The LEO-II Project

Christoph Benzmüller[⋆†], Larry Paulson[⋆]
[⋆]Computer Laboratory
The University of Cambridge, UK
ceb88@cam.ac.uk
lp15@cam.ac.uk

Frank Theiss[†], Arnaud Fietzke[†]
[†]FR Informatik
Universität des Saarlandes, Germany
lime@ags.uni-sb.de
arnaud.fietzke@gmx.net

Abstract

LEO-II, a resolution based theorem prover for classical higher-order logic, is currently being developed in a one year research project at the University of Cambridge, UK, with support from Saarland University, Germany. We briefly discuss the main objectives of the project.

1 Motivation

Automatic theorem provers (ATPs) based on the resolution principle, such as Vampire, E, and SPASS, have reached a high degree of sophistication. They can often find long proofs even for problems having thousands of axioms. However, they are limited to first-order logic. Higher-order logic extends first-order logic with lambda 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. 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. Conversely, elementary identities (such as the distributive law for union and intersection) turn into difficult problems when expressed in first-order form.

The LEO-II project develops a standalone, resolution-based higher-order theorem prover that is designed for fruitful cooperation with specialist provers for first-order and propositional logic. The idea is to combine the strenghts of the different systems. On the other hand, LEO-II itself, as an external reasoner, wants to support interactive proof assistants such as Isabelle/HOL, HOL, and OMEGA by efficiently automating subproblems and thereby reducing the interaction effort (costs). LEO-II is implemented in Objective CAML and its problem representation language is HOTPTP (1).

2 Research Objectives in the LEO-II Project

Logic and Calculus LEO-II is a theorem prover for classical higher-order logic, i.e., Church's simple type theory (6), which is a logic built on top of the simply typed λ-calculus. The target semantics is Henkin semantics (8) in which the Boolean and functional extensionality principles are valid and thus have to be addressed (4). LEO-II's calculus extends and refines the extensional higher-order resolution calculus of its predecessor LEO (2; 3). The aim is to step towards higher-order ordered paramodulation and superposition and to use (partial) term orderings in order to prune the search space. LEO-II will priorily operate on essentially higher-order clauses with the goal to quickly reduce them to essentially first-order ones. Ideally, the set of essentially first-order clauses is continuously growing until it eventually becomes efficiently refutable by an off the shelf automated first-order prover LEO-II collaborates with. Hence, we are interested in term orderings that characterize how "higher-order" a term actually is and, e.g, how much Boolean and functional extensionality reasoning presumably is required for reduction into essentially first-order or propositional arguments.

Another key interest in the project is to study how the cut-simulation effect (5) can be minimized. For this reason, LEO-II supports primitive equality instead of Leibniz equality and provides built-in extensionality rules instead of working with extensionality axioms.

Perfectly Shared Term Representation and Term Indexing Operations on terms in LEO-II are supported by term indexing. Key features of LEO-II's term indexing are the representation of terms in a perfectly shared graph structure and the indexing of various structural properties, such as the occurrence of subterms and their position.

Term sharing is a technique which is widely employed in first-order theorem proving (10; 11; 13), where syntactically equal terms are represented by a single instance. For LEO-II, we have adapted this technique to the higher-order case. We use de Bruijn-notation (7) to avoid blurring of syntactical equality by α-conversion.

A shared representation of terms has multiple benefits. The most obvious is the instant identification of *all* occurrences of a term or subterm structure. Furthermore, it allows an equality test of syntactic structures at constant cost, which allows the pruning of structural recursion over terms early in many operations. Finally, it allows for tabling of term properties at reasonable cost, as the extra effort spent on term analysis is compensated by the reusability of the results.

The indexing approach of LEO-II, which is employed, e.g., to determine candidate clauses for extensional resolution and subsumption, has a strong focus on structural aspects. It differs in this respect from the approach by Pientka (9), which is based on a discrimination tree and which allows for perfect filtering on the basis of higher order pattern unification. In contrast, we are particularly interested also in more relaxed search criteria, such as subterm occurrences or head symbols.

Architecture and Strategies Automation of higher-order logic, whether resolution-based or not, is comparably poorly investigated. This particularly holds for the question about suitable system architectures and search strategies. In LEO-II's context this question becomes even more relevant because of the idea to fruitfully collaborate with theory specific reasoning specialists. In LEO, an agent-based architecture has been employed to support such a collaboration underneath an extended (wrt. to extensionality aspects) set of support search strategy. In LEO-II we currently reinvestigate these and other options in order to come up with a good solution that takes all novel aspects (e.g., the termindexing support and the modified calculus) into account.

Interactive and Automatic Modes The standard mode of LEO-II is fully automatic, just as for first-order provers like Vampire, E, or Spass. However, LEO-II also supports an interactive mode in which resolution proofs can be developed by the user step by step in a simple command line interpreter. The interaction flavor thereby is similar to standard proof assistants; in fact, in interactive mode, LEO-II *is* a proof assistant for classical higher-order logic based on the extensional higher-order resolution calculus. The interactive mode serves different purposes: (i) it supports debugging of the calculus rules and the search strategies during system development, (ii) it can be used to illustrate the calculus of LEO-II to experts in the field, and (iii) it will later be used for teaching extensional higher-order resolution to students.

Proof Objects Since LEO-II wants to support interactive proof assistants such as Isabelle/HOL, HOL, and OMEGA, we consider the generation of proof objects as very important and, hence, LEO-II provides proof objects and slightly sacrifices efficiency for it. These proof objects provide enough information so that they can later be inspected and verified in these proof assistants.

Evaluation As soon as the implementation has sufficiently progressed we will demonstrate and evaluate LEO-II's usefulness and performance on problems from different domains including hardware- and software verification problems, mathematics (e.g. set theory), and logical puzzles.

3 State of Development

In the current prototype of LEO-II the datastructures, the term index framework, the calculus layer, the basic system architecture, and the command line interpreter are sufficiently developed so that interactive proof development is feasible. Furthermore, a first, simple search strategy for proof automation is provided so that some simple higher-order theorems can already be automatically proved. Even though this strategy does not use sophisticated term orderings it can prove these theorems much faster than the predessor LEO — this is a promising first result. LEO-II already provides full support for analyzing the term index and for computing statistical data such as the average sharing rate. This is not only useful for the further development of LEO-II but also as a tool for analyzing structural aspects of arbitrary data that can be represented in HOTPTP syntax and read in to LEO-II. (The LEO-II system can be demonstrated at the workshop.)

References

[1] The THF syntax (HOTPTP). URL `http://www.cs.miami.edu/ tptp/TPTP/Proposals/THFSyntaxBNF.html`.
[1] C. Benzmüller, V. Sorge, M. Jamnik, and M. Kerber. Can a higher-order and a first-order theorem prover cooperate? Proc. of LPAR 2005.
[2] C. Benzmüller. System description: LEO – a resolution based higher-order theorem prover. Proc. of ESHOL WS at LPAR 2005.
[3] C. Benzmüller and M. Kohlhase. LEO – a higher-order theorem prover. Proc. of CADE 1998.
[4] C. Benzmüller, C. Brown, and M. Kohlhase. Higher order semantics and extensionality. *Journal of Symbolic Logic*, 69:1027–1088, 2004.
[5] C. Benzmüller, C. Brown, and M. Kohlhase. Cut-simulation in impredicate logics. Proc. of IJCAR 2006.
[6] A. Church. A Formulation of the Simple Theory of Types. *Journal of Symbolic Logic*, 5:56–68, 1940.
[7] N.G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.
[8] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15:81–91, 1950.
[9] B. Pientka. Higher-order substitution tree indexing. Proc. of ICLP 2003.
[10] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AICOM*, 15(2-3):91–110, 2002.
[11] S. Schulz. E – A Brainiac Theorem Prover. *Journal of AI Communications*, 15(2/3):111–126, 2002.
[12] F. Theiss and C. Benzmüller. Term indexing for the LEO-II prover. Proc. of IWIL WS at LPAR 2006.
[13] Ch. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, Ch. Theobald, and D. Topic. SPASS version 2.0. Proc. of CADE 2002.