System Demonstration: The Higher-Order Prover Leo-III

Alexander Steen[1] and Christoph Benzmüller[2,1]

[1] Freie Universität Berlin, Institute for Computer Science, Berlin, Germany
[2] University of Luxembourg, FSTC, Luxembourg
{a.steen|c.benzmueller}@fu-berlin.de

Abstract

The higher-order ATP system Leo-III is demonstrated. Leo-III supports flexible and effective reasoning in every common semantical variation of normal modal logics.

Many powerful automated and interactive theorem proving systems for first-order and higher-order logics have been developed over the past decades. However, with a few notable exceptions, most available systems focus on classical logics only. In particular for quantified non-classical logics only a small number of implemented systems is available to date. This is in contrast to an increasing number of challenging and interesting applications for such systems in artificial intelligence, computer science, mathematics and philosophy [10, 8, 9, 5, 6, 7]. Metaphysics, for example, is an area where higher-order modal logics (HOMLs) play an important role. The development of ATPs for HOMLs, however, is still in its infancy. The Leo-III prover, which is presented here, is addressing this gap.

Leo-III [4] is in the first place an automated theorem prover for classical higher-order logic (HOL) with Henkin semantics and choice [1]. Despite its primary focus on HOL, Leo-III comes with effective means for reasoning in HOMLs. In fact, reasoning in every normal modal logic variant is supported in Leo-III. To achieve this, the prover internally implements a shallow semantical embedding approach [2, 3]. The key idea of this approach is to provide and exploit faithful mappings for HOML input problems to HOL. This is orthogonal to the direct implementation of specialised theorem provers, which usually focus on a small subset of modal logic systems only. The semantical embedding approach realised in Leo-III, in contrast, allows for a quick adaptation to a broad variety of expressive, non-classical logics.

Leo-III in particular supports (but is not limited to) first-order and higher-order extensions of the well known modal logic cube for different concrete choices of

Quantification semantics, including cumulative, decreasing, constant and varying domains,
Rigidity, including rigid and world-dependent constant symbols, and
Consequence, including the usual notions of local and global consequence.

When taking all possible parameter combinations into account this amounts to more than 120 supported HOMLs [3, §2.2]. The exact number of logics is in fact much higher, since Leo-III also supports multi-modal logics and offers fine-grained control over more specific combinations of the above semantical parameters (e.g. different quantification semantics per type).

Higher-order modal logics. HOMLs as addressed here are extensions of HOL, which has been proposed by Church, and further studied by Henkin, Andrews and others. HOL provides lambda-notation as an elegant means to denote unnamed functions, predicates and sets (by their characteristic functions). HOML, in turn, augments HOL with a set of modal operators $\Box^i$, $i \in I$, for some index set I, and is equipped with a suitable combination of HOL semantics and a Kripke-style modal semantics. In our approach an adequate notion of Henkin semantics for both HOML and HOL is assumed.

Figure 1: Example modal logic problem input for Leo-III. The first three lines specify the exact modal logic (here a S5 logic with rigid constants, constant domain quantification and global consequence) under which the problem is to be analyzed. The conjecture is represented by the last two lines and encodes the formula $\forall P_{\iota\to o}\forall F_{\iota\to\iota}\forall X_\iota\exists G_{\iota\to\iota}(\Diamond\Box P(F(X)) \Rightarrow \Box P(G(X)))$.

```
thf(s5_spec, logic, ($modal := [
      $constants := $rigid, $quantification := $constant,
      $consequence := $global, $modalities := $modal_system_S5 ])).
thf(becker,conjecture,( ! [P:$i>$o,F:$i>$i, X:$i]: (? [G:$i>$i]:
      (($dia @ ($box @ (P @ (F @ X)))) => ($box @ (P @ (G @ X))))))).
```

Automation of HOML. In order to automate reasoning in HOMLs, Leo-III exploits the semantical embedding approach and internally translates modal logic problems into equivalent problems formulated within classical higher-order logic. To that end, the de-facto standard TPTP THF input syntax is augmented to include the modal connectives. Fig. 1 displays an example modal logic formula that is an instance of a corollary of Becker's postulate, with `$box` and `$dia` representing the (mono-)modal operators $\Box$ and $\Diamond$, respectively, and the usual TPTP text representatives of the remaining logical connectives. This example formula is valid in S5 but not in any weaker system.

The logic specification format displayed in the example from Fig. 1 is stemming from an ongoing TPTP language extension proposal.[1] In this logic specification, the identifiers `$constants`, `$quantification` and `$consequence` specify the exact semantical settings for the rigidity of constant symbols, the quantification semantics and the consequence relation, respectively. Finally, `$modalities` specify the properties of the modal connectives. Valid values are either pre-defined identifiers representing the usual modal logic systems, as in `$modalities := $modal_system_S5` for the specification of an S5 modal logic, or lists of individual modal axiom schemes, as in `$modalities := [$modal_axiom_K, $modal_axiom_B]`.

The reasoning process of Leo-III proceeds as follows:

1. The user inputs a HOML problem in the adapted TPTP syntax from above (Fig. 1).
2. Leo-III analyses the logic specification contained within the input and automatically selects the definitions and axioms to be added to the embedded problem representation.
3. The problem statement itself is translated into its embedded equivalent using the definitions from the previous step.
4. Finally, Leo-III starts reasoning in (meta-logic) HOL and returns SZS compliant result information and, if successful, also a proof object just as for standard HOL problems.

Summary. At the ARQNL 2018 event we will demonstrate Leo-III, which, in terms of supported logics, is the most widely applicable automated theorem prover available to date. The embedding procedure is also available as stand-alone implementation at `github.com/leoprover` and can be used in conjunction with every THF-compliant ATP.

[1] See `http://www.cs.miami.edu/~tptp/TPTP/Proposals/LogicSpecification.html` for more details.

A Installation and Usage of Leo-III

Acquisition and Installation

Leo-III is freely available on GitHub (https://github.com/leoprover/Leo-III) under BSD-3 license. The most current release (version 1.2) is accessible under https://github.com/leoprover/Leo-III/releases/latest. To get it, simply download the source archive and extract it so some location.

```
> wget https://github.com/leoprover/Leo-III/archive/v1.2.tar.gz
> tar -xvzf v1.2.tar.gz
```

After extraction, Leo-III can be built using Make. Simply `cd` to the extracted directory and run `make`:

```
> cd Leo-III-1.2/
> make
```

After building, there should be a directory `bin/`, relative from the current directory. This directory contains the binary `leo3` of Leo-III.

Leo-III can optionally be installed by invoking

```
> make install
```

which copies the binary to the directory `$HOME/bin` by default.

Usage

Leo-III is invoked via command-line (assuming the leo3 executable is in `$PATH`):

For the example of Becker's postulate of Fig. 1, running

```
> leo3 becker.p -p
```

will invoke Leo-III for proving this conjecture (the `-p` option enables the output of a proof certificate). This will produce the following result:

```
% Axioms used in derivation (1): mrel_meuclidean
% No. of inferences in proof: 22
% No. of processed clauses: 14
% No. of generated clauses: 77
[...]
% SZS status Theorem for becker.p : 4179 ms resp. 1443 ms w/o parsing
% SZS output start CNFRefutation for becker.p
thf(mworld_type, type, mworld: $tType).
thf(mrel_type, type, mrel: (mworld > (mworld > $o))).
thf(meuclidean_type, type, meuclidean: ((mworld > (mworld > $o)) > $o)).
thf(meuclidean_def, definition, (meuclidean = (^ [A:(mworld > (mworld > $o))]: ! [B:mworld,C:mworld,D:mworld
    ]: (((A @ B @ C) & (A @ B @ D)) => (A @ C @ D))))).
thf(mvalid_type, type, mvalid: ((mworld > $o) > $o)).
thf(mvalid_def, definition, (mvalid = ('!' @ mworld))).
thf(mimplies_type, type, mimplies: ((mworld > $o) > ((mworld > $o) > (mworld > $o)))).
thf(mimplies_def, definition, (mimplies = (^ [A:(mworld > $o),B:(mworld > $o),C:mworld]: ((A @ C) => (B @ C))
    ))).
thf(mdia_type, type, mdia: ((mworld > $o) > (mworld > $o))).
thf(mdia_def, definition, (mdia = (^ [A:(mworld > $o),B:mworld]: ? [C:mworld]: ((mrel @ B @ C) & (A @ C))))).
thf(mbox_type, type, mbox: ((mworld > $o) > (mworld > $o))).
thf(mbox_def, definition, (mbox = (^ [A:(mworld > $o),B:mworld]: ! [C:mworld]: ((mrel @ B @ C) => (A @ C))))
    .
thf(mexists_const__o__d_i_t__d_i_c__type, type, mexists_const__o__d_i_t__d_i_c_: (((($i > $i) > (mworld > $o))
     > (mworld > $o))).
thf(mexists_const__o__d_i_t__d_i_c__def, definition, (mexists_const__o__d_i_t__d_i_c_ = (^ [A:(($i > $i) > (
    mworld > $o)),B:mworld]: ? [C:($i > $i)]: (A @ C @ B)))).
thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__type, type, mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
    : (((($i > (mworld > $o)) > (mworld > $o)) > (mworld > $o))).
```

```
thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c__def, definition, (
     mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ = (^ [A:(($i > (mworld > $o)) > (mworld > $o)),B:mworld]:
      ! [C:($i > (mworld > $o))]: (A @ C @ B)))).
thf(mforall_const__o__d_i_c__type, type, mforall_const__o__d_i_c_: (($i > (mworld > $o)) > (mworld > $o))).
thf(mforall_const__o__d_i_c__def, definition, (mforall_const__o__d_i_c_ = (^ [A:($i > (mworld > $o)),B:mworld
     ]: ! [C:$i]: (A @ C @ B)))).
thf(mforall_const__o__d_i_t__d_i_c__type, type, mforall_const__o__d_i_t__d_i_c_: ((($i > $i) > (mworld > $o))
      > (mworld > $o))).
thf(mforall_const__o__d_i_t__d_i_c__def, definition, (mforall_const__o__d_i_t__d_i_c_ = (^ [A:(($i > $i) > (
     mworld > $o)),B:mworld]: ! [C:($i > $i)]: (A @ C @ B)))).
thf(sk1_type, type, sk1: mworld).
thf(sk2_type, type, sk2: ($i > (mworld > $o))).
thf(sk3_type, type, sk3: ($i > $i)).
thf(sk4_type, type, sk4: $i).
thf(sk5_type, type, sk5: mworld).
thf(sk6_type, type, sk6: (($i > $i) > mworld)).
thf(1,conjecture,((mvalid @ (mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ @ (^ [A:($i > (mworld > $o))]: (
     mforall_const__o__d_i_t__d_i_c_ @ (^ [B:($i > $i)]: (mforall_const__o__d_i_c_ @ (^ [C:$i]: (
     mexists_const__o__d_i_t__d_i_c_ @ (^ [D:($i > $i)]: (mimplies @ (mdia @ (mbox @ (A @ (B @ C)))) @ (mbox
     @ (A @ (D @ C))))))))))))),file('becker.p',1)).
thf(2,negated_conjecture,((~ (mvalid @ (mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_ @ (^ [A:($i > (mworld
     > $o))]: (mforall_const__o__d_i_t__d_i_c_ @ (^ [B:($i > $i)]: (mforall_const__o__d_i_c_ @ (^ [C:$i]: (
     mexists_const__o__d_i_t__d_i_c_ @ (^ [D:($i > $i)]: (mimplies @ (mdia @ (mbox @ (A @ (B @ C)))) @ (mbox
     @ (A @ (D @ C))))))))))))),inference(neg_conjecture,[status(cth)],[1])).
thf(5,plain,((~ (! [A:mworld,B:($i > (mworld > $o)),C:($i > $i),D:$i]: ? [E:($i > $i)]: ((? [F:mworld]: ((
     mrel @ A @ F) & ! [G:mworld]: ((mrel @ F @ G) => (B @ (C @ D) @ G)))) => (! [F:mworld]: ((mrel @ A @ F)
     => (B @ (E @ D) @ F)))))),inference(defexp_and_simp_and_etaexpand,[status(thm)],[2])).
thf(6,plain,((~ (! [A:mworld,B:($i > (mworld > $o)),C:($i > $i),D:$i]: ((? [E:mworld]: ((mrel @ A @ E) & ! [F
     :mworld]: ((mrel @ E @ F) => (B @ (C @ D) @ F)))) => (? [E:($i > $i)]: ! [F:mworld]: ((mrel @ A @ F) =>
     (B @ (E @ D) @ F)))))),inference(miniscope,[status(thm)],[5])).
thf(10,plain,((mrel @ sk1 @ sk5)),inference(cnf,[status(esa)],[6])).
thf(4,axiom,((meuclidean @ mrel)),file('becker.p',mrel_meuclidean)).
thf(15,plain,((! [A:mworld,B:mworld,C:mworld]: (((mrel @ A @ B) & (mrel @ A @ C)) => (mrel @ B @ C)))),
     inference(defexp_and_simp_and_etaexpand,[status(thm)],[4])).
thf(16,plain,(! [C:mworld,B:mworld,A:mworld] : ((~ (mrel @ A @ B)) | (~ (mrel @ A @ C)) | (mrel @ B @ C))),
     inference(cnf,[status(esa)],[15])).
thf(17,plain,(! [C:mworld,B:mworld,A:mworld] : ((~ (mrel @ A @ C)) | (mrel @ B @ C) | ((mrel @ sk1 @ sk5) !=
     (mrel @ A @ B)))),inference(paramod_ordered,[status(thm)],[10,16])).
thf(18,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | (mrel @ sk5 @ A))),inference(pattern_uni,[status(thm)
     ],[17:[bind(A, $thf(sk1)),bind(B, $thf(sk5))]])).
thf(40,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | (mrel @ sk5 @ A))),inference(simp,[status(thm)],[18])).
thf(9,plain,(! [A:mworld] : ((~ (mrel @ sk5 @ A)) | (sk2 @ (sk3 @ sk4) @ A))),inference(cnf,[status(esa)
     ],[6])).
thf(7,plain,(! [A:($i > $i)] : ((~ (sk2 @ (A @ sk4) @ (sk6 @ (A))))),inference(cnf,[status(esa)],[6])).
thf(11,plain,(! [A:($i > $i)] : ((~ (sk2 @ (A @ sk4) @ (sk6 @ (A))))),inference(simp,[status(thm)],[7])).
thf(206,plain,(! [B:($i > $i),A:mworld] : ((~ (mrel @ sk5 @ A)) | ((sk2 @ (sk3 @ sk4) @ A) != (sk2 @ (B @ sk4
     ) @ (sk6 @ (B)))))),inference(paramod_ordered,[status(thm)],[9,11])).
thf(212,plain,((~ (mrel @ sk5 @ (sk6 @ (^ [A:$i]: (sk3 @ sk4)))))),inference(pre_uni,[status(thm)],[206:[bind
     (A, $thf(sk6 @ (^ [C:$i]: (sk3 @ sk4)))),bind(B, $thf(^ [C:$i]: (sk3 @ sk4)))]])).
thf(259,plain,(! [A:mworld] : ((~ (mrel @ sk1 @ A)) | ((mrel @ sk5 @ A) != (mrel @ sk5 @ (sk6 @ (^ [B:$i]: (
     sk3 @ sk4)))))),inference(paramod_ordered,[status(thm)],[40,212])).
thf(260,plain,((~ (mrel @ sk1 @ (sk6 @ (^ [A:$i]: (sk3 @ sk4)))))),inference(pattern_uni,[status(thm)],[259:[
     bind(A, $thf(sk6 @ (^ [B:$i]: (sk3 @ sk4))))]])).
thf(8,plain,(! [A:($i > $i)] : ((mrel @ sk1 @ (sk6 @ (A))))),inference(cnf,[status(esa)],[6])).
thf(12,plain,(! [A:($i > $i)] : ((mrel @ sk1 @ (sk6 @ (A))))),inference(simp,[status(thm)],[8])).
thf(269,plain,(~ ($true)),inference(rewrite,[status(thm)],[260,12])).
thf(270,plain,($false),inference(simp,[status(thm)],[269])).
% SZS output end CNFRefutation for becker.p
```

The line starting with "`% SZS status Theorem`" confirms that the conjecture is indeed a theorem and the contents between "`% SZS output start`" and "`% SZS output end`" are the proof certificate for this claim.

Becker's Postulate Embedded

The semantically embedded variant of `becker.p` that is used internally by Leo-III is as follows (this can also be generated using the stand-alone embedding tool available at https://github.com/leoprover/embed_modal):

```
% declare type for possible worlds
thf(mworld_type,type,(
    mworld: $tType )).

% declare accessibility relations
thf(mrel_type,type,(
    mrel: mworld > mworld > $o )).

% define accessibility relation properties
thf(mreflexive_type,type,(
    mreflexive: ( mworld > mworld > $o ) > $o )).

thf(mreflexive_def,definition,
    ( mreflexive
    = ( ^ [R: mworld > mworld > $o] :
        ! [A: mworld] :
          ( R @ A @ A ) ) )).

thf(meuclidean_type,type,(
    meuclidean: ( mworld > mworld > $o ) > $o )).

thf(meuclidean_def,definition,
    ( meuclidean
    = ( ^ [R: mworld > mworld > $o] :
        ! [A: mworld,B: mworld,C: mworld] :
          ( ( ( R @ A @ B )
            & ( R @ A @ C ) )
         => ( R @ B @ C ) ) ) )).

% assign properties to accessibility relations
thf(mrel_mreflexive,axiom,(
    mreflexive @ mrel )).

thf(mrel_meuclidean,axiom,(
    meuclidean @ mrel )).

% define valid operator
thf(mvalid_type,type,(
    mvalid: ( mworld > $o ) > $o )).

thf(mvalid_def,definition,
    ( mvalid
    = ( ^ [S: mworld > $o] :
        ! [W: mworld] :
          ( S @ W ) ) )).

% define nullary, unary and binary connectives which are no quantifiers
thf(mimplies_type,type,(
    mimplies: ( mworld > $o ) > ( mworld > $o ) > mworld > $o )).

thf(mimplies,definition,
    ( mimplies
    = ( ^ [A: mworld > $o,B: mworld > $o,W: mworld] :
          ( ( A @ W )
         => ( B @ W ) ) ) )).

thf(mdia_type,type,(
    mdia: ( mworld > $o ) > mworld > $o )).

thf(mdia_def,definition,
    ( mdia
    = ( ^ [A: mworld > $o,W: mworld] :
        ? [V: mworld] :
          ( ( mrel @ W @ V )
          & ( A @ V ) ) ) )).
```

```
thf(mbox_type,type,(
    mbox: ( mworld > $o ) > mworld > $o )).

thf(mbox_def,definition,
    ( mbox
    = ( ^ [A: mworld > $o,W: mworld] :
        ! [V: mworld] :
          ( ( mrel @ W @ V )
         => ( A @ V ) ) ) )).

% define exists quantifiers
thf(mexists_const_type__o__d_i_t__d_i_c_,type,(
    mexists_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o ) > mworld > $o )).

thf(mexists_const__o__d_i_t__d_i_c_,definition,
    ( mexists_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,W: mworld] :
        ? [X: $i > $i] :
          ( A @ X @ W ) ) )).

% define for all quantifiers
thf(mforall_const_type__o__d_i_t__o_mworld_t__d_o_c__c_,type,(
    mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_: ( ( $i > mworld > $o ) > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_,definition,
    ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
    = ( ^ [A: ( $i > mworld > $o ) > mworld > $o,W: mworld] :
        ! [X: $i > mworld > $o] :
          ( A @ X @ W ) ) )).

thf(mforall_const_type__o__d_i_c_,type,(
    mforall_const__o__d_i_c_: ( $i > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_c_,definition,
    ( mforall_const__o__d_i_c_
    = ( ^ [A: $i > mworld > $o,W: mworld] :
        ! [X: $i] :
          ( A @ X @ W ) ) )).

thf(mforall_const_type__o__d_i_t__d_i_c_,type,(
    mforall_const__o__d_i_t__d_i_c_: ( ( $i > $i ) > mworld > $o ) > mworld > $o )).

thf(mforall_const__o__d_i_t__d_i_c_,definition,
    ( mforall_const__o__d_i_t__d_i_c_
    = ( ^ [A: ( $i > $i ) > mworld > $o,W: mworld] :
        ! [X: $i > $i] :
          ( A @ X @ W ) ) )).


% transformed problem
thf(1,conjecture,
    ( mvalid
    @ ( mforall_const__o__d_i_t__o_mworld_t__d_o_c__c_
      @ ^ [P: $i > mworld > $o] :
          ( mforall_const__o__d_i_t__d_i_c_
          @ ^ [F: $i > $i] :
              ( mforall_const__o__d_i_c_
              @ ^ [X: $i] :
                  ( mexists_const__o__d_i_t__d_i_c_
                  @ ^ [Q: $i > $i] :
                      ( mimplies @ ( mdia @ ( mbox @ ( P @ ( F @ X ) ) ) ) @ ( mbox @ ( P @ ( Q @ X ) ) ) ) ) ) ) ) ) ) ).
```

References

[1] Peter Andrews. Church's type theory. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2014 edition, 2014.

[2] Christoph Benzmüller and Lawrence Paulson. Quantified Multimodal Logics in Simple Type Theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.

[3] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. Theorem provers for every normal modal logic. In *LPAR-21*, volume 46 of *EPiC Series in Computing*, pages 14–30. EasyChair, 2017.

[4] Alexander Steen and Christoph Benzmüller. The higher-order prover Leo-III. In *IJCAR 2018*, LNCS. Springer, 2018. forthcoming.

[5] Daniel Kirchner and Christoph Benzmüller and Edward N. Zalta. Mechanizing Principia Logico-Metaphysica in Functional Type Theory (Extended Abstract). In 3rd Conference on Artificial Intelligence and Theorem Proving (AITP 2018), Book of Abstracts, 2018.

[6] Christoph Benzmüller, Xavier Parent, and Leendert van der Torre. A Deontic Logic Reasoning Infrastructure. In 14th Conference on Computability in Europe, CiE 2018, Kiel, Germany, July 30-August, 2018, Proceedings, LNAI Vol. 10505, pages 114–127, Springer, 2018.

[7] David Fuenmayor and Christoph Benzmüller. A Case Study on Computational Hermeneutics: E. J. Lowe's Modal Ontological Argument. PhilPapers, https://philpapers.org/rec/FUEACS, 2017.

[8] David Fuenmayor and Christoph Benzmüller. Types, Tableaus and Gödel's God in Isabelle/HOL. *Archive of Formal Proofs*, 2017.

[9] Christoph Benzmüller and Bruno Woltzenlogel Paleo. The Inconsistency in Gödel's Ontological Argument: A Success Story for AI in Metaphysics. In IJCAI 2916, pages 936–942, AAAI Press, 2016.

[10] Christoph Benzmüller, Leon Weber, and Bruno Woltzenlogel Paleo. Computer-Assisted Analysis of the Anderson-Hájek Controversy. *Logica Universalis*, 11(1):139–151, 2017.