LEO II: An Effective Higher-Order Theorem Prover

Prof. Lawrence C. Paulson
Computer Laboratory, University of Cambridge

1 Previous Research and Track Record

Lawrence C. Paulson is Professor of Computational Logic at the University of Cambridge, where he has held established posts since 1983. One of his main activities is developing proof tools. His early work made fundamental contributions to Prof. M. J. C. Gordon's proof assistant, HOL. In 1986, Paulson introduced Isabelle, a generic proof assistant. Isabelle supports higher-order logic (HOL), Zermelo-Fraenkel set theory (ZF) and other formalisms. Many developments are due to Prof. Tobias Nipkow's group at the Technical University of Munich. Automatic proof search, one of Isabelle's particular strengths, is however due to Paulson [17].

The designated Visiting Researcher, Dr. Christoph Benzmüller, is indispensable for this project. He is the principal architect of LEO, the only higher-order theorem prover to incorporate modern techniques. Benzmüller's previous work [11] is the starting point for the current proposal, which is to develop a new automatic theorem prover for higher-order logic. More generally, Benzmüller has an outstanding reputation in the field of automated reasoning. He heads the research group at Saarland University that is developing OMEGA, an integrated mathematics assistance environment.

The work will be done within the Cambridge Automated Reasoning Group. Hardware verification was pioneered here by Prof. Gordon and his students. They introduced what have become standard techniques, such as the use of higher-order logic to model hardware and software systems. The group's work continues to attract worldwide attention. Former members such as Dr. John Harrison have taken formal verification to Intel and other companies. The group has built two of the world's leading proof environments, namely HOL and Isabelle. Institutes using Isabelle as a basis for their research include the University of Edinburgh, Carnegie-Mellon University and Australia's Defence Science and Technology Organisation (DSTO). The Verisoft project, which uses Isabelle extensively, comprises 11 partners, including Infineon Technologies and BMW.[1]

The EPSRC has funded several projects at Cambridge involving Isabelle. They include the following:

[1] `http://www.verisoft.de/ProjectConsortium.html`

- *Verifying Electronic Commerce Protocols* (EPSRC ref. GR/R 01156/01), 2000–03. This project had the objective of verifying security protocols of industrial complexity. The huge SET protocol suite was analysed and some vulnerabilities found. The research assistant, Giampaolo Bella, investigated the Zhou-Gollmann non-repudiation protocol and a certified electronic mail protocol designed by Abadi et al. This project produced numerous publications.

- *Compositional Proofs of Concurrent Programs* (EPSRC ref. GR/M75440), 2000–03. This project investigated the verification of reactive systems using UNITY and the guarantees-calculus of Sanders and Chandy. It made much progress on the problem of formalizing states; toward this objective, it produced an untyped UNITY environment, formalized within Isabelle/ZF. A number of journal and conference papers have appeared.

- *Automation for Interactive Proof* (EPSRC ref. GR/S57198/01), 2004–07. This project aims to give interactive proof tools improved automation through an effective combination of interactive and automatic tools. It is developing techniques for transferring subgoals from an interactive prover to an automatic one and for transferring proofs in the opposite direction. This project has been underway for about 21 months. The basic system has been built, linking Isabelle to the resolution provers E, SPASS and Vampire. The next phase concerns reconstruction of the proofs within Isabelle.

This last project is the main inspiration for the new proposal. The integration between Isabelle and automatic provers is currently restricted to goals and lemmas that are expressed in first-order logic, when many problems are more naturally expressed using higher-order logic. Existing higher-order automatic theorem provers are not powerful enough for this application. The techniques exist now to build an automatic prover for higher-order logic that can cope with large problems; Isabelle generates problems that are very large indeed, but shallow.

2 Description of Proposed Research

The proposal is to build a new higher-order automatic theorem prover incorporating the lessons of recent research. We can expect a huge performance improvement over the current leading system, TPS [2]. The prover will be designed to solve problems of the sort that arise in verification. One design goal is that it should be easy to integrate with interactive verification tools such as HOL and Isabelle.

Background

Automatic theorem provers (ATPs) based on the resolution principle, such as SPASS [22] and Vampire [19], have reached a high degree of sophistication. They can often find long proofs even for problems having thousands of axioms. A fundamental limitation, however, is that they reason in first-order logic.

Higher-order logic extends first-order logic with λ-notation for functions and with function and predicate variables. It supports reasoning in set theory, using the obvious representation of sets by predicates. Higher-order logic is a natural language for expressing mathematics, and it is also ideal for formal verification. Function and predicate variables help express specifications concisely. In hardware verification, signals are often represented as functions over time, and circuits are represented by predicates relating a number of signals [14].

Interactive verification tools such as HOL, Isabelle and PVS use higher-order logic for the reasons noted above. Automatic provers for higher-order logic, however, are rare: the chief ones are TPS [1, 2] and LEO [10]. LEO is still an experimental prototype. TPS is based on a 20 year old architecture and it can only cope with very small problems. Higher-order logic is not inherently inefficient, but TPS uses a naive proof calculus and its search is unfocused. Modern techniques will yield an exponential performance improvement.

First-order ATPs are frequently used as tools by other researchers. A celebrated example is the recent Cambridge-MIT work [23] on breaking security application programming interfaces. Also impressive is Cohen's security protocol verifier [13], which uses SPASS [22]. Current higher-order ATPs are simply not good enough to be used in research. This project's main objective is to deliver such a tool.

Moving from first-order to higher-order logic requires a more complicated proof calculus, but it often allows much simpler problem statements. Higher-order logic's built-in support for functions and sets often leads to shorter proofs [1]. Conversely, elementary identities such as $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ turn into difficult problems when expressed in first-order form, as Benzmüller et al. [11] note. They describe a system where a higher-order theorem prover (LEO) cooperates with a first-order one (Bliksem), and their empirical results demonstrate the system's potential. The system attempts to remove higher-order features from the problem so that it can be solved efficiently by a first-order prover. This cooperation has many advantages: the higher-order prover does not have to duplicate the immensely complicated technologies used in first-order provers; it immediately benefits from enhancements made to the first-order provers; it may extend to co-operation with various decision procedures.

The existing LEO-Bliksem co-operation [11] demonstrates that the ideas are feasible, but it requires further work to become generally useful. Bliksem is no longer supported;[2] so another ATP must take its place. LEO is only available as a part of OMEGA [20], which is very large and complex; a stand-alone version of LEO would be easier for users to install, especially if they want to embed it in their own large system.

Higher-order logic can express both deep and shallow problems. Deep problems, such as Cantor's theorem, have difficult proofs: functions or predicates such as Cantor's diagonal function [2], must be constructed automatically. Shallow problems, which often arise in verification, may involve nothing more than straightforward reasoning involving λ-notation. The λ-calculus can be encoded in first-order logic, just as set theory can, but the resulting performance with first-order automatic provers is disappointing.[3] The approach taken by Isabelle's *blast* tactic [17] is to give a first-order prover a limited ability to reason about λ-expressions. A lean re-implementation of LEO could outperform *blast* while yielding a true higher-order automatic prover. The intrinsic value of higher-order logic makes the development worth while.

Related work in this area is scarce. Beeson's Otter-λ [3] extends the first-order theorem prover Otter [16] with λ-notation. This combination does not yield higher-order logic, and Otter can no longer be regarded as a high-performance ATP.

The proposal is for Benzmüller to visit Cambridge for a year to develop a lean new version of LEO. The design will be informed by the requirements of the Isabelle and HOL communities at Cambridge. The emphasis will be on solving shallow problems efficiently: this goal is feasible and it is relevant to verification. However, the logical calculi will be complete, which means that deep problems can also be tackled. We intend to make progress in both practical and theoretical aspects.

First-order theorem provers can be compared through test data drawn from the Thousands of Problems for Theorem Provers (TPTP) library [21]. Benzmüller and Brown [6] have recently announced a small set of higher-order problems, but its purpose is to discriminate among the variants of higher-order logic rather than to present challenges. A set of hard problems, graded by difficulty, will be a natural by-product of this project.

In summary, here are the *Project Objectives*:

1. to design and implement a high-performance automatic higher-order theorem prover

2. to collect a set of higher-order problems that can be used for evaluating and comparing higher-order theorem provers

[2]See `http://www.mpi-sb.mpg.de/~nivelle/software/bliksem/`

[3]Joe Hurd, private communication.

Programme and Methodology

We envisage the following workplan for the 12 months. Note that some of the tasks take place concurrently. Tasks 2, 3 and 4 are strongly connected.

Integration with Isabelle is not formally part of the programme. Although Isabelle's existing ATP linkup could easily be modified to deliver input to another ATP, reconstruction of the resulting proof is a hard problem. Paulson will work on this (initially without support) after the project finishes.

Task 1: Problem set.

We shall accumulate a corpus of problems that can guide the research. Some of these will be drawn from Isabelle problems, as is currently done for first-order problems in the EPSRC project *Automation for Interactive Proof*. We shall also solicit contributions from colleagues. One category of problems will be those where higher-order logic admits a more efficient formalization than is possible in first-order logic, providing a basis for comparison with first-order ATPs. We shall also collect problems from Isabelle and HOL users. *(months 1–2)*

Task 2: Design issues.

Through investigation of the corpus and the challenges it imposes, we shall redesign LEO's calculus and architecture. This step will also examine the form of co-operation with first-order automatic theorem provers. LEO's use of the OANTS blackboard architecture (a component of OMEGA) may be replaced by a direct integration. *(months 1–4)*

Task 3: Resolution calculus.

We shall design a suitable resolution calculus. This calculus will improve LEO's current extensional higher-order resolution calculus [9, 10] in various ways. Possibilities include equality as a primitive logical connective, lifting of ideas from first-order superposition to higher-order logic, support for non-normal form resolution, extensionality treatment, and basic support from primitive substitution. This work will be based on the ideas of Benzmüller and Brown [4, 5, 12]. *(months 2–4)*

Task 4: Main loop.

We shall design the LEO main loop, in combination with an architecture that supports the integration of the improved calculus with first-order ATPs. The first-order ATPs will be used to tackle sets of first-order clauses accumulated by LEO's search procedure. This architecture will draw ideas from the existing prototype [11]. The possibility of invoking decision procedures and model checkers may also be examined, but this is speculative. *(months 3–5)*

Task 5: Theoretical issues.

We shall investigate soundness and completeness issues. From the theoretical side, we shall employ the recently developed higher-order abstract consistency proof principle [7, 8]. From the practical side, we shall use Benzmüller's collection of proof problems [6]. *(months 5–7)*

Task 6: Coding.

We shall implement the new calculus and the new main loop of LEO, probably using Standard ML. We shall investigate term-indexing and similar techniques for improving efficiency. These are ubiquitous in first-order ATPs, but it is not clear that they are applicable to higher-order logic.*(months 3–9)*

Task 7: Integration.

We shall integrate a fast first-order ATP with LEO, replacing the obsolete Bliksem. Two possible choices are Vampire [19] and SPASS [22]. *(months 6–11)*

Task 8: Testing.

We shall evaluate the new LEO prover using our higher-order problem set and other examples. We shall test it against TPS; we shall also test against first-order ATPs with first-order problem descriptions, as in the preliminary paper [11]. *(months 10–12)*

Relevance to Beneficiaries

The beneficiaries can be found in several branches of computer science.

- The formal methods community uses higher-order logic and will benefit from advances made in its automation.
- The automated reasoning community will benefit from the release of a modern higher-order theorem prover, especially one that is lean and easy to use.
- Higher-order logic is relevant to computational linguistics, as noted by Pulman [18]. Benzmüller has already received enquiries from that community.

Regardless of the success of the project itself, UK-based researchers will benefit from Benzmüller's presence in Britain for one year. As evidence of this point, a letter of support is attached from Prof. Alan Bundy of the University of Edinburgh.

Dissemination and Exploitation

The new theorem prover and problem corpus will be distributed via the Internet under an open source license. Technical and theoretical findings will be presented at conferences and published in academic journals.

Justification of Resources

Staff. Benzmüller will undertake most of the work programme in collaboration with Paulson, who will work part-time to manage the project. Appointing Benzmüller as a visiting researcher will free him from his duties at Saarland University and enable this collaboration. In order to match Benzmüller's existing salary (approximately €53000 per annum, or £36000), the corresponding point on the salary scale is requested. *Note*: this application has not been costed for the new pay structures (single spine point system).

Travel and Subsistence. Benzmüller intends to travel in the UK to consult with experts such as Volker Sorge and Manfred Kerber (Birmingham), Alan Bundy and Jacques Fleuriot (Edinburgh) and Andrei Voronkov (Manchester). Conference attendance is useful for dissemination; we may attend conferences such as CADE, LPAR, MKM and TPHOLs or workshops at Schloß Dagstuhl and elsewhere.

Consumables. The figure shown includes a Linux workstation for Benzmüller. It has a high specification: two processors and 4GB of memory. Theorem proving requires much memory, and dual processors are appropriate in our application, with its multiple co-operating processes. Paulson will use this machine once the project ends in order to investigate integrating LEO II with Isabelle.

References

[1] P. B. Andrews and M. Bishop. On sets, types, fixed points, and checkerboards. In P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, editors, *Theorem Proving with Analytic Tableaux and Related Methods: 5th international workshop, TABLEAUX '96*, LNAI 1071, pages 1–15. Springer, 1996.

[2] P. B. Andrews, D. A. Miller, E. L. Cohen, and F. Pfenning. Automating higher-order logic. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, pages 169–192. American Mathematical Society, 1984.

[3] M. Beeson. Lambda logic. In D. Basin and M. Rusinowitch, editors, *IJCAR*, LNAI 3097, pages 460–474. Springer, 2004.

[4] C. Benzmüller. *Equality and Extensionality in Higher-Order Theorem Proving*. PhD thesis, Universität des Saarlandes, 1999.

[5] C. Benzmüller. Extensional higher-order paramodulation and RUE-resolution. In H. Ganzinger, editor, *Automated Deduction — CADE-16 International Conference*, LNAI 1632, pages 399–413. Springer, 1999.

[6] C. Benzmüller and C. Brown. A structured set of higher-order problems. In J. Hurd and T. Melham, editors, *Theorem Proving in Higher Order Logics: TPHOLs 2005*, LNCS 3603, pages 66–81. Springer, 2005.

[7] C. Benzmüller, C. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69(4):1027–1088, 2004.

[8] C. Benzmüller, C. E. Brown, and M. Kohlhase. Semantic techniques for higher-order cut-elimination. SEKI Technical Report SR-2004-07, Saarland University, Saarbrücken, Germany, 2004.

[9] C. Benzmüller and M. Kohlhase. Extensional higher-order resolution. In Kirchner and Kirchner [15], pages 56–71.

[10] C. Benzmüller and M. Kohlhase. LEO — a higher-order theorem prover. In Kirchner and Kirchner [15], pages 139–143.

[11] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a higher-order and a first-order theorem prover cooperate? In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning — 11th International Workshop, LPAR 2004*, LNAI 3452, pages 415–431. Springer, 2005.

[12] C. E. Brown. *Set Comprehension in Church's Type Theory*. PhD thesis, Carnegie Mellon University, 2004.

[13] E. Cohen. TAPS: A first-order verifier for cryptographic protocols. In *13th Computer Security Foundations Workshop*, pages 144–158. IEEE Computer Society Press, 2000.

[14] M. J. C. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. In G. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. North-Holland, 1986.

[15] C. Kirchner and H. Kirchner, editors. *Automated Deduction — CADE-15 International Conference*, LNAI 1421. Springer, 1998.

[16] W. McCune. OTTER 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, IL, 1994.

[17] L. C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–87, 1999.

[18] S. G. Pulman. Bidirectional contextual resolution. *Comput. Linguist.*, 26(4):497–537, 2000.

[19] A. Riazanov and A. Voronkov. Vampire 1.1 (system description). In R. Goré, A. Leitsch, and T. Nipkow, editors, *Automated Reasoning — First International Joint Conference, IJCAR 2001*, LNAI 2083, pages 376–380. Springer, 2001.

[20] J. Siekmann, C. Benzmüller, A. Fiedler, A. Meier, I. Normann, and M. Pollet. Proof development with Ωmega: The irrationality of $\sqrt{2}$. In F. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, pages 271–314. Kluwer Academic Publishers, 2003.

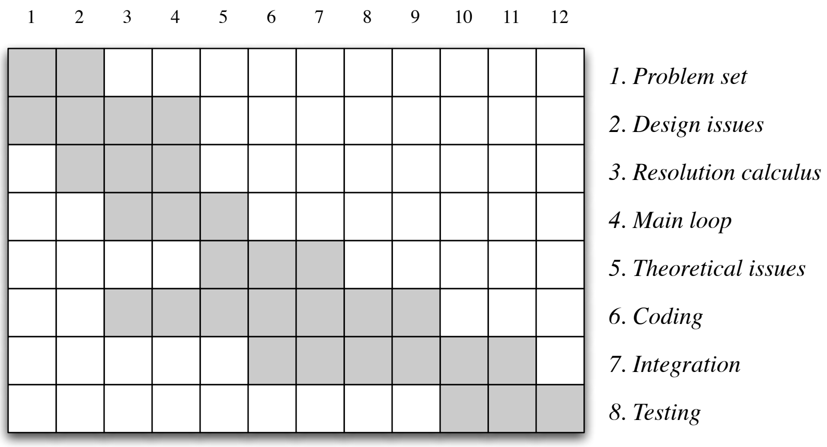
[21] G. Sutcliffe and C. Suttner. The TPTP problem library for automated theorem proving. On the Internet at `http://www.cs.miami.edu/~tptp/`, 2004.

[22] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science, 2001.

[23] P. Youn, B. Adida, M. Bond, J. Clulow, J. Herzog, A. Lin, R. L. Rivest, and R. Anderson. Robbing the bank with a theorem prover. Technical Report UCAM-CL-TR-644, University of Cambridge Computer Laboratory, 2005.

3 Diagrammatic project plan

The diagram shows the approximate distribution of tasks among the twelve months of the project.



1	2	3	4	5	6	7	8	9	10	11	12	
▨	▨											*1. Problem set*
▨	▨	▨	▨									*2. Design issues*
	▨	▨	▨									*3. Resolution calculus*
		▨	▨	▨								*4. Main loop*
				▨	▨	▨						*5. Theoretical issues*
		▨	▨	▨	▨	▨	▨	▨				*6. Coding*
					▨	▨	▨	▨	▨	▨		*7. Integration*
									▨	▨	▨	*8. Testing*