System Description: Leo – A Higher-Order Theorem Prover*

Christoph Benzmüller and Michael Kohlhase

Fachbereich Informatik, Universität des Saarlandes, Germany
chris|kohlhase@cs.uni-sb.de

Many (mathematical) problems, such as Cantor's theorem, can be expressed very elegantly in higher-order logic, but lead to an exhaustive and un-intuitive formulation when coded in first-order logic.

Thus, despite the difficulty of higher-order automated theorem proving, which has to deal with problems like the undecidability of higher-order unification (HOU) and the need for primitive substitution, there are proof problems which lie beyond the capabilities of first-order theorem provers, but instead can be solved easily by an higher-order theorem prover (HOATP) like Leo. This is due to the expressiveness of higher-order Logic and, in the special case of Leo, due to an appropriate handling of the extensionality principles (functional extensionality and extensionality on truth values).

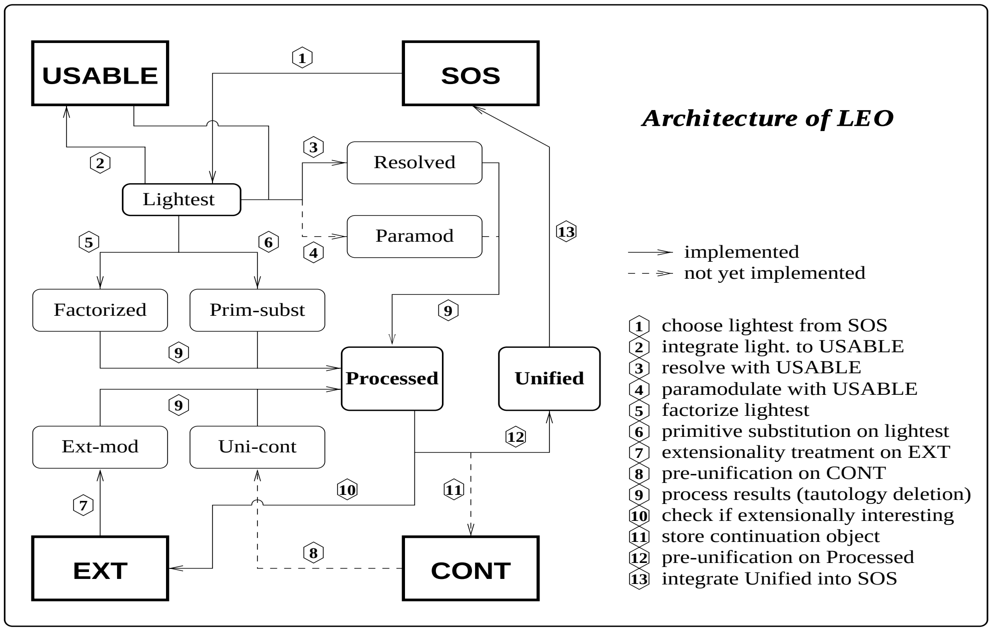
Leo uses a higher-order Logic based upon Church's simply typed λ-calculus, so that the comprehension axioms are implicitly handled by $\alpha\beta\eta$-equality. Leo employs a higher-order resolution calculus $\mathcal{ERES}$ (see [3] in this volume for details), where the search for empty clauses and higher-order pre-unification [6] are interleaved: the unifiability preconditions of the resolution and factoring rules are residuated as special negative equality literals that are treated by special unification rules. In contrast to other HOATP's (such as Tps [1]) extensionality principles are build in into Leo's unification, and hence do not have to be axiomatized in order to achieve Henkin completeness.

Architecture

Leo's architecture is based on a standard set-of-support strategy, extended in order to fulfill the requirements specific to higher-order logic. Furthermore, it uses a higher-order variant [7] of Graf's substitution tree indexing [4] and its implementation is based on the Keim [5] toolkit which provides most of the necessary data structures and algorithms for a HOATP. The four cornerstones of Leo's architecture (see the figure for details see [2]) are the set of usable clauses (USABLE), the set of support (SOS) – well known from theorem provers such as OTTER – and two new constructions: The set of extensionally interesting clauses (EXT) and the set of HOU continuations (CONT). The motivation for

* The work reported in this paper was supported by the Deutsche Forschungsgemeinschaft in grant HOTEL.

C. Kirchner and H. Kirchner (Eds.): Automated Deduction, CADE-15
LNAI 1421, pp. 139–143, 1998. © Springer–Verlag Berlin Heidelberg 1998


USABLE
SOS
Architecture of LEO
1
2
3
4
5
6
9
9
9
10
11
12
13
7
8
Lightest
Resolved
Paramod
Factorized
Prim-subst
Processed
Unified
Ext-mod
Uni-cont
EXT
CONT
implemented
not yet implemented
1 choose lightest from SOS
2 integrate light. to USABLE
3 resolve with USABLE
4 paramodulate with USABLE
5 factorize lightest
6 primitive substitution on lightest
7 extensionality treatment on EXT
8 pre-unification on CONT
9 process results (tautology deletion)
10 check if extensionally interesting
11 store continuation object
12 pre-unification on Processed
13 integrate Unified into SOS

these two additional sets comes from the main idea of using HOU as a filter in order to eliminate in each loop all those newly derived clauses from the search space, which cannot be pre-unified within the given search depth limit (specified by a flag). Unfortunalety this filter is too strong and eliminates clauses which are nevertheless important for the refutation. Such clauses are preserved from elimination and put into CONT or EXT.

In each cycle, Leo selects the lightest clause from SOS and resolves it against all clauses in USABLE, factorizes it and applies primitive substitutions – the higher-order pre-unifiers that license this are not directly computed but residuated as unification literals. In a first-order theorem prover, these processed clauses would simply be integrated into the SOS after unification. A HOATP would thereby loose completeness for two reasons:

- There may be clauses which are non-unifiable within the given unification depth limit, but which have solutions beyond this limit. Upon reaching the depth limit, the unification procedure generates the clauses induced by the open leaves of the unification tree. These are stored in CONT for further processing by pre-unification.
- There are clauses which are not pre-unifiable at all, but which might be necessary for a refutation, if one takes the extensionality principles into account (these are stored in EXT for extensionality treatment).

  As an example consider the unification constraint $[\lambda X_\iota . a_o \vee b_o = \lambda X_\iota . b_o \vee a_o]^F$, which is not pre-unifiable, but leads to a refutation if one applies the extensionality rules to it (first *Func*, then *Equiv*, rest straightforward).

Of course it would be theoretically sufficient to integrate the clauses in CONT and EXT directly into the SOS, but the experiments show that it is better to subject them to specialized heuristics and filters and possibly delay their further processing.

Leo employs a higher-order substitution-tree indexing method [7], for example in the subsumption tests during incorporation into the SOS. Since full HOU is undecidable, it is only possible to use an imperfect filter that rules out all literals where simplification of the induced unification literals fails. However, it is impossible to use these techniques to select possible resolutions and factorizations, since co-simplification does not take extensionality into account – see example 1 below, where the refutation would be impossible, since the set of possible resolutions found by indexing is empty.

Experiments

Leo is able to solve a variety of simple higher-order theorems such as Cantor's theorem and it is specialized in solving theorems with embedded propositions.

Example 1 (Embedded Propositions). $pa \wedge pb \Rightarrow p(a \wedge b)$, where $p_{o \to o}, a_o$ and b_o are constants. Despite it's simplicity, this theorem cannot be solved automatically by any other HOATP such as Tps or Hol.

The clause normal form of the problem consists of three clauses

$$[p(a \wedge b)]^F \qquad [pa]^T \qquad [pb]^T$$

Leo inserts the first one into the SOS and the others into USABLE. In the first cycle $[p(a \wedge b)]^F$ is resolved against $[pa]^T$ and $[pb]^T$ yielding the clauses $[p(a \wedge b) = pa]^F$ and $[p(a \wedge b) = pb]^F$. These are simplified to $[a \wedge b = a]^F$ and $[a \wedge b = b]^F$ and subsequently stored in EXT, since their unification constraints are of boolean type, which makes them extensionally interesting. Since unification fails on these the SOS becomes empty now, leaving extensionality treatment as the only option for further processing. This now interprets the equalities as logical equivalences and yields (after subsumption) the clauses

$$[a]^F \vee [b]^F \qquad [b]^T \qquad [a]^T$$

from which an empty clause can be derived in two resolution steps.

Among the examined examples are also some interesting theorems about sets, where the application of the extensionality principles are essential for finding a proof.

Example 2. $2^{\{X|pX\}} \cap 2^{\{X|qX\}} = 2^{\{X|pX \wedge qX\}}$, where $p_{\iota \to o}$ and $q_{\iota \to o}$ are two arbitrary predicates. In Leo we code this theorem as:

$$\mathcal{P}(\lambda X_\iota.pX) \cap \mathcal{P}(\lambda X_\iota.qx) = \mathcal{P}(\lambda X_\iota.pX \wedge qX)$$

where the power-set $\mathcal{P}$ stands for $\lambda X_{\iota \to o}.\lambda Y_{\iota \to o}.Y \subset X$ and $\cap, \subset$ are similarly defined from $\wedge, \Rightarrow$. The current default heuristic of Leo's clause normalization procedure, does not replace the negated theorem by it's Leibniz formulation, but generates the following clause consisting of exactly one unification constraint

$$[(\lambda X_{\iota \to o}.(\forall Y_\iota.XY \Rightarrow pY) \wedge (\forall Y_\iota.XY \Rightarrow qY)) = (\lambda X_{\iota \to o}.(\forall Y_\iota.XY \Rightarrow (pY \wedge qY)))]^F$$

This unification constraint is not pre-unifiable, but note that the two functions on both hand sides can be identified with the help of the extensionality principles. Thus a higher-order theorem prover without extensionality treatment would either give up or would have to use the extensionality axioms, provided they are available in the search space. Leo instead makes use of its extensionality rules and first derives the following clause with rule *Func* (s_ι is a new Skolem constant):

$$[(\forall Y.sY \Rightarrow pY) \land (\forall Y.sY \Rightarrow qY)) = (\forall Y.sY \Rightarrow (pY \land qY)))]^F$$

Next Leo applies the rule *Equiv*, which first replaces the primitive equality symbol by an equivalence and second applies clause normalization. By this we get 12 new first-order clauses, and the rest of the proof is straightforward.

Similar examples are those discussed in [10]. When coding these theorems as above, then Leo discovers all proofs (except those for examples 56 and 57, where too many simple first-order clauses were generated), most of them within one second on a Pentium Pro 200. On these examples Leo outperforms well known first-order ATPs like Spass, Otter, Protein, Gandalf and Setheo (see `http://www-irm.mathematik.hu-berlin.de/~ilf/miz2atp/mizstat.html`). Especially example 111, which cannot be solved by any of the above provers, is trivial for Leo (10msec on a Pentium Pro 200).

Conclusion and Availability

The next logical steps to enhance the deductive power of Leo will be to extend the system to sorted logics [8], to extend the indexing scheme from co-simplification to higher-order pattern unification [9], to fine-tune the heuristics for extensionality treatment and finally to extend the system by a treatment for primitive equality.

The source code and proof examples (including detailed proofs for the example discussed above and those for the problems in [10]) are available via `http://www.ags.uni-sb.de/projects/deduktion/projects/hot/leo/`.

References

1. P. B. Andrews, M. Bishop, S. Issar, D. Nesmith, F. Pfenning, and H. Xi. TPS: A theorem proving system for classical type theory. *Journal of Automated Reasoning*, 16(3):321–353, 1996.
2. C. Benzmüller. A Calculus and a System Architecture for Extensional Higher-Order Resolution. Research Report 97-198, Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh,USA, June 1997.
3. C. Benzmüller and M. Kohlhase. Extensional Higher-Order Resolution. *Proc. CADE-15*, this volume, 1998.
4. P. Graf. *Term Indexing.* Number 1053 in LNCS. Springer Verlag, 1996.
5. X. Huang, M. Kerber, M. Kohlhase, E. Melis, D. Nesmith, J. Richts, and J. Siekmann. Keim: A toolkit for automated deduction. In Alan Bundy, editor, *Proc. CADE-13*, number 814 in LNAI, pages 807–810, 1994. Springer Verlag.
6. G. P. Huet. An unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975.
7. L. Klein. Indexing für Terme höherer Stufe. Master's thesis, FB Informatik, Universität des Saarlandes, 1997.
8. M. Kohlhase. *A Mechanization of Sorted Higher-Order Logic Based on the Resolution Principle.* PhD thesis, Universität des Saarlandes, 1994.

9. D. Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.
10. Z. Trybulec and H. Swieczkowska. Boolean properties of sets. *Journal of Formalized Mathematics*, 1, 1989.