Preface

The production of high-quality scientific software is a legitimate and valuable contribution to the scientific infrastructure and to science itself. Scientific software can involve both numeric and symbolic calculations. Deductive software consists of symbolic routines for logical manipulations. It includes decision procedures, solvers, rewriters, model checkers, and theorem provers. Deductive software serves as the semantic foundation for a number of scientific and engineering applications in areas such as hardware and software verification, program synthesis and analysis, data- and knowledge-based systems, artificial intelligence, linguistics, and e-commerce.

QPQ, which stands for *QED Pro Quo*, is a forum for publishing, exchanging, and refining deductive software components. The refereed publication and distribution of such scientific software components in open source form yields higher quality, greater visibility, and accelerated productivity.

The QPQ deductive software repository is located at http://www.qpq.org/. It was officially launched at the First QPQ workshop at CADE-19. The development of the QPQ repository is funded by the US National Science Foundation under Grant No. EIA-0224465 and by SRI International. Dr. Mark Stickel (SRI International) is the founding editor-in-chief. The QPQ advisory board consists of an international panel of distinguished researchers in automated deduction. The repository consists of 33 different subfields, each with an associated editor. The QPQ web site was developed by John Pedersen, Patricia Morley, and Nadia Ghamrawi of SRI International.

The present volume collects together papers summarizing talks given at the First QPQ workshop. The workshop also included an invited presentation by Dr. Mark Stickel, and two discussion sessions on the organization and maintenance of the QPQ repository.

We thank NSF and SRI International for their generous support, and the CADE organizers, notably the conference chair Geoff Sutcliffe and the workshop chair Cesare Tinelli, for their help and encouragement.

July 2003

Michael Kohlhase
Natarajan Shankar
Mark Stickel

Table of Contents

Peopleware in Practice: Human Experiences in Tool Integration in ETI and Lessons Learned for QPQ

Tiziana Margaria[1,2]

[1] Chair of Programming Systems, University of Dortmund, Germany
Tiziana.Margaria@cs.uni-dortmund.de
[2] METAFrame Technologies GmbH, Dortmund, Germany
tmargaria@METAFrame.de

Abstract. The Electronic Tool Integration platform (ETI) is designed for the support of communities wishing to set up sites for the project-specific, domain-specific, public or private interactive experimentation with and coordination of heterogeneous tools.
This paper briefly sketches the experience of tool refereeing and integration in the last years, as a possible starting point for a discussion on how to manage the same issues in QPQ.

1 Background: The ETI Goals

Since 1997, the Electronic Tool Integration platform (ETI) supports communities to set up sites for project-specific, domain-specific, public or private interactive experimentation with and coordination of heterogeneous tools[1]. ETI managers, developers, and users are assisted by an advanced personalized Online Service guiding experimentation, coordination and simple browsing of the available tool repository according to their degree of experience. This allows even newcomers to orient themselves in the wealth of existing tools and to identify the most appropriate collection of tools to solve their own application-specific tasks.

The ETI project, which has been started in 1996, is intended to provide a Web-based, open communication platform for tool providers and end users. Here, tool providers can publish their tools and get valuable feedback by the end users. End users can compare different tools within an application domain and can combine functionalities of tools of different application domains to solve problems a single tool never would have been able to. A more detailed exposition, including background and related work can be found in [4], which describes the concrete instantiation for the ETI community associated to the International Journal on Software Tools for Technology Transfer (STTT) [6].

Similar to QPQ, the ETI Online Service plays a public service role, giving users the possibility of direct, *hands-on, experience* with a wealth of available tools and functionalities. Users and contributors are invited to report on their experience with the integrated tools in the context of the service as a

[1] The ETI platform is realized on top of the METAFrame environment [5, 3].

- *directory* for possible tools and algorithms satisfying totally or partially their needs,
- (vendor- and producer-) *independent test site* for trying and comparing alternative products and solutions without any installation overhead,
- *quality assessment site* for the published tools, which are refereed according to requirements like originality, usability, installability, stability, performance, design,
- *independent benchmarking site* for performance on a growing basis of problems and case studies.

Currently the STTT ETI Online Service comprises verification tools for real time systems and model checkers. The integration of programming language tools like type checkers, optimizers and code generators is on the way. The STTT ETI Service can be accessed via its homepage, `http://eti.cs.uni-dortmund.de`. From there, users can

1. **access online information** on the tools via hyperlinks to each tool's home site.
2. **access online a stand-alone version** of each tool, centrally located at the ETI service sites
3. **access the ETI repository** of integrated tools. It contains a collection of functionalities offered by individual tools, classified for ease of retrieval according to behavioural and interfacing criteria.
4. **experiment** at ease with the integrated tools and functionalities, by
   (a) *running the* (stand-alone or integrated) *tools* on libraries of examples, case studies, and benchmarks made available on the ETI platform,
   (b) *testing and running single tool functionalities*, capturing specific features offered by the integrated tools, on the same examples, from within a uniform graphical user interface provided by ETI,
   (c) *constructing own application-specific heterogenous tools* through combination of single functionalities coming from different tools within the ETI platform,
   (d) *loosely specifying coordination tasks*, which can be then automatically completed by means of ETI's coordination support. This, in particular, takes care of data format incompatibilities, as detailed in [2].
5. **experiment with own sets of data**, to be deployed in user-specific, protected home areas.

The wealth of input/output formats that accumulates in ETI over time makes correct tool combination extremely difficult for users who are not familiar with the tool landscape. ETI therefore provides coordination support in order to ease usability: based on its interfacing layer, which organizes a growing library of type transformers, whenever possible, type-incorrect tool sequences are automatically completed to directly executable ones. This mechanism, which is based on model synthesis for temporal logics, is hidden from newcomers. Experts, however, are able to investigate the full potential for type completion in order to flexibly exploit the entire tool repository.

In addition, ETI provides high-level task specification languages, graphical support for specifications and user interaction, as well as prototype animation. Together this eases the access and use of the functionalities offered by different tools, even if implemented in different languages of different programming paradigms (functional, imperative, object-oriented) and running on different platforms. Together with the loose specification of single functionalities (simply in terms of desired properties), this allows even newcomers to develop and test complex tool combinations in a comfortable, intuitive manner.

1.1 Division of Labor

A complex software like the ETI platform should be collectively maintained by a group of people with different, complementary profiles and skills. Five different groups of people are involved with the platform: end users, tool providers, tool integrators, platform developers, and site managers.

End Users They want easy, fast, and reliable access to the functionality provided by the ETI platform. The standard end users look for candidate tools or a combination of tool functionalities to solve a certain problem. Basically, they want to browse through the tool repository, evaluating available tool components and activities with respect to the own problem or application. Using the ToolZone client they are able to combine activities to coordination sequences and execute them. Additionally they can access tool-related information and discussion groups via a standard Web browser.

Tool Providers They are end users with a special focus: instead of searching for tools and activities, tool providers are principally interested in publishing the own tools and getting valuable feedback by the end users. In addition to the tool itself, tool providers supplies benchmarks defining the tool's profile and additional documentation material.

Tool Integrators They make new activities and types available to the ETI tool repository. They investigate the tool to be integrated, identify new activities, and establish connections between the new activities/types and already available ones. For this, they need a good understanding of the tool to be integrated as well as the application domain modelled in the tool repository. Integrators should be familiar with C++, since the tool management application is written in this language. To ease the integration task, we provide a process and utility tools that support and semi-automate integration steps (see [1]).

Platform Developers The ETI platform consists of an online service, the ETI community application, and of the ETI repository with the corresponding management services. Accordingly, platform developers can be split in two groups according to their responsibilities: one group provides new services and functionalities for the ETI community service, for which they need good Java knowledges; the other group adds new functionality to the ToolZone software. This second

group needs good Java and C++ skills, and a good understanding of the platform architecture.

Site Managers They build up and maintain an ETI site. This includes setting up the servers, installing the required software and customizing site-specific versions of the community application. They need skills concerning UNIX-based operating systems (e.g., Linux or SUN's Solaris), and the use the METAFrame environment and HTML to customize the functionality and appearance of the ETI Web application to the specific community needs.

2 Supporting the Tool Integration Process

The technical aspects of tool integration support have been already described in [1]. Here we concentrate on the process of handling the integration of new tools from the point of view of the interaction between the human actors: the tool providers and the ETI support team.

Potential tool providers contact the ETI manager either via the ETI Online service or at the ETI site. They usually first enquire about possibilities for integration, requirements on the tools and on them, and what kind of information they should provide.

Unfortunately, this differs from tool to tool. In particular it differs between

- *stand alone tools*, which are relatively easily installed on the technical platform, and
- *tool functionalities*, which are the basic building blocks for tool coordination. As such they are in the core of ETIs interests and quite likely of QPQ too. Functionalities comprise tool specific or generic pre- and post processing algorithms, verification engines, adapters and translators between different formats, and all the various elements that play a role in the computation, optimization, and communication in and around tools.

After some years of experience in this process, we can summarize the wishes of the tool providers as follows. The typical tool providers would like to go to the ETI website and find a link to some online site where they can check in the code they wish to provide (source or executable), plus some additional files (a mixed and varying collection of documentation, manuals, examples). Ideally, this should be all they need to do, i.e. they would like to come back later on this site and see that it is "all done".

This simple procedure is almost always unrealistic. In fact, it implies that someone

1. has taken over the code (sources/executable) and the documents,
2. has classified and deposited them in the adequate portions of the repository,
3. has taken over the *visible responsibility* for the installation of the software, including
   (a) code adaptions - if necessary, (and usually it is necessary),

(b) retrieving and installing additional packages or software components - if necessary,
(c) licensing them in case of need, - it is often needed, even in cases where the additional software is free and for free,
(d) ensuring and granting the availability of the installed tool(s) to the outside
(e) generic maintenance - e.g. taking care of updates of the environment, when needed
(f) support to external users - i.e. personnel ready to answer questions and able to take countermeasures in case something does not work properly. This is a serious burden, which requires building up expertise about the tool as software package and about its adequate usage in the originally foreseen (or in additional, non-foreseen) application domains.

4. has taken over responsibility for the *actuality* of the software: someone or something should pro-actively monitor further releases of the tool, notice when this is not any longer the most actual (stable) release and notify the tool providers.

Unfortunately, in many of the above points the ETI support team cannot take over responsibility[2], which therefore has to remain with the tool providers. This sometimes dampens the enthusiasm of potential tool providers, who often only then realize that "publishing" a tool is quite different from publishing a paper.

3 The ETI Tool Refereeing Process

In ETI, most tools arrive together with accompanying papers that describe the tool itself and one or more applications, usually through discussion of concrete examples or through case studies. Accordingly, there was so far no need for a tool-specific refereeing process distinct from the one for submitted papers: the merits of the tool contributions are discussed and evaluated as integral part of the paper reviewing process. In this context, the evaluation of a tool's quality is of scientific character: it concerns contributions to the advancement of the state-of-the-art of tool development in that application area. In this sense, no evaluation is performed of the tool as software artefact *prior to integration*. Indeed, the whole integration process plays instead the role of a collaborative evaluation of the tool as a piece of craftmanship. In this process all the issues concerning a tools architecture, modularity, openness, interfaces, etc... naturally emerge, and their more or less elegant solution determines and influences the experimental value of the contributed tool and the effort needed for its integration.

So in the particular ETI context, where there is no need of separate acceptance of tool contributions, this is carried on under the supervision of the ETI Manager. Once the ETI Manager is contacted by the tool providers, the discussion efforts concern the technical characteristics and modalities of cooperation

[2] This is mostly a matter of resources, but involves also legal aspects.

for an integration in the repository that exposes the maximum value to the end-users.

The situation may be different for QPQ, due to the absence of an accompanying "traditional" paper contribution and the presence of a refereeing process on the tools themselves.

4 Lessons Learned

4.1 Integration and Support

In the years since ETI's launch we have collected varied experiences concerning the interaction between the ETI support team, tool providers, and ETI users. They concern the technical aspects of communication and support, but also a surprisingly central social interaction component: *Peopleware*, in the sense of community building and fostering, plays an unexpectedly central role in the acceptance of the offered service, of the subjective evaluation of its value, and in the establishment of a sound, constructive cooperation between the members in different roles. Here follows a brief collection of the main observations, together with the approaches we followed to take adequate care of the feedback.

- Initially we had underestimated the importance for tool providers of the integration-related wishes mentioned above. In particular, we were not aware of the centrality of the "no effort" integration requirement on their side. It is therefore necessary to communicate a perceivable value of tool publishing: tool providers can be much more readily motivated to take up additional efforts once they discover and accept this value.
- Once tool providers have accepted that they must play an important part in it, it is still a matter of agreeing on a distribution of labour. Clearly, establishing who has to do what, evaluating how much effort this costs, and agreeing on the respective parts are tool specific matters. Clear communication of the support provided for tool integration and of the available automation capabilities was here of central relevance for the acceptance of a *cooperative* integration effort.
- More specific requirements, like pro-active monitoring of the actuality of the published release of a tool, have been discovered at a later time. Such requirements have emerged and were articulated only after a number of integrated tools had been available for a while. At that point the legitimate concern arose that the published version may not reflect any more (in an acceptable way) the capabilities and performance of the latest releases, and thus even potentially harm the reputation of a tool.
- The notion of "acceptable minimum level of support" evolves steadily: the degree of support and automation offered in ETI has risen over the years. In particular it is now going to be enhanced via online services that take over some of the communication and history recording tasks. Particularly concerned are monitoring, notification, and general communication tasks. In practice, this requires a continuous development effort on demand, since it mainly arises in response to (maybe urgent) users requirements and feedback.

- Setting up and communicating well-defined procedures for management and monitoring of the repository's content as well as of access to it is important for transparency to the public. It also helps to canalize support requests to the right person.
- We came to the conclusion that an online service, whatever its nature, is likely to be perceived by the users as a *product*. This brings expectations that - consciously or not - quite closely match the expectations from a commercial, B2C product. As a consequence, in case the site has a different character, it must be made visible and clearly communicated. In particular, it is important to separate and perceivably distinguish a "stable" version of the site from any experimental version that contains recently developed, thus possibly unstable features. This clear distinction should concern the repository and the services connected to it, but also the tools made available there.
- Reactivity and responsiveness of the support team are vital. As for any (commercial, B2C) product, it is necessary to build from the beginning also for QPQ a good integration and support team. Equally important is to start from the beginning a process of observation, experience collection, and prompt reaction in the management, integration, and support procedures. It is important to make its existence and evolution clear to the users.

Even after several major integration projects, the expectation of "no effort" integration is still prevalent in potential tool providers. Accordingly, also in the future it will be necessary to communicate very clearly that tool integration in the ETI fashion is a *common task* of the tool providers and of the repository support team. Any distribution of responsibilities must leave several of the above tasks to the tool provider and its own support infrastructure.

4.2 Tool Refereeing: some Observations

The specialistic skills and the intensive work needed to evaluate a tool in a fair and competent way may turn out to be a hurdle in finding enough adequate referees. Two possible policies to remedy this may lead to a different character of the repository:

- One may resort to experts in the field that already know the tool very well and just recommend it for inclusion. In this case, instead of relying on a strictly independent refereeing, QPQ would end up following an "invited tool" acquisition policy.
- Alternatively, we must be aware that reviewers not familiar with a tool must spent considerable effort in its evaluation. This is likely to cause difficulties of scale, both in finding tools that are adequately documented and in finding reviewers that can devote so much time and effort.

In a preliminary enquiry among potential reviewers for the Peripherals Area of QPQ, the following differences with respect to traditional paper review for conference or journals were quite readily pointed out:

- a tool reviewer needs the right equipment, not just a pencil as for reviews of papers,
- a tool reviewer must learn to get along with some tool, which brings up the issues of user interface, ergonomy, esthetic criteria, sheer taste, personal level of "pain" in dealing with these issues,
- tool reviewing is a residential task, with high demands on equipment: one must carry out the review with a computer and (high speed) internet connection, while a normal paper can be read and annotated almost anywhere, in free moments, whenever it fits the schedule (which is a widely appreciated characteristic of reviewing),
- tool reviewing is a time demanding task: a tool is not just easily overflown and classified[3]
- tool reviewing a new thing, one has first to understand what are the important characteristics and metrics of evaluation, there is no "body of experience" to build upon. Meanwhile, several conferences have introduced tool papers and tool demonstrations as specific categories of submission, but such contributions are evaluated in the traditional way, on the basis of a paper, and not on the basis of direct experience with the tools themselves,
- tool reviewing is difficult to subdelegate, at least with good conscience, due to the danger of much higher subjectivity of judgement.

In any case, it would be advantageous for the QPQ management to prepare adequate documentation, guidelines, and support for the reviewers. It would be also important to devise some form of reward (whatever it be, it should be quite attractive) that compensates reviewers for their intensive engagement.

References

1. V. Braun, T. Margaria, C. Weise: *Integrating Tools in the ETI Platform*, [6], pp.31-48.
2. T. Margaria, V. Braun, J. Kreileder: *Interacting with ETI: A User Session*, [6], pp.49-63.
3. T. Margaria, B. Steffen: *Coarse-grain Component Based Software Development: The* METAFrame *Approach*, Proc. STJA'97, "Smalltalk und Java in Industrie und Ausbildung"10.-11. September 1997, Erfurt (D), ISBN 3-00-001828-X, pp.29-34.
4. B. Steffen, T. Margaria, V. Braun: *The Electronic Tool Integration platform: concepts and design*, [6], pp. 9-30.
5. B. Steffen, T. Margaria, A. Claßen, V. Braun: *The* METAFrame*'95 Environment*, Proc. CAV'96, Int. Conf. on Computer-Aided Verification - Juli-Aug. 1996, New Brunswick, NJ, USA, LNCS 1102, pp.450-453, Springer Verlag.
6. *Special section on the Electronic Tool Integration Platform*, Int. Journal on *Software Tools for Technology Transfer*, Vol. 1, Springer Verlag, November 1997

[3] Personally, I think that "overflowing and classifying" is not an advisable review policy, but it is practised, and probably the existence of this lightweight review chance is a reason why some people in some cases accept to review at all.

The QSL platform at LORIA

Olivier Bournez, Mohamed El Habib, Claude Kirchner,
Hélène Kirchner, Jean-Yves Marion, and Stephan Merz

LORIA, Nancy, France
http://qsl.loria.fr/

Abstract. The QSL project is aimed at the development of concepts, methods, techniques, and tools to increase the reliability and the quality of software-intensive systems. Within this project, we are constructing a platform of tools for validation and verification. Besides the tools themselves, it will be a repository of documentation and case studies. We eventually hope to foster cooperation of different teams that pursue common development projects using interoperating tools.

1 The QSL project

The QSL (*Qualité et Sûreté des Logiciels*) project pursued at the LORIA laboratory in Nancy is aimed at the development of concepts, methods, techniques, and tools to increase the reliability and the quality of software-based systems in a broad sense. Roughly a dozen teams within LORIA contribute to the project by carrying out operations on specific subjects such as the analysis of cryptographic protocols, verification and rewriting, an integration of the UML notation and the B method of formal system development, extreme programming and tele-cooperation and others; some of these operations involve teams from other French laboratories. The project was launched in 2000 and is expected to continue in its present form until 2006.

Complementing research on the individual operations within the QSL project, a federating goal is the creation of a platform of relevant verification and validation tools, developed either within LORIA (for example ELAN [1] or CASRUL [2]) or elsewhere. Initially, the platform serves as a one-stop Web portal that makes the tools available to a larger public, and acts as a repository for associated documentation, course material, and case studies. Over a longer term, the platform is intended to evolve toward support for joint development projects by teams who are geographically distributed and use different tools, possibly based on different logical theories. We hope that the availability of the QSL platform can be useful to assemble convincing success stories about the application of verification technology in development projects, to evaluate the relative strengths and weaknesses of different methods, techniques, and tools within their respective application areas, and to encourage the development of interoperable verification and validation components.

2 The current state of the QSL platform

A preliminary version of the QSL platform can be accessed at the URL `http://plateforme-qsl.loria.fr/`. It presently hosts around 30 tools and libraries. For every tool, a short summary and overall administrative information is provided via the Web interface, including a classification according to the ACM categories, home URLs, installation instructions, examples of use, and a FAQ. At present, this information is entered and maintained by local administrators at the LORIA site. However, the entire administration can be done remotely via the Web, and we will ask tool developers to maintain the information concerning their respective tools, and to use our site as a mirror for distribution.

Technically, the QSL platform is based on a 3-tier architecture. The actual data is kept in a PostgreSQL database, which was chosen because of its free availability and its complete functionality that includes checks for referential integrity. The middle tier is built on PHP scripts that act as an interface between the database on the server and the presentation based on HTML forms, both for queries and updates. The presentation tier is provided by the Apache Web server that allows users to connect to the QSL Platform.

In parallel, the tools are installed as executables on a server that presently makes them available to users at the LORIA site. We are currently working on an interface based on Servlets (using JavaServer Pages) that will let users run the tools on demonstration examples or on user-provided models, where allowed by license conditions. This interface is mainly intended for tool evaluation by remote users and for educational purposes, without exposing our site to possible security breaches.

3 The intended evolution of the QSL platform

The version of the platform described in section 2 is but a first step towards the development of a useful repository for verified system development. We foresee its evolution along two main axes that we now describe.

3.1 A repository of tools and case studies

We hope that the platform will evolve into a recognized Web portal as a repository of tools, developments, and know-how relevant to the theme of quality and reliability of software-intensive systems. In this way, we aim at hosting not only tools and libraries, but also development projects and case studies. Despite many common aspects, this focus on documenting the know-how of applying deductive software differentiates our project from the QPQ project, which is primarily interested in archiving peer-reviewed components of deductive software themselves.

This axis should lead to a presentation of validation and verification methods and tools that will hopefully attract new users as well as allow for a comparative

evaluation of different theories, techniques, and implementations. As an opportunity of disseminating formal methods, we encourage academics to make relevant course material available. Students will thus be able to use the QSL platform as a knowledge base that offers lectures as well as worked-out examples.

3.2 A platform for cooperation

Based on a ProofForge concept, we envision an environment that assists the cooperative development of proofs and verified components. Cooperation can be foreseen along two dimensions: firstly, joint projects can be established between geographically distributed teams. It will be even more challenging to enable cooperation involving different tools, possibly based on different logico-mathematical theories. While cooperation can greatly enhance the capabilities of deductive software components, it is in general nontrivial to issue a correctness certificate based on certificates coming from different tools. For a concrete example, let us mention the ongoing work around the integration of the interactive proof assistant coq [3] based on the calculus on inductive constructions, permitting verified development of efficient programs, and the rewriting system elan [1] that offers efficient and user-definable evaluation strategies. The combination of these complementary paradigms promises much better support for calculational proofs in coq; the concepts of "deduction modulo" and the ρ-calculus offer a sound and elegant theoretical basis for a combined certificate in this and similar cases. In general, we can certainly not expect a "silver bullet" that would make underlying incompatibilities disappear.

The key impediments to the achievement of a platform for cooperation are of organizational and of technological nature. In fact, the field of verification techniques and tools itself is subdivided in many different research topics, which are in rapid evolution. A natural tendency for every group working in this area is therefore to continue with their own research and tool development, trying to outpace the competitors within the given subfield (sometimes even at the risk of introducing inconsistencies). One of the central challenges of our project will be to instead convince the community of the benefits of cooperation and to provide the technical framework for its possibility and success. We are certainly not the first to advocate tools that integrate verification capabilities. The integration of decision procedures for various first-order theories into a coherent theorem prover has been a long-standing and fruitful research area of its own since the seminal works of Shostak, Nelson, and Oppen. More recently, the SAL [4] and VeriTech [5] projects attempt to combine the deductive capabilities of interactive theorem provers and of model checkers.

On the technological side, integration is hampered by the fact that most deductive software has been designed as stand-alone programs without well-defined programmatic interfaces. There are no widely agreed-upon standards to represent basic syntactic entities such as formulas, definitions, theories, lemmas or proofs. Initiatives such as the MathWeb [6] and OpenMath [7] projects should help to alleviate these incompatibilities. The model checking community has also started to use standard formats as in the IF [8] project or the Model Checking

Kit [9]. There also exist standard libraries exist for the manipulation of basic data structures such as BDDs [10] or terms [11]. The shift from stand-alone tools to the development of components and APIs and the adoption of standardized middleware for communication among heterogeneous platforms is certainly a prerequisite for integration. Finally, our platform will provide a basis for version management and for tele-cooperation between geographically distributed sites.

While progress on these technological issues is indispensable for tool integration, effective ways of user interaction are a problem that has not received enough attention by the community as a whole. However, realistic verification problems will be beyond the reach of fully automated tools, and users will need to interact at adequate levels of abstraction in order to gain the benefits of deductive analysis. Interactive theorem provers today are based on proof scripts that are too low-level to adequately express the users' intentions, and they are often not even portable across different versions of the same proof assistant. While some projects have attempted to design better user interfaces [12,13], we believe that more fundamental research into the underlying concepts and concrete languages offered at the user level are necessary. The QSL platform is intended as a testbed to enable this kind of research, and to encourage developers to design their tools with cooperation in mind.

References

1. http://www.loria.fr/equipes/protheo/SOFTWARES/ELAN/.
2. http://www.loria.fr/equipes/cassis/softwares/casrul/.
3. http://coq.inria.fr/.
4. http://www.csl.sri.com/projects/sal/.
5. http://www.cs.technion.ac.il/Labs/ssdl/research/veritech/.
6. http://www.mathweb.org/.
7. http://www.openmath.org/cocoon/openmath//index.html.
8. http://www-verimag.imag.fr/PEOPLE/async/IF/.
9. http://wwwbrauer.in.tum.de/gruppen/theorie/KIT/.
10. http://vlsi.coloradu.edu/~fabio/CUDD/cuddIntro.html.
11. http://www.cwi.nl/projects/MetaEnv/aterm/.
12. http://www-sop.inria.fr/lemme/pcoq/.
13. http://www.dcs.ed.ac.uk/home/proofgen/.

The Calculemus Research Training Network – A Short Overview*

Christoph Benzmüller

Saarland University, Saarbrücken, Germany

1 Introduction

Calculemus research aims at the integration of systems for symbolic computation and symbolic reasoning and is related to the QPQ initiative. It is in the spirit of both projects to discuss these connections and potential collaborations. The following text sketches the structure and scientific contributions of the Calculemus Research Training Network (Calculemus RTN; see Figure 1 for the partner sites) since its start in September 2000. It has been reproduced from the networks midterm report [22] and credit is due to all researchers of the Calculemus RTN. More than 28 young visiting researchers (with a sum approx. 150 financed person-months) have been supported by the network so far and approx. 47 senior researchers are involved in the training measures at the different partner sites.

2 Research Objectives and Results

The main research objective of the Calculemus RTN (and the broader Calculemus Interest Group founded in the mid 90s; see `www.calculemus.net`) is to foster the integration of deduction systems (DS) and computer algebra systems (CAS) both at a conceptual and at a practical level. The point of origin for this kind of research is a landscape of heterogeneous approaches and systems on both sides of the spectrum, where the diversity on the DSs side is probably greater than on the side of CASs.

Since its start in September 2000 the Calculemus RTN has contributed to the convergence of DSs and CASs through its research on unifying frameworks for encoding and combining computation and deduction, the identification of the architectural requirements for a new generation of reasoning systems with combined reasoning and computational power, and the prototypical implementation and application of the improved systems. However, a single predominant theoretical framework is currently not possible. Such an approach would particularly involve predominant solutions to the still rather diverging systems at both sides of the spectrum between DSs and CASs. Therefore a strong line of research in the Calculemus RTN focuses on the modelling and integration of CASs and

* This work is supported by the EU research training network CALCULEMUS (HPRN-CT-2000-00102) funded in the EU 5th framework.

- USAAR Saarland University, Saarbrücken, Germany (Jörg Siekmann and Christoph Benzmüller)
- UED The University of Edinburgh, Scotland (Alan Bundy)
- UKA Karlsruhe University, Germany (Jacques Calmet)
- RISC Research Institute for Symbolic Computation, Linz, Austria (Bruno Buchberger)
- TUE Eindhoven University of Technology, Netherlands (Arjeh Cohen) and University of Nijmegen, Netherlands (Henk Barendregt)
- ITC-IRST Instituto per la Ricerca Scientifica e Tecnologica, Trento, Italy (Fausto Giunchiglia)
- UWB University of Bialystok, Poland (Andrzej Trybulec)
- UGE Università degli Studi di Genova (Alessandro Armando)
- UBIR The University of Birmingham, England (Manfred Kerber)

Fig. 1. The CALCULEMUS RTN

DSs at the systems layer. In this research direction, significant progress has been made and several systems of project partners and other research institutes have been connected in order to form networks of cooperating mathematical service systems. The benefits and impacts of such integrations have been investigated in prototypical case studies.

The researchers of the CALCULEMUS RTN and the CALCULEMUS interest group also fostered the Mathematical Knowledge Management (MKM, EU MKM-NET) research initiative; see [40, 8]. This relatively young line of research adopts a broader perspective on the future of mathematics (e.g. research and publication practice, education, and knowledge maintenance) in the 21st century. A significant amount of CALCULEMUS research is MKM relevant and is currently being taken up in this community in order to adopt and integrate it into the MKM perspective.

The extensive research activities of the CALCULEMUS Network and the CALCULEMUS Interest Group are furthermore shown inter alia by three special issues of the Journal of Symbolic Computation [101, 4, 78] and the following international events: CALCULEMUS Symposium 2000 in St. Andrews, Scotland [69, 101], CALCULEMUS Symposium 2001 in Siena, Italy [78], CALCULEMUS Symposium 2002 in Marseilles, France [45, 49], CALCULEMUS Autumn School 2002 in Pisa, Italy [23–25, 129]. The CALCULEMUS Symposium 2003[1] will be held in September in Rome, Italy, and it will join IJCAR conference in 2004.

In the following paragraphs we sketch the highlights of the CALCULEMUS RTN since its start in September 2000; for more detailed reports to all tasks we refer to [22].

Task 1.1: Mathematical Frameworks TUE and Nijmegen University investigated type theory for the purpose of formalising mathematics: Barendregt and Geuvers [21] give an overview of type theory, how it is used to represent logic and

[1] `http://www-calfor.lip6.fr/~rr/Calculemus03/`

mathematics and what issues and choices come up. Type theory (encoded in OPENMATH) as a way for communicating mathematics is proposed in [20] and in [48] it is shown how a proof presentation can be generated from a formalised proof in type theory. This paper argues that 'formal contexts' in Coq can be used as a basis for interactive mathematical documents. This topic is also treated in [99]. An in-depth discussion of the various ways to treat computations in theorem provers is given in [19] and further related work is presented in [36].

The CALCULEMUS RTN has also studied other approaches to theorem proving and their capacities to integrate computations (see also [123]). This includes proof planning, as developed and employed by the nodes USAAR and UED. In the ΩMEGA system [104], at USAAR, symbolic calculations can be integrated into proof planning in two ways: (i) to guide the proof planner and to prune the search space by computing hints with control rules and (ii) to shorten and simplify the proofs by calling a CAS within the application of a method to solve equations. As a side-effect both cases can restrict possible instantiations of meta-variables. These approaches are discussed in [52, 107, 84, 105].

An investigation into the use of deduction for the implementation of correct computations within computer algebra system was considered at UGE and is presented in [1].

The THEOREMA system, developed at RISC, aims at providing one mathematical framework encompassing all aspects of algorithmic mathematics, notably the aspects of *proving*, *computing*, and *solving*; see [39, 37, 38].

In [70, 71] it is critically argued by UBIR that aspects of mathematical concepts, including procedural knowledge, are hard to reconstruct from the formalisation in deduction systems. This work points to limitations of the flexibility of mathematical representations which apply to all our current approaches.

Task 1.2: Definition of Mathematical Service The primary goal of this Task is the enhancement of existing computer algebra systems and deductive systems by turning them into *open* systems capable of using and/or providing mathematical services. After a preliminary analysis of the state-of-the-art of reasoning systems, it was decided to tackle the problem, in parallel, by a top-down and a bottom-up approach.

In the top-down approach, new infrastructures (both at the conceptual, specification, and architectural level) for the seamless integration of mathematical services have been investigated. This was intended not only for current systems, but also and in particular for future implementations. To this extent particular emphasis was on the definition of frameworks (languages, protocols, semantic specifications, architectural schemata) suitable for making mathematical services accessible over the web. The relevant top-down approaches are: OMRS (Open Mechanised Reasoning Systems) developed by UGE and ITC-IRST [2], LBA (Logic Broker Architecture) developed by UGE [6, 7], MathWeb-SB (MathWeb Software Bus) developed by USAAR [130], MathBroker developed by RISC [81]. These networks can themselves be coupled again as, for instance, exemplarily investigated in [128].

In the bottom-up approach, we have investigated how complex mathematical services can be built out of simpler ones. A particular emphasis has been devoted to decision procedures, and in particular to the integration of procedures specific for solving mathematical problems with deductive procedures. Examples for bottom up approaches are CCR (Constraint Contextual Rewriting) developed by UGE and MathSat [61, 11, 10, 9, 12], developed by ITC-IRST.

In Task 1.2 the CALCULEMUS network also closely cooperates with the EU project MONET. In MONET special ontologies comprising mathematical problems, queries and services have been defined and investigated.

Task 2.1: Integration of CASs and DSs via Protocols Cooperation among several software systems can be achieved with indirect, unidirectional and bidirectional communication. The goal of this task is to investigate how protocols can be defined to provide a semantics as well as soundness results for systems exchanging mathematical information. This definition hints at several other tasks in the CALCULEMUS RTN dealing with very similar problems. This is for example true when defining a context for a computation and is partly covered in Task 1. Unidirectional and bidirectional communication protocols are designed when coupling directly different modules. Although there are no direct links between the services with indirect communication, interaction is possible when systems can communicate with a common user interface, central unit, mediator or evaluator. This approach, which is partly based on a joint work with ITC-IRST on OMSCS (Open Mechanised Symbolic Computation Systems), has been investigated within the KOMET system at UKA see [44, 76, 55, 46].

A semantics can be provided by at least three approaches: (a) define a mathematical software bus, (b) define a context from which a semantic can be derived, (c) formulate the problem as a knowledge representation paradigm.

These approaches are shared by several of the partners. Indeed, they lead to introduce multi-agent systems, contexts, and ontologies to just quote a few features (see for instance the LBA and the MathWeb-SB).

Task 2.2: Enhancing the Reasoning Power of Computer Algebra Systems Enhancement of CAS with reasoning power can be attempted at different levels: (a) enhancement of CAS on the System Level, (b) enhancement of CAS on the Theory Level, and (c) enhancement of CAS on the User Level.

Direction (a) can be achieved by adding additional reasoning capabilities, i.e., logical inference systems, to algorithms built into the CAS. The Constraint Contextual Rewriting (CCR) framework developed by UGE can be used in order to integrate the evaluation mechanism of the CAS MAPLE with an appropriate decision procedure for checking side-conditions, see [1] and [5].

Direction (b) can be achieved by adding proven knowledge about CAS functions to the CAS knowledge base. The HR system, developed at UED, has been used to conjecture properties of functions available in the MAPLE algorithm library from empirical patterns detected in computational data produced by the CAS [53].

Direction (c) can be achieved by giving the CAS user the possibility to prove mathematical statements using proof techniques from logic within the CAS in addition to the computing facilities that each CAS offers. In the framework of the CALCULEMUS RTN, the work of RISC represents this aspect of CAS enhancement: The THEOREMA system, see [41], is an add-on package for the widespread and popular CAS *Mathematica* where the user formulates mathematical theorems and proves them entirely within the *Mathematica* environment.

Task 2.3: Enhancing the Computation Power of Deductions Systems UED investigated the combination of the proof-planner $\lambda Clam$ [102] with other systems for computationally costly tasks. This includes (a) an implementation of the GS flexible decision procedure system framework in (Teyjus) LambdaProlog and within the $\lambda Clam$ proof planning system [42] and (b) the integration of the $\lambda Clam$ proof-planner into the MathWeb-SB system [54].

UED also investigated the combination of systems to discover attacks to security protocols [108, 109]. This work makes use of computational power in that it generates a large number of clauses in its processing.

Further relevant work has been done in the $\lambda Clam$ proof-planner to construct very large and modular proof-plans for complicated real analysis theorems [65, 79, 80].

The ΩMEGA proof planner at USAAR has been coupled with different CASs via MathWeb-SB, see [107, 84, 105]. The ΩANTS approach to integrate CASs into mathematical assistant systems is sketched in [29, 28, 34, 35]. This work proposes an agent-based modelling of inference rules and external systems at a very basic level within theorem provers.

Finally, work done at UBIR and UGE which render techniques from automated reasoning highly efficient by using enhanced computational power are presented in [66–68] and [9, 12, 3]. Further relevant work is given in [100].

Task 3.1: Automated Support to Writing Mathematical Publications Typically, a mathematical publication contains the following ingredients: natural language text, mathematical formulae, formal text (i.e. definitions and theorems), proofs, examples (typically with computations), and graphics (tables, drawings, sketches, etc.). In the optimal case, a software system for supporting mathematical publications would support all these facets of mathematical publications. Several systems and languages have been used for case studies in this area:

(a) The MIZAR approach (at UWB) is based on two kinds of software which automate the process of writing formal mathematical papers: (i) software used to prepare an article as a formal text whose correctness is computer verified and (ii) the software for automatic (or semi-automatic) translation into natural language (particularly English); this includes also the software for translation into XML-based formats. The cooperation with other CALCULEMUS sites includes development of the MIZAR Mathematical Library (MML) and also the above mentioned translation into XML formats. Relevant publications are [88, 60, 16–18, 94]. Recently published MIZAR articles in the Journal of Formalized

Mathematics are [114, 74, 95, 63, 118, 73, 103, 15, 14, 64, 89, 97, 90, 112, 113, 98, 93, 117, 59, 115, 91, 92, 62, 116].

(b) THEOREMA is a prototypical software system designed to give computer-support to the working mathematician during all phases of mathematical activity. Several features qualify THEOREMA as a powerful system for creating mathematical publications entirely inside the system. "Classical" mathematical documents can be written that are intended mainly for printout, as for instance the thesis [126] or the conference papers [124], [125], and [127]. In the case studies, however, emphasis has been put on using the THEOREMA system for developing interactive lecture notes for university mathematics courses. Mostly since the THEOREMA language is very similar to the language used in "ordinary mathematics" the system is highly suitable for this approach, both in illustrating computation-based courses as well as in supporting proof-oriented courses.

(c) The OMDOC [72] content markup scheme which has been developed at USAAR, supports authors with writing formal mathematical documents including articles, textbooks, interactive books and courses. OMDOC allows to capture the semantics and structure of these documents. Various tools are available to transform OMDOC documents into other formats for presentation purposes (using, e.g., MathML) or to support inter-system communication (e.g., by transformation into the logic of a theorem prover).

(d) TUE has developed the MATHDOX tool supporting interactive mathematical documents. MATHDOX is based on DOCBOOK but also has similarities to OMDOC.

Task 3.2: Support to the Development of an Industrial-Strength Application of Formal Methods to Program Verification In addition to formal methods, which is undoubtedly the most important application area for our research, we have identified the education sector as another interesting application for DSs and CASs. Actually the systems THEOREMA (RISC) and ACTIVEMATH [87] (USAAR), which make use of tools and approaches developed in the CALCULEMUS RTN, are already employed in education practice. Another example is the MATHDOX tool developed at TUE since the next version of the interactive textbook *Algebra Interactive!* [51] will appear in this format.

Formal method applications currently pursued in the CALCULEMUS RTN include (a) an approach to support the verification of hybrid systems with the help of mathematical services in MathWeb-SB [27, 26], (b) the investigation whether specialised reasoning tools within the MathWeb-SB can fruitfully support the formal verification of information flow properties and error detection in security protocols [12], and (c) the application of proof planning in first-order linear temporal logic (FOLTL) to feature interactions as they arise in large telephone networks [50].

Task 3.3: Support to the Solution of Undergraduate Exam in Calculus and Economics In this Task we focus on simple, mathematics education oriented problems with a strong emphasis on the particular way the problems are solved, how interaction with the user is supported and how the solution is presented.

We analyse whether our systems can be employed in a user friendly and adequate way and whether the interaction and maths presentation capabilities of the systems are appropriate.

A task relevant case pursued at Nijmegen University compares how the problem of proving the irrationality of $\sqrt{2}$, which involves computations, can be proved in fifteen different theorem proving environments (including systems of the CALCULEMUS RTN) [123, 122, 106, 33, 105].

Among the case studies that are currently being started at USAAR are exercises from the German *Bundeswettbewerb Mathematik* and Calculus exercises being encoded and investigated in the ACTIVEMATH project. Empirical studies at USAAR investigates the phenomena of natural language dialog with mathematical assistant systems on proof exercises in naive set theory.

Task 3.4: Modelling of Existing Systems as Mathematical Services The work in this Task so far has concentrated both on developing the required infrastructure (languages, protocols, semantic specifications, architectural schemata) for making existing systems inter-operate, and on studying extensions and enhancements of the reasoning capabilities of some existing tools. The relevant contributions are: (i) MathSat framework developed at ITC-IRST [11, 10], (ii) the RDL (Rewrite and Decision procedure Laboratory), (iii) the LBA [6, 7, 128] developed by UGE, (iv) the modelling of existing systems, for instance, $\lambda Clam$ developed at UED [102], as mathematical services in MathWeb-SB developed at USAAR [54].

Further work at USAAR concentrates on the mediation of mathematical knowledge between the mathematical knowledge base MBASE, which has been integrated to the MathWeb-SB, and mathematical assistant systems such as ΩMEGA [56, 33, 32].

Task 3.5: Challenge Mathematical Problems During the work on the above tasks some challenging mathematical problems had to be tackled already, in order to have non-trivial working examples. Some of the examples were done either by single partner nodes or in collaboration between some of the nodes. The examples include: (i) Fundamental Theorem of Algebra [58, 57], (ii) Involutive Bases [47, 43], (iii) Exploration in Finite Algebra, (iv) The Residue Class Domain [82, 85, 83, 84], (v) Proving with Invariants [86], (vi) The Jordan curve theorem for special polygons, (vii) Continuous lattices [75], (viii) Order sorted algebras [120, 111, 119], (ix) Proofs in Homological Algebra, (x) Proofs in Graph Theory, (xi) Exploration in Zariski Spaces. Further related work is given in [30, 31].

3 Discussion

Prima facie it may appear disappointing that a predominant, single and uniform solution for the integration of deduction and computation is not possible and that the network places an emphasis on integration at the systems level (which requires support for heterogeneous problem representations). However, it is the

authors opinion that mathematical assistant systems, in particular those for theory exploration, generally have to find a good compromise between a well chosen degree of heterogeneity and flexibility of mathematical representations and the enforcement of representational uniformity. Finding "good" representations has been identified as a key issue in artificial intelligence and the author is convinced that it is important for mathematical theory exploration and mathematics education as well. Unfortunately many of todays deduction systems are still strongly afflicted with the spirit of Hilbert's program: the possibility to encode mathematics in a uniform, restrictive manner (e.g. based on set theory) does not imply the usefulness of representational uniformity for theory exploration.

Heterogeneity at the representations and the related systems layer, however, requires support for the semantically validated exchange of information and for transformations of representations (probably including algorithms and proof objects based on them) in various goal directions. For instance, semantical descriptions of system capabilities and uniform information exchange facilities can be used for making heterogeneous systems interoperable in "abstract" level proof development. Transformation mechanisms (if possible and available) may then be later used to generate proof objects in a uniform goal representation format. Alternatively the employed systems may be trusted in the context of their particular usage.

In short, the author claims that a well chosen degree of representational heterogeneity and flexibility should be considered a design requirement for mathematical assistant systems instead of a drawback. A cooperation with the QPQ project could therefore focus on the interoperability aspect and, in the long run, semantical capability descriptions could be envisaged for the systems registering to QPQ. For this, the description languages and ontologies for mathematical algorithms employed and developed in the MONET project and the result state ontologies for theorem provers in [110], for example, may serve as a first point of origin. The validity and quality of the semantical system capabilities descriptions could be taken into consideration in the refereeing process so that clients systems (e.g. in CALCULEMUS) could rely on them when accessing or obtaining service systems from the QPQ repository. QPQ could furthermore encourage the development of proof transformation tools between frequently employed representations.

4 Conclusion

Since several years the integration of tools for deduction and computation (but also other tools such as mathematical databases) has been identified by CALCULEMUS researchers as a promising way to built better mathematical assistant systems. A collaboration with the recently founded QPQ project appears promising, in particular, in the light of the networks current emphasis on integrations at the systems layer.

References

1. A. Armando and C. Ballarin. Maple's evaluation process as constraint contextual rewriting. In B. Mourrain, editor, *ISSAC 2001: July 22–25, 2001, University of Western Ontario, London, Ontario, Canada: Proceedings of the 2001 International Symposium on Symbolic and Algebraic Computation*, pages 32–37, New York, NY 10036, USA, 2001. ACM Press.
2. A. Armando, A. Coglio, F. Giunchiglia, and S. Ranise. The Control Layer in Open Mechanized Reasoning Systems: Annotations and Tactics. *Journal of Symbolic Computation*, 32(4), 2001.
3. A. Armando, L. Compagna, and S. Ranise. System Description: RDL—Rewrite and Decision procedure Laboratory. In *Automated Reasoning. First International Joint Conference (IJCAR'01), Siena, Italy, June 18–23, 2001, Proceedings*, volume 2083 of *LNAI*, pages 663–669, Berlin, 2001. Springer.
4. A. Armando and T. Jebelean, editors. *Calculemus: Integrating Computation and Deduction*, volume 32 (4) of *Special Issue of Journal of Symbolic Computation on Calculemus'99*, October 2001.
5. A. Armando and S. Ranise. Constraint Contextual Rewriting. *Journal of Symbolic Computation. Special issue on First Order Theorem Proving, P. Baumgartner and H. Zhang editors*, 2002.
6. A. Armando and D. Zini. Towards Interoperable Mechanized Reasoning Systems: the Logic Broker Architecture. In *AI*IA-TABOO Joint Workshop: 'Dagli Oggetti agli Agenti: Tendenze Evolutive dei Sistemi Software'*, pages 70–75, Parma, Italy, 2000. Reprinted in AI*IA Notizie Anno XIII (2000) vol. 3.
7. A. Armando and D. Zini. Interfacing Computer Algebra and Deduction Systems via the Logic Broker Architecture. In Kerber and Kohlhase [69], pages 49–64.
8. A. Asperti, B. Buchberger, and J. H. Davenport, editors. *Mathematical Knowledge Management, Second International Conference, MKM 2003*, Bertinoro, Italy, February 16-18 2003. Springer.
9. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In Voronkov [121], pages 195–210.
10. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Efficiently Integrating Boolean Reasoning and Mathematical Solving, 2002. Submitted to Journal of Symbolic Computation.
11. G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements. In Calmet et al. [45].
12. G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Bounded Model Checking for Timed Systems. In D. A. Peled and M. Y. Vardi, editors, *FORTE 2002: Conference on Formal Techniques for Networked and Distributed Systems*, volume 2529 of *LNCS*, pages 243–259, Houston, Texas, 2002. Springer.
13. M. Baaz and A. Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002*, volume 2514 of *LNAI*, Tblisi, Georgia, 2002. Springer.
14. J. Backer and P. Rudnicki. Hilbert basis theorem. *Formalized Mathematics*, 9(**3**):583–589, 2001.
15. J. Backer, P. Rudnicki, and C. Schwarzweller. Ring ideals. *Formalized Mathematics*, 9(**3**):565–582, 2001.

16. G. Bancerek. Development of the theory of continuous lattices in MIZAR. In Kerber and Kohlhase [69].
17. G. Bancerek, N. Endou, and Y. Shidama. Lim-inf convergence and its compactness. *Mechanized Mathematics and Its Applications*, 2(**1**):29–35, 2002.
18. G. Bancerek and P. Rudnicki. A Compendium of Continuous Lattices in MIZAR: Formalizing recent mathematics. *Journal of Automated Reasoning*, 29(**3**):189–224, 2002.
19. H. Barendregt and E. Barendsen. Autarkic computations in formal proofs. *Journal of Automated Reasoning*, 28(3):321–336, 2002.
20. H. Barendregt and A. Cohen. Electronic communication of mathematics and the interaction of computer algebra systems and proof assistants. *Journal of Symbolic Computation*, 32:3–22, 2001.
21. H. Barendregt and H. Geuvers. *Proof Assistants using Dependent Type Systems*, volume 2 of *Handbook of Automated Reasoning*, chapter 18, pages 1149–1238. Elsevier, 2001.
22. C. Benzmüller, editor. *Systems for Integrated Computation and Deduction – Interim Report of the Calculemus IHP Network*, Seki Technical Report. Saarland University, 2003. `http://www.ags.uni-sb.de/~chris/papers/E5.pdf`.
23. C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part I)*, number SR-02-07 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E2.pdf`.
24. C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part II)*, number SR-02-08 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E3.pdf`.
25. C. Benzmüller and R. Endsuleit, editors. *CALCULEMUS Autumn School 2002: Course Notes (Part III)*, number SR-02-09 in SEKI Technical Report, 2002. `http://www.ags.uni-sb.de/~chris/papers/E4.pdf`.
26. C. Benzmüller, C. Giromini, and A. Nonnengart. Symbolic Verification of Hybrid Systems supported by Mathematical Services. In Caprotti and Sorge [49]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.
27. C. Benzmüller, C. Giromini, A. Nonnengart, and J. Zimmer. Reasoning services in the mathweb-sb for symbolic verification of hybrid systems. In *Proceedings of the Verification Workshop - VERIFY'02 in connection with FLOC 2002*, pages 29–39, Kopenhagen, Denmark, 2002.
28. C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. An Agent-oriented Approach to Reasoning. In Linton and Sebastiani [77].
29. C. Benzmüller, M. Jamnik, M. Kerber, and V. Sorge. Experiments with an Agent-oriented Reasoning System. In *KI 2001: Advances in Artificial Intelligence*, Vienna (Austria), 2001.
30. C. Benzmüller and M. Kerber. A Challenge for Automated Deduction. In *Proceedings of IJCAR-Workshop: Future Directions in Automated Reasoning*, Siena (Italy), 2001.
31. C. Benzmüller and M. Kerber. A Lost Proof. In *TPHOLs: Work in Progress Papers*, Edinburgh (Scotland), 2001.
32. C. Benzmüller, A. Meier, and V. Sorge. Distributed assertion retrieval. In *First International Workshop on Mathematical Knowledge Management RISC-Linz*, pages 1–7, Schloss Hagenberg, 2001.
33. C. Benzmüller, A. Meier, and V. Sorge. Bridging Theorem Proving and Mathematical Knowledge Retrieval. In D. Hutter and W. Stephan, editors, *Festschrift in Honour of Prof. Jörg Siekmann*, LNAI. Springer, 2003. To appear.

34. C. Benzmüller and V. Sorge. Oants – an open approach at combining interactive and automated theorem proving. In Kerber and Kohlhase [69], pages 81–97.
35. C. Benzmüller and V. Sorge. Agent-based Theorem Proving. In *9th Workshop on Automated Reasoning*, London (GB), March 2002.
36. A. Bove and V. Capretta. Nested general recursion and partiality in type theory. In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001*, volume 2152 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2001.
37. B. Buchberger. Theorema: A short introduction. *Mathematica Journal*, 8(2):247–252, 2001.
38. B. Buchberger. Theorema: Extending mathematica by automated proving. In D. Ungar, editor, *Proceedings of PrimMath 2001 (The Programming System Mathematica in Science, Technology, and Education)*, pages 10–11, University of Zagreb, Electrotechnical and Computer Science Faculty, September 27-28 2001.
39. B. Buchberger, C. Dupré, T. Jebelean, K. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger. The *Theorema* Project: A Progress Report. In Kerber and Kohlhase [69].
40. B. Buchberger, G. Gonnet, and M. Hazewinkel, editors. *Mathematical Knowledge Management (MKM 2001) – Special issue of Annals in Mathematics and Artificial Intelligence,*. Kluwer, 2003. To appear.
41. B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru. A survey of the *theorema* project. In W. Kuechlin, editor, *Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation*, pages 384–391, Maui, Hawaii, July 1997. ACM Press.
42. A. Bundy and P. Janičić. A General Setting for Flexibly Combining and Augmenting Decision Procedures. *Journal of Automated Reasoning*, 3(28), 2002.
43. J. Calmet. Intas: Final report. Internal Report: http://iaks-www.ira.uka.de/iaks-calmet/intas.html, 2002.
44. J. Calmet, C. Ballarin, and P. Kullmann. Integration of deduction and computation. *Applications of Computer Algebra*, pages 15–32, 2001.
45. J. Calmet, B. Benhamou, O. Caprotti, L. Henocque, and V. Sorge, editors. *CALCULEMUS-2002: Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, volume 2385 of *LNAI*. Springer, 2002.
46. J. Calmet, F. Freitas, and G. Bittencourt. Master-web: An ontology-based internet data mining multi-agent system. In *Proceedings of SSGRR 2001, Computer & e-Business conference*, 2001.
47. J. Calmet, W. Hausdorf, and W. Seiler. A constructive introduction to involution. In R. Akerkar, editor, *Proc. Int. Symp. Applications of Computer Algebra - ISACA 2000*, pages 33–50. Allied Publishers Limited, 2001.
48. O. Caprotti, H. Geuvers, and M. Oostdijk. Certified and portable mathematical documents from formal contexts. In B. Buchberger and O. Caprotti, editors, *MKM 2001 (1st International Workshop on Mathematical Knowledge Management)*, Research Institute for Symbolic Computation, Johannes Kepler University, Hagenberg, September 24-26 2001.
49. O. Caprotti and V. Sorge, editors. *Calculemus 2002, 10th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning: Work in Progress Papers*, Marseilles, France, June 2002. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.
50. C. Castellini and A. Smaill. Proof planning for feature interactions: a preliminary report. In Baaz and Voronkov [13].

51. A. Cohen, H. Cuypers, and H. Sterk. *Algebra Interactive!* Springer, 1999.
52. A. Cohen, S. Murray, M. Pollet, and V. Sorge. Certifying solutions to permutation group problems. Submitted to a major international conference, 2003.
53. S. Colton. Making conjectures about maple functions. In Calmet et al. [45].
54. L. Dennis and J. Zimmer. Inductive theorem proving and computer algebra in the mathweb software bus. In Calmet et al. [45].
55. R. Endsuleit and T. Mie. Protecting co-operating mobile agents against malicious hosts. Internal Report 2002-8, University of Karlsruhe, 2002.
56. A. Franke, M. Moschner, and M. Pollet. Cooperation between the Mathematical Knowledge Base MBase and the Theorem Prover Omega. In Caprotti and Sorge [49]. Seki-Report Series Nr. SR-02-04, Universität des Saarlandes.
57. H. Geuvers, R. Pollack, F. Wiedijk, and J. Zwanenburg. A constructive algebraic hierarchy in coq. *Journal of Symbolic Computation*, 34(4):271–286, 2002.
58. H. Geuvers, F. Wiedijk, and J. Zwanenburg. A constructive proof of the fundamental theorem of algebra without using the rationals. In P. Callaghan, Z. Luo, J. McKinna, and R. Pollack, editors, *Types for Proofs and Programs, Proceedings of the International Workshop, TYPES 2000, Durham*, number 2277 in LNCS, pages 96–111. Springer, 2001.
59. M. Giero. On the general position of special polygons. *Formalized Mathematics*, 10(**2**):89–95, 2002.
60. G. Gierz, K. Hofmann, K. Keimel, J. Lawson, M. Mislove, and D. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, Berlin, Heidelberg, New York, 1980.
61. F. Giunchiglia, R. Sebastiani, and P. Traverso. Integrating SAT solvers with domain-specific reasoners. In Kerber and Kohlhase [69].
62. A. Grabowski. On the decompositions of intervals and simple closed curves. *Formalized Mathematics*, 10(**3**):145–151, 2002.
63. A. Grabowski, A. Korniłowicz, and A. Trybulec. Some properties of cells and gauges. *Formalized Mathematics*, 9(**3**):545–548, 2001.
64. E. Grądzka. The algebra of polynomials. *Formalized Mathematics*, 9(**3**):637–643, 2001.
65. A. Heneveld, E. Maclean, A. Bundy, A. Smaill, and J. Fleuriot. Towards a formalisation of college calculus. In Kerber and Kohlhase [69].
66. M. Jamnik, M. Kerber, and M. Pollet. Automatic learning in proof planning. Technical Report CSRP-02-3, University of Birmingham, School of Computer Science, March 2002.
67. M. Jamnik, M. Kerber, and M. Pollet. Automatic learning in proof planning. In F. van Harmelen, editor, *ECAI-2002: European Conference on Artificial Intelligence*, pages 282–286. IOS Press, 2002.
68. M. Jamnik, M. Kerber, and M. Pollet. LearnOmatic: System description. In Voronkov [121], pages 150–155.
69. M. Kerber and M. Kohlhase, editors. *Symbolic Computation and Automated Reasoning – The CALCULEMUS-2000 Symposium*, St. Andrews, UK, August 6–7, 2000 2001. AK Peters, Natick, MA, USA.
70. M. Kerber and M. Pollet. On the design of mathematical concepts. Cognitive Science Research Papers CSRP-02-06, The University of Birmingham, School of Computer Science, May 2002.
71. M. Kerber and M. Pollet. On the design of mathematical concepts. In B. McKay and J. Slaney, editors, *AI-2002: 15th Australian Joint Conference on Artificial Intelligence*. Springer, LNAI, 2002.

72. M. Kohlhase. OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *Proceedings of AI and Symbolic Computation, AISC-2000*, LNAI. Springer Verlag, 2000.
73. A. Korniłowicz and R. Milewski. Gauges and cages. Part II. *Formalized Mathematics*, 9(**3**):555–558, 2001.
74. A. Korniłowicz, R. Milewski, A. Naumowicz, and A. Trybulec. Gauges and cages. Part I. *Formalized Mathematics*, 9(**3**):501–509, 2001.
75. J. Kotowicz and Y. Nakamura. Go-board theorem. *Formalized Mathematics*, 3(**1**):125–129, 1992.
76. P. Kullmann. *Wissensrepraesentation und Anfragebearbeitung in einer logikbasierten Mediatorumgebung.* PhD thesis, University of Karlsruhe, 2001.
77. S. Linton and R. Sebastiani, editors. *CALCULEMUS-2001 – 9th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning*, Siena, Italy, June 21–22 2001.
78. S. Linton and R. Sebastiani, editors. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, volume 34 (4). Elsevier, 2002.
79. E. Maclean. Automating proof in non-standard analysis (ii). In *Proceedings of ESSLLI 2001*, Helsinki, 2001.
80. E. Maclean, J. Fleuriot, and A. Smaill. Proof-planning non-standard analysis. In *Proceedings of the 7th International Symposium on Aritifical Intelligence and Mathematics*, Fort Lauderdale, 2002.
81. Mathbroker - A Framework for Brokering Distributed Mathematical services. `http://poseidon.risc.uni-linz.ac.at:8080/index.html`.
82. A. Meier, M. Pollet, and V. Sorge. Exploring the Domain of Residue Classes. Seki Report SR-00-04, Fachbereich Informatik, Universität des Saarlandes, Saarbrücken, Germany, December 2000.
83. A. Meier, M. Pollet, and V. Sorge. Classifying Isomorphic Residue Classes. In Moreno-Díaz et al. [96], pages 494–508.
84. A. Meier, M. Pollet, and V. Sorge. Comparing Approaches to the Exploration of the Domain of Residue Classes. *Journal of Symbolic Computation, Special Issue on the Integration of Automated Reasoning and Computer Algebra Systems*, 34(4):287–306, 2002.
85. A. Meier and V. Sorge. Exploring Properties of Residue Classes. In Kerber and Kohlhase [69], pages 175–190.
86. A. Meier, V. Sorge, and S. Colton. Employing theory formation to guide proof planning. In Calmet et al. [45], pages 275–289.
87. E. Melis, E. Andres, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: A generic and adaptive web-based learning environment. *Journal of Artificial Intelligence and Education*, 12(4):385–407, 2001.
88. R. Milewski. Fundamental theorem of algebra. *Formalized Mathematics*, 9(**3**):461–470, 2001.
89. R. Milewski. Upper and lower sequence of a cage. *Formalized Mathematics*, 9(**4**):787–790, 2001.
90. R. Milewski. Upper and lower sequence on the cage. Part II. *Formalized Mathematics*, 9(**4**):817–823, 2001.
91. R. Milewski. Properties of the internal approximation of Jordan's curve. *Formalized Mathematics*, 10(**2**):111–115, 2002.
92. R. Milewski. Properties of the upper and lower sequence on the cage. *Formalized Mathematics*, 10(**3**):135–143, 2002.

93. R. Milewski. Upper and lower sequence on the cage, upper and lower arcs. *Formalized Mathematics*, 10(**2**):73–80, 2002.
94. R. Milewski and C. Schwarzweller. Algebraic requirements for the construction of polynomial rings. *Mechanized Mathematics and Its Applications*, 2:1–8, 2002.
95. R. Milewski, A. Trybulec, A. Korniłowicz, and A. Naumowicz. Some properties of cells and arcs. *Formalized Mathematics*, 9(**3**):531–535, 2001.
96. R. Moreno-Díaz, B. Buchberger, and J.-L. Freire, editors. *Proceedings of the 8th International Workshop on Computer Aided Systems Theory (EuroCAST 2001)*, volume 2178 of *LNCS*, Las Palmas de Gran Canaria, Spain, February 19–23 2001. Springer Verlag, Berlin, Germany.
97. A. Naumowicz. Some remarks on finite sequences on go-boards. *Formalized Mathematics*, 9(**4**):813–816, 2001.
98. A. Naumowicz and R. Milewski. Some remarks on clockwise oriented sequences on go-boards. *Formalized Mathematics*, 10(**1**):23–27, 2002.
99. M. Oostdijk. *Generation and Presentation of Formal Mathematical Documents*. PhD thesis, Eindhoven University of Technology, Sept. 2001.
100. S. Ranise. Combining generic and domain specific reasoning by using contexts. In Calmet et al. [45].
101. T. Recio and M. Kerber, editors. *Computer Algebra and Mechanized Reasoning: Selected St. Andrews' ISSAC/Calculemus 2000 Contributions*, volume 32(1/2) of *Journal of Symbolic Computation*, 2001.
102. J. D. C. Richardson, A. Smaill, and I. Green. System description: proof planning in higher-order logic with Lambda-Clam. In *CADE'98*, volume 1421 of *LNCS*, pages 129–133, 1998.
103. C. Schwarzweller. The binomial theorem for algebraic structures. *Formalized Mathematics*, 9(**3**):559–564, 2001.
104. J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer. Proof development with omega. In Voronkov [121], pages 144–149.
105. J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Irrationality of Square Root of 2 - A Case Study in OMEGA. Submitted to an International Journal, 2002.
106. J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, and M. Pollet. Proof development with omega: Sqrt(2) is irrational. In Baaz and Voronkov [13], pages 367–387.
107. V. Sorge. Non-Trivial Symbolic Computations in Proof Planning. In H. Kirchner and C. Ringeissen, editors, *Proceedings of Third International Workshop Frontiers of Combining Systems (FROCOS 2000)*, volume 1794 of *LNCS*, pages 121–135, Nancy, France, March 22–24 2000. Springer Verlag, Berlin, Germany.
108. G. Steel, A. Bundy, and E. Denney. Finding counterexamples to inductive conjectures and discovering security protocol attacks. *AISB Journal*, 1(2), 2002.
109. G. Steel, A. Bundy, and E. Denney. Finding counterexamples to inductive conjectures and discovering security protocol attacks. In *Proceedings of the Foundations of Computer Security Workshop*, 2002. Appeared in Proceedings of The Verify'02 Workshop as well. Also available as Informatics Research Report EDI-INF-RR-0141.
110. G. Sutcliffe, J. Zimmer, and S. Schulz. Communication standards for automated theorem proving tools. In *Proceedings of the Workshop on Agents and Automated Reasoning, 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003. To appear.

111. A. Trybulec. Many sorted algebras. *Formalized Mathematics*, 5(**1**):37–42, 1996.
112. A. Trybulec. More on the external approximation of a continuum. *Formalized Mathematics*, 9(**4**):831–841, 2001.
113. A. Trybulec. More on the finite sequences on the plane. *Formalized Mathematics*, 9(**4**):843–847, 2001.
114. A. Trybulec. Some lemmas for the jordan curve theorem. *Formalized Mathematics*, 9(**3**):481–484, 2001.
115. A. Trybulec. Introducing spans. *Formalized Mathematics*, 10(**2**):97–98, 2002.
116. A. Trybulec. On the minimal distance between sets in Euclidean space. *Formalized Mathematics*, 10(**3**):153–158, 2002.
117. A. Trybulec. Preparing the internal approximations of simple closed curves. *Formalized Mathematics*, 10(**2**):85–87, 2002.
118. A. Trybulec and Y. Nakamura. Again on the order on a special polygon. *Formalized Mathematics*, 9(**3**):549–553, 2001.
119. J. Urban. Free order sorted universal algebra. *Formalized Mathematics*, 10(**3**):211–225, 2002.
120. J. Urban. Order sorted algebras. *Formalized Mathematics*, 10(**3**):179–188, 2002.
121. A. Voronkov, editor. *Proceedings of the 18th International Conference on Automated Deduction (CADE-19)*, volume 2392 of *LNAI*, Copenhagen, Denmark, 2002. Springer.
122. F. Wiedijk. The fifteen provers of the world. Unpublished Draft available at `http://www.cs.kun.nl/~freek/notes/index.html`.
123. F. Wiedijk. Comparing mathematical provers. In Asperti et al. [8].
124. W. Windsteiger. Building up hierarchical mathematical domains using functors in *mathematica*. In A. Armando and T. Jebelean, editors, *Calculemus 99: International Workshop on Combining Proving and Computation*, volume 23(3) of *Electronic Notes in Theoretical Computer Science*, pages 83–102, Trento, Italy, 1999. Elsevier. CALCULEMUS 99 Workshop, Trento, Italy.
125. W. Windsteiger. A Set Theory Prover in Theorema. In Moreno-Díaz et al. [96], pages 525–539. extended version available as RISC report 01-07.
126. W. Windsteiger. *A Set Theory Prover in Theorema: Implementation and Practical Applications*. PhD thesis, RISC Institute, May 2001.
127. W. Windsteiger. On a Solution of the Mutilated Checkerboard Problem using the Theorema Set Theory Prover. In Linton and Sebastiani [77].
128. J. Zimmer, A. Armando, and C. Giromini. Towards Mathematical Agents – Combining MathWeb-SB and LB. In Linton and Sebastiani [77], pages 64–77.
129. J. Zimmer and C. Benzmüller, editors. *CALCULEMUS Autumn School 2002: Student Poster Abstracts*, number SR-02-06 in SEKI Technical Report, 2002.
130. J. Zimmer and M. Kohlhase. System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. In Voronkov [121], pages 144–149.

Meta Logical Frameworks and QPQ Position Paper

Carsten Schürmann

Yale University
New Haven, CT, USA
carsten@cs.yale.edu

Deductive software as opposed to numerical software is gaining more and more importance in industrial applications. Numerical software can be found in components prevalent in control systems that are built into cars, trains, space vehicles, and plants, recognition systems for voice, vision, and access control, but also our internet infrastructure such as load balancing servers, and routers. Regarding fault tolerance, safety, security, and correctness, many branches of industry are employing or are thinking of employing deductive software such as model checkers, theorem provers, type checkers, and counter example generators to improve designs cost-efficiently before releasing a product.

Complementary to the functionality commonly associated with deductive software is the question of how to represent derivations, proofs, traces of model checkers, counter examples, and other non-numerical domains if at all. Due to efficiency reasons, many implementations of theorem provers refrain from the explicit creation and maintenance of deductions, making it more difficult, almost impossible to connect and import deductions from one component to another, and consequently render independent third party verification of deductions impossible. For others that do, deductions are merely mathematical objects that can be manipulated, constructed, deconstructed, combined, exchanged, inspected and satisfy far reaching mathematical properties and laws.

The structure of deductions is in general richer then that of numbers. Questions about how to represent axioms, inference rules, hypotheses, conclusions, formulas, terms, quantifiers, and judgments, are often a matter of personal choice and preference, leading to a significant amount of syntactical and semantical ambiguity and prohibiting the free exchange and sharing of deductive objects among software components, or entire software systems. In fact once committed to a particular representation, interfacing a deductive software system to another component can be prohibitively difficult, and the associated cost is often thought to outweigh the expected benefits.

The deductions prevalent in deductive software components range from derivations in various logics, such as, temporal, first-order, or modal logics, to witness traces of model checkers. Ultimately, deductions should have adequate representations in software and should not only be described by the properties they satisfy. Of course, all of this complexity is mirrored in the deductions as well as the deductive software components whose implementations are tedious and often difficult to get right. Model checkers, for example, exploit symmetry properties to prune the state space, and theorem provers employ various optimization tech-

niques to make proof search feasible. The soundness of an implementation can be difficult to show, especially in the model checking case where it hinges on a complete traversal of the state space.

A source code repository such as QPQ in form of a passive digital library will clearly be of interest to many contemporary industrial applications. However, if it were also active in that it offers source code related functionality, as for example, operators that combine a heterogeneous set of software components into one single component that share one common data structure by means of partial evaluators and semantics preserving source code transformers, we can hope for a tighter integration of components, efficiency benefits, portability, and simplified maintainability. Combining deductive software components that are written in different programming languages but share the same core data structures can be a real challenge and requires semantic modeling of each programming language, with sound transformations into and from the semantic domain.

The interaction of heterogeneous software components adds another level of complexity to the repository of deductive software. While many safety properties can be inferred directly from a software component (given an appropriate semantic model of the programming language involved), there are others, often relevant for security, privacy, and secrecy, that can only be inferred from communication infrastructure or the protocols that are used to communicate between software component. CORBA or COM and DCOM, for example, are popular software architectures used in practice that interconnect remote applications and facilitate the exchange of data. Their meta-theoretic properties, however, remain informal and can in general not be used when reasoning about the behavior of a software system.

We therefore conjecture that a source code repository such as QPQ should distinguish between the correctness of data representations, the correctness of the functionality of a software component, and the correctness of the the communication infrastructure. To tackle these challenges, we advocate the use of meta-logical frameworks that provide

1. A rich, uniform, expressive, representation language to represent deductions. Logical frameworks, in particular LF [HHP93], are designed as a meta-language for representing logics, and therefore lend themselves as prime candidates for encoding deductions. More research is necessary to scale the present techniques to capture the semantics of real world programming languages, including Java, C#, ML, Haskell or Mathematica, and model the underlying communication infrastructure.
2. Tools to help analyze deductions, deductive software and their properties. Special-purpose reasoning tools that are custom-made for the underlying logical framework ensure that the source code of deductive software entered into the QPQ satisfies the accompanying meta theoretical claims. One example is the special-purpose theorem prover implemented in Twelf [Sch00] for LF.
3. Tools that can convert between different deductions, mapping them from one logical formalism into another. Different software components employ

different deductive systems and often one can be converted into another, for example, by subsumption or because two logics have different but equivalent formulations. One could also speculate that ultimately these conversion tools are useful for faithfully converting source programs from one programming language into another.

Even without automated reasoning techniques, a digital library based on QPQ must make source code available at a "suitable" level of abstraction. Programming language idiosyncrasies, global state, and other programming language related particularities make it often difficult to distinguish the essential from the unimportant, and consequently leave informal human software verification time consuming, tricky, and error prone. Consider, for example, the LINUX approach to open source. Although the source code can be inspected and analyzed by everybody, buffer overflow vulnerabilities in servers and exploits of faulty protocols persist to be hard problems. On the other hand, if its source code could be made accessible on higher levels of abstraction, automatic inspection and certification processes may arguably have better chances to success.

And finally, a digital library based on QPQ will have the most significant impact, if it also offers configuration management capabilities. Updates of deductive software should progress as autonomously as possible, while preserving the prescribed properties by the client side. This task includes again reasoning about deductive software and their properties. A sophisticated and reliable configuration management system seems to play a key role in the deployment and acceptance of such a repository.

In conclusion, deductions are becoming the new primitive data objects of coming generations of industrial software applications. I therefore advocate that a part of the research effort on digital libraries for deductive software should focus on the underlying theories, principles, and meta logical frameworks that model, analyze, reason, and verify deductions, programming languages, and communication infrastructures.

References

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, January 1993.

[Sch00] Carsten Schürmann. *Automating the Meta-Theory of Deductive Systems*. PhD thesis, Carnegie Mellon University, 2000. CMU-CS-00-146.

Resurrecting the ANALYTICA Theorem Prover

Edmund Clarke, Michael Kohlhase, Joël Ouaknine, Klaus Sutner

Carnegie Mellon University
\{emc|kohlhase|ouaknine|sutner\}@cs.cmu.edu

Abstract. The ANALYTICA system is a theorem proving system for 19^{th} century mathematics written on top of the MATHEMATICA computer algebra system. It was developed in the early 1990's by Xudong Zhao and Edmund Clarke and has since been dormant. In this note we describe recent work to resurrect the theorem prover, port it to newer versions of MATHEMATICA, to restructure and document the code, so that it can be submitted to the QPQ repository.

1 Introduction

The ANALYTICA system [CZ92,BCZ98] is a theorem proving system for 19^{th} century mathematics. It has been able to prove theorems from elementary calculus and number theory, including a chapter from Ramanujan's second notebook[Ber85]. The system was developed in the early 1990's by Xudong Zhao and Edmund Clarke and has since been dormant.

ANALYTICA is written on top of the MATHEMATICA computer algebra system [Wol02], a large commercial computer algebra system that offers a highly developed graphical front-end that facilitates communication with the kernel and that allows for the development of multi-modal electronic documents, so-called notebooks, that can contain code, text, graphics, and data. Notebooks can render mathematical formulae in near-typeset quality and provide an interactive mathematical canvas for the user. Moreover, Notebooks are symbolic structures that can be manipulated by the MATHEMATICA kernel like any other symbolic expression in the system. They can also be exported in LaTeX and MATHML format, and thus can be used to present very high quality printed representations. We suggest that this computational environment naturally supports the design and implementation of fairly complicated software systems using symbolic computation. A description of a similar effort in the area of computational automata theory can be found in [Sut02].

2 Porting the Code Base to MATHEMATICA Version 5

ANALYTICA was originally written for MATHEMATICA version 1.2, which lacked many of the features of current versions of the product. In particular, no graphical front-end was available and all communication to the kernel was handled by a text-based interface similar to a command shell. In our work on the ANALYTICA prover we made substantial use of three new capabilities of MATHEMATICA:

the notebook front-end We have used the front-end in the documentation of the code base (see Section 3), and as user interface: As formula output in the MATHEMATICA frontend approaches that of TEX and notebooks supports a powerful folding operation, ANALYTICA's original LATEX output routines for proofs are now obsolete and were deleted from ANALYTICA.

the external system interface JLINK is used for interfacing to knowledge exchange formats like OMDOC (see Section 4).

the native XML processing capabilities are used heavily in communication with XML based services.

the added symbolic computation capabilities in the MATHEMATICA kernel: For instance, support for symbolic summation and trigonometric simplification has dramatically improved in MATHEMATICA since version 1.2. Nonetheless, we have retained existing ANALYTICA modules for these areas as plug-ins, loadable on demand. These implementations are transparent to the theorem prover and can thus be used to document proofs and computations that would be opaque if carried out by MATHEMATICA's built-in version.

3 Documenting the Prover

The first step in working with the ANALYTICA code base was to convert the formerly 50 plus source files into two large notebooks, one each for the prover and knowledge base parts (see Section 4). Code for the prover is represented using a special SOURCE style sheet that tightly integrates the actual MATHEMATICA code with accompanying documentation, examples and test code. From the SOURCE style notebook one can automatically generate files that augment the MATHEMATICA help browser and provide online help for the ANALYTICA system. From the same source document one can also extract pure code files that can be bundled together with the online documentation into an add-on package. Installation of this package is very straightforward, and requires no more than to copy the package files to the appropriate place in the MATHEMATICA file structure.

Take for instance the original code fragment in Figure 1, which defines the `WeakSimplify` function (this is the central ANALYTICA simplifier, which implements much of the integration of the MATHEMATICA symbolic algebra capabilities into ANALYTICA).

In Figure 2, we can see the SOURCE style notebook entry for this code: Each function, functional, constructor, option and system variable has a section, which contains documentation, (optionally) some examples, test code, and finally the code itself. The decorations on the right hand side of the window are the manipulators of the folding mechanism for the sections (called "cells" in MATHEMATICA) in the front end. The last cell that contains the actual code is a so-called input cell and can be directly evaluated in the MATHEMATICA kernel.

Figure Figure 3 shows the rendition of this segment in the MATHEMATICA help browser, properly classified under "Simplification" and "Prover". The necessary information is directly generated from the upgraded source in Figure 3.

```
SimplifyRules :=
        Join[OperatorRules, AbsRule, ExpressionRules,
                MaxMinRules, EquationRules, InequalityRules];

(* WeakSimplify is used to simplify sub-formula and is
a tactic in StrongSimplify. *)

WeakSimplify[f_] :=
        Simplify[f //. SimplifyRules] /.
        RulesFromGiven /. RulesForRelations;
```

Fig. 1. A piece of original ANALYTICA Code

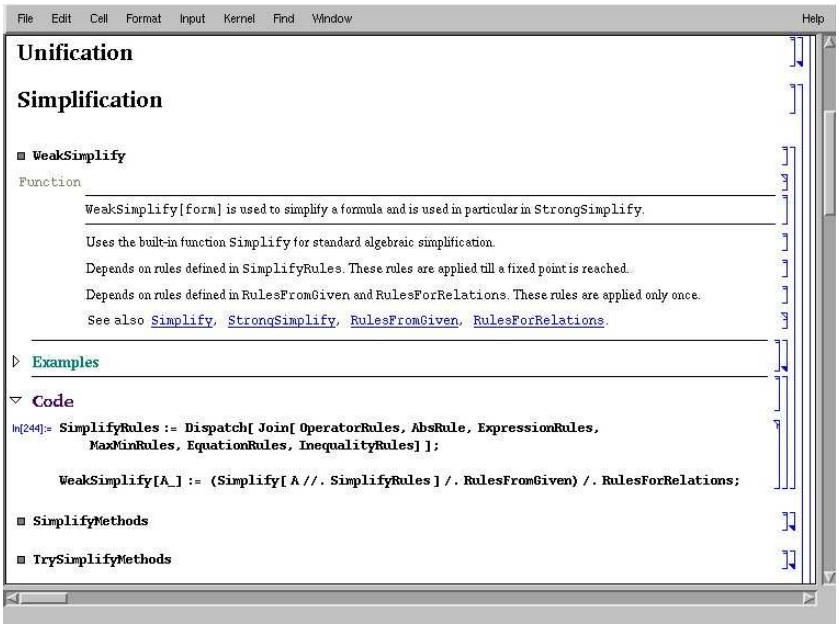

Fig. 2. ANALYTICACode in a source Notebook.

4 Separating Mathematical Knowledge from Code

There are two kinds of code in ANALYTICA: the program code from the prover we have discussed above, and mathematical knowledge used in proof search. To separate causes and make ANALYTICA easier to port to other mathematical domains, these are separated in the resurrected ANALYTICA. Originally, the mathematical knowledge used in ANALYTICA was represented as MATHEMATICA code of the form given in Figure 4.

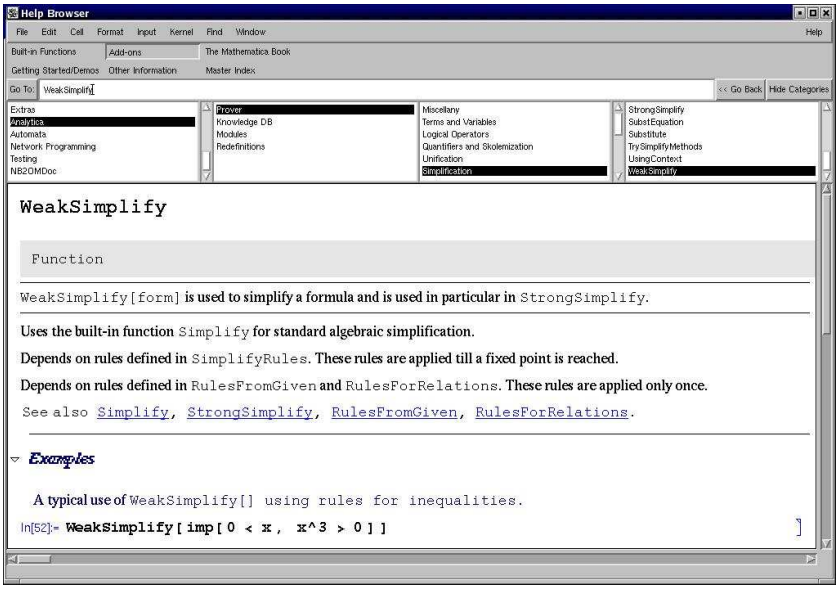

Fig. 3. The corresponding online help message in the MATHEMATICA browser.

```
(* Rules for simplifying expressions involving the absolute value function. *)
UnProtect[Abs];
Abs[a_ b_] := Abs[a] Abs[b];
Abs[a_^n_] := Abs[a]^n;
Protect[Abs];

(*  Local rule  used in simplification. *)
AbsRule = {Abs[a_] :> If[TrueQ[WeakSimplify[a >= 0]], a,
                If[TrueQ[WeakSimplify[a <= 0]], -a,
                        Abs[Factor1[a]]]]};
```

Fig. 4. Encoding Mathematical Knowledge in ANALYTICA

The first block specifies some rewriting rules for the MATHEMATICA symbol `Abs` that are subsequently be used by MATHEMATICA's built-in simplifier. The second code block specifies a rewrite rules used in a special simplification engine in ANALYTICA. The correctness of the ANALYTICA system depends on a couple of hundred of such rules.

These rules are now collected in a notebook using as special Knowledge Representation style that captures the information implicit in the original code fragments. We have used a variant of the `nb2omdoc` transformer [Sut03] to transform a KNOWLEDGE style notebooks into the OMDOC format (Open Mathematical DOCuments [Koh03]), an XML-based format for representing mathematical

knowledge in the large. OMDoc can be used as a basis for communicating with other mathematical software systems and in particular, the MBase mathematical knowledge base [KF01], which acts as an external knowledge repository for the new Analytica.

In the transformation we have made explicit and thus documented the mathematical knowledge used in Analytica. In the case of our example, this is given by the 4 theorems:

#	formalization	the absolute value function ...
1	$\forall a, b.\|a \cdot b\| = \|a\| \cdot \|b\|$	commutes with multiplication
2	$\forall a, n.\|a^n\| = \|a\|^n$	commutes with exponentiation
3	$\forall a.a \geq 0 \Rightarrow \|a\| = a$	is the identity on $\mathbb{R}^+$
4	$\forall a.a \leq 0 \Rightarrow \|a\| = -a$	is the negative identity on $\mathbb{R}^-$

The last line of the code in Figure 4 is purely pragmatic allowing to factorize polynomials inside absolute values, which of course preserves meaning. In the generated OMDoc representation, these theorems are represented in a special `assertion` element that combines the formalization with the natural language. As the Mathematica code fragments are also embedded, and Mathematica has a native Xml (and thus OMDoc) parser, Analytica can directly read the OMDoc documents served by the MBase system. We plan to augment the proof output of Analytica to point to the justifying theorems to make Analytica proofs independent of the Analytica prover itself. Eventually, we plan to supply proofs from first principles for all the knowledge used in the prover, so that Analytica proofs are grounded in axiomatics, as they should be for a theorem prover for mathematics.

5 Conclusion

We have described a recent effort to port the code base of the Analytica theorem prover to the newest version of the Mathematica language and to restructure it, so that it is more suitable for submission to QPQ. Analytica is an interesting example of a QPQ submission, since it is written not in one of the standard programming languages, but in a "mathematical development environment", and the new version of the system makes extended use of the new features of the Mathematica system for code documentation. The knowledge part of Analytica is translated to the OMDoc framework for mathematical knowledge representation. This general setup seems ideal for a knowledge-rich deduction component like the Analytica theorem prover. The port of Analytica to Source and Knowledge style notebooks, while laborious, is amply repaid by the increased ability to modify and extend the system, and to automatically generate a complete Mathematica add-on, including online help.

References

[BCZ98] A. Bauer, E. Clarke, and X. Zhao. Analytica — an Experiment in Combining Theorem Proving and Symbolic Computation. *Journal of Automated Reasoning*, 21(3):295–325, 1998.

[Ber85] B. C. Berndt. *Ramanujan's Notebooks, Part I*, pages 25–43. Springer Verlag, 1985.

[CZ92] Edmund Clarke and Xudong Zhao. Combining symbolic computation and theorem proving: Some problems of ramanujan. In D. Kapur, editor, *Proceedings the 11th Conference on Automated Deduction*, volume 607 of *LNCS*, pages 66–78, Saratoga Spings, NY, USA, 1992. Springer Verlag.

[KF01] Michael Kohlhase and Andreas Franke. MBase: Representing knowledge and context for the integration of mathematical software systems. *Journal of Symbolic Computation; Special Issue on the Integration of Computer algebra and Deduction Systems*, 32(4):365–402, September 2001.

[Koh03] Michael Kohlhase. OMDoc an open markup format for mathematical documents (version 1.2). Technical report, Computer Science, Carnegie Mellon University, 2003. forthcoming.

[Sut02] K. Sutner. `automata`, a hybrid system for computational automata theory. In J.-M. Champarnaud and D. Maurel, editors, *CIAA 2002*, pages 217–222, Tours, France, 2002.

[Sut03] Klaus Sutner. Converting mathematica notebooks to OMDoc. to appear in [Koh03], 2003.

[Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge Unviersity Press, 2002.

The Interface is the Message*

Sam Owre and Harald Rueß and Natarajan Shankar

Computer Science Laboratory
SRI International
Menlo Park CA 94025 USA
shankar@csl.sri.com
URL: http://www.csl.sri.com/~{ruess, shankar}/
Phone: +1 (650) 859-5272 Fax: +1 (650) 859-2844

A decision procedure for the satisfiability (validity) problem is a total predicate on a class of formulas that determines whether a given formula is satisfiable. It might seem that a simple binary interface for a decision procedure as a function that takes a formula and returns *Yes* or *No* according to whether the formula is satisfiable, would be adequate. On the contrary, we have observed that decision procedures need rich interfaces in order to be deployed most effectively. Decision procedures are frequently used in an online manner so that atomic formulas are added to a context incrementally, and claims are tested against the context. The application programmer interface (API) should include operations for asserting and retracting information, testing claims, and for creating, deleting, and browsing contexts. The decision procedures might need to exchange information with other inference procedures such as a rewriter, typechecker, or an external constraint solver. We discuss the basic issues in the design of modular interfaces for decision procedures that allow a rich range of interactions with other decision procedures and external applications. Our experience includes the integration of ground decision procedures, BDD packages, model checkers, and rewriters into PVS, and with the integration of ground decision procedures with SAT solving. The discussion here is mostly centered around the design of the API for ICS.

Boyer and Moore [BM86] indicate how even a simple decision procedure can have a complex interaction with the other components, so that it is not merely a black box that returns a yes/no decision. To quote Boyer and Moore, "*The black box nature of the decision procedure is frequently destroyed by the need to integrate it.*" They discuss the specific case of the interaction between the linear arithmetic package and the other routines in the Boyer–Moore theorem prover. Most binary decision diagram (BDD) packages for Boolean manipulation have fairly standardized set of operations for constructing BDDs corresponding to Boolean functions, constructing image and fixpoint BDDs, and converting BDDs to other representations.

The design of the application programmer interface is one of the hardest engineering challenges in the construction of usable decision procedures. The diffi-

* Funded by NSF Grants CCR-0082560 and CCR-9712383, DARPA/AFRL Contract F33615-00-C-3043, and NASA Contract NAS1-20334.

culty arises from anticipating and supporting a rich range of possible interactions without compromising efficiency and modularity. We identify a number of desirable features for decision procedures for building a library of automated reasoning routines. Many of these features have either been stated before or they are standard software engineering practice, but we still consider such a summary to be useful as a guideline and measuring stick for developers of inference procedures.

Incremental and Resettable. Inference procedures are commonly used as backend verification systems for formal methods tasks such as static analysis, type checking, abstraction, symbolic simulation, or proof search. In all these applications it is common to generate verification conditions relative to some context, which usually consists of a conjunction of formulas. In the symbolic simulation of software, for example, branch conditions are dynamically added to some context when processing a conditional statement. To allow for backtracking, such conditions must be retracted when exploring alternative paths. A binary decision procedure is ineffective in such a dynamic domain, since contextual information needs to be repeatedly reprocessed. Consequently, an effective inference procedure must be *incremental* and *resettable*, that is, it must be possible to add and remove atomic formulas without restarting the procedures. Nelson's thesis [Nel81] is a compendium of techniques for building incremental and resettable decision procedures.

Integrability. Combination methods in the style of Nelson and Oppen [NO79] and of Shostak [Sho84,RS01] explicate the APIs needed to integrate theory-specific decision procedures into a decision procedure for the union of theories. On the other hand, the combination of decision procedures with model checking in predicate and data abstraction involves the integration of complementary techniques [SS99].

We will investigate the case of combining ground decision procedures with propositionaly satisfiability (SAT) more closely [dMR02]. Let ϕ be the formula whose satisfiability is being checked. Let L be an injective map from fresh propositional variables to the atomic subformulas of ϕ such that $L^{-1}[\phi]$ is a propositional formula. We can use a SAT solver to check that $L^{-1}[\phi]$ is satisfiable, but the resulting truth assignment, say $l_1 \wedge \ldots \wedge l_n$, might be spurious, that is $L[l_1 \wedge \ldots \wedge l_n]$ might not be ground-satisfiable. If that is the case, we can repeat the search with the added *lemma* clause $(\neg l_1 \vee \ldots \vee \neg l_n)$ and invoke the SAT solver on $(\neg l_1 \vee \ldots \vee \neg l_n) \wedge L^{-1}[\phi]$. This ensures that the next satisfying assignment returned is different from the previous assignment that was found to be ground-unsatisfiable. The lemma that is added should be minimized so as to provide greater pruning of the search space. The ground decision procedure can be also be used to precompute a set Λ of lemma clauses of the form $l_1 \vee \ldots \vee l_n$, where $\neg L[l_1] \wedge \ldots \neg L[l_n]$ is unsatisfiable according to the ground decision procedures. The SAT solver can then be reinvoked with $\Lambda \wedge L^{-1}[\phi]$. For such an

integration to be most effective, both the ground decision procedures and the SAT solver must support the incremental introduction of information.

Checkability. How do we know that our inference procedures are sound? This question is often asked by those who wish to apply inference procedures in contexts where a high level of manifest assurance is required. This question has been addressed in a number of ways. The LCF approach [GMW79] requires inference procedures to be constructed as tactics that generate a fully expanded proof in terms of low level inferences when applied. Proof objects have also been widely used as a way of validating inference procedures [Nec97], and models may be generated to justify satisfiable outcomes.

The formal verification of decision procedures is actually well within the realm of feasible, and recently, there have been several successful attempts in this direction [FS02]. Using reflection, such a correctness proof may be used to produce proof objects for each decision problem by mere instantiation.

Other Issues. Managing failure, interruption, error reporting, resource management, and memory behavior across component boundaries remains a significant challenge for interface design. The API should include operations for observing and controlling the relevant behavior of the component, and for handling errors, failures, and garbage collection.

The ICS Interface. Based on previous experience, we addressed the interface issues as discussed above already in the early design stages of the ICS decision procedures(`ics.csl.sri.com`). There is a well-defined API for manipulating ICS terms, asserting formulas to the current database, switching between databases, and functions for maintaining normal forms and for testing the validity of assertions by means of canonization. This API is packaged as a C library, an Ocaml module, and a CommonLisp interface. The main function `(ics_process ctxt atom)` in the Lisp interface of ICS takes a logical context `ctxt` and an `atom`. Its results satisfies exactly one the recognizers `ics_is_consistent`, `ics_is_inconsistent`, or `ics_is_valid`. In case `atom` has not been shown to be valid in `ctxt` or inconsistent with `ctxt`, a representation of `ctxt` conjoined with `atom` is obtained using a destructor `ics_d_consistent`. In addition, ICS values can be freed in memory using `ics_deregister value`. We now develop a simple interface for hiding memory management issues by directing the Lisp garbage collector to free ICS value, which are, from the point of view of Lisp, just addresses in foreign space. In a first step, these addresses are wrapped into a Lisp structure, and the Lisp garbage collector is directed by a call to `(wrap-finalize! w)` to free the ICS value wrapped inside of `w` whenever `w` itself is garbage collected.

```
(defstruct wrap () address)

(defun wrap-finalize! (w)
```

```
(excl:schedule-finalization w
      #'(lambda (w) (ics_deregister (wrap-address w)))))
```

Since `ics_deregister address` does not immediately free memory, but only schedules the value at `address` to be garbage collected in the ICS memory space, in effect, this finalization scheme coordinates the action of two separate garbage collectors. Using wrappers, it is now straightforward to lift the ICS interface to a functional interface, which, also incorporates memory management. Function arguments and results are all wrapped, and newly generated ICS values are wrapped and registered at the LISP garbage collector.

```
(defun ics-empty-state ()
   (let ((empty (make-wrap (ics_context_empty))))
     (wrap-finalize! empty) empty))

(defun ics-process (state atom)
  (let* ((result (ics_process (wrap-address state)
                              (wrap-address atom))))
     (cond ((holds (ics_is_consistent result))
            (let ((newstate
                    (make-wrap (ics_d_consistent result))))
               (wrap-finalize! newstate)
               newstate))
      ((holds (ics_is_inconsistent result))
       :unsat)
      ((holds (ics_is_redundant result))
       :valid))))
```

Error handling on the Lisp level is provided by calling the ICS error handling function `ics_error` to get the function name `fname` and an error messages `msg`. These arguments are used to create a Lisp exception.

```
(ff:defun-foreign-callable ics_error (fname msg)
  (error (format nil "~a: ~a" (excl:native-to-string fname)
                 (excl:native-to-string msg))))
```

As a result, ICS can now be used in an opaque way for discharging proof obligations for formal methods applications developed in Lisp. Although the details of this integration are specific to Lisp—in fact, Allegro Common Lisp—garbage collection schemes in other modern languages usually include similar interfaces.

Conclusions. We have discussed some of the key issues in the design of interfaces for incremental, resettable, integrable, interactive, and verificable decision procedures. Many research challenges remain. We need to systematically catalog the possible interactions between inference procedures for equality, inequality, rewriting, propositional logic, quantificational logic, and model checking. The interface between decision procedures and libraries of formalized mathematical knowledge remains largely unexplored. The tradeoff between online and offline decision procedures is a topic that merits further study. There is also only a preliminary understanding about efficient proof production and representation. Decision procedures operate on a mathematically precise domain and should therefore be a touchstone for modern software engineering practice.

References

[BM86] R. S. Boyer and J S. Moore. Integrating decision procedures into heuristic theorem provers: A case study with linear arithmetic. In *Machine Intelligence*, volume 11. Oxford University Press, 1986.

[dMR02] Leonardo de Moura and Harald Rueß. Lemmas on demand for satisfiability solvers. Presented at SAT 2002, accepted for journal publication, May 2002. Available at `http://www.csl.sri.com/users/demoura/sat02_journal.pdf`.

[FS02] Jonathan Ford and Natarajan Shankar. Verifying Shostak. In A. Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction*, volume 2392 of *Lecture Notes in Computer Science*, pages 347–362, Copenhagen, Denmark, July 2002. Springer-Verlag.

[GMW79] M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF: A Mechanized Logic of Computation*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[Nec97] George C. Necula. Proof-carrying code. In *24th ACM Symposium on Principles of Programming Languages*, pages 106–119, Paris, France, January 1997. Association for Computing Machinery.

[Nel81] G. Nelson. Techniques for program verification. Technical Report CSL-81-10, Xerox Palo Alto Research Center, Palo Alto, Ca., 1981.

[NO79] G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–257, October 1979.

[RS01] Harald Rueß and Natarajan Shankar. Deconstructing Shostak. In *16th Annual IEEE Symposium on Logic in Computer Science*, pages 19–28, Boston, MA, July 2001. IEEE Computer Society.

[Sho84] Robert E. Shostak. Deciding combinations of theories. *Journal of the ACM*, 31(1):1–12, January 1984.

[SS99] Hassen Saïdi and N. Shankar. Abstract and model check while you prove. In Nicolas Halbwachs and Doron Peled, editors, *Computer-Aided Verification, CAV '99*, volume 1633 of *Lecture Notes in Computer Science*, pages 443–454, Trento, Italy, July 1999. Springer-Verlag.