ProofPeer: Collaborative Theorem Proving

ProofPeer: Collaborative Theorem Proving

A Position Paper

Steven Obua Jacques Fleuriot Phil Scott David Aspinall

School of Informatics
University of Edinburgh
Scotland, United Kingdom

We define the concept of *collaborative theorem proving* and outline our plan to make it a reality. We believe that a successful implementation of collaborative theorem proving is a necessary prerequisite for the formal verification of large systems.

Submitted to:
arXiv.org

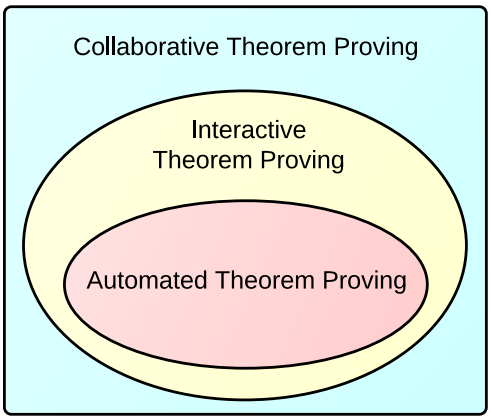

Figure 1: Collaborative Theorem Proving

	Duration	Number of people	Lines of verification code
Four colour theorem	2000 - 2005	1	60 000
seL4 microkernel	2004 - 2009	17	200 000
CompCert verified compiler	2005 - 2008	1 - 3	42 000
Feit-Thompson theorem	2006 - 2012	15	170 000
Flyspeck project	2003 - ?	16	325 000

Table 1: Major mechanised theorem proving efforts

1 The Challenge

In today's computerised and scientific era, making our claims *indefeasible* is more critical and more relevant than ever. We need to make indefeasible claims about the stability of mission critical hardware and software components [29, 32]. We need to make indefeasible claims about technology used in medicine where mistakes literally *kill* [14]. And we are now entering an age where malware is transforming from a nuisance into a weapon of warfare [31].

Mechanised Theorem Proving

In response to the challenge of making indefeasible claims, the fields of *automated* and *interactive theorem proving* have emerged. *Automated theorem proving* (ATP) is the mechanical checking of computerised proofs by mostly black box software components. *Interactive theorem proving* (ITP) builds on this by allowing human insight to guide and coordinate ATP systems in almost arbitrary ways as they struggle in non-trivial domains.

ITP has had noteworthy successes. It has been used to prove the *four colour theorem* [33] in Coq [4], the correctness of the *seL4 microkernel* [45, 27] in Isabelle [8], the correctness of a (large subset of a) C compiler in Coq [49], and most recently, the proof of the *Feit-Thompson odd order theorem* [34] in Coq. Finally, the complete mechanisation of the formal proof of the Kepler conjecture in HOL Light [7] is the ongoing goal of the *Flyspeck project* [38].

With these projects, we already have evidence that we can indefeasibly verify claims of both industrially relevant software *and* deep mathematical problems, *all using the same technology*. Table 1 lists the mentioned projects together with their duration and a rough estimate of their size in terms of lines of

verification code. The scale here is impressive and might come as a surprise to those outside ITP circles. However, these are toy examples compared to serious problems such as the verification of a software system as large as, say, the Linux operating system kernel. Here, we would need to tackle a problem that is based on (as of Linux 3.2) about *fifteen million lines of C code*, contributed by *over 1300 developers* and *over 200 companies*. The following seems obvious to us: *Linux itself is the result of a large collaborative effort, therefore verifying it (if possible at all) will require an* ***even larger collaborative effort***.

While we manage to develop software at scale, it is as yet unknown how to do verification at scale. We therefore propose to expand the capabilities of ITP by creating **collaborative theorem proving** (Figure 1). We shall take the lessons and technologies of interactive theorem proving and integrate the lessons and technologies of the *social web* [28], in particular: 1) *commons-based peer production* [23, 24] and *crowd-sourcing* [40], 2) *reputation systems* [53] and 3) *recommender systems* [54, 55].

The Web

One obvious step towards integrating theorem proving and web technology is to simply publish machine-checked proofs on the web. Prominent examples are the *Mizar Mathematical Library* [12] (MML) and the *Archive of Formal Proofs* [3] (AFP). Both libraries are hosted at a central location and managed by a small fixed committee to which mechanised proofs are submitted. There is an inevitable maintenance burden [36], as changes to the underlying theorem proving system might render previously valid proofs invalid.

An alternative approach is *Logiweb* [37], which dispenses with the central authority and advocates a distributed model. Each user hosts their own Logiweb server, and while they may reference proofs hosted on other servers, the curation of the theory libraries themselves is down to the individual.

The Social Web

The next step beyond mere publication of mechanised proofs on the web is to add interactivity. To this end, the goal of the newest version of the *Nuprl* ITP system [13] is to be primarily accessible as a web service on top of which web user interfaces could be implemented. A similar approach is to equip an existing ITP system with a web interface [42, 21, 50].

We think a truly *cohesive* system for *collaborative theorem proving* (CTP) will be obtained by going further than merely plumbing together existing ITP systems and web software. We define collaborative theorem proving via

CTP = The social machine of ITP.

The term *social machine* was coined by Berners-Lee in 1999 to describe processes on the web where people do the creative work and computers play the role of administrators and assistants [25]. This is exactly how we see collaborative theorem proving.

We are interested in applying the following ideas from the social web to ITP:

1. Commons-based peer production and crowd-sourcing. A prime example for *commons-based peer production* (CBPP) [23, 24] is *Wikipedia* [16]. It is regularly used by about half of the Internet population. The number of actual contributors is around 30 000 per month. Its success, and the success of the general wiki format, has inspired *Wikiproofs* [17], which in turn has been inspired by the *Metamath* [11] system, while ITP researchers have combined a wiki frontend with the Coq [19] and Mizar [57] systems.

Regarding mathematics, a smaller, but nevertheless significant experiment is the *Polymath project*, which uses an online forum open to anyone. The goal is to collaborate on finding new proofs of mathematical theorems. The first of a series of such projects was the density Hales-Jewett theorem, which

was started by a post on the blog of Fields Medallist Timothy Gowers as an example of what he calls *massively collaborative mathematics* [35].

Another example of CBPP, mentioned already, is the Linux kernel. This project spawned off the version control system *Git* which is the basis for another example of CBPP: *Github* [5]. Github is a site for hosting and sharing source code. It has around 3 million users sharing over 2.6 million public source code repositories.

Closely related to CBPP is its commercial cousin *crowd-sourcing* [40]. Examples for crowd-sourcing are sites like *Kaggle* [9], which has attracted sixty thousand scientists who compete to produce the best solutions in machine learning problems, and *Amazon mechanical turk* [2] which connects over 260 000 HITs (human intelligence tasks) to 3 million workers.

To apply CBPP and crowd-sourcing, it must be possible to decompose a task into many smaller tasks. There must then be a way to reintegrate the results of the smaller task into a result for the larger task. Mechanised theorem proving has unique advantages here. *It can **guarantee** two things: that the smaller tasks completed correctly; and that the reintegration is valid.*

This potential for crowd-sourcing proof is also pursued by the DARPA project *Crowd Sourced Formal Verification*[18] which started in 2011 and attacks the problem of formal verification of computer software by providing innovative user interfaces, for example in the form of iPad games. This project is different to what we are proposing in that we are focusing on extending ITP with the capabilities of the social web, while they are working on specific ways of making theorem proving accessible to laymen who might not even realise that their output is repurposed for theorem proving. It is imaginable that both approaches could be very fruitfully combined (see also Section 2 / Task 5).

2. Reputation systems. *Stack Overflow* [15] is a Q&A site for programmers, with around 1.5 million users, and a total of 4 million questions and 7.5 million answers. *Math Overflow* [10] is a Q&A site for mathematicians, with around 23 000 users and 40 000 questions asked and answered.

Both sites rely on commons-based peer production. To control for quality, they employ *reputation systems* [53]. Reputation is measured by *reputation points*, which are usually earned when a user adds content, such as questions, answers, links to news and comments, that other users "vote up". In this way, they increase the content producer's reputation, who is then afforded more power at the site. For example, they might earn the ability to delete items submitted by other users.

We will use reputation to instill *confidence*. Confidence is already guaranteed for mechanically verified theorems, but collaborative theorem proving is broader than this. Confidence is also needed in the process of library curation, burdening a centralised panel of expert curators (as in the case of the MML or the AFP). We intend to use reputation to replace this centralised authority by a dynamic committee consisting of peers with high reputation. We will also use reputation to deal with another important confidence issue. Peers are more likely to invest time on small problems within a large verification project if they are confident that their work will pay off in solving the larger problem. This means they must have confidence in the original analysis of the problem and its proposed decomposition into subproblems. They can gain this confidence if those proposing the decomposition have high reputation.

Using reputation to instill confidence in the decomposition mirrors the reality of large verification projects like Flyspeck and the mechanisation of the Feit-Thompson theorem, the decompositions of which are managed by Hales and Gonthier respectively.

3. Recommender systems. Hales estimates[1] that almost 50% of his time formalising the Jordan curve theorem and working on Flyspeck was spent looking up theorems in the HOL Light library, which

[1] personal communication by email

suggests we need much better search tools in ITP. Contemporary approaches are all pattern-based: given a search pattern, look up the corresponding matches in the library, and possibly take subpatterns into account. This approach is pursued, for example, with Whelp [20], MathWebSearch [46] and MIaS [56].

An alternative is to use *recommender systems* [54, 55]. These are software tools and techniques which provide suggestions for items of use to a user. For theorem proving, these items would typically be theorems, but they could just as well be *peers* of interest. Finding these theorems and peers will use the two main techniques of recommender systems (often combined in hybrid approaches): *collaborative filtering* and *content-based filtering*.

Recommender systems typically include a ranking component which can be coupled with reputation systems. They are widely deployed in the industry, e.g. collaborative filtering is used by *Netflix* [22] for making movie recommendations.

Preliminary work by Fleuriot on modifying and applying the Slope One item-based collaborative filtering algorithm [48] to the Jordan Curve Theorem corpus in HOL Light indicates that it can offer relevant theorem recommendations when tested against previously unseen (but already mechanised) results.

Content-based recommender systems are already an important part of ITP. Tools like Sledgehammer [26] use relevance filtering [51] and premise selection techniques [47] to scan the context of an interactive proof for useful theorems. Once identified, these theorems are passed to ATP systems which are often able to prove the goal. While the success rate is highly dependent on the quality of the context information, astonishing 40% success rates have been reported [43]. Other advanced machine learning techniques not only recommend theorems, but *proof patterns* [39].

2 Implementing Collaborative Theorem Proving

We think that:

Collaborative theorem proving *can tackle verification tasks of unprecedented magnitude.*

Under this assumption, the obvious question is: how do we make collaborative theorem proving a reality? In the remainder of this paper, we will outline our plan for doing so. In particular:

- We will prove that the technical requirements of CTP can be met, by defining mechanisms for collaborative theorem proving: these include 1) machinery for allowing commons-based peer production and crowd-sourcing, 2) a reputation system that applies to peers and the artefacts created by them, and 3) a hybrid recommender system based on *both* collaborative and content-based filtering which supports the theorem proving activities of peers and also feeds into automation.
- We will grow a community called **ProofPeer** which uses an experimental research prototype which implements these mechanisms for collaborative theorem proving. This prototype will be called ProofPeer, and will be hosted as a central hub at `http://proofpeer.net`. It will be accessible by any modern web browser and will not require any additional installation on the user's behalf. This keeps the barriers to entry low, so as not to discourage participants.
- We will show that the community can create a mechanically verified library of mathematical theorems. We will measure the speed at which this library is being created, the quality of the library and the size of the library, and compare these measurements to what we know about the growth of libraries for current popular proof assistants. Most importantly, we will measure the extent to which the mechanisms for CTP are being used to create this library, and try to determine the impact that these mechanisms have on growth speed, quality and size. Note that at this stage, it is not obvious how to define "quality". Coming up with an appropriate definition is part of the challenge.

Growing a Community

A key challenge of our plan is that we depend on having a community of peers that we can experiment on. Such a community will allow us to continuously gather data and combine theory and experiment in an iterative feedback loop. We will solve this problem by introducing students to ProofPeer as part of their course and project work and by making the system available to members of the research community. At some point, the system will be open to the public, and we will reach out to the wider audience to recruit peers. We must be able to cope with any potential influx of new users, so one of our goals is to have a system which can scale to arbitrary numbers of peers. To provide formalisation goals, we will setup many mathematical theorems as conjectures (see Task 1) in ProofPeer and award reputation points for proofs of them. This list will be seeded from Wiedijk's list of theorems [1] which contains theorems of wildly varying difficulty, ranging from basic theorems like the triangle inequality to celebrated theorems of mathematics like Fermat's last theorem. We will also add open problems like the Goldbach conjecture and problems that may be added in the future to the Polymath project.

We want to be flexible in our exploration of the design space of collaborative theorem proving. In particular we want to deal with the higher-level implications of collaboration and scale, and not so much with the lowest-level ones. We think we can best do this by creating a new theorem proving system which can run on top of a proven cloud computing platform like Google App Engine [6]. This platform already provides many important abstractions like a scalable data store and scalable session management. We also want a *quick start*; therefore, we strive to make our system compatible with HOL Light. This is a good choice of system for two reasons: firstly, it contains one of the largest libraries of mechanically verified mathematical theorems; secondly, it has a very simple and elegant implementation which can be feasibly incorporated with ProofPeer.

The following subsections break down our work programme in more detail.

Task 1: CBPP / crowd-sourcing

The main problem to solve for effective CBPP / crowd-sourcing is how to manage the decomposition of a task and the reintegration of the resulting smaller tasks. To this end, we will study mechanisms for *version control*, *continuous integration*, *conjectures / gaps* and *access control*.

We will define and implement a mechanism for **version control**. We have already experimented with such a mechanism and have partially implemented it on top of Google App Engine as a versioned file system based on 3-way-merge and the diff3 [44] merge algorithm for text files. We will need to determine how best to specialise such merging for theory files (written in ProofScript, see Task 4).

Closely connected to version control is the issue of **continuous integration**. Formal theories depend on each other, so a change in one theory may imply that depending theories do not mechanically verify anymore, or are subject to more subtle changes. We will be looking at mechanisms to reduce costly and time consuming rechecking of theories, exploit the potential for parallel and asynchronous processing [58], and also explore how far related work in the area of algebraic specifications [41] is applicable to our specific case.

A common phenomenon of formal verification is the presence of **conjectures** and more generally **gaps** in intermediate stages of the verification [30, 60]. We will study how to explicitly support such gaps in ProofPeer (this is interdependent with Task 4), and how to advertise the existence of such gaps to peers who might be willing to close them.

Finally, we will be studying **access control** mechanisms. Peers might prefer to keep certain formal verification projects private or only accessible by a certain group of peers. Access control is particularly

important for industrial verification projects where intellectual property is an issue. We will connect access control to the reputation system (Task 2) so that peers with high reputation can curate the library effectively.

Task 2: reputation system

We will define and implement a reputation system for peers based on points, which should track the value of a user's contributions. A contribution can be, for example, a verified theorem, a definition, a conjecture, or the closing of a gap (see Task 1). It might also be a more informal artefact such as a *comment* about another contribution, which itself might just be another comment.

The value of a contribution could be influenced by where, how often and by whom it is used. It can also be influenced by other peers explicitly voting on that value, e.g. either by "liking" or "disliking" a contribution. We will experiment with a special form of voting which consists of setting up a **bounty** for proving a conjecture (first).

There are links to the issue of version control from Task 1: how do we transfer direct votes from peers between different versions? Assume somebody made a change to the statement of a theorem. Will that negate all previous votes for this theorem? Or if a bounty has been set up for a conjecture, should that conjecture be allowed to change?

The goal of our reputation system is threefold: 1) to enable us to carve out a dynamic group of peers who have high reputation and who are granted special (access control) powers for library curation; 2) to motivate peers; 3) to serve as input to our recommender system (see Task 3).

Task 3: recommender system

There are two separate motivations for the use of recommender systems in ProofPeer.

1) To allow a peer to explore topics of interest to them. These might be theorems, but they might also be other peers. To characterise the topics, we will add **social tagging**, and will then experiment with a recommender system that uses both the tags and the reputation points as input for suggesting interesting topics to peers.

2) To support a peer during interactive theorem proving. We will define and implement a **context aware recommender system** that will suggest theorems and proof patterns that might be useful in a given theorem proving situation based on implicit and explicit observations. Our first iteration for such a recommender system will be content-based. We will then try to combine this content-based approach with the recommender system from 1). E.g., we might restrict the content we are using to that produced by peers with high reputation, or to content with related tags.

Task 4: ProofScript

We will define a language called **ProofScript** that will be the primary interface between peers and the collaborative theorem proving system. ProofScript will serve both as a **programming language** (similar in function to Standard ML in HOL-based systems) and as a **proof language** (similar to the *Isabelle/Isar* language). An important design constraint is that we need to make conversion from HOL light to ProofScript possible.

Starting from these initial design constraints we will codevelop ProofScript with the other features of ProofPeer. An example of why this is necessary lies in the fact that ProofScript should support **namespaces** which align with the hierarchal structure of our versioned filesystem. Another example is

that ProofScript should have language constructs for stating **conjectures** and marking **gaps** within the formalisation which can be tracked by ProofPeer.

We will implement an interpreter for ProofScript which is deployed both on the server (i.e., on Google App Engine) and the client. We will try to shift most of the computing to the distributed peer machines by allowing proof scripts to run not only on the server, but also on the client. This means devising a strategy for how server and client side work together. Ideally, the client will be responsible for *finding* proofs. *Checking* those proofs is the duty of the server.

Task 5: user interface

We will have the entire user interface running in a web browser. Our starting point will be the Clide [50] system which builds on the PIDE [59] approach to proof interaction. We will also investigate integration of the proof editor / viewer with our hiproof visualisation [52]. Furthermore, we must provide appropriate user interface abstractions for dealing with all features of ProofPeer like versioning, and we will need to seek straightforward and intuitive ways for users to interact with the features developed in Tasks 1–4. These features will be exposed as part of an application programming interface (API), which will be the sole means by which the user interface will communicate with ProofPeer. This means other researchers can use the API to connect their own applications and interfaces to ProofPeer in a seamless way. For example, innovative user interfaces like Xylem [18] for crowd-sourced proof via gaming could be integrated with ProofPeer so that certain proof obligations arising within ProofPeer are not handled entirely within the system, but are outsourced to Xylem instead, and then reintegrated with ProofPeer via proof certificates.

3 Conclusion

This position paper is based on our EPSRC grant proposal *EP/L011794/1*. The grant started in March 2014 and will allow us to work on making collaborative theorem proving a reality. We believe that one of the main applications of CTP will be in specifying and verifying large systems. And because growing a community is such an important aspect of CTP we believe that it is important to get in touch with possible beneficiaries of CTP such as systems designers as soon as possible.

Acknowledgements. Many thanks to everyone who commented on (earlier drafts) of our grant proposal and therefore indirectly also on this paper, in particular: Alan Bundy, Brian Campbell, Ursula Martin, James McKinna, Alison Pease, and Norbert Schirmer.

References

[1] *100 Theorems*. Available at `http://www.cs.ru.nl/~freek/100/`.

[2] *Amazon Mechanical Turk*. Available at `http://www.mturk.com`.

[3] *Archive of Formal Proofs*. Available at `http://afp.sourceforge.net`.

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

[5] *Github*. Available at `http://github.com`.

[6] *Google App Engine*. Available at `http://appengine.google.com`.

[7] *HOL Light*. Available at `http://www.cl.cam.ac.uk/~jrh13/hol-light`.

[8] *Isabelle*. Available at `http://isabelle.in.tum.de`.

[9] *Kaggle*. Available at http://www.kaggle.com.

[10] *Math Overflow*. Available at http://mathoverflow.net/.

[11] *Metamath*. Available at http://us.metamath.org.

[12] *Mizar Mathematical Library*. Available at http://www.mizar.org/library.

[13] *Nuprl*. Available at http://www.nuprl.org/html/NuprlSystem.html.

[14] *Open-source medical devices*. Available at http://www.economist.com/node/21556098.

[15] *Stack Overflow*. Available at http://stackoverflow.com/.

[16] *Wikipedia*. Available at http://www.wikipedia.org.

[17] *Wikiproofs*. Available at http://www.wikiproofs.org.

[18] *Xylem: Crowd-Sourced Gaming for Formal Verification*. Available at http://www.sri.com/work/projects/xylem.

[19] Jesse Alama, Kasper Brink, Lionel Mamane & Josef Urban (2011): *Large Formal Wikis: Issues and Solutions*. In: *Intelligent Computer Mathematics*, *LNCS* 6824, Springer, doi:10.1007/978-3-642-22673-1_10.

[20] Andrea Asperti, Ferruccio Guidi, Claudio Sacerdoti Coen, Enrico Tassi & Stefano Zacchiroli (2006): *A Content Based Mathematical Search Engine: Whelp*. In: *Types*, *LNCS* 3839, Springer, doi:10.1007/11617990_2.

[21] Andrea Asperti & Wilmer Ricciotti (2012): *A Web Interface for Matita*. In: *Intelligent Computer Mathematics*, 7362, Springer, doi:10.1007/978-3-642-31374-5_28.

[22] Robert M. Bell & Yehuda Koren (2007): *Lessons from the Netflix prize challenge*. *SIGKDD Explor. Newsl.* 9(2), doi:10.1145/1345448.1345465.

[23] Yochai Benkler (2002): *Coase's Penguin, or, Linux and The Nature of the Firm*. *The Yale Law Journal* 112(3).

[24] Yochai Benkler & Helen Nissenbaum (2006): *Commons-based Peer Production and Virtue*. *Journal of Political Philosophy* 14(6).

[25] Tim Berners-Lee & Mark Fischetti (2008): *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. Paw Prints.

[26] Jasmin Blanchette, Sascha Böhme & Lawrence Paulson (2013): *Extending Sledgehammer with SMT Solvers*. *Journal of Automated Reasoning* 51, pp. 109–128, doi:10.1007/s10817-013-9278-5.

[27] Timothy Bourke, Matthias Daum, Gerwin Klein & Rafal Kolanski (2012): *Challenges and Experiences in Managing Large-Scale Proofs*. In: *Intelligent Computer Mathematics*, *LNCS* 7362, Springer, pp. 32–48, doi:10.1007/978-3-642-31374-5_3.

[28] Ed H. Chi (2008): *The Social Web: Research and Opportunities*. *Computer* 41(9), doi:10.1109/MC.2008.401.

[29] Barry Cipra (1995): *How Number Theory Got the Best of the Pentium Chip*. *Science* 267(5195), p. 175, doi:10.1126/science.267.5195.175.

[30] Lucas Dixon & Jacques Fleuriot (2006): *A proof-centric approach to mathematical assistants*. *Journal of Applied Logic* 4(4), doi:10.1016/j.jal.2005.10.007.

[31] James P. Farwell & Rafal Rohozinski (2011): *Stuxnet and the Future of Cyber War*. *Survival* 53(1), pp. 23–40, doi:10.1080/00396338.2011.555586.

[32] James Gleick: *A bug and a crash*. Available at http://www.around.com/ariane.html.

[33] Georges Gonthier (2008): *Formal Proof - The Four-Color Theorem*. *Notices of the AMS* 55(11).

[34] Georges Gonthier et al. (2012): *Mathematical Components Project*. Available at http://www.msr-inria.inria.fr/Projects/math-components.

[35] Timothy Gowers & Michael Nielsen (2009): *Massively collaborative mathematics*. *Nature* 461(7266), pp. 879–881, doi:10.1038/461879a.

[36] Adam Grabowski & Christoph Schwarzweller (2007): *Revisions as an Essential Tool to Maintain Mathematical Repositories*. In: *Towards Mechanized Mathematical Assistants*, *LNCS* 4573, Springer, doi:10.1007/978-3-540-73086-6_20.

[37] Klaus Grue (2007): *The Layers of Logiweb*. In: *Towards Mechanized Mathematical Assistants*, *LNCS* 4573, Springer, doi:10.1007/978-3-540-73086-6_21.

[38] Thomas Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua & Roland Zumkeller (2010): *A Revision of the Proof of the Kepler Conjecture*. *Discrete and Computational Geometry* 44(1), pp. 1–34, doi:10.1007/s00454-009-9148-4.

[39] Jónathan Heras & Ekaterina Komendantskaya (2013): *Statistical Proof-Patterns in Coq/SSReflect*. *arXiv:1301.6039*. Available at `http://arxiv.org/abs/1301.6039`.

[40] Jeff Howe (2006): *The Rise of Crowdsourcing*. *Wired*. Available at `http://www.wired.com/wired/archive/14.06/crowds.html`.

[41] D. Hutter: *Management of change in structured verification*. In: *The Fifteenth IEEE International Conference on Automated Software Engineering, 2000. Proceedings ASE 2000*, pp. 23–31, doi:10.1109/ASE.2000.873647.

[42] Cezary Kaliszyk (2007): *Web Interfaces for Proof Assistants*. *Electronic Notes in Theoretical Computer Science* 174(2), doi:10.1016/j.entcs.2006.09.021.

[43] Cezary Kaliszyk & Josef Urban (2012): *Learning-assisted Automated Reasoning with Flyspeck*. *arXiv:1211.7012*. Available at `http://arxiv.org/abs/1211.7012`.

[44] Sanjeev Khanna et al. (2007): *A Formal Investigation of Diff3*. In: *Proceedings of the 27th international conference on Foundations of software technology and theoretical computer science*, *LNCS* 4855, pp. 485–496, doi:10.1007/978-3-540-77050-3_40.

[45] Gerwin Klein et al. (2009): *seL4: formal verification of an OS kernel*. In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ACM, doi:10.1145/1629575.1629596.

[46] Michael Kohlhase et al. (2012): *MathWebSearch 0.5: Scaling an Open Formula Search Engine*. In: *Intelligent Computer Mathematics*, *LNCS* 7362, Springer, pp. 342–357, doi:10.1007/978-3-642-31374-5_23.

[47] Daniel Kühlwein et al. (2012): *Overview and Evaluation of Premise Selection Techniques for Large Theory Mathematics*. In: *Automated Reasoning*, *LNCS* 7364, Springer, doi:10.1007/978-3-642-31365-3_30.

[48] Daniel Lemire & Anna Maclachlan (2005): *Slope One Predictors for Online Rating-Based Collaborative Filtering*. In: *Proceedings of 2005 SIAM Data Mining (SDM'05)*, abs/cs/0702144. Available at `http://arxiv.org/abs/cs/0702144`.

[49] Xavier Leroy (2009): *Formal verification of a realistic compiler*. *Commun. ACM* 52(7), p. 107–115, doi:10.1145/1538788.1538814.

[50] Christoph Lüth & Martin Ring (2013): *A Web Interface for Isabelle: The Next Generation*. In: *Intelligent Computer Mathematics*, *LNCS* 7961, pp. 326–329, doi:10.1007/978-3-642-39320-4_22.

[51] Jia Meng & Lawrence C. Paulson (2009): *Lightweight relevance filtering for machine-generated resolution problems*. *Journal of Applied Logic* 7(1), pp. 41–57, doi:10.1016/j.jal.2007.07.004.

[52] Steven Obua, Mark Adams & David Aspinall (2013): *Capturing Hiproofs in HOL Light*. In: *Intelligent Computer Mathematics*, *LNCS* 7961, doi:10.1007/978-3-642-39320-4_12.

[53] Paul Resnick et al. (2000): *Reputation systems*. *Commun. ACM* 43(12), p. 45–48, doi:10.1145/355112.355122.

[54] Francesco Ricci, Lior Rokach & Bracha Shapira (2011): *Introduction to Recommender Systems*. In: *Recommender Systems Handbook*, Springer, Boston, MA, pp. 1–35, doi:10.1007/978-0-387-85820-3_1.

[55] Toby Segaran (2007): *Programming Collective Intelligence: Building Smart Web 2.0 Applications*. O'Reilly.

[56] Petr Sojka & Martin Líška (2011): *Indexing and Searching Mathematics in Digital Libraries*. In: *Intelligent Computer Mathematics*, *LNCS* 6824, Springer, doi:10.1007/978-3-642-22673-1_16.

[57] Josef Urban et al. (2010): *A Wiki for Mizar*. In: *Intelligent Computer Mathematics*, *LNCS* 6167, Springer, doi:10.1007/978-3-642-14128-7_38.

[58] Makarius Wenzel (2012): *Asynchronous Proof Processing with Isabelle/Scala and Isabelle/jEdit*. *Electronic Notes in Theoretical Computer Science* 285, pp. 101–114, doi:10.1016/j.entcs.2012.06.009.

[59] Makarius Wenzel (2012): *Isabelle/jEdit – A Prover IDE within the PIDE Framework*. In: *Intelligent Computer Mathematics*, *LNCS* 7362, Springer, pp. 468–471, doi:10.1007/978-3-642-31374-5_38.

[60] Freek Wiedijk (2004): *Formal Proof Sketches*. In: *Types*, *LNCS* 3085, Springer, pp. 378–393, doi:10.1007/978-3-540-24849-1_24.