

Automated Reasoning in Non-classical Logics in the TPTP World

Alexander Steen^{1,2}, David Fuenmayor^{2,5}, Tobias Gleißner³, Geoff Sutcliffe⁴ and Christoph Benzmüller^{5,6}

¹University of Greifswald, Germany

²University of Luxembourg, Luxembourg

³Fraunhofer FOKUS, Germany

⁴University of Miami, USA

⁵University of Bamberg, Germany

⁶Freie Universität Berlin, Germany

Abstract

Non-classical logics are used in a wide spectrum of disciplines, including artificial intelligence, computer science, mathematics, and philosophy. The de-facto standard infrastructure for automated theorem proving, the TPTP World, currently supports only classical logics. This paper describes the latest extension of the TPTP World, providing languages and infrastructure for reasoning in non-classical logics. The extension integrates seamlessly with the existing TPTP World.

Keywords

TPTP World, Non-classical Logics, Automated Reasoning

1. Introduction

The TPTP World [1] is a well established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems. The TPTP World includes the TPTP problem library, the TSTP solution library, standards for writing ATP problems and reporting ATP solutions, tools and services for processing ATP problems and solutions, and it supports the CADE ATP System Competition (CASC). Various parts of the TPTP World have been deployed in a range of applications, in both academia and industry. The web page <https://www.tptp.org> provides access to all components.

The TPTP languages [2] are one of the keys to the success of the TPTP World. The languages are used for writing both TPTP problems and TSTP solutions, which enables convenient communication between different systems and researchers. It also enables

PAAR'22: 8th Workshop on Practical Aspects of Automated Reasoning, August 11–12, 2022, Haifa, Israel

EMAIL: alexander.steen@uni-greifswald.de (A. Steen); david.fuenmayor@uni.lu (D. Fuenmayor);


tobias.gleissner@fokus.fraunhofer.de (T. Gleißner); geoff@cs.miami.edu (G. Sutcliffe);

c.benzmueller@gmail.com (C. Benzmüller)

ORCID: 0000-0001-8781-9462 (A. Steen); 0000-0002-0042-4538 (D. Fuenmayor); 0000-0002-7730-5852

(T. Gleißner); 0000-0001-9120-3927 (G. Sutcliffe); 0000-0002-3392-30935 (C. Benzmüller)

© 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)

tool exchange, tool integration, and comparable experimental results. Originally the TPTP World supported only first-order clause normal form (CNF) [3]. Over the years full first-order form (FOF) [4], typed-first order form (TFF) [5, 6], and typed higher-order form (THF) [7, 8] have been added. The TFF and THF languages include constructs for arithmetic.

This paper describes the latest extension of the TPTP World, providing languages and infrastructure for reasoning in non-classical logics [9, 10], via the (new) non-classical typed extended first-order (NXF) and non-classical typed higher-order (NHF) languages, based on the existing typed extended first-order (TXF) and typed higher-order (THF) languages. The default typing rules of TFF/TXF/NXF (see Sections 2.2 and 3.1) means that a non-classical untyped first-order form is also supported. NXF and NHF support a broad range of non-classical logics. Problems, solutions, and the logic to be used for reasoning, are expressed in the same language framework. In this paper we exemplify the languages using modal logics [11]. However, at all times the reader should keep in mind that the intention is for the languages to have much broader capability. For example, syntactically, the new languages allow multiple non-classical logics to be used together (while, of course, the semantic implications of using such combinations need to be carefully considered). TPTP-related tools e.g. parsers, syntax checkers, encoders, etc. are easily applicable to any non-classical logic formulated in this uniform syntax. In the medium to long term it is hoped that experts in various non-classical logics will use the TPTP framework to develop specifications that can be assimilated into the TPTP World. Stakeholders are invited to contribute!¹

Motivation.

The development of standards for ATP systems for first- and higher-order logic has traditionally focused mostly on classical logic, while many real-world applications often also require non-classical reasoning. These applications include artificial intelligence (e.g., knowledge representation, planning, multi-agent systems), philosophy (e.g., formal ethics, metaphysics), natural language semantics (e.g., generalized quantifiers, modalities), and computer science (e.g., software and hardware verification). There are also recent developments in natural and life sciences that employ logical reasoning (e.g., modelling of biochemical processes).

There has been a gradual disconnect between classical and non-classical logics in the practical development and handling of automated reasoning technology, with classical logics receiving greater attention. This is unfortunate because there exist ATP systems for non-classical logics but their usage, interoperability, and incorporation within larger contexts is hampered by their heterogeneous input formats and non-uniform modes of result reporting. Furthermore, various non-classical logics can be reduced to classical logics, e.g., the well-known standard translation of modal logics to classical first-order logic [12], but classical logic ATP systems have not yet been fully exploited for non-classical reasoning modulo such translations. This work aims to provide a fruitful bridge between the different communities, and foster the interoperability of classical and non-classical

¹Send an email to the fourth author, geoff@tptp.org.

reasoning systems. A preliminary format proposal was discussed in earlier work [13].

Related work.

Knowledge representation formats for non-classical logics have been developed and applied in the well known ILTP [14] and QMLTP [15] projects, which anticipated and largely coincide with our goals. These two projects have contributed significantly to the practical application of first- and higher-order theorem provers for non-classical logics [16]. However, the representation formats provided in ILTP and QMLTP focus on unary modal connectives, and do not support the much broader range of non-classical logics that are the target of our work, e.g., logics that require support for arbitrary n -ary operators, or generalizations of modal operators indexed by (lists of) terms. An example is dynamic epistemic logics, e.g., the recent mechanization of public announcement logic with relativized general knowledge [17].

The “DFG syntax” [18], a format for problem and proof interchange developed in the DFG Schwerpunktprogramm Deduktion, contains a meta-information tag called `logic` that can be used to specify “non-standard quantifiers or operators” in informal natural language. This has had some limited use [19, 20].

The Knowledge Interchange Format (KIF) [21] is a comprehensive format for knowledge representation, including numbers, lists, sets, and non-monotonic rules. KIF could be considered a language for non-classical logic. However, KIF is based on a first-order language and comes with a fixed semantics. It is not flexible enough to capture different logics.

Common Logic (CL) is an ISO standard [22] for the representation of logical information, with several dialects and a common general XML-based syntax. While allowing expressing both first-order and higher-order concepts, it also comes with a fixed semantics.

The OMDoc format [23] is also XML-based, and is geared primarily towards uniform representation of mathematical knowledge. MMT [24] extends and heavily redesigns the formal subset OMDoc. MMT aims at providing foundation independent means of specifying formal systems.

Paper structure.

Section 2 reviews the general structure of the TPTP languages and the existing TFF and THF languages, and introduces extensions to TFF and THF that underlie the new non-classical languages. Section 3 presents the new NXF and NHF languages for non-classical logics. This includes the form of the non-classical connectives, and a format for specifying the logic to be used when reasoning. Section 5 exemplifies the new languages with multi-modal logic, including an illustrative example. Section 6 describes some tools that are being developed to process and reason in non-classical logics. Section 7 concludes. The resources being developed, including example ATP problems in modal logic, are available at <https://github.com/TPTPWorld/NonClassicalLogic>

2. The TPTP Languages

The TPTP languages are human-readable, machine-parsable, flexible, and extensible languages, suitable for writing both problems and solutions. The new TPTP languages described in this paper support the representation of problems and solutions in non-classical logics². In this section the general structure of the TPTP languages is reviewed, and key features of the TXF and THF languages that underlie the new non-classical languages are presented. The syntax of the TPTP languages is available in an extended BNF [26].³

2.1. The Structure of the TPTP Languages

The top-level building blocks of the TPTP languages are *annotated formulae*. An annotated formula has the form:

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

The *languages* supported are clause normal form (**cnf**), first-order form (**fof**), typed first-order form (**tff**), and typed higher-order form (**thf**). The *name* assigns a (unique) identifier to each formula, for referring to it. The *role*, e.g., **axiom**, **lemma**, **conjecture**, defines the use of the formula in an ATP system. In the *formula*, terms and atoms follow Prolog conventions. The TPTP language also supports interpreted symbols, including: "the type of types" **\$tType**; types for individuals **\$i** (ι) and booleans **\$o** (o); types for numbers **\$int** (integers), **\$rat** (rationals), and **\$real** (reals); numeric constants such as 27, 43/92, -99.66; arithmetic predicates and functions such as **\$greater** and **\$sum**; the truth constants **\$true** and **\$false**. The basic logical connectives are \wedge , $!$, $?$, $@$, \sim , $|$, $\&$, \Rightarrow , \Leftarrow , \Leftrightarrow , and $\langle \sim \rangle$, for λ , \forall , \exists , higher-order application, \neg , \vee , \wedge , \Rightarrow , \Leftarrow , \Leftrightarrow , and \oplus respectively. Equality and inequality are expressed as the infix operators $=$ and \neq . The *source* and *useful_info* are optional (extra-logical) information about the origin and useful details about the formula. See [1] or the TPTP web site <https://www.tptp.org> for all the details. An example annotated first-order formula defining the set-theoretic union operation, supplied from a file named SET006+1.ax, is ...

```
fof(union,axiom,
  ( ! [X,A,B] :
    ( member(X,union(A,B))
      <=> ( member(X,A) | member(X,B) ) ),
  file('SET006+0.ax',union),
  [description('Definition of union'), relevance(0.9)]).
```

2.2. The Existing TFF and THF Languages

The typed first-order form (TFF) language extends FOF with types and type declarations. Predicate and function symbols can be declared before their use, with type signatures

²The development of TPTP World standards for writing ATP solutions beyond common derivations and models is still necessary – see, e.g., [25]

³ <https://www.tptp.org/TPTP/SyntaxBNF.html>

that specify the types of their arguments and result. An expression $(t_1 * \dots * t_n) > \$o$ is the type of an n -ary predicate, where the i -th argument is of type t_i , and it returns a Boolean. Analogously, and expression $(t_1 * \dots * t_n) > t$ is the type of a function that returns a term of type t . TFF supports arithmetic (which requires types, i.e., arithmetic is not supported in CNF or FOF). A useful feature of TFF is default typing for symbols that are not declared: predicates default to $(\$i * \dots * \$i) > \$o$, and functions default to $(\$i * \dots * \$i) > \$i$. This allows TFF to effectively degenerate to untyped FOF.

The monomorphic variant of TFF is called TF0. For example ...

```
tff(dog_decl,type,      dog: $tType ).
tff(human_decl,type,    human: $tType ).
tff(owner_of_decl,type, owner_of: dog > human ).
tff(bit_decl,type,      bit: (dog * human * $int) > $o ).
tff(hates_decl,type,    hates: (human * human) > $o ).

tff(hate_the_multi_biter_dog,axiom,
  ! [D: dog,H: human,N: $int] :
    ( ( H != owner_of(D) & bit(D,H,N) & $greater(N,1) )
      => hates(H,owner_of(D)) ) ).
```

The typed higher-order form (THF) extends TFF with higher-order notions, including the curried form of type declarations, lambda terms with a lambda binder \sim for λ , application with $@$, a choice binder $@+$ for ϵ , and a description binder $@-$ for ι . In THF all symbols must be declared before their use (default typing is not possible).

The monomorphic variant of THF is called TH0. For example ...

```
thf(dog_decl,type,      dog: $tType ).
thf(human_decl,type,    human: $tType ).
thf(owner_of_decl,type, owner_of: dog > human ).
thf(owns_decl,type,     owns: human > dog > $o ).
thf(bit_decl,type,      bit: dog > human > $int > $o ).
thf(hates_decl,type,    hates: human > human > $o ).

thf(owns_defn,definition,
  ( owns = ( ~ [H: human,D: dog] : ( H = ( owner_of @ D ) ) ) ) ).

thf(hate_the_multi_biter_dog,axiom,
  ! [D: dog,H: human,N: $int] :
    ( ~ ( owns @ H @ D ) & ( bit @ D @ H @ N ) & ( $greater @ N @ 1 )
      => ( hates @ H @ ( owner_of @ D ) ) ) ).
```

The polymorphic variants of TFF and THF, called TF1 [6] and TH1 [8], add type constructors, type variables, and polymorphic symbols. TH1 also adds five polymorphic constants: $!!$ for Π (universal quantification), $??$ for Σ (existential quantification), $@@+$ for ϵ (choice), $@@-$ for ι (definite description), and $@=$ for typed equality. The monomorphic variants TF0 and TH0 are currently more widespread than the polymorphic variants, and they are the basis for the languages introduced in this paper.

2.3. The TXF and THF Languages

The TXF Language.

The typed extended first-order form (TXF) [27]⁴ augments TFF with FOOL logic [28] constructs: formulae of type \$o as terms; variables of type \$o as formulae; tuple types and tuple terms; conditional (if-then-else) and let (let-defn-in) expressions (these are particularly useful in software verification applications [29]). TXF can be translated to first-order logic [28]. TXF provides the basis for the non-classical typed extended first-order form (NXF) described in Section 3. The monomorphic variant of TXF is called TX0. Augmenting the TF0 example from above ...

```
tff(odie_decl,type,      odie: dog ).
tff(jon_decl,type,       jon: human ).
tff(feeds_decl,type,     feeds: (human * dog) > $o ).
tff(chases_decl,type,    chases: (human * dog) > $o ).
tff(says_decl,type,      says: (human * $o) > $o ).

tff(feed_the_non_biter_dog,axiom,
  ! [D: dog,H: human] :
    $ite(
      ? [N: $int] : ( bit(D,H,N) & $greater(N,0) ),
      chases(H,D), feeds(H,D)) ).

tff(jon_says_a_dog_bit_him_twice,axiom,
  ? [D: dog] :
    ( D != odie & jon != owner_of(D) & says(jon,bit(D,jon,2)) ) ).

tff(jon_says_truth,axiom,
  ! [S: $o] : ( says(jon,S) => S ) ).
```

The (not really extended) THF Language.

In parallel to the development of TXF, THF has been revised to have the same structures as TXF for tuples, conditional expressions, and let expressions. The revised THF provides the basis for the non-classical typed higher-order (NHF) language described in Section 3. In THF the features of FOOL are naturally available, and thus their presentation in the TXF context is immediately adopted in THF. Augmenting the TH0 example from

⁴The language was called TFX in [27].

above ...

```

thf(odie_decl,type,      odie: dog ).
thf(jon_decl,type,       jon: human ).
thf(feeds_decl,type,     feeds: human > dog > $o ).
thf(chases_decl,type,    chases: human > dog > $o ).
thf(says_decl,type,      says: human > $o > $o ).

thf(feed_the_non_biter_dog,axiom,
  ! [D: dog,H: human] :
    $ite(
      ? [N: $int] : ( (bit @ D @ H @ N) & ($greater @ N @ 0) ),
      chases @ H @ D, feeds @ H @ D) ).

thf(jon_says_a_dog_bit_him_twice,axiom,
  ? [D: dog] :
    ( ( D != odie ) & (~ ( owns @ jon @ D ))
      & ( says @ jon @ ( bit @ D @ jon @ 2 ) ) ) ).

thf(jon_says_truth,axiom,
  ! [S: $o] : ( ( says @ jon @ S ) => S ) ).

```

3. The NXF and NHF Languages

The non-classical typed extended first-order form (NXF) and non-classical typed higher-order form (NHF) languages are the new TPTP languages for non-classical logics, extending TXF and THF respectively (note the mnemonic ‘*N*’ in the names NXF and NHF, indicating *Non*-classical). The design of NXF and NHF adopted the following principles: (i) syntactic consistency with the underlying classical languages, (ii) a uniform syntax for a wide range of non-classical logics and connectives, and (iii) requiring minimal changes to existing parsing and reasoning software. The underlying typed languages provide users with useful expressive power, and types are necessary for some commonly needed concepts, e.g., arithmetic. Recall, however, from Section 2.2, that default typing provides an untyped first-order language too.

The new languages add new non-classical connectives (Section 3.1), and a syntax for specifying the logic to be used for reasoning (Section 4). QMLTP library problems and other sample problems have been translated to the new languages, which has provided an initial test of adequacy. Further testing with more logics is planned.

3.1. The Non-Classical Connectives

NXF and NHF add a new interpreted functor-like connective form ...

{connective_name}

The *connective_name* is a TPTP defined symbol or system symbol, i.e., starting with \$ or \$\$, naming a non-classical connective. If the *connective_name* is a TPTP defined symbol then its meaning is documented in the TPTP. If the *connective_name* is a system symbol then its meaning is defined by the user/ATP system being used, thus allowing the TPTP

syntax to be used when experimenting with logics that have not been formalized in the TPTP. A *connective_name* may optionally be parameterized, as explained below. In NXF the non-classical connectives are applied in a mixed “higher-order applied”/“first-order functional” style, with the connectives applied to a ()ed list of arguments.⁵ In NHF the non-classical connectives are applied in higher-order style ...

- In NXF $\{connective_name\} @ (arg_1, \dots, arg_n)$ is a formula, where each arg_i is an NXF term. NXF terms are defined as for TXF, including formulae.
- In NHF $\{connective_name\} @ arg_1 @ \dots @ arg_n$ is a formula, where each arg_i is an NHF term. NHF terms are defined as for THF, including formulae.

Despite their functional appearance, non-classical operators are different from usual predicates, as indicated by their enclosing braces. They can be supplied with an arbitrary number of both formulae and terms as arguments. The chosen format is a trade-off between conciseness and generality, and allows for a uniform representation of non-classical operators with arbitrary arity. Augmenting the TXF and THF examples from above, using the box and diamond connectives from normal modal logic ...

```
tff(possible_dog_bit_owner,axiom,
    {$dia} @ (? [D: dog] : bit(D,owner_of(D),1)) ).
```

```
tff(jon_says_necessary_truth,axiom,
    ! [S: $o] : ( says(jon,S) => {$box} @ (S) ) ).
```

```
thf(possible_jon_owns_biter,axiom,
    ! [D: dog] :
      ( ( bit @ D @ jon @ 1 )
        => ( {$dia} @ ( owns @ jon @ D ) ) ) ).
```

```
thf(jon_says_he_must_feed_odie,axiom,
    says @ jon @ ({$box} @ (feeds @ jon @ odie)) ).
```

A *connective_name* may optionally be parameterized to reflect more complex non-classical connectives, e.g., in multi-modals logics where the modal operators are indexed, in epistemic logics [31] where the common knowledge operator can specify the agents under consideration, and in dynamic logics [32] where the connectives are parameterized with (complex) programs. The form is ...

$\{connective_name(param_1, \dots, param_n)\}$

If the connective is indexed, i.e., representing a family of connectives parameterized over some index set of constants, the index is given as the first argument as a constant (uninterpreted constant, number, or TPTP defined constant) prefixed with a #. All other parameters are key-value pairs of the form ...

$parameter_name := parameter_value$

⁵This slightly unusual form was chosen to reflect pure first-order functional style, but by making the application explicit the formulae can be parsed in Prolog - a long standing principle of the TPTP languages [30].

where the *parameter_name* is a constant, and the *parameter_value* is any term. In many logics, including the examples from modal logics below, the parameter values (including index values) are on the meta level. They are thus distinct from symbols (even of the same name) occurring at the object level, and are not declared with types. In the future, more complex logics such as term-modal logics [33] or term-sequence modal logics [34] might merge these levels; the syntax does not prohibit this, and any kind of parameterization of connectives on the object or meta level is permitted by this dictionary-like syntactical structure.

Augmenting the unparameterised examples from above, using connectives from epistemic logic where $\$knows(\#agent)$ is the knowledge operator for *agent*, and $\$common$ is the common knowledge operator for a set of agents encoded as a key-value parameter $\$agents:=[...]$...

```
tff(alice_knows_its_possible_odie_bit_jon,axiom,
    {$knows(#alice)} @ ({$dia} @ (bit(odie,jon,1)) ).

tff(jon_says_common_knowledge,axiom,
    ! [S: $o] :
      ( says(jon,S) => {$common($agents:=[alice,bob,claire])} @ (S) ) ).

tff(alice_knows_jon_owns_a_dog,axiom,
    {$knows(#alice)} @
      ? [D: dog] : ( owns @ jon @ D ) ).

tff(alice_and_bob_know_jon_might_lie,axiom,
    ! [S: $o] :
      ( (says @ jon @ S )
        => {$common($agents:=[alice,bob])} @ ({$dia} @ ~ S ) ) ).
```

As was noted in Section 2.2, the default typing rules of TFF and TXF, and hence also of NXF, allows them to degenerate to untyped languages. In the following example *bird* and *fly* default to predicates of type $\$i > \o , *tweety* defaults to a constant of type $\$i$, and *X* defaults to a variable of type $\$i$. It uses an exemplary system-defined non-classical binary connective $\$usually$, denoting some kind of (not further specified) non-monotonic conditional:

```
tff(birds_fly,axiom,
    ! [X] : {$usually} @ (bird(X),fly(X)) ).

tff(tweety_is_bird,axiom, bird(tweety) ).

tff(tweety_conjecture, fly(tweety) ).
```

4. Logic Specifications

In the world of non-classical logics the intended logic cannot always be inferred from the language used for the formulae – the same language can be used for formulae while

different logics are used for reasoning. A paradigmatic example of this underspecification phenomenon occurs in intuitionistic logic. In modal logic, when reasoning about metaphysical necessity **S5** is usually used, but when reasoning about deontic necessities a more suitable choice might be **D**. Thus when a formula uses a modal connective it is unknown what notion of necessity is intended, and in quantified logics it is unknown how necessity interacts with quantification, e.g., if $\forall x. \Box P(x)$ entails $\Box \forall x. P(x)$. It is therefore necessary to provide (meta-)information that specifies the logic to be used. A new kind of TPTP annotated formula has been introduced for this, with the role `logic`, and a “logic specification” as the formula.

A logic specification consists of a defined logic (family) name identified with a list of properties, e.g., in NXF ...

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

where *properties* is a []ed list of key-value identities ...

```
property_name == property_value
```

where each *property_name* is a TPTP defined symbol or a system symbol, and each *property_value* is either a term of the language (often a defined constant) or a []ed list that might start with a term (often a defined constant), and otherwise contains key-value identities. If the first element of a *property_value* list is a term then that is the default value for all cases that are not specified by the following key-value identities. A simple example from modal logic is ...

```
tff(simple_spec,logic,
    $modal == [
        $constants == $rigid,
        $quantification == [ $constant, some_user_type == $varying ],
        $modalities == $modal_system_S5 ] ).
```

See Section 5 for more sophisticated examples, and Section 5.1 in particular for the explanation of the *property_names* and *property_values* used here.

The BNF grammar for logic specifications is available at the TPTP page, see footnote 3. The grammar is quite unrestrictive, and allows quite complicated specifications, e.g., arbitrary formulae can be used as *property_values*. It is flexible enough to be used for many different logics, users can create specifications for logics that are not defined in the TPTP, and it is possible to specify the same logic in different ways. It is also possible (users beware) to write meaningless specifications in a syntactically well-formed way – a tool to check the sanity of a specification is available (see Section 6). It is also clear that the logic specification resources provided at this stage of development should not already be considered exhaustive and conclusive; the property-value pairs currently supported arise from needs and experience with the range of logics currently under consideration, and might be further modified and extended.

A NXF or NHF problem file must have one logic specification, and it typically comes first in the file. The logic specification binds meta-logical information to the object-level information in the problem formulae. It is an error to use a non-classical connective without a logic specification, or to underspecify the logic. Note that the logic specification can change the meaning of language features such as truth-values, universal quantification,

etc. – existing meanings from classical logic should not be confused with the meanings in the declared logic.

5. Case Study: Multi-Modal Logics

Quantified normal multi-modal logics [11] is the first family of non-classical logics defined in the TPTP. The standardization originates from preliminary work [35, 36] based on the QMLTP syntax.⁶ In this section the TPTP representation of quantified normal multi-modal logics is introduced, and the logic specification properties are discussed. A logic puzzle is presented to exemplify its usage.

5.1. Syntax and Logic Specification

The formula language of quantified normal multi-modal logics is that of classical logics without equality, augmented with a unary connective \Box , with an indexed form \Box_i . The reading of $\Box\varphi$ depends on the application context, such as “ φ is necessary”, “ φ is obligatory”, and “ φ is known”. From here forward these connectives are used without any assumption about the intended reading unless stated. The dual \Diamond (and similarly \Diamond_i) is defined by $\Diamond\varphi := \neg\Box\neg\varphi$. Note that any multi-modal language can also be regarded a mono-modal language if there is only one index value.

Quantified normal multi-modal logic is named `$modal` in the TPTP. The connectives are `{ $box }` and `{ $dia }`, with the indexed forms `{ $box(#i) }` and `{ $dia(#i) }`. The indices are uninterpreted constants on the meta-level, as described in Section 3.1. For increased readability, the TPTP also defines specialized modal logics with more specific names for the connectives. The logics are `$alethic_modal`, `$deontic_modal`, and `$epistemic_modal`. Each of these is identical to `$modal` in terms of syntax and parameterization except that `{ $box }` and `{ $dia }` are renamed to `{ $necessary }` and `{ $possible }`, `{ $obligatory }` and `{ $permissible }`, and `{ $knows }` and `{ $believes }`, respectively.

Logic specifications for `$modal` use three semantically oriented properties that characterize the logic to be used. The property names and their possible values are shown in Table 1.⁷

- The `$constants` property specifies whether symbols are interpreted as `$rigid`, i.e. interpreted as the same domain element in every world, or as `$flexible`, i.e., possibly interpreted as different domain elements in different worlds. The property can provide a single value for all *symbols*, or a default value and individual values for some symbols.
- The `$quantification` property specifies restrictions on the quantification domain across the accessibility relation [37], with the possible values `$constant`, `$varying`,

⁶The NXF and NHF translations of QMLTP problems can be found at <https://github.com/TPTPWorld/NonClassicalLogic/QMLTP>

⁷The properties for `$modal` could also be characterized by proof-theoretic properties. For example, `$quantification` can be characterized by properties that express whether or not the (converse) Barcan formula is a tautology.

Table 1Logic specification properties of `$modal`.

Property	Values
<code>\$constants</code>	<code>\$rigid</code> , <code>\$flexible</code>
<code>\$quantification</code>	<code>\$constant</code> , <code>\$varying</code> , <code>\$cumulative</code> , <code>\$decreasing</code>
<code>\$modalities</code>	<code>\$modal_system_X</code> $X \in \{K, KB, K4, K5, K45, KB5, D, DB, D4, D5, D45, T, B, S4, S5, S5U\}$ <i>or a list of axioms</i> <code>[\$modal_axiom_X₁, \$modal_axiom_X₂, ...]</code> $X_i \in \{K, T, B, D, 4, 5, CD, BoxM, C4, C\}$

`$cumulative`, and `$decreasing`. The property can provide a single value for all *types*, or a default value and individual values for some types.

- The `$modalities` property specifies properties of the connectives. Possible values are defined for well-known modal logic systems, e.g., `$modal_system_K`, and individual modal axiom schemes, e.g., `$modal_axiom_5`. They refer to the corresponding systems and axiom schemes of the modal logic cube [38]. The property can provide a single value for all *indices*, or a default value and individual values for some indices.

An example (a more sophisticated version of the example in Section 4) is ...

```
tff(complex_spec,logic,
  $modal == [
    $constants ==      [ $flexible, sun == $rigid ],
    $quantification == [ $constant,
                        planet_type == $varying],
    $modalities ==     [ $modal_system_K,
                        {$box(#1)} == $modal_system_KB,
                        {$box(#2)} == [ $modal_axiom_K,
                                        $modal_axiom_4 ] ] ).
```

In this example: • all symbols are flexible except for the symbol `sun` that is rigid, • quantification is over a constant domain, except for terms of type `planet_type` that are over varying domains, and • the default modality is **K**, but index #1 uses **KB** and index #2 uses the axiom schemes **K** and **4**.

The TPTP provides multiple roles to distinguish between various types of formulae that are assumed to be true at the start of reasoning, e.g., `axiom`, `hypothesis`, `lemma`, etc. Following the generalized notion of consequence by Fitting and Mendelsohn [37] in `$modal` the role `hypothesis` is used to indicate that the formula is assumed to be true *locally*, i.e., in the current world, and all other axiom-like roles, e.g., `axiom`, `lemma`, etc., are used to indicate that the formula is assumed to be true *globally*, i.e., in all worlds. TPTP *subroles* are used to override the local/global defaults, e.g., a formula with the role `axiom-local` is a local assumption (instead of a global one), and a formula with the role `hypothesis-global` is a global assumption (instead of a local one). For further background information on the local-global distinction see [37] and [39, Chap. 1.5].

5.2. Application Example

Four non-classical logicians, Tim, Fred, Betty and Nancy, walked into a bar.⁸ They form the steering committee (SC) of a non-classical logic conference. As the night goes on, and the empty glasses pile up, they start discussing the conference bylaws. Since one of the agreed rules is that all SC decisions are made by majority vote, they start arguing about the following (quite reasonable) rule:

“The number of SC members is necessarily an odd number.”

The situation is formalized in NXF using logic `$alethic_modal` as follows (`eq` represents an adequately axiomatized equality predicate ...

```
tff(four_members,hypothesis, eq(scMemberCount,4) ).
tff(four_not_odd,hypothesis, ~ odd(4) ).
tff(agreed_rule, hypothesis, {$necessary} @ (odd(scMemberCount)) ).
```

The discussion goes on as follows:

Tim: *This rule is hopelessly inconsistent: 4 is not an odd number. It cannot possibly be! Let's better forget about it.*

Fred: *I disagree, the rule per se is not inconsistent. The reason is that you take the term “the number of SC members” to rigidly denote the number 4.*

Tim's assumption that constants denote rigidly can be written in a logic specification ...

```
tff(tim,logic,
    $alethic_modal ==
    [ $constants == $rigid,
      $quantification == $constant,
      $modalities == $modal_system_S5 ] ).
```

In this setting the state of affairs is indeed inconsistent as confirmed by Leo-III.

Fred continues: *A better alternative is to take the term “the number of SC members” as flexibly denoting whatever number of SC members there happen to be. So if we were, say, 3 SC members, the rule would be perfectly fine. But I agree with you that, right now, the rule is of no use for us, since we can derive a contradiction that 4 is an odd number, so anything would follow ...*

Unlike Tim, Fred reasons assumes that `scMemberCount` denotes flexibly. However, he also employs an (alethic) modal logic that assumes necessity implies truth, i.e., adopting the modal axiom **T** ($\Box A \rightarrow A$) ...

```
tff(fred,logic,
    $alethic_modal ==
    [ $constants == [ $rigid, scMemberCount == $flexible ],
      $quantification == $constant,
      $modalities == [ $modal_axiom_K, $modal_axiom_T ] ] ).
```

Betty: *I agree with interpreting the term “the number of SC members” flexibly as you suggest. However, I don't see the rule deriving a contradiction. That something needs*

⁸They were probably proponents of the FDE logic [40]: Tim ordered a whisky (*true*), Fred ordered a glass of water (*false*), Betty ordered *both*, and Nancy ordered *neither*. But that's just a humorous coincidence that does not impact this example.

to be the case does not imply that something is actually the case. So the number of SC members is necessarily odd, yet it is four in the actual world. I don't see any trouble with this!

Betty assumes `scMemberCount` denotes flexibly, while using a modal logic that does not assume the modal axiom **T**. For instance, this can be the modal logic **D** (aka. *standard deontic logic* – SDL, where \Box is read normatively, e.g., as “it is obligatory that”) ...

```
tff(betty,logic,
  $alethic_modal ==
  [ $constants == [ $rigid, scMemberCount == $flexible ],
    $quantification == $constant,
    $modalities == $modal_system_D ] ).
```

Nancy: *Yes, I agree. The rule is perfectly consistent and, moreover, we should adopt it now! However, this means that we are actually violating the rule, so either someone else must come or one of us must go!* she says, looking at Tim.

Nancy also assumes that constants denote flexibly, while employing a more sophisticated logic, e.g., the deontic system **E** [41, 42]. In contrast to SDL this logic is suitably extended to deal with norm violations (e.g., *contrary-to-duty* reasoning) so that they do not result in inconsistencies. Alas, such a logic is not easily captured in `$modal`, and it might be necessary to use a more expressive logic employing a different specification.

Tim: *But we have to decide this by majority vote!*

6. Tools for the TPTP

The TPTP problem library v9.0.0 will include modal logic problems. There is a tool chain in place that has been used to convert QMLTP library problems to NXF and NHF, to provide an initial set of problems. The TPTP4X utility [43] will be extended to output formats for existing non-classical ATP systems, to provide those systems with a bridge to the TPTP problems, until they adopt the TPTP language natively. Contemporary systems to bridge to include, e.g., KSP [44, 45], nanoCoP 2.0 [46], MleanCoP [47], MetTeL2 [48], LoTREC [49], and MSPASS [50].

A suite of tools that can read, manipulate, and reason over problems written in the NXF and NHF languages is available in the Leo-III framework [51].⁹ Leo-III's parser is available as a stand-alone parsing library [52]. Problems in non-classical logics (including modal logic) are translated to THF using a shallow embedding [53, 54, 35, 36], and reasoning proceeds using Leo-III's THF capabilities. A generalization of the modal logic embedding procedure is available as an extensible library and executable, called LET (Logic Embedding Tool) [55], also available online,¹⁰ allowing any TPTP-compliant higher-order ATP system to be used as the backend in this tool chain. Currently LET supports the range of modal logics presented above, a range of first-order quantified hybrid logics, public announcement logic, and two different dyadic deontic logics. A tool to sanity check logic specifications for modal logics is available [56].

⁹Available online in SystemOnTPTP: <https://www.tptp.org/cgi-bin/SystemOnTPTP>

¹⁰<https://www.tptp.org/cgi-bin/SystemB4TPTP>

In order to compactly represent a set of problems using different logics with the same set of formulae, multiple logic specifications can be put in a *problem generator* file, with multiple corresponding **Status** values in the problem header. These will be distributed in the **Generators** directory of the TPTP problem library. The TPTP4X utility will expand such files to multiple individual files with a single logic specification and corresponding **Status** value. Selected individual files will be in the **Problems** directory of the TPTP.

7. Conclusion

This paper has described the new TPTP languages, NXF and NHF, for writing problems and solutions in non-classical logics. NXF and NHF support a new syntactic construct for non-classical logic connectives, and define a new type of annotated formula used to specify the logic to be used when reasoning. The use and flexibility of the proposed languages have been exemplarily demonstrated with modal logic. The proposed syntax is quite general and unrestrictive, and makes no a priori statement about semantics. Rather, the syntax provides a template that can be used with logics defined in the TPTP, and also by users who would like to have a TPTP-oriented input syntax for their specialized context. In both cases users will benefit from the TPTP infrastructure.

Further work.

The SZS success and failure ontologies [30] specify result values for ATP system reporting. In the light of the non-classical TPTP extension, the ontologies need to be extended to reflect additional success and failure situations, e.g., success values that are meaningful in (only certain) non-classical logics, and failure values for malformed logic specifications. The SZS dataform ontology needs similar attention.

In the medium-term more non-classical logics will be standardised in the TPTP, and problems in all the defined logics will be added to the TPTP problem library. The TPTP technical manual will document the defined symbols used in these logics – the connectives and their properties, and the various components of their logic specification. As soon as an adequate number of problems and TPTP-compatible ATP systems are available for a specific non-classical logic, a division for that logic will be added to CASC [57]. This will foster robust ATP system development for non-classical logics. In conjunction with the technical manual, a suite of *logic files* that provide semi-formal machine-readable information about the non-classical logics defined in the TPTP is being developed. The logic files will contain information such as logics' syntax, semantic, and proof-theoretic properties, etc. This is “work in progress”, which can be seen in the **Logics** directory of the project repository.

The TPTP syntax aims to provide a very general framework for automated reasoning in expressive formalisms, not yet addressed by this work. For example, in knowledge representation it is often necessary to flexibly combine multiple logics to capture the different information dimensions [58, 59]. Typical examples include, e.g., combinations of temporal logic with (multi-agent) epistemic logics, and deontic logic with action languages. Standard notions for systematically deriving combined logics from constituent logics in

the context of normal modal logics are, among others, fusions [60] and fibrings [61]. In the context of the TPTP syntax, it is intriguing to consider supporting fusions or fibrings by simply providing multiple logic specifications, yielding a very expressive and flexible representation for domain-specific logics.

Acknowledgments

The first and second authors acknowledge financial support from the Luxembourg National Research Fund (FNR), under grant CORE C20/IS/14616644. The third author acknowledges financial support from the German Federal Ministry for Economic Affairs and Energy within the project “KI Wissen – Entwicklung von Methoden für die Einbindung von Wissen in maschinelles Lernen”, project number 19A20020J.

References

- [1] 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.
- [2] G. Sutcliffe, The Logic Languages of the TPTP World, *Logic Journal of the IGPL* (2022) To appear.
- [3] G. Sutcliffe, C. Suttner, The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning* 21 (1998) 177–203.
- [4] G. Sutcliffe, The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0, *Journal of Automated Reasoning* 43 (2009) 337–362.
- [5] G. Sutcliffe, S. Schulz, K. Claessen, P. Baumgartner, The TPTP Typed First-order Form with Arithmetic, in: N. Bjørner, A. Voronkov (Eds.), *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 7180 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2012, pp. 406–419.
- [6] J. Blanchette, A. Paskevich, TFF1: The TPTP Typed First-order Form with Rank-1 Polymorphism, in: M. Bonacina (Ed.), *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2013, pp. 414–420.
- [7] G. Sutcliffe, C. Benz Müller, Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure, *Journal of Formalized Reasoning* 3 (2010) 1–27.
- [8] 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.
- [9] G. Priest, *An Introduction to Non-Classical Logic: From If to Is*, Cambridge University Press, 2008.
- [10] L. Goble, *The Blackwell Guide to Philosophical Logic*, Wiley-Blackwell, 2001.
- [11] P. Blackburn, J. van Benthem, F. Wolther, *Handbook of Modal Logic*, number 3 in *Studies in Logic and Practical Reasoning*, Elsevier Science, 2006.

- [12] H. Ohlbach, Translation Methods for Non-Classical Logics: An Overview, *Logic Journal of the IGPL* 1 (1993) 69–89.
- [13] M. Wisniewski, A. Steen, C. Benz Müller, TPTP and Beyond: Representation of Quantified Non-Classical Logics, in: C. Benz Müller, J. Otten (Eds.), *Proceedings of the 2nd International Workshop on Automated Reasoning in Quantified Non-Classical Logics*, number 1770 in *CEUR Workshop Proceedings*, 2016, pp. 51–65.
- [14] T. Rath, J. Otten, C. Kreitz, The ILTP Problem Library for Intuitionistic Logic - Release v1.1, *Journal of Automated Reasoning* 38 (2007) 261–271.
- [15] T. Rath, J. Otten, The QMLTP Problem Library for First-Order Modal Logics, in: B. Gramlich, D. Miller, U. Sattler (Eds.), *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2012, pp. 454–461.
- [16] C. Benz Müller, J. Otten, T. Rath, Implementing and Evaluating Provers for First-order Modal Logics, in: L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P. Lucas (Eds.), *Proceedings of the 20th European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications*, IOS Press, 2012, pp. 163–168.
- [17] C. Benz Müller, S. Reiche, Automating Public Announcement Logic with Relativized Common Knowledge as a Fragment of HOL in LogiKEy, *Journal of Logic and Computation* *exac029* (2022) 1–27.
- [18] R. Hähnle, M. Kerber, C. Weidenbach, Common Syntax of the DFG-Schwerpunktprogramm Deduction, Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1996.
- [19] U. Hustadt, R. Schmidt, On Evaluating Decision Procedures for Modal Logics, in: P. M.E. (Ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1997, pp. 202–207.
- [20] U. Hustadt, R. Schmidt, Using Resolution for Testing Modal Satisfiability and Building Models, *Journal of Automated Reasoning* 28 (2002) 205–232.
- [21] M. Genesereth, R. Fikes, Knowledge Interchange Format, Version 3.0 Reference Manual, Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.
- [22] ISO/IEC, Information technology - Common Logic (CL) - A Framework for a Family of Logic-based Languages, 2018. ISO/IEC 24707:2018.
- [23] M. Kohlhase, OMDoc - An Open Markup Format for Mathematical Documents [version 1.2], number 4180 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2006.
- [24] M. Kohlhase, F. Rabe, QED Reloaded: Towards a Pluralistic Formal Library of Mathematical Knowledge, *Journal of Formalized Reasoning* 9 (2016) 201–234.
- [25] J. Otten, G. Sutcliffe, Using the TPTP Language for Representing Derivations in Tableau and Connection Calculi, in: B. Konev, R. Schmidt, S. Schulz (Eds.), *Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 5th International Joint Conference on Automated Reasoning*, 2010, pp. 90–100.
- [26] A. Van Gelder, G. Sutcliffe, Extending the TPTP Language to Higher-Order Logic with Automated Parser Generation, in: U. Furbach, N. Shankar (Eds.), *Proceedings*

- of the 3rd International Joint Conference on Automated Reasoning, number 4130 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2006, pp. 156–161.
- [27] 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.
 - [28] E. Kotelnikov, L. Kovacs, A. Voronkov, A First Class Boolean Sort in First-Order Theorem Proving and TPTP, in: M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, V. Sorge (Eds.), Proceedings of the International Conference on Intelligent Computer Mathematics, number 9150 in Lecture Notes in Computer Science, Springer-Verlag, 2015, pp. 71–86.
 - [29] E. Kotelnikov, L. Kovacs, A. Voronkov, A FOOLish Encoding of the Next State Relations of Imperative Programs, in: D. Galmiche, S. Schulz, R. Sebastiani (Eds.), Proceedings of the 9th International Joint Conference on Automated Reasoning, number 10900 in Lecture Notes in Computer Science, 2018, pp. 405–421.
 - [30] 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.
 - [31] H. van Ditmarsch, J. Halpern, W. van der Hoek, B. Kooi, Handbook of Epistemic Logic, College Publications, 2015.
 - [32] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.
 - [33] M. Fitting, L. Thalman, A. Voronkov, Term-Modal Logics, *Studia Logica* 69 (2001) 133–169.
 - [34] T. Sawasaki, K. Sano, T. Yamada, Term-Sequence-Modal Logics, in: P. Blackburn, E. Lorini, M. Guo (Eds.), Proceedings of the 7th International Workshop on Logic, Rationality and Interaction, number 11813 in Lecture Notes in Computer Science, Springer-Verlag, 2019, pp. 244–258.
 - [35] 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.
 - [36] 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.
 - [37] M. Fitting, R. Mendelsohn, First-Order Modal Logic, Kluwer, 1998.
 - [38] J. Garson, Modal Logic, in: E. Zalta (Ed.), Stanford Encyclopedia of Philosophy, Stanford University, 2018.
 - [39] P. Blackburn, M. de Rijke, Y. Venema, Modal Logic, Cambridge University Press, 2001.
 - [40] N. Belnap, A Useful Four-valued Logic: How a Computer Should Think, in: A. Anderson, N. Belnap, J. Dunn (Eds.), Entailment: The Logic of Relevance and Necessity, Volume II, Princeton UP, 1992, pp. 506–541.

- [41] L. Åqvist, Deontic Logic, in: D. Gabbay, F. Guentner (Eds.), *Handbook of Philosophical Logic*, volume 2, D. Reidel, 1984, p. 605–714.
- [42] C. Benzmüller, A. Farjami, X. Parent, Åqvist’s Dyadic Deontic Logic E in HOL, *Journal of Applied Logics* 6 (2019) 733–755.
- [43] G. Sutcliffe, TPTP, TSTP, CASC, etc., in: V. Diekert, M. Volkov, A. Voronkov (Eds.), *Proceedings of the 2nd International Symposium on Computer Science in Russia*, number 4649 in *Lecture Notes in Computer Science*, Springer-Verlag, 2007, pp. 6–22.
- [44] C. Nalon, U. Hustadt, C. Dixon, KSP: Architecture, Refinements, Strategies and Experiments, *Journal of Automated Reasoning* 64 (2020) 461–484.
- [45] 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.
- [46] 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.
- [47] 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.
- [48] D. Tishkovsky, R. Schmidt, M. Khodadadi, The Tableau Prover Generator MetTeL2, in: L. Fariñas del Cerro, A. Herzig, J. Mengin (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.
- [49] L. Fariñas del Cerro, D. Fauthoux, O. Gasquet, A. Herzig, D. Longin, F. Massacci, LoTREC: The Generic Tableau Prover for Modal and Description Logics, in: R. Gore, A. Leitsch, T. Nipkow (Eds.), *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2001, pp. 453–458.
- [50] U. Hustadt, R. Schmidt, MSPASS: Modal Reasoning by Translation and First-Order Resolution, in: R. Dyckhoff (Ed.), *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 1847 in *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 2000, pp. 67–71.
- [51] A. Steen, C. Benzmüller, Extensional Higher-Order Paramodulation in Leo-III, *Journal of Automated Reasoning* 65 (2021) 775–807.
- [52] A. Steen, Scala TPTP Parser v1.5, 2021. DOI: 10.5281/zenodo.5578872.
- [53] C. Benzmüller, L. Paulson, Quantified Multimodal Logics in Simple Type Theory, *Logica Universalis* 7 (2013) 7–20.
- [54] C. Benzmüller, T. Rath, HOL Based First-order Modal Logic Provers, in: K. McMillan, A. Middeldorp, A. Voronkov (Eds.), *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number

- 8312 in *Lecture Notes in Computer Science*, Springer-Verlag, 2013, pp. 127–136.
- [55] A. Steen, *logic-embedding v1.6*, 2022. DOI: 10.5281/zenodo.5913216.
 - [56] A. Steen, *tptp-utils v1.1*, 2021. DOI: 10.5281/zenodo.5877564.
 - [57] G. Sutcliffe, The CADE ATP System Competition - CASC, *AI Magazine* 37 (2016) 99–101.
 - [58] W. Carnielli, M. Coniglio, D. Gabbay, P. Gouveia, C. Sernadas, *Analysis and Synthesis of Logics - How to Cut and Paste Reasoning Systems*, number 35 in *Applied Logic Series*, Springer Verlag, 2008.
 - [59] W. Carnielli, M. Coniglio, *Combining Logics*, in: E. Zalta (Ed.), *Stanford Encyclopedia of Philosophy*, Stanford University, 2020.
 - [60] R. Thomason, *Combinations of Tense and Modality*, in: D. Gabbay, F. Guenther (Eds.), *Handbook of Philosophical Logic*, volume 2, D. Reidel, 1984, pp. 135–165.
 - [61] D. Gabbay, *Fibring Logics*, number 38 in *Oxford Logic Guides*, Clarendon Press, 1998.