Automated theorem provers: a practical tool for the working mathematician?

Citation for published version:
Bundy, A 2011, 'Automated theorem provers: a practical tool for the working mathematician?', *Annals of Mathematics and Artificial Intelligence*, vol. 61, no. 1, pp. 3-14. https://doi.org/10.1007/s10472-011-9248-8

Digital Object Identifier (DOI):
10.1007/s10472-011-9248-8

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
Annals of Mathematics and Artificial Intelligence

Noname manuscript No.
(will be inserted by the editor)

Automated Theorem Provers: A Practical Tool for the Working Mathematician?

Alan Bundy

June 6, 2011

Abstract In contrast to the widespread use of computer algebra systems in mathematics automated theorem provers have largely met with indifference. There are signs that this is at last beginning to change. We argue that it is inevitable that automated provers will be adopted as a practical tool for the working mathematician. Mathematical applications of automated provers raises profound challenges for their developers.

Keywords Automated theorem proving · Mathematician's assistant · Very large proofs · Proof understanding

1 Introduction

Automated theorem proving is the application of computer programs to the proof of theorems. Axioms and theorems are formalised in a logic, and new theorems are derived from old ones using logical rules of inference. The field ranges from totally automated provers via semi-automated, interactive provers, to mere proof checkers. Its main application has been to resolving the proof obligations that arise from formal methods of ICT system development, but it also finds applications in mathematics and to commonsense reasoning in artificial intelligence.

The past decade has seen a number of significant developments in automated theorem proving, for instance: the growing importance of SAT, SMT, description-logic reasoners and other decision procedures; the industrial uptake of formal methods of ICT system design; and the central role of automated inference in the Semantic Web. In this essay, however, I want to focus on a development that is perhaps

The research reported in this paper was supported by EPSRC grants EP/F033559/1 and EP/H023119/1. I am indebted to Bogdan Grechuk, Lucas Dixon, Ursula Martin and an anonymous AMAI referee for valuable feedback on earlier drafts, and to Michael Chan for help with the submission.

Alan Bundy
School of Informatics, University of Edinburgh
Edinburgh EH8 9AB, UK.
E-mail: bundy@ed.ac.uk

less conspicuous, but which promises a far-reaching impact on the World's oldest science.

Mathematicians have keenly adopted a number of computational tools. Foremost among these have been: the use of latex for typesetting mathematical papers; the use of computer algebra systems for formula manipulation and calculation; and the use of the Polymath Blog[1] for collective mathematical problem solving. The mathematical community has been involved in the development or even creation of some of these tools, for instance, the AMS latex packages and Polymath. A more comprehensive account of the computational tools that mathematicians have adopted can be found in [Martin, 1999].

Conspicuous by its absence has been the use of automated theorem provers to assist with the development of proofs. Among the reasons that mathematicians frequently give me when I ask them why they are not using automated theorem provers are the following.

1. The detailed logical proofs typically produced by automated provers can have an order of magnitude more steps than a typical textbook or journal paper proof. This extra detail can be tedious to wade through. Mathematicians find informal proofs more accessible and understandable. This point was developed at length in [Millo *et al*, 1979] and is further discussed in [Martin, 1999].
2. Automated provers are not powerful enough to prove novel conjectures of interest to mathematicians. In particular, they do not have the libraries of previously proved theorems that a mathematician would want to use without having to prove them first.
3. Some of the manipulations that mathematicians do routinely and quickly require tedious logical manipulations, such as: reinterpreting integers as reals[2]; using symmetry and analogy to avoid proving new cases that are similar to old ones; mapping theorems from one theory to another.
4. Automated provers are quite hard to use. Theorems have to be translated into the format used by the prover, which may require an unexpected level of detail, such as declaring the types of constants and variables. The translation of informal mathematical concepts into formal logic may itself introduce errors, so that the prover attempts to prove the wrong theorem. When the prover's automated techniques fail, it must be guided interactively, which involves some understanding of its proof technique.
5. Theorem proving is what mathematicians most enjoy. Why let a machine have all the fun?

On this last point, [Gowers, 2000][p2] says: "The idea that all our creativity and insight might be reduced to something mechanical was, after all, not very appealing".

In §7, we will reconsider each of these reasons in the light of our discussion in the rest of this paper.

Despite these obstacles, the last decade has seen a small, but growing mathematical interest in automated theorem proving. Consider, for instance, the series of articles on automated proof that have recently appeared in the Notices of the American Mathematical Society, e.g., [Hales, 2008, Gonthier, 2008]. I will argue

[1] `http://polymathprojects.org/`

[2] An example of *type casting*

that this interest is not only inevitable, but is likely to grow and to have a profound influence on mathematical methodology.

2 Simple Theorems with Enormous Proofs

The cause of this interest in automated proof is the emergence of simple theorems whose shortest proof is very large. The existence of theorems with enormous proofs is a direct corollary of the undecidability of predicate calculus provability. In 1936, Alonzo Church and Alan Turing independently proved that there was no algorithm that would determine whether or not a conjecture in predicate calculus was provable [Church, 1936a, Church, 1936b, Turing, 1936]. This undecidability of provability extends to nearly all nontrivial areas of Mathematics.

To see that this result entails the existence of theorems with enormous proofs, suppose, on the contrary, that there was a limit to the size of proofs. To be concrete, suppose that there was an arithmetic function, f, that set an upper limit on the size of a theorem's proof given the size of the theorem. Contrary to the above Church/Turing result, we could now design an algorithm for the provability of a conjecture. The algorithm is as follows:

– Given a conjecture, apply f to its size to determine the maximum size of its proof.
– Generate all proofs in the theory up to that maximum size. This requires an astronomical, but finite, amount of work. [Note that there are only finitely many proofs of any given size.]
– If one of these proofs proves the conjecture, then it is a theorem.
– Otherwise, if when the process terminates no proof has been found, then the conjecture is unprovable.

Since assuming an upper limit to proof size contradicts Church/Turing's undecidability result, then there can be no such upper limit. This result is even stronger than the mere existence of enormous proofs, it says that however big you set the proof size N, then there is a theorem of size n, much smaller than N, whose shortest proof is bigger than N.

This might have remained a mere logical curiosity, were it not for the emergence of a series of simple theorems which appear to require proofs of very large size[3]. Three of the most prominent recent examples have been: Appel and Haken's proof of The Four Color Theorem [Appel *et al*, 1977] (see Figure 1); the Classification of Finite Simple Groups [Gorenstein, 1982] (see Figure 2); and Hales' proof of The Kepler Conjecture [Hales, 2005b] (see Figure 3).

As important theorems requiring larger and larger proofs emerge, Mathematics faces a dilemma: either these theorems must be ignored or computers must be used to assist with their proof.

3 The Use of Computers in Proofs

Appel and Haken's proof of The Four Color Theorem reduced it to 1,482 'reducible configurations', each of which then had to be discharged. They chose to use a

[3] But as yet nothing compared to the enormous proofs that the Church/Turing result predicts.

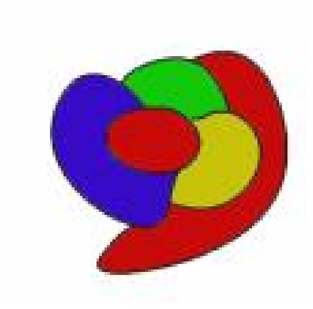

Fig. 1 The Four Color Theorem. This map is coloured with just four colours, so that no two adjacent countries share a colour. Will four colours always suffice?

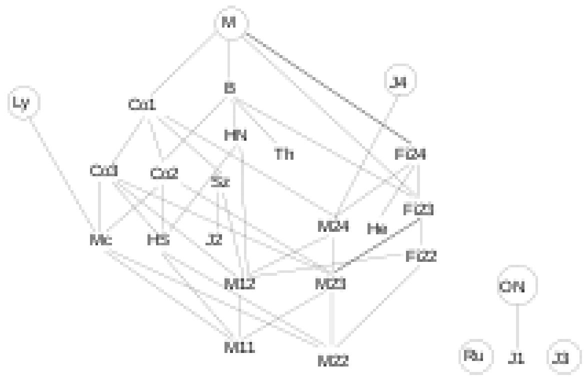

Fig. 2 The Classification of Finite Simple Groups. This graph shows a small part of the classification: the relations between the sporadic groups. Most of them are related to an enormous group, called *The Monster*. Many of these sporadic groups were first constructed using computer algebra systems.

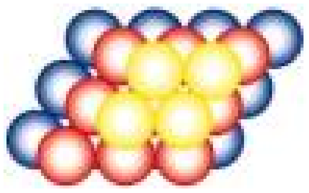

Fig. 3 The Kepler Conjecture. These spheres are packed closely together. Is this always the tightest possible packing method?

custom-built computer program to execute this discharge process. This was partly to avoid the estimated couple of months of work that a manual calculation would have taken, but mainly to ensure the absence of errors in the calculations. The use of computers to automate part of the proof proved hugely controversial within the mathematical community [Mackenzie, 2001][Chap. 4]. The main cause of concern was that there might have been a bug in the computer program, so that the correctness of the proof could not be guaranteed.

One response to this concern about the correctness of the proof was to automate it in a theorem prover. If the prover was built in the 'LCF' style [Gordon *et al*, 1979], then any proof could only be constructed by a small kernel of logical rules of inference. This ensured a high degree of confidence in the proof. This programme was eventually carried out by Gonthier in 2005 [Gonthier, 2008] using the Coq LCF-style prover.

Essentially the same story was repeated with Hales' proof of The Kepler Conjecture. This proof also involved reducing the problem to lots of cases and then using a custom-built computer program to check each case. When the proof was submitted to the Annals of Mathematics, the editor appointed two teams of referees; a regular team to examine the human part of the proof and a 12-person team to examine the computer-generated part. After months of effort, the team examining the computer part announced that they were unable confidently to verify the proof. The human part was accepted for publication, but published with a disclaimer about the correctness of the computer part. Unsatisfied with this outcome, Hales has set up the Flyspeck Project [Hales, 2005a] to apply LCF-style provers to prove the whole theorem, and thus confirm its unqualified correctness.

The proof of The Classification of Finite Simple Groups has also involved computers, but a more conventional use of computational algebra to help construct some of the extremely large groups. Hand-crafted programs were used to find matrices with certain properties. These programs were fore-runners of today's computer algebra systems. Perhaps surprisingly, given the notorious unsoundness[4] of many computer algebra systems, this use of computers has not produced anything like the controversy generated by the Four Color and Kepler Conjecture computer proofs. The explanation in this case may rely on the specific use made of computers: they were used to construct candidate groups, whose properties were then relatively easy to check by hand. The proof that all groups had now been identified was done purely by hand. However, computer algebra systems are in widespread use in Mathematics, and it is not always so easy to explain away the lack of concern about their unsoundness. Why is this?

4 Mathematicians' Attitude to Error

In 2004, I co-ran a Royal Society meeting on "The Nature of Mathematical Proof" [Bundy *et al*, 2005a]. This meeting brought together mathematicians, computer scientists, logicians, philosophers, sociologists and others, to compare and contrast their different attitudes to 'proof'. This uncovered huge cultural differences, in which the logicians were closer to the computer scientists than to the other mathematicians.

Aschbacher, one of the principal researchers involved in the Classification of Finite Simple Groups, was a speaker and summarised the main cultural difference when he said that "the probability of an error in the proof is one" [Aschbacher, 2005][p2403]. So here we have a paradox. On the one hand, Aschbacher is asserting that one

[4] This most often arises when general results are returned without the conditions under which they have been calculated. For instance, they may be false in degenerate cases, such as at boundary values, but this is not stated.

of the triumphs of modern mathematics and a cornerstone of much subsequent mathematics, definitely has at least one error. The mathematicians present seemed neither surprised nor perturbed by this revelation. On the other hand, many mathematicians were deeply concerned with the possibility of error in the proof of The Four Color Theorem — even though computers had been principally used in its proof to *avoid* the strong possibility of error had the computer-generated parts of the proof been done by hand.

One resolution of this paradox is that mathematicians distinguish two kinds of error:

1. Minor errors that are readily corrected and that are not critical to the integrity of the proof; and
2. Major errors that undermine the proof and that must be corrected.

Aschbacher presumably meant that the errors in the Classification proof were of type 1, so were of little importance. Although, of course, it would still be considered very important to correct any type 1 errors that *were* identified. The concern about the computer parts of the Four-Color and Kepler proofs was that they might be of type 2, that is, that a buggy program had erroneously 'checked' a case that was actually a counter-example to the theorem.

Computer Scientists make a similar distinction between kinds of program bugs: type 1 roughly corresponding to coding errors and type 2 to design errors. Automated prover developers, however, do not make such a distinction between kinds of erroneous *proofs*. Firstly, errors rarely arise within proofs produced by mature provers, especially LCF-style provers. Secondly, if they do, then the problem is likely be blamed on the program, not the proof. Moreover, since most computer proof terms are flat and low-level, all proof bugs are at the same level of granularity, and all are considered equally fatal to the correctness of the proof.

This goes some way to explain the lack of mathematical concern over an important proof that definitely contains an error, and the extreme concern over even the possibility of error in a computer-produced proof. Confidence in this mathematical judgement, of course depends on its accuracy, i.e., can experienced mathematicians reliably distinguish between the two kinds of proof error?[5] It seems that they can distinguish between routine parts of the proof, e.g., standard definitions and lemmas, and the critical parts where the key novel contribution is made. They assume that faults in the routine parts can be readily corrected without significant impact on the rest of the proof. Attention is focused on the critical parts. From experience they will have a collection of common counter-examples and common missteps that they can use to test this critical part, exposing possible errors. They will also be alert to the misquoting or inappropriate use of third-party lemmas.

That automated reasoners can be used to find and correct errors in informal human-produced proofs is ably illustrated by Fleuriot's formalisation of the derivation in, Newton's Principia, of Kepler's Laws of planetary motion [Fleuriot & Paulson, 1999]. His formalisation uncovered a previously undiscovered error in Newton's proof. The error, the cancellation of an infinitesimal quantity on either side of an equation, was one that Newton himself had elsewhere highlighted but had unwittingly made himself on this occasion. Fleuriot was able to correct the error and complete the automation of the proof. *It is remarkable that three centuries of analysis of*

[5] Especially when, as in the Aschbacher quote above, they do not know what the error is.

this keynote Principia proof failed to uncover this error, but that it was readily discovered on the first attempt to automate the proof.

5 The Importance of Understanding Proofs

Although correctness is the usual reason cited for concern over computer proofs, it is not the only reason — arguably not even the most important one. Mathematicians also require to *understand* proofs. [Gowers, 2000][p3], for instance, says: "...we tend to prefer questions that are interesting, comprehensible and seemingly not completely out of reach, and we like our proofs to provide explanations rather than just formal guarantees of truth". This is not just so they can verify the proof's correctness, but also because they want to recycle new proof ideas in their own future proofs. In fact, a new theorem can be important for two reasons: not just the new knowledge that the theorem represents, with its potential for new applications both within and outwith[6] mathematics, but also any novel proof method, which has the potential to be applied to future conjectures. Proofs generated by computer are inherently hard to understand. Indeed, the actual reasoning process may not even be available for inspection, but only the binary result of the checking: yes or no.

LCF-style proofs do offer *some* potential for understanding. A low-level proof term is usually available to be inspected, but (a) this can be extremely large and (b) it facilitates understanding only at the lowest level of detail: that one proof line is a logical consequence of previous ones. Mathematicians also desire a higher-level understanding: what are the main cases into which the proof is divided? what is the key, novel proof idea? which bits of the proof are hard and/or surprising and which bits routine and easy? [Gowers, 2000][p3], for instance, says: "Proofs are usually clearer if they have a hierarchical structure" and "Many good proofs are variants of existing better known arguments. This makes them easier to understand, because all one has to do is concentrate on the parts that are new".

Most research in automated proof *presentation* is focused on turning machine-oriented proof representations into human-oriented ones, e.g., by translating them into sequent calculus, natural deduction or natural language form. For instance, the ISAR format [Wenzel, 2007] for Isabelle [Paulson, 1990] provides a more readable 'deductive' style of presentation rather than the previous tactic-based 'procedural' one, but the presentation is still low level.

This points to an important new direction for automated reasoning: multi-level proof presentation in which the user can choose the level of granularity of the proof and which highlights the key ideas and the hard parts of the proof. Tactic-based provers already provide some basis for grouping proof steps and lifting the granularity of the proof, but this functionality has been under-exploited in existing provers. My research group has been addressing this issue in its work on *proof plans*, which provide a high-level proof outline, and *proof critics* which highlight the point at which standard proof techniques have broken down and have needed to be adapted or repaired [Bundy *et al*, 2005b]. We have also experimented with a hierarchical, graphical presentation of these proof plans, called

[6] 'Outwith' is an invaluable Scottish word. It is the antonym of 'within', whereas 'without' is the antonym of 'with' and 'outside' the antonym of 'inside'.

hiproofs, which the user can use to browse the proof at different levels of granularity [Denney *et al*, 2006]. This provides the functionality both to get an overview of the proof at the highest level of granularity, but also to delve down to any level of detail. In contrast, journal presentations of human-produced proofs can only provide one level of detail, leaving some readers bored and others mystified. In future, one might also add functionality to explore the interrelationships within a proof, e.g., where and why is it that this surprising condition is required? Help information might be attached to definitions, axioms, etc. both to access their formal definitions and explain the intuitions behind them.

6 Future Developments

The adoption of automated theorem provers in mathematical research will be slow and incremental. Provers will not be used routinely until there is a solid basis of well-developed mathematical theories from which novel research can grow. But this basis must be provided by experienced mathematicians adopting provers and formalising existing mathematics. Some initial formalisation has been started by a few strongly motivated mathematicians and those computer scientists with an interest in both automated proof and mathematics, but this is insufficient as a basis for novel mathematics. So, we have deadlock.

In this section, we speculate on five potential applications that might overcome this deadlock and kick-start the process.

6.1 Automated Theorem Synthesis

While coming up with a novel idea for a challenging theorem is both fun for mathematicians and beyond the ability of current provers, there are more routine activities in theorem proving where mathematicians might welcome automated assistance, for instance, the initial exploration of alternative axiomatisations of a domain. The MATHsAiD system [McCasland *et al*, 2006] was designed to address this requirement[7]. Given a set of axioms, it will deduce simple theorems from them, filtering out uninteresting ones. Precision/recall comparisons with theorems from standard textbooks has shown that MATHsAiD's judgement of interestingness results in a similar outcome to those of the authors of those textbooks.

MATHsAiD will enable mathematicians quickly to explore alternative axiomatisations, reject those that lead to unwanted theorems or fail to produce wanted ones, and even compare their relative ease of proof.

6.2 Automated Proof Refactoring

It is a common mathematical experience that well into the development of a new theory one realises that the initial definitions or axioms are sub-optimal. Either you decide to leave matters in this rather unsatisfactory state, or you commit to an extensive, tedious and error-prone programme of what programmers call

[7] There are other theorem synthesis systems, but most are focused on recursive theories. These are of more interest to computer scientists than mathematicians.

refactoring, i.e., of tracking the consequences of the changed definitions or axioms through a development of, perhaps, hundreds of lemmas, theorems and corollaries. If, however, the original theory was developed using a theorem prover, then there is the potential to have the prover automate the refactoring. It can identify just those theorems that might need to be changed, try to automate their proof in the changed environment, then report just those (we hope few) places which require human intervention. Prototype tools already exist for this function [Autexier & Hutter, 2005].

6.3 Automated Theorem Search

The field of Mathematics has existed for millennia. In its history it has accumulated a huge number of theorems — no individual can be acquainted with all of them. Any of these theorems might be just the lemma needed at some critical stage of a proof. How can the working mathematician discover such lemmas, given that they might use different notation, might require a small amount of bridging inference to link them to the current problem and might be in a completely different area of mathematics to the current conjecture? Provers such as Isabelle provide tools for conducting such lemma search, i.e., by trying to prove a proposed lemma from the previously proved theorems. It is, however, limited to those theorems already in its own library. The wider problem of lemma search over the internet, is the subject of the Mathematical Knowledge Management community (see `www.mkm-ig.org`), which has many publications in this area.

6.4 Automated Referee Assistance

Referees for mathematical journals are responsible for checking the correctness of the proofs in submitted papers. This is a time-consuming and enervating task. As a result, referees often take a long time to return their reviews; delays of the order of years, from submission to acceptance, are not unusual. The refereeing task is also error prone; many ‘proofs’ pass refereeing but are later found to be faulty. I often ask mathematicians how they check proofs. As I suspected, they do not painstakingly check each line. Rather, as mentioned in §4, they use their experience to identify where any type 2 error is likely to lie, then they analyse this critical sub-proof using a collection of common counter-examples and common missteps.

An alternative approach would be for authors to check the proofs in their paper by formalising them in a trusted automated prover. Each journal might licence provers that it considered trustworthy, for instance, due to their LCF style. Referees would merely have to check that the formalisation actually captured the intended concepts and that the automated proof really did prove the theorem claimed. Most of their referee effort could be directed to assessing the significance, originality, relevance and presentation of the results, leaving validity to the prover. Of course, this places an additional burden on authors. Their reward for this additional effort would be a quicker refereeing process and the confidence that no embarrassing error would be found in their proofs. The burden could be eased by not insisting that these proofs be from first principles. Rather, authors would be free to encode other mathematicians’ theorems as axioms in their formalisation.

Of course, referees would have to check that these third party theorems had been correctly encoded.

Ironically, this process of automated referee assistance might be initiated outwith mathematics. Theoretical papers in computer science, for instance, also contain proofs. However, (a) these proofs are usually simpler, so easier to formalise, than typical mathematics proofs and (b) the referees of these papers typically are less mathematically skilled. So, outwith mathematics, the need is greater and the task simpler. Success here, however, might then encourage experimentation within mathematics.

6.5 Automated Exam Marking

I once asked two mathematicians which mathematical task they would be happiest to have automated. Without collusion or hesitation, they simultaneously replied "exam marking". Automated provers are ideal for automating this task.

- Firstly, the kind of exam questions set to students are within the scope of state-of-the-art automated provers — much easier, for instance, than the proofs from novel, current research.
- Secondly, automated provers are well placed to deal with the variation of student's answers. It would be impossible to anticipate all possible correct answers and pre-store them. The theorem prover, on the other hand, can just prove that each step follows from the previous ones. If the gap between the steps is too large, then the proof attempt might fail even though the step is correct, but this is unlikely to be a serious practical problem.
- Thirdly, automated provers could be used to rank incorrect answers by classifying each proof step as correct or incorrect. Furthermore, if we formalised common missteps as 'malrules', i.e., syntactically well-formed, but semantically unsound rules, then we might be able to classify some errors and even to associate standard marking penalties with them.

Many problems remain, of course, such as how to input the student's answer into the prover, but, in principle, these problems are soluble.

7 Conclusion

We have argued that there are signs that mathematicians are becoming more interested in automated theorem provers. The main driver of this interest is the existence of simple theorems whose smallest proof is very large. Only with machine assistance can we both produce such proofs and ensure that they are correct. In particular, if LCF-style provers are used, then their proofs have a high assurance of correctness. An initial indifference and, sometimes, hostility to computer-produced proofs is giving way to curiosity and, sometimes, enthusiasm.

This interest creates new challenges for developers of automated provers. In particular, they must find ways to present computer-produced proofs that make them easy to understand — even when they are enormous. This requires the presentation of proofs at varying levels of granularity, so that mathematicians can get an overview of the proof but also explore them in more detail, as required. The presentations must highlight the key ideas of the proof and identify the hard bits.

A major barrier to the more widespread uptake of automated provers is that existing theory libraries contain only a relatively small, albeit growing, area of mathematics. Most interactive provers are accompanied by a growing library of theories developed with their aid. Perhaps the best known, oldest and most developed is the Mizar library [Rudnicki, 1992], but even this library is relatively very small compared to the vast body of mathematical knowledge. The QED Manifesto was a call to build a library of all known mathematics [Boyer, 1994], but this project floundered in a disagreement about logics. So, the immediate entry-level applications must only depend on the small number of libraries of pre-existing theories. We have discussed five such applications: automated theorem synthesis from axiomatic presentations; automated proof refactoring; automated theorem search; automated referee assistants; and automated exam marking.

As promised, we end by returning to discuss the reasons given by mathematicians for not using automated theorem provers.

1. **Logic proofs are too detailed and long.** Presenting a proof hierarchically, e.g., as a hiproof, can provide both an overview of the high-level structure of a proof, and the opportunity to investigate the proof in detail, explore the dependencies between its parts and link it to help information to explain the definitions and concepts involved. This would be a considerable improvement over the current situation with long and complex *human-produced* proofs. Currently, only a handful of mathematicians understand each such proof. A hiproof presentation would facilitate the understanding of a much wider group.
2. **Provers are insufficiently powerful.** The power of automated provers has increased significantly over the last few decades. We see the results of this increased power in the much more challenging theorems that are being proved with them. In certain areas, for instance, decidable domains requiring long and complicated calculations, automated provers are orders of magnitude faster than humans and are not prone to the kind of errors that plague human calculations. New libraries of proved theorems are gradually being added, but in the meantime one can always add any required third-party lemmas as extra axioms.
3. **Provers are too tedious to use.** Prover developers are gradually adding the functionality to simplify 'tedious' manipulation. Such developments would be stimulated by customers identifying such problems and demanding their solution.
4. **Provers are hard to use.** Similar remarks apply as to the last point. Bogdan Grechuk, a mathematician, who was a Visiting Researcher in my group, has tried to address this problem with a short Isabelle primer aimed at the mathematician user [Grechuk, 2010].
5. **Why give up the fun of proving?** As we have seen in §6, there are a variety of potential applications for provers that automate aspects of mathematics that are not fun. These will leave more time for the fun bits.

References

[Appel *et al*, 1977]	Appel, K., Haken, W. and Koch, J. (1977). Every planar map is four colorable. I: Discharging. *Illinois J. Math*, 21:429–490.
[Aschbacher, 2005]	Aschbacher, M. (2005). Highly complex proofs and the implications of such proofs. In *The nature of mathematical proof* [Bundy *et al*, 2005a], pages 2401–2406.
[Autexier & Hutter, 2005]	Autexier, S. and Hutter, D. (2005). Formal software development in MAYA. In *Mechanizing Mathematical Reasoning*, volume 2605 of *LNCS*, pages 407–432. Springer, Berlin/Heidelberg.
[Boyer, 1994]	Boyer, R. et al. (1994). The QED manifesto. In Bundy, A., (ed.), *Automated Deduction, CADE 12: 12th International Conference on Automated Deduction*, volume 814 of *LNCS*, page 238251. Springer-Verlag.
[Bundy *et al*, 2005a]	Bundy, A., Atiyah, M., Macintyre, A. and Mackenzie, D. (2005a). The nature of mathematical proof. *Philosophical Transactions of the Royal Society*, 363(1835).
[Bundy *et al*, 2005b]	Bundy, A., Basin, D., Hutter, D. and Ireland, A. (2005b). *Rippling: Meta-level Guidance for Mathematical Reasoning*, volume 56 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press.
[Church, 1936a]	Church, A. (1936a). A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, pages 40–41 & 101–102.
[Church, 1936b]	Church, A. (1936b). An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363.
[Denney *et al*, 2006]	Denney, E., Power, J. and Tourlas, K. (May 2006). Hiproofs: A hierarchical notion of proof tree. *Electronic Notes in Theoretical Computer Science*, 155:341–359.
[Fleuriot & Paulson, 1999]	Fleuriot, J. D. and Paulson, L. C. (1999). Proving Newton's Propositio Kepleriana using geometry and nonstandard analysis in Isabelle. In *Automated Deduction in Geometry 1998*, volume 1669 of *Lecture Notes in Artificial Intelligence*, pages 47–66.
[Gonthier, 2008]	Gonthier, G. (December 2008). Formal proof — the four-color theorem. *Notices of the AMS*, 55(11):1382–1393.
[Gordon *et al*, 1979]	Gordon, M. J., Milner, A. J. and Wadsworth, C. P. (1979). *Edinburgh LCF - A mechanised logic of computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag.
[Gorenstein, 1982]	Gorenstein, D. (1982). *Finite simple groups: An introduction to their classification.* Plenum Press (New York).
[Gowers, 2000]	Gowers, W.T. (2000). Rough structure and classification. *GAFA: Geom. funct. anal.*, pages pp 1–39. Special volume.
[Grechuk, 2010]	Grechuk, B., (2010). Isabelle primer for mathematicians, Available from http://dream.inf.ed.ac.uk/projects/isabelle/.
[Hales, 2005a]	Hales, T.C., (2005a). The Flyspeck project fact sheet. http://code.google.com/p/flyspeck/wiki/FlyspeckFactSheet.
[Hales, 2005b]	Hales, T.C. (2005b). A proof of the Kepler conjecture. *Annals of Mathematics*, 162(3):1065–1185.
[Hales, 2008]	Hales, T.C. (January 2008). Formal proof. *Notices of the AMS*, 55(11):1370–80.
[Mackenzie, 2001]	Mackenzie, D. (2001). *Mechanizing Proof.* MIT Press.
[Martin, 1999]	Martin, Ursula. (1999). Computers, reasoning and mathematical practice. In Berger, Ulrich and Schwichtenberg, Helmut, (eds.), *Computational Logic, Proceedings of the NATO Advanced Study Institute on Computational Logic, Marktoberdorf, Germany, 1997*, volume 165 of *NATO ASI Series*, pages 301–346. Springer-Verlag.
[McCasland *et al*, 2006]	McCasland, R.L., Bundy, A. and Smith, P.F. (2006). Ascertaining mathematical theorems. *Electronic Notes in Theoretical Computer Science*, 151:21–38.
[Millo *et al*, 1979]	Millo, Richard A. De, Lipton, Richard J. and J.Perlis, Alan. (May 1979). Social processes and proofs of theorems and programs. *Communications of the ACM*, 22(5):214–225.

[Paulson, 1990]	Paulson, L.C. (1990). Isabelle: the next 700 theorem provers. In Odifreddi, P., (ed.), *Logic and Computer Science*, pages 77–90. Academic Press.
[Rudnicki, 1992]	Rudnicki, P. (1992). An overview of the Mizar project. In *1992 Workshop on Types for Proofs and Programs*, Bastad. Chalmers University of Technology. See http://mizar.org for up-to-date information on Mizar and the Journal of Formalized Mathematics.
[Turing, 1936]	Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society (2)*, 42:230–265.
[Wenzel, 2007]	Wenzel, M. (2007). Isabelle/Isar — a generic framework for human-readable proof documents. In Matuszewski, R. and Zalewska, A., (eds.), *From Insight to Proof — Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar, and Rhetoric*. University of Bialystok.