An Extensible Logic Embedding Tool for Lightweight Non-Classical Reasoning

Alexander Steen

University of Greifswald, Walther-Rathenau-Straße 47, 17489 Greifswald, Germany

Abstract

The logic embedding tool (LET) encodes non-classical reasoning problems into classical higher-order logic. It is extensible and can support an increasing number of different non-classical logics as reasoning targets. When used as a pre-processor or library for higher-order theorem provers, the tool admits off-the-shelf automation for logics for which otherwise few to none provers are currently available.

Keywords

Non-Classical Logic, Higher-Order Logic, Logic Encoding

1. Introduction

Non-classical logics (NCLs) deviate from various principles of classical logics such as bivalence, truth-functionality, idempotency of entailment, etc. [1]. NCLs have numerous topical applications in artificial intelligence, mathematics, computer science, philosophy and other fields; and increasingly many domain-specific NCLs are being introduced. Despite the relevance of NCL reasoning, for many formalisms automated theorem proving (ATP) systems do not exist. One major reason is that the development of ATP systems requires not only suitable theoretical foundations, but it also requires considerable resources for software development and related aspects. It is not surprising that these efforts are only rarely made for logics that are still the subject of active research and discussion (i.e., moving targets), and might be superseded with novel formalisms in the near future. This situation impedes the deployment of methods in practical AI research, and it also hampers the systematic evaluation of available formalisms. Of course, there are notable exceptions of well-established NCLs for which both standards and/or ATP systems do exist, such as linear logics [2, 3], intuitionistic logics [4, 5, 6, 7, 8, 9, 10] and modal logics [11, 12, 10, 13, 14, 15].

Orthogonal to the development of special-purpose provers for individual NCLs is the use of logic translations that encode the logic under consideration (the *source logic*) into another logic formalism (the *target logic*) for which there exist means of automation [16, 17]. In this setting, improvements to ATP systems for the target logic inherently benefit

PAAR'22: 8th Workshop on Practical Aspects of Automated Reasoning, August 11–12, 2022, Haifa, Israel
EMAIL: alexander.steen@uni-greifswald.de (A. Steen)
URL: https://www.alexandersteen.de/ (A. Steen)
ORCID: 0000-0001-8781-9462 (A. Steen)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

reasoning in the source logic. A special type of logic translation are *shallow embeddings* [18], in which the source logic's semantics is directly expressed in the target logic (as opposed to *deep embeddings* where source logic expressions are represented as uninterpreted data [18]). One well-known example of a shallow embedding is the *standard translation* of modal logic [19] in which reasoning in certain modal logics is reduced to reasoning in classical first-order logic.

In this paper, the *logic embedding tool* (**LET**) is presented that provides a library of shallow embeddings of different NCLs into classical higher-order logic (the target logic considered throughout this work), and an executable for applying these embeddings on input problems. Special attention is paid to the extensibility of the tool's underlying library of embeddings. Currently, the following logics are supported:

- Many quantified normal multi-modal logics
- Many quantified hybrid logics
- Public announcement logic
- Carmo and Jones' dyadic deontic logic
- Åqvist's dyadic deontic logic **E**

LET is implemented in Scala and freely available as open-source software (BSD-3 license) via Zenodo [20] and GitHub[1]. The input format is a non-classical TPTP syntax extension [21] that allows non-classical reasoning problems to be written within the common TPTP framework [22], see below for an overview. **LET** can be used as library or as external pre-processor to higher-order ATP systems, effectively enabling automated reasoning for various NCLs.

Higher-Order Logic. Extensional type theory, commonly referred to as higher-order logic (HOL), is an expressive higher-order logical formalism based on a simply typed λ-calculus which originates from works of Church, Henkin and others [23, 24, 25]. HOL is the target logic of **LET**.

The term *higher-order* refers to the ability of HOL to represent quantifications over predicate and function variables — as opposed to first-order logics, in which quantification is restricted to individuals only. Furthermore HOL provides λ-notation as an expressive binding mechanism to denote unnamed functions, predicates and sets (by their characteristic functions), and it comes with built-in principles of Boolean and functional extensionality as well as type-restricted comprehension. It constitutes the foundation of most contemporary higher-order ATP systems [26].

Related work. The translation tool FMLtoHOL [27] implements a shallow embedding of first-order modal logic to HOL. However, the tool only supports modal logic, the set of supported mono-modal logics is strictly smaller, and multi-modal logics are not addressed. **LET** generalizes and extends previous work on modal logic embedding tools [28, 29]: As opposed to previous tools, **LET** is not fixed to modal logic. It, e.g., supports deontic logics

[1] `github.com/leoprover/logic-embedding`

not based on standard modal logic. Also, it is the first tool to make use of the novel TPTP language standard for non-classical reasoning; hence offering a uniform input language front-end (as opposed to previous tools, which all only supported specifically tailored input formats for their respective logic). **LET** also offers more encoding variants for modal logics, including embedding into polymorphic higher-order logic in TH1 format [30] to allow for a more compact output.

2. TPTP Problem Representation Format

As input syntax **LET** accepts the NXF and NHF languages [21], recent non-classical extensions of the well-established TPTP syntax standard for ATP systems. The TPTP syntax is part of the TPTP World infrastructure [22], and defines several languages for representing reasoning problems, including languages for typed first-order logic extended with Boolean terms (*denoted TXF*) [31] and higher-order logic (*denoted THF*) [32]. Reasoning problems in TPTP are built-up from *annotated formulae* of the form ...

language`(`*name*`,` *role*`,` *formula*`,` *source*`,` *useful_info*`).`

where *name*, *role*, *source* and *useful_info* are extra-logical annotations, the two latter being optional. The *name* assign a (unique) identifier to the formula, the *role* describes whether the formula is used, e.g., as axiom or conjecture of a reasoning problem. A comprehensive survey of these and further TPTP languages is available in the literature [22].

The NXF and NHF languages extend TXF and THF, respectively, with new generic non-classical operators of the form `{`*connective_name*`}`, where *connective_name* is either a TPTP-defined name (starting, by convention, with a `$`-sign) or a system-specific name (starting with `$$`), that are applied like functional symbols. They can be parameterized with arbitrary key-value pairs or a simple index. As an example, assuming that `{$$obligation(bearer := alice)}` represents a directed dyadic obligation operator of some deontic logic, the formula `{$$obligation(bearer := alice)}(a,b)` represents the conditional obligation of agent `alice` to adhere to `b` given `a` in NXF. The concrete interpretation of a non-classical operator is defined externally by the TPTP (in case of TPTP-defined operators) or a third party (in case of system-specific operators).

Although NXF is a typed language, it may also represent untyped formalisms. Following the conventions from TXF, any predicate or function symbol with undeclared type implicitly defaults to a canonical n-ary predicate or n-ary function type.

Additionally, NXF and NHF introduce so-called *logic specifications*, special kinds of annotated formulas with the role `logic`, that specify the logic being used within the problem file. In NXF they are of form ...

`tff(`*name*`,logic,`*logic_name* `==` *properties*`).`

where *logic_name* is some TPTP or user defined name for a logic (or logic family), and *properties* is a list of key-value parameters that optionally further specify the intended NCL. In NHF the THF formula identifier `thf` is used instead.

A detailed introduction of non-classical connectives and logic specifications is presented in the respective TPTP proposal [33, 21], and they are informally illustrated via the application examples in Section 5 below.

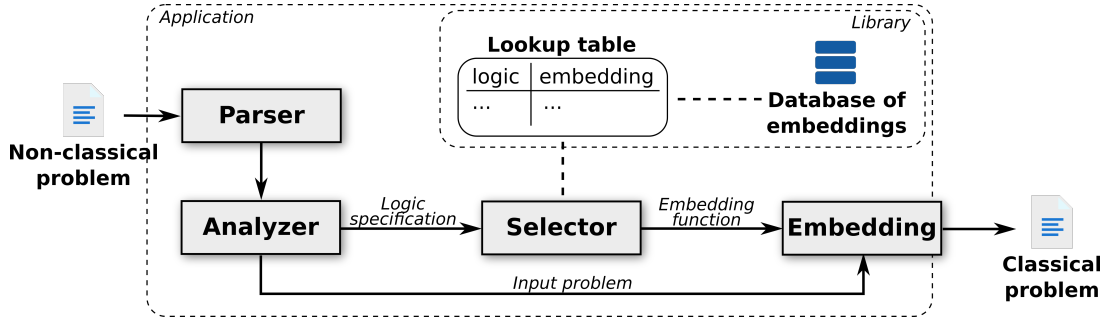

Figure 1: The top-level architecture of **LET**. The solid arrows indicate directed data flow between different components, the dashed lines represent access to additional resources.

3. Architecture

The components of **LET** and their relationship are displayed in Fig. 1. It is structured into two main modules:

1. The *library* module defines a common embedding interface, and constitutes the collection of shallow embeddings for different NCLs.
2. The *application* module implements a stand-alone executable on top of the library module. It finds and applies the correct shallow embedding on a given input problem.

Note that the library module is independent from the application module, and can be included in existing ATP systems via a simple API. The application module, in contrast, can be employed as external pre-processor executable.

The general procedure implemented by the application module is as follows:

1. The input problem is parsed [34] and scanned for a logic specification (an annotated TPTP formula with the role `logic`),
2. the logic name and the parameters are extracted from the logic specification,
3. the internal database of supported logics is queried for the given logic and, if supported, the respective embedding procedure is returned, and finally,
4. the embedding procedure is invoked on the input problem and the result is returned as classical TPTP THF problem.

If the problem does not contain any logic specification, the original problem is returned. If the logic specified in the input problem is malformed or not supported by **LET** an error is reported. The separation of library and application facilitates **LET**'s extensibility, as the library can be easily extended with new embeddings of further NCLs while the application module remains unchanged.

Note that the output of the tool is a classical higher-order problem represented in THF syntax. Hence every higher-order ATP system that supports reasoning in THF can be employed for reasoning in the respective NCL. Additionally, **LET** supports TSTP-compatible result reporting [35] for seamless integration into TPTP/TSTP tool chains.

4. Overview of Supported NCLs

The NCLs currently supported by **LET** are the following:

Modal logics. The logic name `$modal` represents the family of propositional and first-order quantified normal multi-modal logics [19, 36]. The modal operators $\Box$ and $\Diamond$ are represented by the non-classical connectives `{$box}` and `{$dia}`, respectively. In the case of multiple modalities, the connectives are indexed with uninterpreted user constants, prefixed with a `#` (hash sign) as, e.g., in `{$box(#i)}` and `{$dia(#i)}`. Global assumptions are expressed via annotated formulas of role `axiom`, and local assumptions via role `hypothesis` [37]. Relevant embeddings are described in [38, 39, 28].

Hybrid logics. Hybrid logics, referred to as `$$hybrid`, extend the modal logic family `$modal` with the notion of nominals, a special kind of atomic formula symbol that is true only in a specific world [40]. The logics represented by `$$hybrid` are first-order variants of $\mathcal{H}(\mathsf{E}, @, \downarrow)$ [40, 36]. A nominal symbol n is represented as `{$$nominal}(n)`, the shift operator $@_s$ as `{$$shift(#s)}`, and the bind operator $\downarrow x$ as `{$$bind(#X)}`. All other aspects are analogous to the modal logic representation above. Preliminary shallow embeddings results are reported in [41]. The embedding implemented in **LET** simplifies and extends these.

Public announcement logic. Public announcement logic (PAL), `$$pal`, is a propositional epistemic logic that allows for reasoning about knowledge. In contrast to `$modal`, PAL is a dynamic logic that supports updating the knowledge of agents via so-called announcement operators [42]. The knowledge operator K_i is given by `{$$knows(#i)}`, the common knowledge operator C_A, with A a set of agents, by `{$$common($$group := [...])}`, and the announcement $[!\varphi]$ is represented as `{$$announce($$formula := phi)}`. An embedding of PAL is presented in [43].

Dyadic deontic logics. Deontic logics are formalisms for reasoning over norms, obligations, permissions and prohibitions. In contrast to modal logics used for this purpose (e.g., modal logic **D**), dyadic deontic logics (DDLs), named `$$ddl`, offer a more sophisticated representation of conditional norms using a dyadic obligation operator $\bigcirc(\varphi/\psi)$. They address paradoxes of other deontic logics in the context of so-called contrary-to-duty (CTD) situations [44]. The concrete DDLs supported by **LET** are the propositional system by Carmo and Jones [45] and Åqvist's propositional system **E** [46]. The dyadic deontic operator $\bigcirc$ is represented by `{$$obl}` (short for obligatory). An embedding of the above DDL is studied in [47, 48]

Note that the name `$modal` of modal logics and that of its connective names are given by TPTP defined names (starting with a single dollar sign) since it is the first non-classical logic standardized by the TPTP [33, 21]. All further logics are **LET**-specific logic representations that have not (yet) been included in the collection of TPTP curated NCLs; following the TPTP naming convention, their identifiers hence start with two dollar signs (system defined names).

Table 1
Overview of the parameters of the different NCLs supported by **LET**

Logic	Parameter	Description
`$modal`	`$quantification`	Selects whether quantification semantics is varying domains, constant domains, cumulative domains or decreasing domains. Accepted values: `$varying`, `$constant`, `$cumulative`, `$decreasing`
	`$constants`	Selects whether constant and functions symbols are interpreted as rigid or flexible. Accepted values: `$rigid`, `$flexible`† †: Not yet supported by **LET**.
	`$modalities`	Selects the properties for the modal operators. Accepted values, for each modality: `$modal_system_X` where `X` ∈ {`K`, `KB`, `K4`, `K5`, `K45`, `KB5`, `D`, `DB`, `D4`, `D5`, `D45`, `T`, `B`, `S4`, `S5`, `S5U`} *or a list of axiom schemes* [`$modal_axiom_X`$_1$, ..., `$modal_axiom_X`$_n$] `X`$_i$ ∈ {`K`, `T`, `B`, `D`, `4`, `5`, `CD`, `C4`}
`$$hybrid`	*see* `$modal`	*see* `$modal`
`$$pal`	*none*	—
`$$ddl`	`$$system`	Selects which DDL logic system is employed: Carmo and Jones or Åqvist's system **E**. Accepted values: `$$carmoJones` or `$$aqvistE`

Non-classical logic languages quite commonly admit different concrete logics using the same syntax. In order to choose the exact logic intended for the input problem, suitable parameters are given as properties to the logic specification as introduced in Sect. 2. For the above NCLs supported by **LET**, Table 1 gives an overview of the individual parameters and their meaning. We refer to Fitting and Mendelsohn [37] for an explanation of the modal logic properties.

5. Application Examples

The functionality of **LET** is illustrated by a number of examples. Exemplary ATP system results are produced by the higher-order prover Leo-III [49], version 1.6.8, in which **LET** is integrated as a library and accessed via its API. Leo-III parses the problems, invokes the embedding API, and then applies standard proof search on the resulting THF problem. Of course, any other THF-compliant HOL ATP system can be used instead when **LET** is used as external pre-processor.

The presented NXF problems and the complete THF output produced by **LET** on the individual problems is available as small data set at Zenodo [50].

Example 1: Modal logic reasoning. The Barcan formula [51], given by

$$\forall x.\, \Box p(x) \Rightarrow \Box(\forall x.\, p(x))$$

in a first-order variant, is a modal logic formula that is valid if and only if the quantification domain of the underlying first-order modal logic model is non-cumulative [37]. This is written in NXF as ...

```
tff(modal_k5, logic, $modal == [
   $constants == $rigid,
   $quantification == $decreasing,
   $modalities == [$modal_axiom_K, $modal_axiom_5]
 ] ).

tff(bf, conjecture, ( ![X]: ({$box}(f(X))) ) => {$box}(![X]: f(X)) ).
```

This specifies a modal logic with rigid function symbols, decreasing quantification domains and box operators satisfying modal axiom schemes K and 5. Leo-III returns ...

```
% SZS status Theorem for barcan.p
```

However, when the parameter `$quantification` is changed to `$cumulative` or `$varying` the problem becomes unprovable (as expected).

As an optimization of earlier embedding tools, **LET** can output polymorphic HOL problems encoded in TH1 (given the parameter `-p POLYMORPHIC` when used as executable). TH1 extends the monomorphic THF format with polymorphic types (rank-1 polymorphism) [30]. An excerpt of the output of **LET** for the reasoning problem is:

```
1  thf(mworld, type, mworld: $tType).
2  thf(mrel_type, type, mrel: mworld > (mworld > $o)).
3
4  thf(mactual_type, type, mactual: mworld).
5  thf(mlocal_type, type, mlocal: (mworld > $o) > $o).
6  thf(mlocal_def, definition, mlocal = (^ [Phi:(mworld > $o)]: ((Phi @ mactual))) ).
7  [...]
8  thf(mbox_type, type, mbox: (mworld > $o) > (mworld > $o)).
9  thf(mbox_def, definition, mbox = (^ [Phi:(mworld > $o),W:mworld]:
10                                    (! [V:mworld]: ( (mrel @ W @ V) => (Phi @ V) )) )).
11 thf(mrel_euclidean, axiom, ! [W:mworld,V:mworld,U:mworld]:
12                               ( ((mrel @ W @ U) & (mrel @ W @ V)) => ( mrel @ U @ V) ) ).
13
14 thf(eiw_type, type, eiw: !> [T:$tType]: (T > (mworld > $o))).
15 thf(eiw_nonempty, axioms, ! [T:$tType,W:mworld]: ((? [X:T]: ((((eiw @ T) @ X) @ W))))).
16 thf(eiw_di_cumul, axiom, ! [W:mworld,V:mworld,X:$i]:
17                           (( (eiw @ $i @ X @ W) & (mrel @ V @ W) ) => ( eiw @ $i @ X @ V ))).
18 thf(mforall_vary, type, mforall_vary: !> [T:$tType]: ((T > (mworld > $o)) > (mworld > $o))).
19 thf(mforall_vary_def, definition, mforall_vary = (^ [T:$tType,A:(T > (mworld > $o)),W:mworld]:
20                                      (! [X:T]: ( (eiw @ T @ X @ W) => (A @ X @ W) ))) ).
21 [...]
22 thf(f_type, type, f: $i > (mworld > $o)).
23 thf(bf, conjecture, mlocal @ ( mimplies
24                                @ (mforall_vary @ $i @ (^ [X:$i]: (mbox @ (f @ X))))
25                                @ (mbox @ ((mforall_vary @ $i) @ (^ [X:$i]: ((f @ X))))) ).
```

In lines 1–12, meta-logical notions and the semantics of the modal logic connectives are defined. Intuitively, a new type for possible worlds `mworld` and a relation symbol

`mrel` as their accessibility relation is introduced, following the usual Kripke semantics for modal logics. Modal logics formulas are encoded as predicates over possible worlds. Then, the box operator is defined to hold whenever the original formula, encoded as a predicate, holds at every accessible world, and the accessibility relation is defined to be euclidean (according to modal modal **K5**). We refer to the literature for details on the encoding process [39, 28]. Subsequently (lines 14–17), an *exists-in-world* predicate is defined to modal varying domain semantics [28]. In the TH1 variant, this is encoded as a polymorphic symbol `eiw` such that objects of any type can be asserted to exist at a particular world using a single predicate. Analogously (lines 18–20), the modal logic universal quantifier is defined as a polymorphic symbol `mforall_vary`. In monomorphic embeddings, the exists-in-world predicates and the quantifier symbols need to be defined for every relevant type occurring in the original problem file individually. Finally, lines 22–25 show the translated conjecture of the Barcan reasoning problem.

Example 2: Hybrid logic reasoning. Hybrid logics can talk about the satisfaction relation of the modal logic at the object language level. Up to the author's knowledge, there are no ATP systems available that can reason in first-order hybrid logics, let alone in the many variants offered by **LET**. An example tautology is given by

$$\forall X.\, \Box @_n \big(\downarrow Y.\, (Y \wedge p(X)) \Leftrightarrow (n \wedge p(X)) \big)$$

that is encoded as ...

```
1  tff(hybrid_s5,logic, $$hybrid == [
2      $constants == $rigid,
3      $quantification == $varying,
4      $modalities == $modal_system_S5
5    ] ).
6
7  tff(1, conjecture, ![X]: {$box}({$$shift(#n)}(
8             {$$bind(#Y)}((Y & p(X))
9                          <=> ({$$nominal}(n) & p(X)) ))) ).
```

Example 3: Contrary-to-duty (CTD) reasoning in deontic logics. In deontic logics, CTD situations arise when reasoning with obligations that prescribe what to do if other (primary) obligations are violated. Simple approaches, e.g., using modal logic **D**, lead to inconsistencies that allow arbitrary conclusions to be inferred. This is addressed by dyadic deontic logics that encode conditional norms using a special operator $\bigcirc(\psi, \varphi)$ (read: *it ought to be ψ, given φ*) represented as `{$$obl}(`ψ`, `φ`)`. An example is ...

```
1  tff(spec_e, logic, $$ddl == [ $$system == $$aqvistE ] ).
2
3  tff(a1, axiom, {$$obl}(go,$true)).
4  tff(a2, axiom, {$$obl}(tell, go)).
5  tff(a3, axiom, {$$obl}(~tell, ~go)).
6  tff(situation, axiom, ~go).
7  tff(c, conjecture, {$$obl}(~tell,$true)).
```

This example encodes that (a1) you ought to go and help your neighbors, (a2) if you go then you ought to tell them that you are coming, and (a3) if you don't go, then you ought not tell them. It can consistently be inferred that if you actually don't go, then you ought not tell them. After embedding into HOL using **LET**, higher-order ATP systems like Leo-III confirm this.

6. Summary

LET is a library and pre-processing tool for encoding non-classical reasoning problems into classical higher-order logic, by means of shallow embeddings. **LET** makes use of the novel non-classical TPTP language extension, as briefly described in Sect. 2, for reading reasoning problems in various non-classical logics as input. The output of the tool is TPTP THF, and any compatible ATP system can be used in conjunction with it, offering off-the-shelf automation for various non-classical logics. The library of **LET** is included into the Leo-III prover so that no external processing steps are required.

LET allows for the automation of more than 60 different first-order modal logics (including all logics from the modal cube), 60 different hybrid logics, dynamic epistemic logic (PAL), and different dyadic deontic logics.[2] For some of these logics there exist no other ATP systems to date. The tool is designed to be easily extensible with new embeddings of further logics.

Shallow embeddings and hence **LET** target rapid logic prototyping, and will, in general, not be as effective as ATP systems specifically designed for the respective NCL. The embedding approach allows for the automation of logics that otherwise would have no automation at all. **LET** aims at closing automation gaps for interesting NCLs, rather than challenging available ATP systems. Nevertheless, previous studies indicate that embeddings perform quite competitively in the context of quantified benchmarks [52]. For many of the logic families currently covered by **LET**, in particular for quantified hybrid logics and deontic logics, there are no benchmark sets or competitors available, and hence comparisons are not possible. Automation via embeddings can also be employed in an educational context for low-threshold student experiments.

It is important to note that proofs found in conjunction with **LET** are expressed in some calculus for classical HOL (depending on the back-end ATP system). It is an open question to what extent they can be automatically and flexibly transformed to genuine proofs in the respective NCL.

As further work, **LET** aims at gradually including more embeddings for further NCLs in order to constitute a rich and flexible NCL reasoning framework. Also, it is planned to extend the tool's portfolio of accepted input formats to further existing logic-specific input standards, such as the QMLTP format for modal logics [11].

[2] The number of modal logics is at least 15 *(modality axiomatizations)* $\times$ 4 *(quantification semantics)* $= 60$. Many more modal logics are supported since **LET** allows arbitrary combinations of different modalities. Also, quantification semantics can be controlled on a per-type basis.

References

[1] G. Priest, An Introduction to Non-Classical Logic: From If to Is, Cambridge University Press, 2008.

[2] K. Chaudhuri, F. Pfenning, A focusing inverse method theorem prover for first-order linear logic, in: R. Nieuwenhuis (Ed.), Proceedings of the 20th International Conference on Automated Deduction, volume 3632 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 69–83.

[3] H. Mantel, J. Otten, lintap: A tableau prover for linear logic, in: N. V. Murray (Ed.), Proceedings of the 8th Conference on Automated Reasoning with Analytic Tableaux and Related Methods, volume 1617 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 217–231.

[4] T. Raths, J. Otten, C. Kreitz, The ILTP Problem Library for Intuitionistic Logic - Release v1.1, Journal of Automated Reasoning 38 (2007) 261–271.

[5] S. Schmitt, L. Lorigo, C. Kreitz, A. Nogin, Jprover: Integrating connection-based theorem proving into interactive proof assistants, in: R. Goré, A. Leitsch, T. Nipkow (Eds.), Proceedings of the First International Joint Conference on Automated Reasoning, volume 2083 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 421–426.

[6] S. McLaughlin, F. Pfenning, Efficient intuitionistic theorem proving with the polarized inverse method, in: CADE, volume 5663 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 230–244.

[7] K. Claessen, D. Rosén, SAT modulo intuitionistic implications, in: LPAR, volume 9450 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 622–637.

[8] G. Ebner, Herbrand constructivization for automated intuitionistic theorem proving, in: TABLEAUX, volume 11714 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 355–373.

[9] C. Fiorentini, Efficient sat-based proof search in intuitionistic propositional logic, in: CADE, volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 217–233.

[10] J. Otten, The nanoCoP 2.0 Connection Provers for Classical, Intuitionistic and Modal Logics, in: A. Das, S. Negri (Eds.), Proceedings of the 30th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, number 12842 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2021, pp. 236–249.

[11] T. Raths, J. Otten, The QMLTP problem library for first-order modal logics, in: B. Gramlich, D. Miller, U. Sattler (Eds.), Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, volume 7364 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 454–461. doi:10.1007/978-3-642-31365-3_35.

[12] F. Papacchini, C. Nalon, U. Hustadt, C. Dixon, Efficient Local Reductions to Basic Modal Logic, in: A. Platzer, G. Sutcliffe (Eds.), Proceedings of the 28th International Conference on Automated Deduction, number 12699 in Lecture Notes in Computer Science, Springer-Verlag, 2021, pp. 76–92.

[13] J. Otten, MleanCoP: A Connection Prover for First-Order Modal Logic, in: S. Demri, D. Kapur, C. Weidenbach (Eds.), Proceedings of the 7th International Joint Conference on Automated Reasoning, number 8562 in Lecture Notes in Artificial Intelligence, 2014, pp. 269–276.
[14] D. Tishkovsky, R. Schmidt, M. Khodadadi, The Tableau Prover Generator MetTeL2, in: A. Platzer, G. Sutcliffe (Eds.), Proceedings of the 13th European conference on Logics in Artificial Intelligence, number 7519 in Lecture Notes in Computer Science, Springer, 2012, pp. 492–495.
[15] O. Gasquet, A. Herzig, D. Longin, M. Sahade, LoTREC: Logical tableaux research engineering companion, in: B. Beckert (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEAUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings, volume 3702 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 318–322. URL: https://doi.org/10.1007/11554554_25. doi:10.1007/11554554_25.
[16] H. Ohlbach, Semantics Based Translation Methods for Modal Logics, Journal of Logic and Computation 1 (1991) 691–746.
[17] H. Ohlbach, Translation Methods for Non-Classical Logics: An Overview, Logic Journal of the IGPL 1 (1993) 69–89.
[18] J. Gibbons, N. Wu, Folding domain-specific languages: deep and shallow embeddings (functional pearl), in: J. Jeuring, M. M. T. Chakravarty (Eds.), Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, ACM, 2014, pp. 339–347.
[19] P. Blackburn, J. van Benthem, Modal logic: a semantic perspective, in: P. Blackburn, J. F. A. K. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, volume 3 of *Studies in logic and practical reasoning*, North-Holland, 2007, pp. 1–84.
[20] A. Steen, logic-embedding v1.7, 2022. DOI: 10.5281/zenodo.5913215.
[21] A. Steen, D. Fuenmayor, T. Gleißner, G. Sutcliffe, C. Benzmüller, Automated Reasoning in Non-classical Logics in the TPTP World, 2022. URL: https://arxiv.org/abs/2202.09836. doi:10.48550/ARXIV.2202.09836.
[22] G. Sutcliffe, The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0, Journal of Automated Reasoning 59 (2017) 483–502.
[23] A. Church, A Formulation of the Simple Theory of Types, Journal of Symbolic Logic 5 (1940) 56–68.
[24] L. Henkin, Completeness in the Theory of Types, Journal of Symbolic Logic 15 (1950) 81–91.
[25] P. Andrews, General Models and Extensionality, Journal of Symbolic Logic 37 (1972) 395–397.
[26] C. Benzmüller, P. Andrews, Church's Type Theory, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, summer 2019 ed., Metaphysics Research Lab, Stanford University, 2019.
[27] C. Benzmüller, J. Otten, T. Raths, Implementing and evaluating provers for first-order modal logics, in: ECAI, volume 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2012, pp. 163–168.
[28] T. Gleißner, A. Steen, C. Benzmüller, Theorem Provers for Every Normal Modal

Logic, in: T. Eiter, D. Sands (Eds.), Proceedings of the 21st International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, number 46 in EPiC Series in Computing, EasyChair Publications, 2017, pp. 14–30.
[29] T. Gleißner, A. Steen, The MET: The Art of Flexible Reasoning with Modalities, in: C. Benzmüller, F. Ricca, X. Parent, D. Roman (Eds.), Proceedings of the 2nd International Joint Conference on Rules and Reasoning, number 11092 in Lecture Notes in Computer Science, 2018, pp. 274–284.
[30] C. Kaliszyk, G. Sutcliffe, F. Rabe, TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism, in: P. Fontaine, S. Schulz, J. Urban (Eds.), Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning, number 1635 in CEUR Workshop Proceedings, 2016, pp. 41–55.
[31] G. Sutcliffe, E. Kotelnikov, TFX: The TPTP Extended Typed First-order Form, in: B. Konev, J. Urban, S. Schulz (Eds.), Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning, number 2162 in CEUR Workshop Proceedings, 2018, pp. 72–87.
[32] G. Sutcliffe, C. Benzmüller, Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure, Journal of Formalized Reasoning 3 (2010) 1–27.
[33] T. Gleißner, A. Steen, G. Sutcliffe, C. Benzmüller, TPTP proposal: Non-classical logics, http://tptp.org/NonClassicalLogic, 2021.
[34] A. Steen, Scala TPTP Parser v1.6, 2022. DOI: 10.5281/zenodo.4468958.
[35] G. Sutcliffe, J. Zimmer, S. Schulz, TSTP Data-Exchange Formats for Automated Theorem Proving Tools, in: W. Zhang, V. Sorge (Eds.), Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems, number 112 in Frontiers in Artificial Intelligence and Applications, IOS Press, 2004, pp. 201–215.
[36] T. Braüner, S. Ghilardi, First-order modal logic, in: P. Blackburn, J. F. A. K. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, volume 3 of *Studies in logic and practical reasoning*, North-Holland, 2007, pp. 549–620.
[37] M. Fitting, R. Mendelsohn, First-Order Modal Logic, Kluwer, 1998.
[38] C. Benzmüller, Combining and automating classical and non-classical logics in classical higher-order logics, Ann. Math. Artif. Intell. 62 (2011) 103–128. doi:10.1007/s10472-011-9249-7.
[39] C. Benzmüller, L. Paulson, Quantified Multimodal Logics in Simple Type Theory, Logica Universalis 7 (2013) 7–20.
[40] C. Areces, B. ten Cate, Hybrid logics, in: P. Blackburn, J. F. A. K. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, volume 3 of *Studies in logic and practical reasoning*, North-Holland, 2007, pp. 821–868.
[41] M. Wisniewski, A. Steen, Embedding of quantified higher-order nominal modal logic into classical higher-order logic, in: C. Benzmüller, J. Otten (Eds.), Workshop on Automated Reasoning in Quantified Non-Classical Logics, volume 33 of *EPiC Series in Computing*, EasyChair, 2014, pp. 59–64.
[42] H. van Ditmarsch, J. Y. Halpern, W. van der Hoek, B. P. Kooi, An introduction to logics of knowledge and belief, in: Handbook of epistemic logic, 2015.
[43] C. Benzmüller, S. Reiche, Automating Public Announcement Logic and the Wise Men Puzzle in Isabelle/HOL, Arch. Formal Proofs 2021 (2021). URL: https://www.

isa-afp.org/entries/PAL.html.
[44] R. M. Chisholm, Contrary-to-duty imperatives and deontic logic, Analysis 24 (1963) 33–36.
[45] J. Carmo, A. J. I. Jones, Completeness and decidability results for a logic of contrary-to-duty conditionals, J. Log. Comput. 23 (2013) 585–626.
[46] L. Åqvist, Deontic logic, in: Handbook of philosophical logic, Springer, 2002, pp. 147–264.
[47] C. Benzmüller, A. Farjami, X. Parent, Dyadic deontic logic in HOL: Faithful embedding and meta-theoretical experiments, in: New Developments in Legal Reasoning and Logic, Springer, 2022, pp. 353–377.
[48] C. Benzmüller, A. Farjami, X. Parent, Åqvist's Dyadic Deontic Logic E in HOL, Journal of Applied Logics 6 (2019) 733–755.
[49] A. Steen, C. Benzmüller, Extensional Higher-Order Paramodulation in Leo-III, Journal of Automated Reasoning 65 (2021) 775–807.
[50] A. Steen, Supplemental data to paper "An Extensible Logic Embedding Tool for Lightweight Non-Classical Reasoning", 2022. URL: https://doi.org/10.5281/zenodo.6651452. doi:10.5281/zenodo.6651452.
[51] R. Barcan, A Functional Calculus of First Order Based on Strict Implication, Journal of Symbolic Logic 11 (1946) 1–16.
[52] A. Steen, Extensional Paramodulation for Higher-Order Logic and its Effective Implementation Leo-III, volume 345 of *DISKI – Dissertations in Artificial Intelligence*, Akademische Verlagsgesellschaft AKA GmbH, Berlin, 2018. Dissertation, Freie Universität Berlin, Germany.