (exists c (c*c=b))
g282. [a, b] : exists c (inv(c)=b & c*c=b)
g307. [a, b] : exists c ((c*c)*(c*c)=b)
```

The remaining 9 concepts produced subgroups for some, but not all of the initial groups.

```
g10. [a, b] : all c (inv(c)=b)
g16. [a, b] : all c (c*c=b))
g29. [a, b] : inv(b)=b
g67. [a, b] : exists c d (c*d=b & (all e (e*e=c)))
g198. [a, b] : all c ((exists d (d*c=b)) & (exists e (e*e=c)))
g202. [a, b] : all c (c*b=c & (exists d (d*d=c)))
```

```
g262. [a, b] : all c (exists d (d*c=b & d*d=c))
g267. [a, b] : exists c (c*(c*c)=b)
g303. [a, b] : all c ((c*c)*(c*c)=b)
```

In the course of setting the exercises, we looked at the first two concepts (g43 and g93) in more detail. Concept g43 simply defines the set of all elements which appear on the diagonal of the Cayley Table of the group. This concept gives a subgroup for each of the groups of order 8 or less, as demonstrated by HR using MACE during the theory formation. However, we were fairly certain that this construction does not always yield a subgroup. We therefore asked MACE to generate a counterexample, by asking it to find a group for which there are two elements on the diagonal which multiply to give an element which is not on the diagonal. If it could find one, then the subset would not be closed under multiplication and hence not a subgroup. We were unsuccessful in finding a counterexample up to order 11, but at order 12, MACE took 142.84 seconds of cpu time to generate the non-Abelian group in figure 4. We see that elements $0, 1, 2$ and 4 appear on the diagonal, but the product of 1 and 2 is 3 which is not on the diagonal. Hence concept g43 does not form a subgroup for this group.

```
 * |  0  1  2  3  4  5  6  7  8  9 10 11
 --+-----------------------------------
 0 |  0  1  2  3  4  5  6  7  8  9 10 11
 1 |  1  0  3  2  5  4  7  6  9  8 11 10
 2 |  2  3  4  5  0  1  8  9 10 11  6  7
 3 |  3  2  5  4  1  0  9  8 11 10  7  6
 4 |  4  5  0  1  2  3 10 11  6  7  8  9
 5 |  5  4  1  0  3  2 11 10  7  6  9  8
 6 |  6  7 10 11  8  9  1  0  5  4  3  2
 7 |  7  6 11 10  9  8  0  1  4  5  2  3
 8 |  8  9  6  7 10 11  3  2  1  0  5  4
 9 |  9  8  7  6 11 10  2  3  0  1  4  5
10 | 10 11  8  9  6  7  5  4  3  2  1  0
11 | 11 10  9  8  7  6  4  5  2  3  0  1
```

Fig. 4. Group produced by MACE disproving subgroup closure property

It is perhaps not entirely obvious that the second concept in the top list, g93, actually generates the centre of a group. To see this, first solve both equations for d, to give $d = b{*}inv(c)$ and $d = inv(c){*}b$, or more succinctly, $b{*}inv(c) = inv(c){*}b$. Now note that since the concept applies to all elements c in the group a, we can substitute $inv(c)$ for c, and it becomes clear that the concept is equivalent to: $[a, b] : \forall\ c\ (b * c = c * b)$. We asked Otter to prove that this set is a subgroup, by giving it the concept and the group axioms, and then having it prove, each in turn, that the concept definition is closed with respect to the products, the inverses and the identity. That is, we ran Otter three times, first with the group axioms and these two additional lines:

```
all a (f(a) <->  (all b (((exists c (c*b=a & b*c=a)))))).
-(all a b (f(a) & f(b) -> f(a*b))).
```

[Note that the goal is negated as Otter is a resolution prover]. This proved that the subset is closed under multiplication. Secondly, we ran Otter with the group axioms and these two additional lines:

```
all a (f(a) <->  (all b (((exists c (c*b=a & b*c=a)))))).
-(all a (f(a) -> f(inv(a)))).
```

This proved that the inverse axiom applies to the subset. Finally, we ran Otter with the group axioms and these two additional lines:

```
all a (f(a) <->  (all b (((exists c (c*b=a & b*c=a)))))).
-(exists a (f(a) & (all b (f(b) -> (a * b = b & b * a = b))))).
```

This proved that there is an identity element in the subset. Given that the parent multiplication operation was inherited by the subset, we had no need to prove associativity in the subgroup. Hence, with the three proofs completed, we had shown that the centre of a group forms a subgroup. The statistics Otter used in this way are given in table 2.

Axiom	Statistic	Value
Identity	Proof Length	1
	Proof level	1
	Cpu time	0.20 seconds
Inverse	Proof Length	36
	Proof level	10
	Cpu time	24.67 seconds
Closure	Proof Length	59
	Proof level	14
	Cpu time	54.55 seconds

Table 2. Otter statistics from proving that central elements form a subgroup

Having run HR and explored the results as described above, the mathematician then set the exercises, with the resulting exercise sheet given in appendix A. It is difficult to assess the success of this work in objective terms, and we have included in appendix A a brief summary of how the exercises were influenced by HR's theory. We hope that this demonstrate that HR, Otter and MACE contributed in a non-trivial way to the setting of the exercises, and that there is much potential for similar use in future.

6 Conclusions and Future Work

We have demonstrated that the HR theory formation program, in conjunction with the Otter theorem prover and MACE model generator can be used to help write tutorial questions for mathematics students. The programs acted in the role of an aid to a mathematics teacher, rather than as an autonomous mathematician. In particular, HR was used to empirically suggest subgroup constructions, and HR called MACE as part of this process. Furthermore, Otter and MACE were both used to check whether certain element types form a subgroup in the general case. We documented a case where Otter proved that a subset of elements always forms a group and a case where MACE showed that another subset of elements doesn't necessarily form a group (as HR had suggested based on the evidence of the groups up to order 8).

To undertake this application, we implemented two new functionalities, namely a reactive search, where HR's search can be interrupted in response to certain events, and a new production rule which identifies when certain algebras are embedded in old concepts. We believe that a sustained application of HR in this manner to other algebras could yield further interesting tutorial questions. However, the main aim of this project is to use HR to discover new results in the domain of Zariski spaces, and we are already getting positive results and feedback from this project. To succeed further in this project, our future work will centre around:

• The implementation of more user interface functionality, so that the user can act themselves as the concept formation/conjecture making/theorem proving/counterexample finding process.

• Storage of knowledge between sessions. This will be important in order to (i) reduce the time spent by the mathematician sifting through results he/she has already come across and (ii) import and apply in a new context important results derived during previous sessions.

• Further exploration of Zariski spaces. This will proceed as planned – HR will be used in successively more complicated domains until eventually able to work with Zariski spaces.

• Further pruning of uninteresting conjectures. This will be achieved by using Knuth-Bendix completion more fully and via a variety of other techniques, some using automated theorem provers such as Otter.

We believe that for automated deduction, the role of an aid for mathematics lecturers is a more plausible prospect in the short term than the role of a discoverer of important new proofs. However, such an application needs to be fuelled by automated theory formation in order to identify interesting results that the lecturer would possibly miss otherwise.

Acknowledgments

This work is supported by EPSRC grants GR/M98012 and GR/R84559/01. We would like to thank Derek Sleeman for his interesting comments about this work. We would also like to thank Alison Pease for her input to the reaction mechanism and other aspects of HR's development. Simon Colton is also affiliated with the Department of Computer Science at the University of York.

References

1. M. Abell and J. Braselton. *Maple V by Example.* Associated Press Professional, 1994.
2. A. Barr, P. Cohen, and E. Feigenbaum, editors. *The Handbook of Artificial Intelligence, IV.* Addison-Wesley, 1989.
3. S. Colton. *Automated Theory Formation in Pure Mathematics.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 2000.
4. S. Colton. The HR program for theorem generation. In *Proceedings of the Eighteenth Conference on Automated Deduction*, 2002.
5. S. Colton. Making conjectures about maple functions. In *Proceedings of the Tenth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, 2002.
6. S. Colton, A. Bundy, and T. Walsh. HR: Automatic concept formation in pure mathematics. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1999.
7. S. Colton, A. Bundy, and T. Walsh. Automatic identification of mathematical concepts. In *Machine Learning: Proceedings of the Seventeenth International Conference*, 2000.
8. S. Colton, A. Bundy, and T. Walsh. On the notion of interestingness in automated mathematical discovery. *International Journal of Human Computer Studies*, 53(3):351–375, 2000.
9. A. Franke and M. Kohlhase. System description: MathWeb, an agent-based communication layer for distributed automated theorem proving. In *Proceedings of Sixteenth Conference on Automated Deduction*, pages 217–221, 1999.
10. J. Golan. *The theory of Semirings with Applications in Mathematics and Theoretical Computer Science.* John Wiley and Sons, 1992.
11. I. Lakatos. *Proofs and Refutations: The logic of mathematical discovery.* Cambridge University Press, 1976.
12. R. McCasland, M. Moore, and P. Smith. An introduction to Zariski spaces over Zariski topologies. *Rocky Mountain Journal of Mathematics*, 28:1357–1369, 1998.
13. W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.
14. W. McCune. MACE 2.0 Reference Manual and Guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA, 2001.
15. A. Pease, S. Colton, A. Smaill, and J. Lee. Lakatos and machine creativity. In *Proceedings of the ECAI Creative Systems Workshop*, 2002.
16. G. Steel. Cross domain concept formation using HR. Master's thesis, Division of Informatics, University of Edinburgh, 1999.
17. G. Steel, S. Colton, A. Bundy, and T. Walsh. Cross domain mathematical concept formation. In *Proceedings of the AISB-00 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science*, 2000.

A Group Theory Exercise Sheet

The following exercise sheet was producing using the theory HR formed as described in §5.1 above. Below, we give a brief summary of how HR's results inspired these questions.

Let G be a multiplicative group with identity 1. Problems marked ** should be considered honors exercises.

1. Let $A = \{a \in G : \text{for some } c, d \in G,\ ac = d,\ ad = c \text{ and } c^{-1} = c\}$ and let $B = \{b \in G : b^2 = 1\}$. Prove or disprove that $A = B$. Determine whether A is a subgroup of G. If so, prove it. If not, give the smallest example of G such that A is not a subgroup of G.

** 2. Let $A = \{a \in G : a = b^2 \text{ for some } b \in G\}$. Determine whether A is a subgroup of G. If A is not a subgroup, give a characterisation for the groups G for which A is a subgroup.

3. Let $A = \{a \in G : ab = ba \text{ for all } b \in G\}$. Prove or disprove that A is a subgroup of G.

4. Let $A = \{a \in G : \text{for some } c, d \in G,\ ac = d \text{ and } da = c\}$ and let $B = \{b \in G : bcb = c \text{ for some } c \in G\}$. Prove or disprove that $A = B$. Determine whether A is a subgroup of G. If so, prove it. If not, give the smallest example of G such that A is not a subgroup of G.

5. Let $A = \{a \in G : aca = c \text{ for all } c \in G\}$. Recall that if $ac = c$ for some $c \in G$, then $ac = c$ for all $c \in G$. With this in mind: Determine whether the set A is equal to B in exercise 4. Determine whether A is a subgroup of G.

6. Let $A = \{a \in G : \text{for some } x \in G,\ x^2a = a \text{ and } (xa)^{-1} = xa\}$. Let $B = \{b \in G : b = b^{-1}\}$. Let $C = \{c \in G : cyc = y \text{ for some } y \in G \text{ such that } y^2 = 1\}$. Is $B \subseteq A$? Is $A \subseteq B$? Is $A = C$? Is this A the same as the A in exercise 4? Prove your answers. Determine whether A is a subgroup of G.

7. Let $A = \{a \in G : \text{for some } c, d, f \in G,\ ac = d,\ ad = c,\ a = f^2 \text{ and } c = c^{-1}\}$ and let $B = \{b \in G : b^2 = 1 \text{ and } b = g^2 \text{ for some } g \in G\}$. Is $A = B$? How does this A compare with the A in exercise 1? Is this A a subgroup of G? Prove your answers.

8. Let $A = \{a \in G : a = b^3 \text{ for some } b \in G\}$. Determine whether A is a subgroup of G. If so, prove it. If not, give the smallest example of G such that A is not a subgroup of G.

** 9. Characterise the groups G for which every element of G can be written as a fourth power, (i.e., for all $a \in G$, there exists $b \in G$ such that $b^4 = a$).

10. Let $A = \{a \in G : a = (a^{-1})^2\}$ and let $B = \{b \in G : b^3 = 1\}$. Is $A = B$? Is A a subgroup of G? Prove your answers.

A.1 Summary of Motivations

Referring to the concept numbers as specified in §5.1, in Question 1, set B is in fact equivalent to the set defined by concept g29, since $inv(b) = b$ iff $b * b = id$. Set A came from concept g30. These two concepts are indeed equivalent, and the proof thereof makes a reasonable exercise for undergraduate students. In Question 2, set A is clearly the set defined by concept g43, which is discussed in §5.1. Set A in Question 3 comes from concept g93, also discussed in §5.1.

Set A of Question 4 comes from concept g102, and the A of Question 5 is derived from g110. The set B of Question 4 was conceived by us, merely by simplifying the definition of the corresponding set A. Having looked at these three sets, we came up with the first part of Question 5. We feel that students need to realise that, even though the existential and universal quantifiers are clearly different, nevertheless, on occasion they turn out to produce the same results. It should perhaps be pointed out that on this occasion, they of course do make a difference. It seemed to us a good opportunity to force the students to think a little, rather than merely give an instinctive reaction.

In Question 6, the set A is given by concept g124, and set B is just concept g29 again. We wanted a situation where students would find that, although the two sets were not equal, one set was contained in the other. Set C resulted from combining this latter concept with the set B of Question 4. In Question 7, set B is derived from concept g224, and set A is simply a combination of the set A of Questions 1 and 2. The sets in Questions 8 and 9 are given by concepts g267 and g303, respectively. The set A in Question 10 is from concept g282, and set B is clearly just a rewrite of the definition for set A.

On the Design of Mathematical Concepts[⋆]

Manfred Kerber[1] and Martin Pollet[1,2]

[1] School of Computer Science, The University of Birmingham
Birmingham B15 2TT, England, UK
{M.Kerber|M.Pollet}@cs.bham.ac.uk http://www.cs.bham.ac.uk/
[2] Fachbereich Informatik, Universität des Saarlandes
66041 Saarbrücken, Germany
pollet@ags.uni-sb.de http://www.ags.uni-sb.de/

Abstract. That foundational systems like first-order logic or set theory can be used to construct large parts of existing mathematics and formal reasoning is one of the deep mathematical insights. Unfortunately it has been used in the field of automated theorem proving as an argument to disregard the need for a diverse variety of representations. While design issues play a major rôle in the formation of mathematical concepts, the theorem proving community has largely neglected them. In this paper we argue that this leads not only to problems at the human computer interaction end, but that it causes severe problems at the core of the systems, namely at their representation and reasoning capabilities.

A Introduction

In the traditional theorem proving community, which we consider as our home community, a seemingly strong argument against new approaches in theorem proving has been of the type "Everything you can do in your system X, I can do in Y," where Y stands typically for standard first-order logic. This kind of thought is so strong that proponents of the conventional approach to theorem proving seem to fail to even understand why this argument – albeit it may be true – is unhelpful and misleading.

For instance, Pat Hayes said in 1974 ([11] quoted from [4, p.18]):

A more recent attack on conventional theorem-proving ... is that it is too concerned with "machine-oriented" logic, and not enough with "human oriented" logic. I confess to being quite unable to understand what this could possibly mean.

In this paper we try to clarify why the argument, although it may be technically correct, is pragmatically flawed and has led to very negative consequences. The argument is pragmatically wrong, since it is meant to say "Your system X is redundant and uninteresting, since we have system Y already, which suffices for everything you want to do." Since this argument was widely accepted the focus in the field was set much too narrow on the study of foundational systems. For other communities, our observations in this paper may be trivial.

⋆ This work was supported by European Commission IHP Calculemus Project grant HPRN-CT-2000-00102.

Of course there are exceptions and we claim in no way that we are the first to have a look at this relationship of mathematical practice and fundamental systems. In this paragraph we do not claim to give a comprehensive overview of this type of work. We just mention some work in this direction, which we think provides very important starting points to the support of the design of mathematical concepts. In AUTOMATH, N.G. de Bruijn developed the idea of a mathematical vernacular [5], which should allow to write everything mathematicians do in informal reasoning, in a computer assisted system as well. In this tradition, Hugo Elbers looked in [9] at aspects of connecting informal and formal reasoning, in particular the integration of computations in formal proofs. Francis Jeffry Pelletier [18] as well as Henk Barendregt and Arjeh Cohen [1] discuss related philosophical questions on the nature of proof. Proof planning [6] in general can be viewed as an attempt to simulate informal reasoning (and integrate it with formal reasoning). Following this paradigm, Alan Bundy made first steps towards a Science of Reasoning [7], which goes beyond a narrow focus on a particular calculus. Ursula Martin took in [15] a close look at the mathematical practice and its relationship to computer algebra and computer-assisted reasoning. Michael Beeson presented in [2] a system that combines deductive and computational reasoning steps. He proposed to use the mathematical standard for checking the correctness of proofs generated by the system, namely peer reviews.

We are not aware, however, of work in which the design process in mathematics is compared to the design possibilities of computer based mathematical support systems. In this paper, we look at design problems, which current automated reasoning systems suffer from, and relate them in the rest of this introduction to an old debate in mathematical philosophy, which summarises the relationship between foundational systems and informal systems very well.

We do not doubt that one of the deepest insights in the foundations of mathematics resulted from the epochal work by Bertrand Russell and Alfred North Whitehead. Russell articulated the idea in the *Principles of Mathematics* [21] namely, to reduce mathematics to formal logic, and carried it through together with Whitehead in the famous *Principia Mathematica* [25] – a work often quoted and seldom read. Russell was at this time also in inspiring discussions with Ludwig Wittgenstein and strongly acknowledges Wittgenstein's contribution to his thoughts. He writes in [22] (quoted from the reprint in [23, p.178]):

> *As I have attempted to prove in* The Principles of Mathematics, *when we analyse mathematics we bring it all back to logic. It all comes back to logic in the strictest and most formal sense.*

Although there is evidence that Wittgenstein shared Russell's view, he later took the opposite stance and attacked Russell's approach. In particular he discusses the important notion of proof (quoted from [26, p. 143]):

> *'A mathematical proof must be perspicuous.' ... I want to say: if you have a proof-pattern that cannot be taken in, and by a change in notation you turn it into one that can, then you are producing a proof, where there was none before.*

One of the reasons why the Principia are so rarely read is that the main ideas of the proofs are no longer visible in very long and very detailed proofs. Wittgenstein continues (p. 176f) to question the idea to try to reduce everything to a very small number of primitives (had the resolution calculus already been invented at that time his attack might have been to try to reduce everything to one single rule):

> *Mathematics is a* MOTLEY *of techniques of proof. – And upon this is based its manifold applicability and its importance. ...*
> *Now it is possible to imagine some – or all – of the proof systems of present-day mathematics as having been co-ordinated in such a way with one system, say that of Russell. So that all proofs could be carried out in this system, even though in a roundabout way. So would there then be only the single system – no longer the many? – But then it must surely be possible to shew of the* one *system that it can be resolved into the many. – One part of the system will possess the properties of trigonometry, another those of algebra, and so on. Thus one* can *say that different techniques are used in these parts.*

He continues then to counter the argument that Russell and Whitehead have constructively shown the possibility to reduce everything to one single system (p. 185)[3]:

> *If someone tries to shew that mathematics is not logic, what is he trying to shew? He is surely trying to say something like: – If tables, chairs, cupboards, etc. are swathed in enough paper, certainly they will look spherical in the end.*
> *He is not trying to shew that it is impossible that, for every mathematical proof, a Russellian proof can be constructed which (somehow) 'corresponds' to it, but rather that the acceptance of such a correspondence does not lean on logic.*

We try to exemplify in the rest of the paper why these matters are crucial for automated theorem proving systems. One important issue is that of acceptance among mathematicians. Current automated theorem provers do not find acceptance among mathematicians, while computer algebra systems do. In many computer algebra systems things are *as they should be, as an inexperienced but mathematically educated user would expect them to be.* That is, these systems are typically *well-designed.* In automated theorem proving systems they are typically *not as they should be, not as an inexperienced user would them expect to be.* We will argue that this is *not* just a deficiency of the interface, but that the problem with automated theorem provers is much deeper, it goes to the core of these systems, namely to the formal representation of mathematical knowledge and the reasoning that can be performed with this knowledge.

[3] By the way, Gödel's proof that formal systems like the Principia are necessarily incomplete is irrelevant for this argument, since not only the Principia but any other powerful system suffers from the same problems.

B Design Issues Exemplified with Multiplication Tables

In this section we want to have a close look at one particular example of a mathematical concept which is carefully designed. Before we do so, let's have a more general look at design issues. Donald Norman gives a fascinating introduction into "The Design of Everyday Things." His insights are of a very general nature and we will see that the principles for good design hold in mathematics as well.

Although Norman does not relate design to mathematics, most observations can be translated to a mathematical context. For instance, design principles allow us to answer questions like "Why and how do we find a proof without major search effort, although it is a difficult one and we don't know it?" Norman states four principles why we get certain things right, although we don't know precisely what to do [17, p.55]:

1. *Information in the world. Much of the information a person needs to do a task can reside in the world. Behavior is determined by combining the information in memory (in the head) with that in the world.*
2. *Great precision is not required. Precision, accuracy, and completeness of knowledge are seldom required. Perfect behavior will result if the knowledge describes the information or behavior sufficiently to distinguish the correct choice from all others.*
3. *Natural constraints are present. The world restricts the allowed behavior. The physical properties of objects constrain possible operations: the order in which parts can go together and the ways in which an object can be moved, picked up, or otherwise manipulated. Each object has physical features – projections, depressions, screwthreads, appendages – that limit its relationship to other objects, operations that can be performed to it, what can be attached to it, and so on.*
4. *Cultural constraints are present. In addition to natural, physical constraints, society has evolved numerous artificial conventions that govern acceptable social behavior. These cultural conventions have to be learned, but once learned they apply to a wide variety of circumstances.*

We argue that humans make use of such principles, not only in their relationship to everyday objects like door handles, but also in their relationship to mathematical objects like multiplication operators. However, conventional theorem proving systems do not. We will take a closer look at multiplication tables, a concept that seems on a first view easy, and on a second difficult to model in existing theorem proving systems. Multiplication tables are part of rigorous mathematics in the sense that they appear not only as comments or illustrations in mathematical textbooks, but are usually introduced in definitions and their properties are stated as theorems.[4] We believe that the features of the concept "multiplication table" as well as those we present in the next section are not

[4] We use here the word "rigorous" and not "formal" in order to distinguish it from "formal logic" and mechanical systems. Mathematicians would probably use the word "formal," since they are happy with these concepts as a level of formalisation that clarifies concepts unambiguously.

only a matter of presentation but that they are used to encode and retrieve information about mathematical concepts in an efficient way and that they ease the actual process of finding and presenting proofs.

Now let's look concretely at multiplication tables. They were first introduced by Cayley to represent the operation of finite abstract groups. The information encoded into the tables is that the operation is a binary operation, defined on $\{d_1, \ldots, d_n\} \times \{d_1, \ldots, d_n\}$ and with range $\{c_{11}, \ldots, c_{nn}\}$, the operation is discrete and has a finite domain and codomain.

$$\begin{array}{c|ccc} \circ & d_1 & \cdots & d_n \\ \hline d_1 & c_{11} & \cdots & c_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ d_n & c_{n1} & \cdots & c_{nn} \end{array}$$

The table has its own notion of well-formedness, that is, all d_i have to occur and have to be different, the table must be fully filled. Here you find Norman's principles 1, 3, and 4. Multiplication tables are designed in a way that their structure puts "information in the world" that makes it difficult to violate well-formedness. It is hard to imagine when you've got the task to define a specific operation starting with an empty multiplication table that you forget a case, since that would leave a hole in the structure. The table itself is of the form that it constrains the possibilities. For instance, it is impossible to enter more than one entry per field. This prevents any over-specification of $\circ$. Furthermore, although the order of the d_i in the columns and rows could in principle be different, cultural conventions prevent that.

Note that there are particular reasoning methods connected to the representation. To check the basic property of closedness, one has to go through the elements of the table and check whether for all elements holds $c_{ij} \in \{d_1, \ldots, d_n\}$. The commutativity of $\circ$ is checked by verifying that the table is symmetric with respect to the diagonal. This intuitive form of reasoning depends on the cultural convention to use the same order for rows and columns. Another cultural convention is to write a (potential) unit element as first element (or second element in the presence of a zero, which typically goes in the first place). With this convention, it is checked that d_1 is a unit element by establishing the equality of the columns under $\circ$ and d_1 and the rows right to $\circ$ and d_1. Inverse elements can be checked by establishing that each column and each row contain the neutral element exactly once. These reasoning patterns follow partly the design principle number 2, since they are natural and easy to reconstruct. From the group properties only associativity requires a logic level proof.[5]

Of course, it is possible to define the same operation in a logic, possible formalisations are for example:

- First order: extend the signature by a function constant $\circ$ and add the assumptions $d_1 \circ d_1 = c_{11} \wedge d_1 \circ d_2 = c_{12} \wedge \ldots \wedge d_n \circ d_n = c_{nn}$.

[5] Surely, for people experienced with this type of proof, there isn't anything to prove anymore, but it boils all down to trivial computations, which according to the Poincaré principle [1] can be considered as not being a part of the proof. This is different for beginners, whose perspective we have taken here.

– Higher order: use the description operator[6] to define the operation as $\circ \equiv \lambda x.\iota y.(x = (d_1, d_1) \wedge y = c_{11}) \vee \ldots \vee (x = (d_n, d_n) \wedge y = c_{nn})$.

While the special representation of multiplication tables can be translated into these general logical formalisms, parts of the information stored in the mathematical representation are lost. In the first case the compoundness of the table representation is hard to reconstruct. We are speaking about a set of equations suitable for equational reasoning steps, but to recognise that the set of equations is suitable for an abstract method for closedness or commutativity as mentioned before is not so obvious. Also the special reasoning methods for proving commutativity, inverse elements, and neutral element are not so obvious, but require search in a set of formulae. From a human interface point of view, the lack of structure in the formulae puts the burden of guaranteeing well-definedness on the human. He or she has to be careful not to forget the definition of one element or to over-define one expression by inserting two formulae like, for instance, $d_1 \circ d_1 = a$ and at a different place $d_1 \circ d_1 = b$.

Although, the higher order formalisation of the operation seems preferable over the first order one since it encodes $\circ$ as one compound object, here as well it is hard to recognise what kind of function is encoded. Actually, there is a proof obligation to be shown in an application of the function to arguments, namely that there is really a unique element with these properties.

While programming languages like Caml (see e.g., http://caml.inria.fr/) foster for the transition between different data structures, this is typically difficult in deduction systems. We will briefly discuss the little theory approach in IMPS [10] later (see section D.2).

C Mathematical Representations

In this section we look at further examples – in addition to the multiplication tables – for mathematical concepts and procedures which are difficult to represent directly in a foundational system. Although we do not relate them in detail to the design principles discussed in the previous section, it wouldn't be hard to establish similar relationships here as well. We try to argue later that all these examples show that mathematical representations are very flexible and extendible, that new concepts may require new representations and that closed systems do not offer the necessary flexibility to design concepts to the level of sophistication which is achieved in informal mathematics.

C.1 Matrices

A matrix is a two dimensional array that may contain numbers or more complex objects. It has similarities with multiplication tables and a possible formal definition as lists of lists is the same as the one of multiplication tables. However its usage is very different, and human mathematicians have different methods

[6] The description operator ι returns the element of a singleton. $\iota y.P[y]$ denotes the unique element c such that $P[c]$ holds, if such a unique element exists.

attached to these concepts. Matrices are typically used to represent linear mappings and other transformations in vector spaces. The equation[7]

$$\left(\begin{array}{c|ccc}\alpha & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & T^{-1} & \\ 0 & & & \end{array}\right) \cdot \left(\begin{array}{c|ccc}1 & & uT & \\ \hline 0 & & & \\ \vdots & & BT & \\ 0 & & & \end{array}\right) = \left(\begin{array}{c|ccc}\alpha & & \alpha uT & \\ \hline 0 & & & \\ \vdots & & T^{-1}BT & \\ 0 & & & \end{array}\right)$$

is taken from a proof about the tridiagonalisation of matrices.

In principle matrices over a field F can be represented in logic by functions from an index set into F. However, this representation does not lend itself to the definition of multiplication of matrices in which product elements are calculated by traversing the left matrix left-right and the right matrix of the product top-down. The generalisation of this principle makes it easy to establish the relationship accounted for above. Furthermore the explicit matrix representation exhibits advantages mentioned above for multiplication tables.

Note that matrices are available in Computer Algebra Systems (CASs) as primitives and that direct manipulations are possible. This, however, does not make it obsolete to introduce them directly into automated theorem proving systems as well. For instance, the matrices used in the equation above cannot be easily defined in a CAS, since they represent more than one instance, they are generalised matrices representing any matrix that has the same 'form.' Establishing the equation itself cannot be done by computation, but requires reasoning. Once established it can become a further computation rule.

C.2 Dynamic Representations

A feature of mathematical representations is that they are dynamic in the sense that as new knowledge is available the basic representation of objects may change. For instance, the existence of inverse elements of a *group* G can be formalised by $\forall_{x\in G}\exists_{y\in G}\, x\circ y = e \wedge y \circ x = e$ with the unit element e. Since for each group element there exists exactly one inverse element, the inverse of an element x is usually denoted with the help of a function as $inv(x)$. Whereas the introduction of a function denoting the inverse elements is possible in most of the interactive theorem proving systems, the question whether the concept group should be introduced as $group(G,\circ)$, $group(G,\circ,e)$, $group(G,\circ,e,inv)$ seems to be more subtle.

The first formalisation eases the possibility to inherit properties of subsumed concepts with $group(G,\circ) \Rightarrow monoid(G,\circ)$. The latter formalisation makes the unit elements and inverse elements directly accessible but already contains the uniqueness of the inverse element of an element in form of the inverse function. The process of the actual exploration of basic principles of group theory that starts with the tuple $(G,\circ)$ and later introduces the neutral and inverse elements as useful parameters of the concept can hardly be modelled in current automated

[7] Mathematicians store information even in the letters they choose for their objects. Even without further information it is relatively easy to reconstruct that T^{-1}, B, and T should represent submatrices since they make use of upper-case letters, while the Greek α stands for a scalar, and u denotes a vector.

reasoning systems. Rather it is necessary to choose one formalisation and to stick to it. This way it is not only the case that the introduction of a concept cannot be modelled adequately, but perhaps more seriously re-representations of concepts are not supported. However, while one representation may be best suited for one task, it may turn out to be unsuitable for a different one. In the latter situation mathematicians change representations. Different formalisations model different views on something that is one single mathematical concept to a mathematician. If a mathematician had to use one of the standard theorem proving systems, he or she would need to know in advance which choice of representation to make, since the choice of a good formalisation is crucial for the success. But how do you know which formalisation is best, when you start to explore something?

Another example of dynamic re-representation, which is very simple, but for which the different reasoning complexities are striking, is when associativity holds for an operation $+$. Once associativity is established, no mathematician would still use brackets, but the notation for a term like $((1+x)+y)$ would change to $1 + x + y$. The property is encoded into the notation and can be retrieved from the given term. On a reasoning level this simple shift in representation can make a dramatic difference. For a term with $n+1$ summands there are $\frac{1}{n+1}\binom{2n}{n}$ different ways to put brackets in.[8] That means for a medium sized expression with just 10 summands there are already 4862 ways to represent it. If all these representations are part of the search process it is no surprise that automated theorem provers find such expressions difficult. The design of these systems does not allow for a change in representation, a user must write down unnecessary brackets in order to be syntactically correct, the brackets, however, don't help but unnecessarily confuse the reasoner. These all are signs of bad design. In the next sub-section we will look more closer at the change of representation.

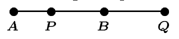
C.3 Change of Representations

Sometimes an appropriate reformulation of a problem into another representation is already the key step to find a proof. Different representations allow to apply knowledge from different sources to a problem. The importance of re-representation was pointed out by George Pólya [20, vol.2, p.80]:

> *When you are handling material things (for instance, when you are about to saw a limb off a tree) you automatically put yourself in the most convenient position. You should act similarly when you are facing any kind of problem; you should try to put yourself in such a position that you can tackle the problem from the most accessible side. You turn the problem over and over in your mind; try to turn it so that it appears simpler. The aspect of the problem that you are facing at this moment may not be the most favourable: Is the problem as simply, as clearly, as suggestively expressed as possible? Could you restate the problem?*
> *Of course you want to restate the problem (transform it into an equivalent problem) so that it becomes more familiar, more attractive, more accessible, more promising.*

[8] Even notation can be an object for mathematical investigations. The formula gives the Catalan numbers.

We exemplify this importance by different forms of re-representations:

- *functions:* Given an Euclidean space $\mathcal{R}$ with a metric function $|\ |: \mathcal{R} \times \mathcal{R} \to \mathbb{R}$ then for each pair $A, B \in \mathcal{R}$ of disjoint points there exists exactly one distance preserving function $g_A^B : \mathbb{R} \to \mathcal{R}$ with $g(0) = A$, $g(|AB|) = B$. With the help of this function the lines in Euclidean space can be interpreted as images of the real numbers. Notions of the real numbers, as intervals, correspond to notions in the abstract Euclidean space, namely line segments.
- *representation theorems:* Poincaré's model for the hyperbolic plane, where a line has infinitely many parallel lines through one point, is a unit disk, where circle segments correspond to hyperbolic lines. With this representation it becomes possible to re-represent constructions in the hyperbolic plane as constructions in the Euclidean plane.
- *theory change:* Geometric constructions can be represented as field extensions. The possibility to construct a square equal in area to a circle using compass and ruler can be re-represented to the question whether π belongs to a particular class of algebraic numbers (which it does not since it is transcendental).
- *inheritance:* The properties of monoids and groups are inherited by the multiplicative and additive substructures of rings and fields.
- *diagrams:* $P \in [AB], B \in [AQ] \Rightarrow P \in [AQ], B \in [PQ]$ which is obvious when this situation is expressed in a diagram: 

And of course there exist re-representations between different theories. Some of them changed the structure of mathematics itself:

- *Cartesian geometry:* arithmetic representation for geometry. This makes it possible to reduce geometrical problems to arithmetic problems and solve them arithmetically. This is actually the starting point of Descartes' idea to take arithmetic as a foundational system so that all problems should be translated to arithmetic and then solved by equation solving.
- *set theory:* mathematical concepts are representable as sets. Set theory is another foundational system on which most of mathematics can be based *in principle.*
- *group theory:* the group concept can be used to represent geometric transformations, permutations, and the solvability of polynomials.

Certain forms of representations are very important to form new concepts. The theorem that every permutation can be decomposed into transpositions, that is, can be represented as a product of transpositions, makes the definition of even and odd permutations suggestive. It is hard to imagine how to define the concept without this particular representational form. Once these concepts have been formed they become the starting point for the introduction of other concepts like the alternating group.

Let's look at a different example, the concept of real numbers. Real numbers can be defined as Dedekind cuts or Cauchy sequences. However, Cantor's second diagonalisation proof that the reals are uncountable is difficult to imagine without having the representation in the decimal (dual, or another) number system.

Often the opposite of a change in the representation happens in mathematics, that is, so-called overloading is used. The use of the same notation for different concepts, e.g. $|\ |$ in the example above for the concept of distance, $+$ for operations that behave like the well-known addition on natural numbers, etc. Even formally incorrect notation, as equality for isomorphisms, is used to enable the transfer of knowledge from a known concept to a new concept. Certain *sort* and *type* systems allow for some kind of overloading, but they do not offer the kind of knowledge transfer that humans achieve this way in using overloaded symbols in an analogical way.

C.4 Limited Coverage of Mathematical Activities by Logic

When we take mathematical textbooks as basis for what is part of mathematics and what not, we can find statements that are not expressible in formal languages at all. Naturally comments or diagrams, and a number of models are not represented. Historic statements, statements about the relevance of properties and concepts and so on are part of mathematics, but not part of logic, although they are often important for a deeper understanding and an informed proof search.

But even at the level of problem formulation, it is not always possible to apply a formal system in a straightforward way. Let's look for instance at the mathematical task (provided in some context): "Determine the maximum of $f(x)$ in the interval $[a, b]$." Does the obvious formalisation "$\exists_{x\in[a,b]}\ \max(f, x)$" reflect the meaning of the sentence? Not necessarily, assume somebody comes up with the argument "Since f is a continuous function on a compact interval it has a maximum." This might be a correct argument to prove the logical formulation but it would not solve the original task. On a closer look adequacy of the logical statements depends on the logic used. If interpreted in a constructive way the logical formulation is adequate; if interpreted classically it is not. While there are constructive and classical systems around, it seems to be inappropriate that one has to decide for a constructive system once and for all, in order to be able to formulate a standard task like the one above.

While one can hardly cover all aspects of mathematics in a computer-based system, it seems for many applications – like the recently emerging applications in education – important to find a coverage which is as broad as possible.

C.5 A 'Natural' Calculus

The suggestions for a 'Mathematical Vernacular' [5] seem not to question that all concepts of mathematics are *sufficiently* expressible in a language consisting of functions and relations, potentially enriched by types or sorts. The design of the Vernacular seems not to be focused on the objects mathematicians are interested in, but on the reasoning framework.

The strength of a formal system can be measured by the de Bruijn factor, that is, the ratio of the length of the proof in the formal system compared to its version in a mathematical textbook.[9] A reassuring observation is that in the experiments with AUTOMATH and MIZAR the de Bruijn factor remained constant for the proofs of a wide range of differently complex theorems.

Even if we agree with the conclusion that the proofs constructed in formal calculi are already in principle an approximation of standard mathematical proofs, it is important to note that such a comparison is based on the *output* of mathematical work, the language used by mathematicians to communicate proofs in textbooks and articles. While a comparison of such completed proofs, proofs after all search is finished may be interesting, they often do not resemble the proof construction. As an example look at standard ϵ-δ-proofs. In the proof construction you compute a sufficient criterion on δ, choose δ as a function of ϵ and than you prove that with this δ the difference of the function values is indeed smaller than ϵ. In a finished proof a crucial part of the proof construction, namely the construction of δ, is redundant and hence not presented. For this reason it is impossible to understand prima facie why δ has been selected as it is.

Traditionally, the formulations of final proofs are minimalist and mathematicians repress their original ideas, even the order of steps can be different, in favour of an objective rigorous style. As Pólya [19, p. vi], pointed out:

> *We secure our mathematical knowledge by demonstrative reasoning, but we support our conjectures by plausible reasoning ... Demonstrative reasoning is safe, beyond controversy, and final. Plausible reasoning is hazardous, controversial, and provisional. ... In strict reasoning the principal thing is to distinguish a proof from a guess, a valid demonstration from an invalid attempt. In plausible reasoning the principal thing is to distinguish a guess from a guess, a more reasonable guess form a less reasonable guess. ... [plausible reasoning] is the kind of reasoning on which his [a mathematician's] creative work will depend.*

The 'demonstrative reasoning' corresponds to a formulation in reasoning steps that were investigated by logicians. In this sense current deduction systems are suitable as proof checkers for existing and well-understood parts of mathematics but lack to act as proof assistants for the exploration and construction of new mathematical knowledge. How can 'plausible reasoning' be modelled? Proof planning follows the paradigm of proof search on an abstract level and can be seen as an important step into this direction. But as described in [3] even proof planning depends on structural restrictions of the underlying calculus and uses a formal language as the only representation for mathematical concepts.

There might be the view that we can come up with a more powerful logic which provides the best possible representation. Marvin Minsky gave a strong argument why this wouldn't be the case in artificial intelligence in general, but why multiple representations are necessary. He recommends [16]:

[9] The notion is not without problems, since mathematical proofs are not standardised with respect to their detailedness. Furthermore there are other aspects for the quality of a proof than its length.

"First decide what kinds of reasoning might be best for each different kind of problem – and then find out which combination of representations might work well in each case."

As we have seen for multiplication tables different types of representations allow for specialised and efficient reasoning methods connected to them, opposed to search on the logic level. Mathematicians carefully design their concepts to keep the search spaces small. We need to understand this aspect of mathematical reasoning much better in order to understand the versatility of the reasoning capabilities of human mathematicians.

Historically the development of many concepts went the way that certain meta expressions were introduced, which were later reified and become object expression. The development of number systems might illustrate this. Having natural numbers and fractions, irrational numbers and negative numbers as well were considered as odd ones out, which only later became first class citizens. Then imaginary numbers were the odd ones and negative square roots were considered as strange entities which were used only for convenience. Likewise, functions were first concrete in nature, and only much later it was possible to speak about them.

D Representations of Mathematics on Computers

Up to here we have strongly argued how important a broad range of specialised constructs is for the adequate representation of mathematical knowledge and for proof construction. Representations find their analogues in data structures used in existing implementations of mathematical software systems, a range of general purpose and specialised theorem proving systems, computer algebra systems, and educational software. These data structures provide functionality and the ability to implement mechanisms working on them. In this section we want to take a brief look at some important features of systems from which ideas can be borrowed to realise a flexible system of the kind we envisage.

D.1 Sorted Extensions to Logic

Sorted logic is a good example to exemplify the importance of careful design. It is an old insight that sorted logic (more carefully put, some classes of sorted logics) can have significant advantages over unsorted logic. Let us just consider the case of standard first-order logic with and without its order sorted extension. Sorts can be considered as special unary predicate symbols. Typical formulations in sorted logic are $\forall x_{\mathsf{Human}}.\mathsf{Mortal}(x)$, $\mathsf{socrates} \lessdot \mathsf{Human}$, $\neg\mathsf{Mortal}(\mathsf{socrates})$. The equivalent in unsorted logic is $\forall x.\mathsf{Human}(x) \Rightarrow \mathsf{Mortal}(x)$, $\mathsf{Human}(\mathsf{socrates})$, $\neg\mathsf{Mortal}(\mathsf{socrates})$. The translation from the sorted to the unsorted logic is called relativisation. The possibility to relativise any sorted problem formulation can be and is used as an argument *against* the use of sorted logic in line with the standard argument "Everything you can do in your system of *sorted logic*, I can do in *unsorted logic*." This argument is – although correct – unhelpful because of the counterargument "Everything you can do in your *unsorted logic*, I can do in *sorted logic*, but in a smaller search space!"

The reason for the smaller search space is due to a better design realised by the sorted system. Although the formalism of sorted logic is equivalent in strength to the unsorted one, a concrete formulation (which makes use of sorts in a non-trivial way, that is, whose relativisation is not equal to itself) is *not*. It is actually *weaker*, since certain things can *not* be derived in sorted logic which can be derived in unsorted logic (concretely, formulae like $\mathsf{Human}(1) \Rightarrow \mathsf{Mortal}(1)$ are syntactically possible in unsorted logic, but the equivalent is rejected in sorted logic). To generalise this observation: A particular design is *better* than an alternative one if certain redundant, or heuristically uninteresting derivations cannot be made.

The integration of sorts in a system also demonstrates how subtle design issues can be. We can't discuss this in detail here, but we will give some indication. It should be noted that sorts are not necessarily exclusively beneficial.[10] If we take standard order-sorted logic we have for each relation like Human and Mortal to decide whether we want to formalise it by a sort symbol or by a predicate symbol. If we want to prove some statement like $\neg\mathsf{Human}(\mathsf{pegasus})$, we cannot formalise Human as a sort symbol. This is a serious flaw in standard sort systems, also it is not possible to model the genesis of concepts in an adequate way. In order to perform the reasoning above we need to know $\mathsf{socrates}{<}\mathsf{Human}$ a priori. It is not possible to infer that Socrates is a human being later in the reasoning process. This unduly limits the flexibility of the system. Ideally you would want to have the advantages of a sorted formulation whenever possible, but also benefit from the flexibility of the unsorted formulation when necessary. To our knowledge only Christoph Weidenbach's system [24] offers these possibilities.

D.2 Little Theories

Another important logic-based approach which approximates mathematical reasoning well is the little theory approach of IMPS [10]. As we can see in textbooks from different areas of mathematics, there is no fixed hierarchy of theories. Each textbook presumes knowledge from different areas of mathematics and by this induces a partial order of theories. That there exists a total order of theories is rather a theoretical result than used in everyday mathematics. The attempt to realise this total order prohibits the flexibility to use theorems constructed for one area in another area. The little theory approach also allows to relate different formulations with each other without necessitating a hierarchy.

D.3 Data Structures in Computer Algebra Systems

CASs offer many more primitive data structures such as matrices than standard theorem proving systems. With these structures it is possible to cover large parts of the 'computational' part of mathematics. As we tried to show, there is a grey area in which deduction and computation go hand in hand (cf. subsection C.1), since any structure in a CAS is concrete, while mathematical expressions often make use of ellipses, for instance, expressions which contain dots like $x_1, x_2, \ldots, x_n$, or the multiplication table and matrix in sections B and C.1.

[10] For a detailed discussion why sorts/types may be harmful see [14].

A promising first approach in the direction of formalising ellipses in reasoning can be found in [8].

D.4 Data Structures for Reasoning

Koedinger and Anderson [13] introduce a representation different from a purely logical formalisation called diagram configuration model (DC) for diagrammatic reasoning in geometry. The representation is based on DC schemas that encode typical geometric situations that were identified through observations on the problem solving behaviour of experts in this domain. The schemas contain the main property of the situation, the subsumed properties and the different ways under which the schema can be established. The level of abstraction allows for an efficient inference algorithm that introduce the DCs as inference steps.

Mateja Jamnik describes in [12] a diagrammatic representation for theorems and inference steps based on this data structure that allow to infer theorems in arithmetic. The proofs constructed in the diagrammatic representation can be translated to proof planning and then formally verified with a theorem prover.

These examples are important in our context since they show that at least for particular domains it is possible to build special reasoners for diagrammatic reasoning which are distinct from a purely logic based system, but which can be formally linked to such a system. An adequate data structure for diagrams seems to be the key point for the success of both approaches. Only this data structure allows the implementation of an efficient inference mechanism.

E Discussion

One of the deepest insights in the foundations of computing was Turing's paper on computable numbers, which clarified by a construct which we now call a Turing machine what can and cannot be computed by computers. This does not mean, however, that the core field of Computer Science would offer to users just Turing machines in which they have to write their programs and if they don't like the idea tell them that everything they may possibly want to do with computers can be written as a Turing machine. It is not even the case that computer languages are built as extensions of Turing machines or compiled into Turing machines. The field of automated theorem proving seems to have followed a different approach. While it is built upon the deep logical insights of the first half of the 20th century, the rich wealth of structure and representations in mathematics has not been mirrored sufficiently yet in formal systems.

In this paper, we presented aspects of mathematical representations that we think are highly relevant for mathematical theorem proving and that can – if at all – be implemented in traditional theorem proving systems only with great difficulty. We think that the lack of acceptance of theorem proving systems (compared with the success of computer algebra systems) is also due to these shortcomings. Deduction systems offer – compared to computer algebra systems – only limited added value: formal correctness, but at a very high price, namely a significant overhead to formulate and prove mathematics. While the production of a large body of formally correct proofs for known theorems does usually not correspond to the research interests of mathematicians, the exploration of new problems is not very well supported as we observed in section C.

Current deduction systems ideally offer the potential benefit that they can relieve users from the tedious task of checking trivial subproblems, so that the human can concentrate on the interesting parts of the problem. In practice, however, it is typically more work even for an experienced human mathematician to formulate the problems in the first place so that an automated theorem prover can prove them than to do the job directly him/herself. We think, when a theorem proving system wants to have a real application as either a proof assistant or a proof tutor, it has to take care about the representation that is used by mathematicians.

The mathematical representation is not only important for the user interface and presentation of proofs, but for theorem proving itself, since the representation is used to optimise problem solving and the transfer and access of knowledge. The observations presented in section C reveal that mathematical knowledge is highly structured and special representations are important for mathematical problem solving. Data structures that realise these aspects may allow for the definition of detailed problem classifications for which special but efficient mechanisms can be found. In sections D.1 and D.4 we gave examples for data structures and implementations of this kind. Mathematical representation appears furthermore to be dynamic and flexible. Modelling this flexibility is on the one hand important for the user interface, because it would give the choice for a representation to the user. On the other hand, this flexibility is the key point for the combination of different representations.

We have summarised some approaches to good design in section D. More work in this direction is necessary to offer well-designed tools to mathematicians and other people interested in computer assisted mathematics. While certain parts might turn out to be straightforward other aspects will require a much deeper understanding. To give one example for the size of the task, if we wanted to integrate multiplication tables as a primitive in automated reasoning systems, it should be relatively easy to offer concrete multiplication tables (or matrices) with all the advantages mentioned in sections B and C (as it is relatively easy to offer order-sorted sorts). However, it will be difficult to offer general type multiplication tables (or matrices) which contain ellipses, and flexible ways to reason about them. Human beings have an amazing capability to reify structures in their space of discourse. We seem not to have yet a deep understanding of these capabilities.

This paper compared approaches in mathematics and in existing theorem proving system. We did not aim to offer solutions, but want to conclude with some thoughts how a strong reasoning system would have to look like to model mathematical practice more adequately. Such a system should firstly be its own meta-system, so that it can speak about itself (care about paradoxes has to be taken). Secondly, the definition of mathematical structures as well as establishing their formal relationships would have to be provided by such a system. Thirdly, a graphical user interface should facilitate the possibility to relate mathematical structures to a form, which is familiar to mathematicians. Particularly important are in this context spatial and diagrammatic representations. While there are approaches to all these points, to our knowledge no single system offers yet an integrated flexible approach.

Acknowledgements

We thank Mateja Jamnik for fruitful discussions and suggestions. Martin Pollet thanks Peter Pröhle for the insights he gave him into the process of mathematical problem solving. Furthermore we thank the anonymous referees for their valuable comments.

References

1. H. Barendregt and A.M. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, **32**(5):3–22, 2001.
2. M. Beeson. Automatic generation of epsilon-delta proofs of continuity. In J. Calment and J. Plaza, editors, *Artificial intelligence and symbolic computation*, pages 67–83. Springer Verlag, Berlin, Germany, LNCS 1476, 1998.
3. C. Benzmüller, A. Meier, E. Melis, M. Pollet, J. Siekmann, and V. Sorge. Proof planning: A fresh start? In M. Kerber, editor, *IJCAR-Workshop: Future Directions in Automated Reasoning*, pages 30–37, Siena, Italy, 2001.
4. R. Brachman and H. Levesque, editors. *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, Los Altos, 1985.
5. N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In R. Nederpelt, J. Geuvers, and R. de Vrijer, editors, *Selected Papers on Authomath*, pages 865–935. Elsevier, North-Holland, Amsterdam, 1994.
6. A. Bundy. The use of explicit plans to guide inductive proofs. In E. Lusk and R. Overbeek, editors, *Proc. of the 9th CADE*, pages 111–120, Argonne, Illinois, USA, 1988. Springer Verlag, Berlin, Germany, LNCS 310.
7. A. Bundy. A science of reasoning. In *Computational Logic: Essays in Honor of Alan Robinson*. MIT Press, 1991.
8. A. Bundy and J. Richardson. Proofs about lists using ellipsis. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th LPAR*, pages 1–12. Springer Verlag, Berlin, Germany, LNAI 1705, 1999.
9. H. Elbers. *Connecting Informal and Formal Mathematics*. PhD thesis, Eindhoven University of Technology, 1998.
10. W. Farmer, J. Guttman, and F. Thayer. Little theories. In D. Kapur, editor, *Proc. of the 11th CADE*, pages 567–581, Saratoga Springs, New York, USA, June 1992. Springer Verlag, Berlin, Germany, LNAI 607.
11. P. Hayes. Some problems and non-problems in representation theory. In *Proc. of the AISB Summer Conference*, pages 63–79, Sussex, 1974.
12. M. Jamnik. *Mathematical Reasoning with Diagrams: From Intuition to Automation*. CSLI Press, 2001.
13. K. Koedinger and J. Anderson. Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, 14:511–550, 1990.
14. L. Lamport. Types are not harmless. Technical report, 1995.

15. U. Martin. Computers, reasoning and mathematical practice. In U. Berger and H. Schwichtenberg, editors, *Proceedings of the NATO Advanced Study Institute on Computational Logic, Marktoberdorf, Germany, July 29 - August 10, 1997*. Springer Verlag, 1999.
16. M. Minsky. Future of AI technology. *Toshiba Review*, 1992. http://web.media.mit.edu/ minsky/papers/CausalDiversity.txt.
17. D. Norman. *The Design of Everyday Things*. The MIT Press, London, 1998.
18. F.J. Pelletier. The philosophy of automated theorem proving. In John Mylopoulos and Ray Reiter, editors, *Proc. of the 12th IJCAI*, pages 538–543, Sydney, 1991. Morgan Kaufmann, San Mateo, California, USA.
19. G. Pólya. *Mathematics and Plausible Reasoning*. Princeton University Press, New Jersey, USA, 1954.
20. G. Pólya. *Mathematical Discovery – On Understanding, Learning, and Teaching Problem Solving*. Princeton University Press, New Jersey, USA, 1962/1965.
21. B. Russell. *The Principles of Mathematics*. George Allen & Unwin Ltd, London, UK, 2nd edition, 1937 edition, 1903.
22. B. Russell. The philosophy of logical atomism. *The Monist*, 28/29:495–527/32–63,190–222,345–380, 1918/1919. republished in [23, p.177-281].
23. B. Russell. *Logic and Knowledge*. Allen & Unwin, London, 1956.
24. C. Weidenbach. Extending the resolution method with sorts. In *Proc. of the 13th IJCAI*, pages 60–65, Chambéry, France, 1993.
25. A.N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, UK, 1910.
26. L. Wittgenstein. *Remarks on the Foundations of Mathematics*. Basil Blackwell, Oxford, England, third edition edition, 1956.

Symbolic Deduction in Mathematical Databases based on Properties

Christoph Schwarzweller

Wilhelm-Schickard-Institute for Computer Science
University of Tübingen
Sand 13, D-72076 Tübingen
schwarzw@informatik.uni-tuebingen.de

Abstract. We claim that mathematical databases should be more than a collection of domains with associated theorems; in particular theorems should be stated as general as possible, that is independent of domains. A database then should be able to check whether such a general theorem holds in a particular domain. To this end we use a properties-based representation for both theorems and domains, and present a deduction calculus that using additional rules about the problem domain allows to perform such a theorem check.

A Introduction

Numerous mathematical theorems have been proven with various mechanized reasoning systems. Among them are, for example, the proof of Robbin's conjecture in Otter [McC97], a proof of the Jordan curve theorem in Mizar [RT99], and a proof of the Chinese Remainder theorem in RRL [ZH93]. However, mathematical databases allowing to reuse such theorems in a general sense can hardly be found. The reason is that mechanized reasoning systems rely on rather involved logics and proof languages, so that proofs cannot be translated from one system to another easily. Thus the only possibility is often to prove such a theorem again or to include it as an axiom. Furthermore, the domain used to prove a theorem often includes unnecessary restrictions, for instance the above mentioned Chinese remainder theorem is proven for the integers and not for rings (or even more general domains). Hence even inside a reasoning system it is sometimes a non-trivial task to reuse an already proven theorem because the current domain does not fit to the one that has been used for the proof.

In this paper we present an approach to mathematical databases focussing on the reuse of theorems in various domains. We have proposed to organize mathematical databases by decoupling proving theorems and reusing them in other domains [Sch01]. To this end, theorems are decorated with sets of properties describing conditions under which a theorem holds. Thus by using as less properties as possible theorems are given in a general setting. These properties-based theorems then allow for checking their validity in particular domains by just checking whether the domain fulfills the properties connected with the theorem. Here, we present a calculus to perform this check in a Prolog-style manner. We think of such a checker as being part of a mathematical database.

B Representation of Theorems and Domains

In this section we describe the representation of theorems and domains underlying our database approach. The key idea is to separate the content of a theorem from the properties necessary to prove a theorem correct [Sch01]. The content $\mathit{Cont}(T)$ of a theorem T states the proposition the theorem is about. It can be compared to a first-order formula. However, the domain and the operations necessary to express $\mathit{Cont}(T)$ are given separately in a signature $\mathit{Sig}(T)$. This allows to distinguish between the proposition of the theorem and conditions under which it holds. This is further elaborated in the third component of a theorem T. Here, a set of properties $\mathit{Prop}(T)$ is given. The intended meaning is that using these properties $\mathit{Cont}(T)$ can be proven correct. To enable easier deduction we represent properties by predicate symbols. Thereby, the arity of these symbols corresponds to the carriers and operations necessary to formulate the property as a first order formula. Summarized we consider a theorem T as a triple

$$T \;=\; (\mathit{Sig}(T), \mathit{Cont}(T), \mathit{Prop}(T))$$

where the statements of $\mathit{Cont}(T)$ and $\mathit{Prop}(T)$ fit to the given signature $\mathit{Sig}(T)$. The other way round $\mathit{Sig}(T)$ should not include more than necessary for the statements given in $\mathit{Cont}(T)$ and $\mathit{Prop}(T)$. Note, that we do not use a formal definition of properties in the sense of first-order logic here. We assume that the meaning of a property is indicated by its name, that is by the chosen predicate symbol. Consider, for example, the following theorem T.

Let R be a (commutative) ring. Then $\{0\}$ is an ideal in R.

Then we get in our notation

$$\mathit{Cont}(T) \;=\; \{0\} \textit{ is an ideal in } R.$$

It is a straightforward task to expand the right-hand phrase "$\{0\}$ *is an ideal in R*" into a first-order formula. More importantly, the signature necessary to formulate this proposition is

$$\mathit{Sig}(T) \;=\; (R, +, *, 0),$$

that is the symbol 1 usually part of a ring signature is not included. Furthermore, in order to prove that $\{0\}$ is an ideal in R it is only necessary that $+$ is associative, provides a right zero as well as right inverses and that $+$ and $*$ are distributive. So we get

$$\begin{aligned}\mathit{Prop}(T) = \{&\text{associative}(R,+),\ \text{right-zero}(R,+,0),\\ &\text{right-inverse}(R,+,0),\ \text{distributive}(R,+,*)\},\end{aligned}$$

that is the properties connected with T are much weaker than the properties of a ring required in the original version of the theorem. Note that the arguments $R, +, *$ and 0 can be interpreted as variable symbols since they represent arbitrary carriers and operations respectively. This will play an important role later for the calculus.

Domains D can be represented in a similar way. They also consist of a signature $Sig(D)$ giving carriers and operations of the domain and a set $Prop(D)$ containing properties the domain fulfills, thus

$$D \;=\; (Sig(D), Prop(D)).$$

This works for both abstract domains such as rings or fields and concrete domains. For example, the ring of integers $\mathbb{Z}$ would look like

$$\begin{aligned} Sig(\mathbb{Z}) &\supseteq (\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}, 0_{\mathbb{Z}}, 1_{\mathbb{Z}}) \\ Prop(\mathbb{Z}) &\supseteq \{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}), \text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}), \\ &\quad\ \text{commutative}(\mathbb{Z}, +_{\mathbb{Z}}), \text{commutative}(\mathbb{Z}, *_{\mathbb{Z}}), \\ &\quad\ \text{Euclidean}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}, 0_{\mathbb{Z}})\} \end{aligned}$$

where $\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}, 0_{\mathbb{Z}}$ and $1_{\mathbb{Z}}$ are now constant symbols. Thus the approach allows for the description of both domains and theorems with regard to properties of domains and properties ensuring the correctness of theorems. This gives rise to a straightforward criterion whether a theorem T holds in a domain D where D may be both an abstract or a concrete domain. It has only to be checked whether D provides both the necessary signature and the properties connected with T, thus

$$Cont(T) \text{ holds in } D \;:\Longleftrightarrow\; Sig_D(T) \subseteq Sig(D) \wedge Prop_D(T) \subseteq Prop(D).$$

The notations $Sig_D(T)$ and $Prop_D(T)$ resp. mean that the variable symbols occurring in the theorem T, actually in $Sig(T)$, are replaced by the corresponding symbols of the domain D. Note that the failure of this test not necessarily implies that a theorem does not hold in a domain, the check is relative to the properties stated about the theorem and the domain. Reasoning is not necessary here, just checking whether certain properties, that is predicates connected with domains and theorems, are present. Nevertheless the deduced results are correct provided that the meaning of the properties was defined properly. Note, that this method also allows for straightforward error messages by collecting the properties of T not included in the ones of D.

C A Calculus for Deducing Properties

The distinction between the content of a theorem and properties under which this content can be proven allows for checking whether theorems hold in a domain by comparing sets of properties with respect to inclusion. However, this setting is too restricted. For example, if a theorem T requires the property left-distributive, and a domain D obeys the property distributive, it is not desirable that left-distributive has to be added to the domain's properties. The mathematical database should rather be able to conclude that T holds in D, although the sets of properties involved are not related by inclusion.

In the following we replace the subset relation between two sets P_1 and P_2 of properties by a relation $P_1 \Longrightarrow P_2$ with the meaning that every domain D that fulfills the properties of P_1 also fulfills the ones in P_2, or more formally

$$\models P_1 \Longrightarrow P_2 \quad :\Longleftrightarrow \quad \forall D : D \models P_1 \text{ implies } D \models P_2$$

where $\models$ on the right-hand side is the model operator well-known from first-order logic. Thus the content of a properties-based theorem $T = (Sig(T), Cont(T), Prop(T))$ holds in a domain $D = (Sig(D), Prop(D))$ if both $Sig_D(T) \subseteq Sig(D)$ and $\models Prop(D) \Longrightarrow Prop_D(T)$ are valid. Note that $\models P_1 \Longrightarrow P_2$ corresponds to the usual semantic implication. However, in a mathematical database as proposed in the last section the formulas occurring in P_1 and P_2 are given by a set of predicates only, whose arguments are either variables or constants.

Obviously, an implication $P_1 \Longrightarrow P_2$ cannot be checked in this generality, in particular if we represent properties by predicate symbols without giving the definition of properties as a first-order formula. Therefore we incorporate a set of rules L describing basic relations between sets of properties. For example, the following rule

$$\begin{array}{l}\{\text{distributive}(R,+,*)\} \longrightarrow \\ \{\text{left-distributive}(R,+,*),\ \text{right-distributive}(R,+,*)\}\end{array} \qquad (\mathbf{A})$$

states that structures $(R,+,*)$ that are distributive are also both left- and right-distributive. In this way a mathematical database is provided with additional knowledge about the problem domain. The deduction of $\models P_1 \Longrightarrow P_2$ is then performed relative to such a set of rules.

The calculus has two axioms. The first mirrors the fact, that an implication $P_1 \Longrightarrow P_2$ trivially holds, if $P_2 \subseteq P_1$. The second axiom allows to incorporate the external rules: if $l \longrightarrow r \in L$, then $\sigma(l)$ implies $\sigma(r)$ where σ is an arbitrary substitution compatible with the signature. Further on, there are a rule allowing to combine different implications P_2 and P_3 both made from P_1 and a rule for concatenating implications $P_1 \Longrightarrow P_2$ and $P_2 \Longrightarrow P_3$.

$$\frac{P_2 \subseteq P_1}{\vdash P_1 \Longrightarrow P_2} \qquad (\mathbf{AX1})$$

$$\frac{l \longrightarrow r \in L}{\vdash \sigma(l) \Longrightarrow \sigma(r)} \qquad (\mathbf{AX2})$$

$$\frac{\vdash P_1 \Longrightarrow P_2,\ \vdash P_1 \Longrightarrow P_3}{\vdash P_1 \Longrightarrow P_2 \cup P_3} \qquad (\mathbf{R1})$$

$$\frac{\vdash P_1 \Longrightarrow P_2,\ \vdash P_2 \Longrightarrow P_3}{\vdash P_1 \Longrightarrow P_3} \qquad (\mathbf{R2})$$

Provided that the rules in L are correct, that is if from $l \longrightarrow r \in L$ indeed follows $\models \sigma(l) \Longrightarrow \sigma(r)$ for the substitutions σ used in a deduction, it is straightforward to see that the calculus is correct. In other words, we have that (relative to L) $\vdash P_1 \Longrightarrow P_2$ implies $\models P_1 \Longrightarrow P_2$. However, if no deduction sequence is found this does not necessarily mean that $\models P_1 \Longrightarrow P_2$ is not valid. The reason is that the calculus checks for implications with respect to the rule set L only. In other words, if L does not contain enough knowledge about the problem domain, the deduction of an implication may fail, although this implication is true.

The calculus can be extended with some straightforward derived rules, among them

$$\frac{\vdash P_1 \Longrightarrow P_2 \cup P_3}{\vdash P_1 \Longrightarrow P_2} \qquad (\mathbf{L1})$$

$$\frac{\vdash P_1 \Longrightarrow P_2, \quad P_1 \subseteq P_3}{\vdash P_3 \Longrightarrow P_2} \qquad (\mathbf{L2})$$

These rules can be easily proven correct in the sense that their consequences can be deduced from their premises in the original calculus. To see how the calculus works let us deduce the following implication.

$$\begin{aligned}\vdash \{&\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}),\ \text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \Longrightarrow \\ &\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}), \\ &\ \text{left-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}),\ \text{right-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\}\end{aligned}$$

To do so, we assume that the distributivity rule **A** from above is present in the set of rules L. Then, using Lemma **L2**, we get the following deduction sequence. Note that the actual definition of the properties involved has no influence on the deduction, that is the deduction is purely symbolic.

$$\begin{aligned}
(1) \vdash\ &\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}),\ \text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \Longrightarrow \\
&\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}})\} \\
&\text{by } \mathbf{AX1} \\
(2) \vdash\ &\{\text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \Longrightarrow \\
&\{\text{left-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}),\ \text{right-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \\
&\text{by } \mathbf{AX2} \text{ with } \mathbf{A} \text{ and } \sigma(R) = \mathbb{Z},\ \sigma(+) = +_{\mathbb{Z}},\ \sigma(*) = *_{\mathbb{Z}} \\
(3) \vdash\ &\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}),\ \text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \Longrightarrow \\
&\{\text{left-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}),\ \text{right-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \\
&\text{by } \mathbf{L2}(2) \\
(4) \vdash\ &\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}},\ \text{distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \Longrightarrow \\
&\{\text{associative}(\mathbb{Z}, +_{\mathbb{Z}}), \\
&\ \text{left-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}}),\ \text{right-distributive}(\mathbb{Z}, +_{\mathbb{Z}}, *_{\mathbb{Z}})\} \\
&\text{by } \mathbf{R1}(1,3)
\end{aligned}$$

Thus a theorem T requiring for instance the properties associative$(R, +)$, left-distributive$(R, +, *)$ and right-distributive$(R, +, *)$ holds in particular for $\mathbb{Z}$. Though for $\mathbb{Z}$ only the property distributive$(R, +, *)$ has been stated, this can be checked by a mathematical database using the calculus. Note that, by just taking the identity substitution for σ, the above sequence can be easily transformed in a deduction sequence using $R, +$ and $*$ instead of $\mathbb{Z}, +_{\mathbb{Z}}$ and $*_{\mathbb{Z}}$, that is general lemmas can be shown.

Finding a deduction sequence for an implication $P_1 \Longrightarrow P_2$ requires some amount of guessing in which way the set on the left-hand side of an implication has to be extended, that is guessing which property should be additionally considered in order to combine already deduced implications. This can be seen, for example, in step (3) of the deduction above where using Lemma **L2** the set on the left-hand side is extended from $\{\text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\}$ to $\{\text{associative}(\mathbb{Z}, +_\mathbb{Z}),\ \text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\}$; any other extension would have been a correct application of **L2**, too. Fortunately, this problem can be avoided using backward propagation. The idea is, given an implication $P_1 \Longrightarrow P_2$, to successively remove properties from P_2 that are implied by the ones from P_1. We use the following three rules.

(B1) Replace $\vdash P_1 \Longrightarrow P_2$ by $\vdash P_1 \Longrightarrow P_2 \backslash (P_1 \cap P_2)$.
(B2) Replace $\vdash P_1 \Longrightarrow P_2$ by $\vdash P_1 \Longrightarrow (P_2 \backslash (\sigma(r) \cap P_2)) \cup \sigma(l)$ if there are a rule $l \longrightarrow r \in L$ and a substitution σ with $\sigma(r) \cap P_2 \neq \emptyset$.
(B3) Accept $\vdash P_1 \Longrightarrow \emptyset$.

Thus an implication $P_1 \Longrightarrow P_2$ is accepted, if it can be transformed into an implication of the form $P_1 \Longrightarrow \emptyset$. The rules are correct with respect to the above calculus in the sense that every deduction starting with $P_1 \Longrightarrow P_2$ and ending with $P_1 \Longrightarrow \emptyset$ using **B1** - **B3** can be translated into a correct sequence of the original calculus. For the example from above we get

$$\begin{aligned}
&\vdash \{\text{associative}(\mathbb{Z}, +_\mathbb{Z}),\ \text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \Longrightarrow \\
&\quad \{\text{associative}(\mathbb{Z}, +_\mathbb{Z}), \\
&\quad\ \text{left-distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z}),\ \text{right-distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \\
&\vdash \{\text{associative}(\mathbb{Z}, +_\mathbb{Z}),\ \text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \Longrightarrow \\
&\quad \{\text{left-distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z}),\ \text{right-distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \\
&\quad \text{by } \mathbf{B1} \\
&\vdash \{\text{associative}(\mathbb{Z}, +_\mathbb{Z}),\ \text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \Longrightarrow \\
&\quad \{\text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \\
&\quad \text{by } \mathbf{B2} \text{ with } \mathbf{A} \text{ and } \sigma(R) = \mathbb{Z},\ \sigma(+) = +_\mathbb{Z},\ \sigma(*) = *_\mathbb{Z} \\
&\vdash \{\text{associative}(\mathbb{Z}, +_\mathbb{Z}),\ \text{distributive}(\mathbb{Z}, +_\mathbb{Z}, *_\mathbb{Z})\} \Longrightarrow \emptyset \\
&\quad \text{by } \mathbf{B1}
\end{aligned}$$

which is accepted by **B3**. Note, that the only choice throughout the deduction consists of determining which rule of L should be applied. Also the left-hand side P_1 of the goal does not change throughout the whole deduction, so that keeping track of the changes occurring in P_2 is sufficient. Finally it may be worth mentioning that for rules $l \longrightarrow r$ with a right-hand side r consisting of one property only, **B2** can be simplified to

(B2') Replace $\vdash P_1 \Longrightarrow P_2$ by $\vdash P_1 \Longrightarrow (P_2 \backslash \sigma(r)) \cup \sigma(l)$ if there are a rule $l \longrightarrow r \in L$ and a substitution σ with $\sigma(r) \in P_2$.

Thus no intersection has to be computed in this case. Note, that this kind of rules can be easily obtained by splitting up given rules as for example the distributivity rule from above into the following two ones.

$$\begin{array}{rcl}\{\text{distributive}(R,+,*)\} & \longrightarrow & \{\text{left-distributive}(R,+,*)\} \\ \{\text{distributive}(R,+,*)\} & \longrightarrow & \{\text{right-distributive}(R,+,*)\}\end{array}$$

However, transforming all the rules of L this way would heavily increase the number of steps in a deduction and only detailed experiments will show which method is to prefer.

D Conclusion

The calculus presented provides mathematical databases with additional knowledge: it allows to infer implications of sets of properties with respect to a given set of basic rules. In the database both theorems and domains are represented based on properties so that the implication of sets of properties is sufficient to check whether a theorem holds in a particular domain. Thus theorems stated in this general manner can be checked for validity in special domains easily.

Stating theorems with respect to properties rather than domains may at first glance be somewhat unfamiliar. However, if working in a particular domain only, the familiar representation of theorems can be easily regained. For example, a theory of rings can be constructed as follows. First—if this has not already been done—the database has to be extended by a domain $R = (Sig(R), Prop(R))$ with the appropriate signature and properties defining rings. Then all theorems T with $Sig_R(T) \subseteq Sig(R)$ and $\models Prop(R) \Longrightarrow Prop_R(T)$ can be extracted from the database, in this way building a new database for rings.

Mechanized reasoning systems can be incorporated the following way. Theorems and properties included in the library as well as rules used for deduction can be proven with a mechanized reasoning system (provided that first-order formulae has been attached to the properties' predicate symbols). Note, that this indeed is a realization of proving theorems and properties of domains on the one side, and storing and reusing knowledge in a mathematical database on the other side. Mechanized reasoning systems allowing to formulate and prove properties-based theorems are for example Mizar [RT99], Imps [Far93], and Theorema [BJK97].

The properties-based approach has applications in other areas, for instance in the area of generic programming. Here, generic algorithms obey type parameters which are instantiated later to get a running instance of the algorithm. Both the feasibility and the correctness of such an instance depends on whether the operations instantiated fulfill certain properties which is usually not checked. Using an properties-based approach, that is providing generic algorithms and possible instantiations with properties required resp. fulfilled, this can be done in terms of the calculus presented.

References

[BJK97] B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru, A Survey on the Theorema Project, in: Proceedings of ISSAC'97

(International Symposium on Symbolic and Algebraic Computation), ed. W. Küchlin, ACM Press, 1997, pp.384-391.

[Far93] W. M. Farmer, J. D. Guttman, and F. J. Thayer, IMPS: An Interactive Mathematical Proof System, Journal of Automated Reasoning, 11 (1993) 213-248.

[McC97] W. McCune, *Solution of the Robbins Problem*; in: Journal of Automated Reasoning (19), p. 263-276, 1997.

[RT99] Piotr Rudnicki and Andrzej Trybulec, *On Equivalents of Well-foundedness. An Experiment in Mizar.* in: Journal of Automated Reasoning, 23:197–234, 1999.

[Sch01] C. Schwarzweller, *Designing Mathematical Libraries based on Minimal Requirements for Theorems.* in: Proceedings of the First International Workshop on Mathematical Knowledge Management (MKM2001), 2001.

[ZH93] H. Zhang and X. Hua, *Proving the Chinese Remainder Theorem by the Cover Set Induction*; in: D. Kapur (ed.), Proceedings of the 1992 International Conference of Automated Deduction, LNAI 607, Springer-Verlag, p. 431-445, 1993.