TPTP World Infrastructure for Non-classical Logics

Alexander Steen[1], Geoff Sutcliffe[2]

[1] University of Greifswald, Germany
[2] University of Miami, USA

Abstract

The TPTP World is a well established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems. Until now the TPTP World has focused on classical logic, while many real-world applications of ATP also require non-classical reasoning. This paper describes the latest extensions to the TPTP World, providing languages, problems, solutions, and infrastructure for non-classical logics. These are the keys steps towards releasing TPTP v9.0.0, with normal modal logic problems.

Keywords

TPTP World, Non-classical logic

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 [2], the TSTP solution library [3], standards for writing ATP problems and reporting ATP solutions [4, 5], tools and services for processing ATP problems and solutions [3], and it supports the CADE ATP System Competition (CASC) [6]. Various parts of the TPTP World have been deployed in a range of applications, in both academia and industry. The web page `www.tptp.org` provides access to all components.

Until now the TPTP World has focused on classical logic, while many real-world applications of ATP also require non-classical reasoning [7]. These applications include artificial intelligence (e.g., knowledge representation [8], planning [9], multi-agent systems [10]), philosophy (e.g., formal ethics [11], metaphysics [12]), natural language semantics (e.g., generalized quantifiers [13], modalities [14]), and computer science (e.g., software and hardware verification [15]). This paper describes the latest extension of the TPTP World, providing languages and infrastructure for reasoning in non-classical logics [16, 17]. The non-classical logics supported so far are normal modal [18], alethic modal [19], deontic [20], epistemic [21], doxastic [22], and instant-based temporal [23]. In this paper the languages and infrastructure are exemplified in normal modal logic [24], as will be used in TPTP v9.0.0. This paper extends [25] – there has been significant progress and development in the last two years:

- The classical connectives $\Box$ and $\Diamond$ now have both long and short forms in the TPTP syntax (see Section 3).
- The property names in logic specifications have been improved, and another property has been added (see Section 3.1).
- The collection of non-classical problems for the TPTP has made substantial progress (see Section 4).
- The ATP systems have been used to generate solutions for the problems, saved in the TSTP (see Section 5).
- The TPTP format for Kripke interpretations has stabilized (see Section 5).

PAAR'24: 9th Workshop on Practical Aspects of Automated Reasoning, July 2, 2024, Nancy, France
alexander.steen@uni-greifswald.de (A. Steen); geoff@cs.miami.edu (G. Sutcliffe)
0000-0001-8781-9462 (A. Steen); 0000-0001-7116-9338 (G. Sutcliffe)

- Tools for processing non-classical logic problems and solutions have been significantly advanced (see Section 6).
- The ATP systems and tools for non-classical logic have been added to SystemOnTPTP (see Section 6.4).

Paper structure: Section 2 provides a review of the classical TPTP languages, as a foundation for Section 3 that described the TPTP languages. Section 4 describes the collection of non-classical problems for the TPTP problem library, and Section 5 the collection of solutions to such problems in the TSTP solution library. Section 6 gives an overview of ATP systems and tools for non-classical logics. Section 7 concludes, including plans for further development of the non-classical TPTP World.

2. The Classical TPTP Languages

The TPTP languages for first-order clause normal form (CNF) [26], full first-order form (FOF) [2], typed-first order form (TFF) [27, 28], and typed higher-order form (THF) [29, 30] are by now well known and documented. An overview that is relevant to this paper is provided in [25], and the detailed syntax of the languages is given in an extended BNF[1] [31]. The TPTP has a hierarchy of languages that underlie the non-classical languages. The languages are:

- Clause normal form (CNF), which is the "assembly language" of many modern ATP systems.
- First-order form (FOF), which hardly needs introduction.
- Typed first-order form (TFF), which adds types and type signatures, with monomorphic (TF0) and polymorphic (TF1) variants.
- Typed extended first-order form (TXF), which adds Boolean terms, Boolean variables as formulae, tuples, conditional expressions, and let expressions. TXF has monomorphic (TX0) and polymorphic (TX1) variants.
- Typed higher-order form (THF), which adds higher-order notions including curried type declarations, lambda terms, partial application, and connectives as terms. THF has monomorphic (TH0) and polymorphic (TH1) variants.

A brief summary of the syntax is provided here.

Problems and solutions are built from *annotated formulae* of the form ...

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

The *language*s supported are `cnf` (clause normal form), `fof` (first-order form), `tff` (typed first-order form), and `thf` (typed higher-order form). The *role*, e.g., `axiom`, `lemma`, `conjecture`, defines the use of the formula. In a *formula*, terms and atoms follow Prolog conventions – functions and predicates start with a lowercase letter or are `'`single quoted`'`, and variables start with an uppercase letter. The language also supports interpreted symbols that either start with a `$`, e.g., the truth constants `$true` and `$false`, or are composed of non-alphabetic characters, e.g., integer/rational/real numbers such as 27, 43/92, -99.66. The logical connectives in the TPTP language are `!`, `?`, `~`, `|`, `&`, `=>`, `<=`, `<=>`, and `<~>`, for the mathematical connectives $\forall$, $\exists$, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$, and $\oplus$ respectively. Equality and inequality are expressed as the infix operators `=` and `!=`. The *source* and *useful_info* are optional.

The typed first-order form (TFF) language adds types and type declarations. Predicate and function symbols can be declared before their use, with type signatures that specify the types of their arguments and result. An expression `(`t_1 `*` ... `*` 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 `*` ... `*` 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). The typed

[1] `www.tptp.org/TPTP/SyntaxBNF.html`

higher-order form (THF) extends TFF with higher-order notions, including the curried form of type declarations, lambda terms with a lambda binder `^` for λ, application with `@`, a choice binder `@+` for ϵ, and a description binder `@-` for ι.

As a simple example, here is an example of a monomorphic typed extended first-order (TX0) annotated formula. The type declarations for `inhabitant`, `is_knave`, and `says` are as expected, and can be seen in the TPTP problem `PUZ081_8.p`[2]. It expresses one of the axioms of the "Knights and Knaves" puzzles [32], that for every inhabitant `I` and utterance `S`, if `I` is a knave (knaves always lie) and **I** says **S** then `S` is not true.

```
tff(leaf_knaves_lie,axiom,
    ! [I: inhabitant,S: $o] :
      ( ( is_knave(I) & says(I,S) ) => ~ S ),
    file('PUZ081_8.p',knaves_lie),
    [description('Knaves always lie'), relevance(0.9)]).
```

3. The Non-classical TPTP Languages

The new non-classical typed form (NTF) family of TPTP languages for non-classical logics has two top level variants based on the classical typed extended first-order (TXF) and typed higher-order (THF) languages. They are the non-classical typed extended first-order form (NXF), and the non-classical typed higher-order form (NHF). As with TXF and THF, NXF and NHF have monomorphic (NX0, NH0) and polymorphic (NX1, NH1) subvariants. All constructs of the underlying TXF and THF languages are available in the NXF and NHF languages.

The non-classical connectives of NTF have the form `{$`*name*`}`. Examples are `{$box}` ($\Box$ in logic texts), `{$dia}` ($\Diamond$ in logic texts), `{$possible}`, `{$necessary}`, `{$obligatory}`, `{$permissible}`, `{$knows}`, `{$believes}`, etc. A connective can be parameterized to reflect more complex non-classical connectives, e.g., in multi-modal logics where the modal operators are indexed [33], in epistemic logics where the knowledge operators can specify the agents under consideration [21], and in dynamic logics [34] where the connectives are parameterized with (complex) programs. The form is `{$`*name*`(`*param*$_1$`,...,`*param*$_n$`)}`. If the connective is indexed the index is given as the first parameter prefixed with a `#`, e.g., `{$knows(#manuel)}@(nothing)`[3]. All other parameters are key-value assignments, e.g., to list the agents of common knowledge the form might be `{$common($agents:=[alice,bob,claire])}`.

In NXF the non-classical connectives are applied in a mixed higher-order-applied/first-order-functional style, with the connectives applied using `@` to a `()`ed list of arguments.[4] In NHF the non-classical connectives are applied using `@` in usual higher-order style with curried function applications. There are also short form unary connectives for unparameterised `{$box}` and `{$dia}`: `[.]` and `<.>`, e.g., `{$box}@(p)` can be written `[.]p`. Full specification of the connectives and their use in formulae is in the BNF starting at `<nxf_atom>` and `<thf_defined_atomic>`.

The TPTP language extension for non-classical connectives is arguably conservative: It only adds one uniform construct to both TXF and THF, i.e., `{$`*name*`(`*param*$_1$`,...,`*param*$_n$`)}`, that is uniformly applied to arguments using `@`. Of course, a simpler and more concise syntax could be envisioned when only focusing on syntactically simple logics (like modal logics with its unary box operators). The presented extension however aims at providing a future-proof syntax that can represent logics with arbitrary n-ary operators that themselves may carry non-trivial parameters (like programs in PDL, or groups of agents in epistemic logic). In addition the syntax makes clear the distinction between object-logic and meta-logic syntax: An expression `{$knows(#manuel)}@(nothing)` is clearly presented as the unary connective

[2] `tptp.org/cgi-bin/SeeTPTP?Category=Problems&Domain=PUZ&File=PUZ081_8.p`

[3] As in `www.youtube.com/watch?v=ISD86-oM4Ow`

[4] This slightly unusual form was chosen to reflect the first-order functional style, and by making the application explicit the formulae can be parsed in Prolog – a long standing principle of the TPTP languages [35].

`{$knows(#manuel)}` applied to one argument, rather than a binary expression `$knows` applied to the parameter (index) `#manuel` and the (only) real argument `nothing`. This addresses at least two problems of the latter (syntactically somewhat simpler) approach: Non-classical connectives may have optional parameters (the latter approach would have connectives occuring with varying number of arguments), and expressions like `#manuel` are not part of the object-logic term/formula language, hence cannot be given a meaningful type (the latter approach would require to assign to them a type in order to be well-typed, as they occur at term-level).

As non-classical consequence relations may be more complex than the standard classical one (e.g., in modal logics where consequence may be global or local), the roles of annotated formulae can have subroles (e.g., `local` or `global`). In modal logic problems, the `conjecture` is `local` by default, i.e., is to be proved in only one local world, but can be given the subrole `global` to indicate it is to be proved in all worlds. By default axioms are `global` in modal logics, i.e., hold in all worlds, but if the `conjecture` is `local` then axioms can have the subrole `local` to indicate they hold in only the one world where the conjecture is being proved.

Figure 1 shows an example NX0 problem in alethic modal logic (the formulae with the `logic` role is explained next), and Figure 2 shows the axiom and conjecture formulae in NX0. Note how the conjecture defaults to being proved in one local world. The `rotten_banana_here` holds in only that local world while the other axioms hold in all worlds.

```
%----------------------------------------------------------------------------
tff(semantics,logic,
    $alethic_modal ==
      [ $domains == $constant,
        $designation == $rigid,
        $terms == $local,
        $modalities == $modal_system_M ] ).

tff(fruit_type,type,   fruit: $tType).
tff(apple_decl,type,   apple: fruit).
tff(banana_decl,type,  banana: fruit).
tff(healthy_decl,type, healthy: fruit > $o).
tff(rotten_decl,type,  rotten: fruit > $o).

%----Apples are different from bananas
tff(apple_not_banana,axiom,
    apple != banana ).

%----All fruit are necessarily healthy
tff(necessary_healthy_fruit_everywhere,axiom,
    ! [F: fruit] : ( {$necessary} @ (healthy(F)) ) ).

%----All fruit are possibly not rotten
tff(fruit_possibly_not_rotten,axiom,
    ! [F: fruit] : ( {$possible} @ (~ rotten(F)) ) ).

%----Bananas are rotten in the conjecture's (local) world
tff(rotten_banana_here,axiom-local,
    rotten(banana) ).

%----Prove it's possible for apples to be healthy and bananas not rotten
tff(possible_fruit,conjecture,
    ( {$possible} @ ( healthy(apple) & ~ rotten(banana) ) ) ).
%----------------------------------------------------------------------------
```

Figure 1: NX0 example

```
%----------------------------------------------------------------------
thf(apple_not_banana,axiom,
    apple != banana ).

thf(necessary_healthy_fruit_everywhere,axiom,
    ! [F: fruit] : ( {$necessary} @ ( healthy @ F ) ) ).

thf(fruit_possibly_not_rotten,axiom,
    ! [F: fruit] : ( {$possible} @ ( ~ ( rotten @ F ) ) ) ).

thf(rotten_banana_here,axiom-local,
    rotten @ banana ).

thf(not_true,conjecture,
    ( {$possible} @ ( ( healthy @ apple ) & ~ ( rotten @ banana ) ) ) ).
%----------------------------------------------------------------------
```

Figure 2: NH0 formulae

3.1. Logic Specification

In non-classical logics the same language can be used for formulae while different logics are used for reasoning. It is therefore necessary to provide (meta-) information that specifies the logic to be used. A TPTP annotated formula with the role `logic` is used for this, with a *logic specification* as the formula. A logic specification consists of a defined logic (family) name identified with a list of properties[5]. It is assumed that there is a single logic specification specifying all relevant properties. However, if it should prove to be convenient in practice, future developments may allow the logic specification to be split over multiple annotated formulae with the role `logic`. The `$domains` property specifies whether each quantification domain is constant, varying, cumulative, or decreasing, across the accessibility relation. The `$designation` property specifies whether symbols are interpreted rigidly, i.e., interpreted as the same domain element in every world, or flexibly, i.e., possibly interpreted as different domain elements in different worlds. The `$terms` property specifies whether interpretation is local to the current world or global to all worlds. The `$modalities` property specifies properties of the connectives, either as a well-known logic system, e.g., the modal system **S5**, or as axiom schemes, e.g., the modal axiom **5** as in [36]. Further details of the logic specifications are in [25]. Figure 1 includes a logic specification in NXF, which specifies:

- `$alethic_modal` - Alethic modal logic [37] is being used.
- `$domains == $constant` - The domains are the same in all worlds.
- `$designation == $rigid` - The interpretation of symbols is the same in all worlds.
- `$terms == $local` - All terms are interpreted in the local world.
- `$modalities == $modal_system_M` - This modality is built from:
  - The distribution axiom **K**: $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$
  - The **M** axiom: $\Box A \rightarrow A$, i.e., the accessibility relation is reflexive.

4. TPTP Problems

The TPTP problem library of test problems for ATP systems has been extended in v9.0.0 to include problems in the NX0 and NH0 languages. To start with, monomodal normal modal logic problems are being collected, including problems from (the citations are just some examples) books [38, 18, 39, 40], conference and journal papers [41, 42, 43, 44], and use cases [45, 46].

[5]The property names have been improved since their presentation in [25]

Each problem file in the TPTP problem library has three parts: a header, optional includes, and annotated formulae. The header section contains information for users, formatted as comments identifying and describing the problem, providing information about occurrences of the problem in the literature and elsewhere, providing semantic and syntactic characteristics of the problem, and finally comments and bugfix information. The include section contains include directives for axiom files to avoid duplication of formulae in commonly used axiomatizations. Annotated formulae are described in Section 4.

The headers of the NX0 and NH0 problems include relevant augmented information. Figure 3 shows the `Syntax` and `SPC` (Specialist Problem Class[6]) fields for the problem in Figure 1. The relevant new information is:

- The number of non-classical connectives, indexed and unindexed. For example, as shown in Figure 3, the problem in Figure 1 has three non-indexed connectives and no indexed connectives, written "`3 {.}; 0 {#}`" in the header field.
- The `SPC` field has values for non-classical logic problems. For example, as shown in Figure 3, the SPC of the problem in Figure 1 is `NX0_THM_NEQ_NAR` – it's in the NX0 language, is a theorem, and there is no use of equality or arithmetic.

```
%------------------------------------------------------------------------------
% Syntax   : Number of formulae    :   10 (   2 unt;   5 typ;   0 def)
%            Number of atoms       :   12 (   1 equ)
%            Maximal formula atoms :    2 (   2 avg)
%            Number of connectives :   10 (   3   ~;   0   |;   1   &)
%                                         (   0 <=>;   0  =>;   0  <=;   0 <~>)
%                                         (   3 {.};   0 {#})
%            Maximal formula depth :    3 (   2 avg)
%            Maximal term depth    :    1 (   1 avg)
%            Number of FOOLs       :    3 (   3 fml;   0 var)
%            Number of types       :    2 (   1 usr)
%            Number of type conns  :    2 (   2   >;   0   *;   0   +;   0  <<)
%            Number of predicates  :    5 (   2 usr;   0 prp; 1-2 aty)
%            Number of functors    :    2 (   2 usr;   2 con; 0-0 aty)
%            Number of variables   :    2 (   2   !;   0   ?;   2   :)
% SPC      : NX0_THM_NEQ_NAR
%------------------------------------------------------------------------------
```

Figure 3: NX0 header fields

The files in the `Documents` directory have been augmented:

- The `TFFSynopsis`, `THFSynopsis`, and `OverallSynopsis` files give the numbers of non-classical logic problems.
- The `ProblemAndSolutionStatistics` file gives the numbers of non-classical connectives in each problem.

TPTP v9.0.0 contains 147 NTF problems. There are 12 `GRA` (graph theory) problems, 12 `LCL` (logic calculi) problems, 7 `PHI` (philosophy) problems, 10 `PLA` (planning) problems, and 106 `SYO` (syntactic) problems. Fifteen problems are in the NH0 language, 132 in the NX0 language. Fifty problems are propositional, 91 are first-order without equality, and 6 are first-order with equality. Eight-four of the problems are known to be theorems, 43 countersatisfiable, and 20 have unknown status.

[6]The problems in the TPTP library are divided into Specialist Problem Classes (SPCs) – classes of problems that are homogeneous wrt recognizable logical, language, and syntactic characteristics.

5. TSTP Solutions

The TSTP solution library of solutions from ATP systems will be updated to include the results of running the ATP systems KSP, nanoCoP-M, MleanCoP, and Leo-III (see Section 6.2) on the NX0 and NH0 problems in the TPTP problem library. The TPTP format for derivations [4] can immediately be used for writing derivations in non-classical logic. The new TPTP format for interpretations [47] can be used to write Kripke models [48]. However, at the time of writing, none of the solutions in the TSTP solution library are in the TPTP format using NX0 or NH0, because none of the ATP systems output their solutions in NX0 or NH0. The task of writing postprocessors that translate ATP systems' native proofs and models to the TPTP format might be future work, but providing those reduces the incentive for system developers to output TPTP format natively. That said ...

5.1. TSTP Derivations

Leo-III outputs its proofs of NXF/NHF problems in TPTP format, but the proofs are in the language of the embedded problems, i.e., TF0 or TH0 (see Section 6.1). It is also often the case that the proof includes one large step resulting from a call to an underlying ATP system, e.g., E [49]. The NX0/NH0 input can be grafted onto Leo-III's proofs, and the details of the proof from underlying ATP added in. This has been done by hand for the proof of the problem in Figure 1, and the result is shown in Figure 4 (the type declarations are omitted for brevity). In this case the underlying ATP system was E. Details on the proof construction process of Leo-III, including a description of its proof search procedure, can be found in the literature [50, 51]

5.2. TSTP Interpretations

The new TPTP format for interpretations, as used for writing Kripke interpretations, deserves some introduction. Interpretations are captured in *interpretation-formulae*, which are written in TXF syntax. For interpretations of classical logic formulae the semantics is the standard classical semantics of TXF. In contrast, for Kripke interpretations the semantics is that of modal logic enriched with four new defined symbols:

- A defined type `$world` is used for the worlds of the interpretation. Different constants of type `$world` are known to be unequal (but as yet no ATP systems implement that, so it's necessary to encode that explicitly using inequalities or the `$distinct` predicate).
- A defined predicate `$accessible_world` of type `($world * $world) > $o` is used to specify the accessibility relation between worlds.
- A defined predicate `$in_world` of type `($world * $o) > $o` is used to specify the interpretations in the worlds.
- A defined constant `$local_world` of type `$world > $o` is used to specify the world in which the conjecture was found to not hold (in case the interpretation represents a countermodel).

A Kripke interpretation-formula is a conjunction of a specification of the worlds, explication of the distinctness of the worlds (until that is built into ATP systems and tools), the accessibility relation, specification of the local world if any, and, for each world, its Tarskian interpretation (also in the new TPTP format for interpretations). The `logic` specification of the problem is included to specify that the interpretation is for formulae of that logic. This information is needed when processing an interpretation, e.g., in verification (see Section 6.3). The interpretation-formula does not provide this information because it underspecifies the logic in use, e.g., it's usually not possible to see whether the interpretation exemplifies modal system K or modal system S5 – in both cases the interpretation could interpret the accessibility relation as an equivalence relation (this is required for S5 but it is also OK for K). The interpretation-formula is preceded by the necessary type declarations.

```
%----------------------------------------------------------------------------------------
tff(reflexive_M,axiom,
    ! [X1: '$world'] : '$accessible_world'(X1,X1) ).

tff(necessary_healthy_fruit_everywhere,axiom,
    ! [F: fruit] : ( {$necessary} @ ( healthy(F) ) ) ).

tff(fruit_possibly_not_rotten,axiom,
    ! [F: fruit] : ( {$possible} @ ( ~ rotten(F) ) ) ).

tff(possible_fruit,conjecture,
    ( {$possible} @ ( healthy(apple) & ~ rotten(banana) ) ) ).

tff(possible_fruit_expanded,conjecture,
    ? [X1: '$world'] :
      ( '$accessible_world'('$local_world',X1) & healthy(X1,apple) & ~ rotten(X1,banana) ),
    inference(expand,[status(thm)],[possible_fruit]) ).

tff(necessary_healthy_fruit_everywhere_expanded,plain,
    ! [X1: '$world',X2: fruit,X3: '$world'] :
      ( '$accessible_world'(X1,X3) => healthy(X3,X2) ),
    inference(expand,[status(thm)],[necessary_healthy_fruit_everywhere]) ).

tff(fruit_possibly_not_rotten_expanded,plain,
    ! [X1: '$world',X2: fruit] :
    ? [X3: '$world'] :
      ( '$accessible_world'(X1,X3) & ~ rotten(X3,X2) ),
    inference(expand,[status(thm)],[fruit_possibly_not_rotten]) ).

tff(c_0_4,negated_conjecture,
    ~ ? [X1: '$world'] :
        ( '$accessible_world'('$local_world',X1) & healthy(X1,apple) & ~ rotten(X1,banana) ),
    inference(assume_negation,[status(cth)],[possible_fruit_expanded]) ).

tff(c_0_7,negated_conjecture,
    ! [X4: '$world'] :
      ( ~ '$accessible_world'('$local_world',X4) | ~ healthy(X4,apple) | rotten(X4,banana) ),
    inference(fof_nnf,[status(thm)],[c_0_4]) ).

tff(c_0_5,plain,
    ! [X8: '$world',X9: fruit,X10: '$world'] :
      ( ~ '$accessible_world'(X8,X10) | healthy(X10,X9) ),
    inference(fof_nnf,[status(thm)],[necessary_healthy_fruit_everywhere_expanded]) ).

tff(c_0_12,plain,
    ! [X1: '$world',X2: fruit] : healthy(X1,X2),
    inference(spm,[status(thm)],[c_0_5,reflexive_M]) ).

tff(c_0_13,plain,
    ! [X5: '$world',X6: fruit] :
      ( '$accessible_world'(X5,esk1_2(X5,X6)) & ~ rotten(esk1_2(X5,X6),X6) ),
    inference(skolemize,[status(esa)],[fruit_possibly_not_rotten_expanded]) ).

tcf(c_0_14,negated_conjecture,
    ! [X1: '$world'] :
      ( rotten(X1,banana) | ~ '$accessible_world'('$local_world',X1) ),
    inference(cn,[status(thm)],[inference(rw,[status(thm)],[c_0_7,c_0_12])]) ).

tcf(c_0_15,plain,
    ! [X1: '$world',X2: fruit] : '$accessible_world'(X1,esk1_2(X1,X2)),
    inference(split_conjunct,[status(thm)],[c_0_13]) ).

tcf(c_0_16,plain,
    ! [X1: '$world',X2: fruit] : ~ rotten(esk1_2(X1,X2),X2),
    inference(split_conjunct,[status(thm)],[c_0_13]) ).

tcf(c_0_17,negated_conjecture,
    ! [X2: fruit] : rotten(esk1_2('$local_world',X2),banana),
    inference(spm,[status(thm)],[c_0_14,c_0_15]) ).

cnf(c_0_18,negated_conjecture,
    $false,
    inference(spm,[status(thm)],[c_0_16,c_0_17]),
    [proof] ).
%----------------------------------------------------------------------------------------
```

Figure 4: NX0 proof example

The problem in Figure 1 can be made into a non-theorem by changing the `{$possible}` in the conjecture to `{$necessary}`. Figure 5 shows the worlds and the first world's Tarskian interpretation for a Kripke (counter)model (the type declarations are omitted for brevity) of the modified problem. The second world is the same except that bananas are not rotten. Note that the quantification semantics of subformulae in the second argument of the `$in_world` predicate such as `? [DP: d_fruit] : ( DP = d_apple )` is not classical but instead that of modal logic with varying domains, i.e., the quantification is over the domain elements that exist in `w1`, i.e., the subformulae requires that the domain element `d_apple` exists in `w1`. Outside of the `$in_world` predicate classical quantification is used. The model was found – after embedding to classical logic – by Nitpick [52], and manually transformed into TPTP format.

```
%----------------------------------------------------------------------
tff(semantics,logic,
    $alethic_modal ==
      [ $domains == $constant,
        $designation == $rigid,
        $terms == $local,
        $modalities == $modal_system_M ] ).

tff(d_fruit_type,type,d_fruit: $tType).
tff(d2fruit_decl,type, d2fruit: d_fruit > fruit ).
tff(d_apple_decl,type,d_apple: d_fruit).
tff(d_banana_decl,type,d_banana: d_fruit).

tff(w1_decl,type,w1: $world).
tff(w2_decl,type,w2: $world).

tff(fruity_worlds,interpretation,
%----There are two worlds, w1 and w2
    ( ( ! [W: $world] : ( W = w1 | W = w2 )
%----The conjecture was disproved in world w1
      & $local_world = w1
%----World accessibility is reflexive, and ws is accessible from w1
      & $accessible_world(w1,w1)     %----Logic is M
      & $accessible_world(w2,w2)
      & $accessible_world(w1,w2) )
%----Tarskian interpretation in world w1
    & $in_world(w1,
%----There are two fruit in the domain, apple and banana
        ( ( ! [F: fruit] : ? [DF: d_fruit] : F = d2fruit(DF)
          & ! [DF: d_fruit] : ( DF = d_apple | DF = d_banana )
          & $distinct(d_apple,d_banana)
%----The local domain elements
          & ? [DP: d_fruit] : ( DP = d_apple )
          & ? [DP: d_fruit] : ( DP = d_banana )
%----The type-promotion is reflexive
          & ! [DF1: d_fruit,DF2: d_fruit] :
              ( d2fruit(DF1) = d2fruit(DF2) => DF1 = DF2 ) )
%----The constant apple is interpreted as the domain element
        & ( apple = d2fruit(d_apple)
%----The constant banana is interpreted as the domain element
          & banana = d2fruit(d_banana) )
%----Apples and bananas are healthy
        & ( healthy(d2fruit(d_apple))
          & healthy(d2fruit(d_banana))
%----Apples are not rotten, bananas are rotten in the local world w1
          & ~ rotten(d2fruit(d_apple))
          & rotten(d2fruit(d_banana)) ) ) )
%----------------------------------------------------------------------
```

Figure 5: NX0 Kripke (counter)model example

6. Software Support

TPTP World software support for non-classical logics has been developed, and continues to be developed, for access to and manipulation of problems, ATP systems, solutions, and processing tools. All the software is freely available from GitHub[7], and mostly available for use in SystemOnTPTP (see Section 6.4).

6.1. Parsers and Printers

The TPTP4X utility and the BNF-based suite of parsers [31] can parse NXF and NHF formulae. TPTP4X parses problems and solutions, can apply various transformations, and pretty-prints the formulae. The BNF-based parsers offer stricter parsing than TPTP4X, can present the parse trees in various forms, but cannot transform or pretty-print the formulae.

In addition to the TPTP World's own tools, a suite of tools that can parse and manipulate NXF and NHF formulae is available in the Leo-III framework [51]. The `tptp-utils` tool [53] can read formulae in all the TPTP languages, including NXF and NHF. It does syntax checking, translations, generation of parse trees, (basic) linting, and pretty-printing. For NXF and NHF in particular, it can sanity check logic specifications for modal logics. It comes with a complete definition of abstract syntax trees for the internal representation. Its underlying parser written in Scala, which is also used by Leo-III, is available as the stand-alone parsing library `scala-tptp-parser` [54].

In order to support existing ATP systems that do not (yet) read TPTP NX0 or NH0 formulae, some syntax translators have been implemented to convert NX0 formulae into the systems' native syntaxes. Thus far front-end translators have been implemented for KSP, nanoCoP-M, and MleanCoP. Thankfully this has been quite easy, and implemented in `sed`. In a related effort, the NTFLET Logic Embedding Tool [55] does a shallow embedding of NXF problems into TXF or THF, and NHF problems into THF [56, 57, 58, 59]. By default NTFLET produces TX0 or TH0, but it can optionally produce polymorphic TH1 or TX1 (which can be significantly shorter if the input problem contains many user types). Currently NTFLET supports a range of modal logics, a range of first-order quantified hybrid logics [60], public announcement logic [61, 62], and two different dyadic deontic logics [63, 64]. Any TPTP-compliant TXF/THF ATP system can be added as a backend to NTFLET to form an ATP system for NXF/NHF (see Section 6.2).

6.2. ATP Systems

ATP for non-classical logics is a well established endeavour (particularly for propositional non-classical logics), but there are significantly fewer ATP systems available than for classical logics. The ATP systems that we know of are KSP [65, 44], nanoCoP-M [66], MleanCoP [67], MetTeL2 [68], Leo-III [51], LoTREC [69], and MSPASS [70]. As noted in Section 6.1, translators have been implemented from the TPTP syntax to the native syntaxes of KSP, nanoCoP-M, and MleanCoP, so that those systems can attempt the NX0 problems in the TPTP problem library.

There have been successful efforts that translate/embed non-classical logic into a classical logic, and apply a classical logic ATP system [71, 72, 73, 58, 74] (i.e., including that used in Leo-III). However, none of them (other than Leo-III) provide full generality. Leo-III can be used on NXF and NHF problems natively, via the embedding approach described in Section 6.1. As noted there, any TPTP-compliant TXF/THF ATP system can replace Leo-III as a backend after the embedding, thus offering a suite of system variants. A comparative study of the performance of these systems is given in [75].

[7] `github.com/TPTPWorld`

6.3. Verifiers and Viewers

The GDV derivation verifier [76] can verify such proofs in TPTP format. GDV does structural verification, e.g., checking that the derivation is acyclic, origin verification, i.e., checking that the leaves of the derivation are (derivable) from the problem formulae, inference verification using trusted ATP systems, e.g., checking that an inferred formula is a theorem of the parents, and completeness verification, e.g., checking that the root of a refutation is *false*. As there are no ATP systems that output TPTP format proofs for non-classical problems yet, it has been tested with examples created by hand. GDV is available as a standalone tool, and also in SystemOnTSTP (see Section 6.4).

The AGMV model verifier [47] can verify Kripke models in TPTP format. AGMV does syntax and type checking, verifies that the interpretation-formula is satisfiable using a trusted model finder, and verifies that the problem formulae are theorems of the interpretation-formula using a trusted theorem prover. AGMV has been tested with examples created by hand. AGMV is available as a standalone tool, and also in SystemOnTSTP (see Section 6.4).

The IDV Interactive Derivation Viewer [77] is able to display NX0 derivations in TPTP format. Figure 6 shows the derivation in Figure 4. The pointer is hovering over the node `c_0_12`, whose formula shown in the lefthand panel, expresses that in all worlds all fruit are healthy. The red and blue coloring shows the ancestors and descendants of the node in the derivation. IDV is available in SystemOnTSTP (see Section 6.4).

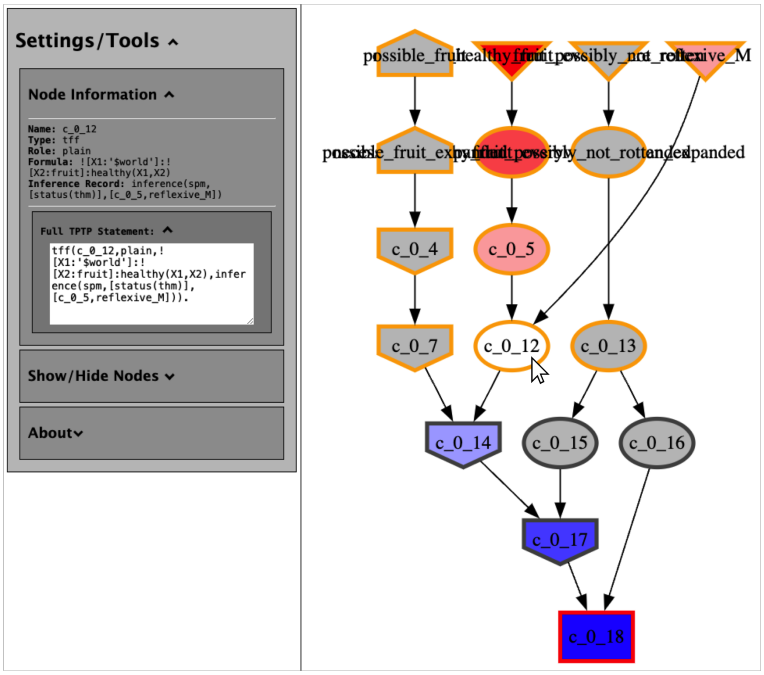

Figure 6: IDV view of the proof in Figure 4

The IIV Interactive Interpretation Viewer [47, 78] is able to display the Tarskian interpretation of a world in a Kripke model in TPTP format. Figure 7 shows the interpretation of world `w1` in Figure 5. The pointer is hovering over the node `d_banana`. The red `$o` ancestor indicates that `rotten` has a boolean result type. The blue `$true` descendant indicates that the interpretation of `rotten` for the domain element `d_banana` is *true*, and the blue `$o` descendant indicates that the domain type is boolean. IIV is available in SystemOnTSTP (see Section 6.4). A wrapper to view the worlds and accessibility relationship of a Kripke model is being developed, which will be linked to IIV to view a chosen world's Tarskian interpretation.

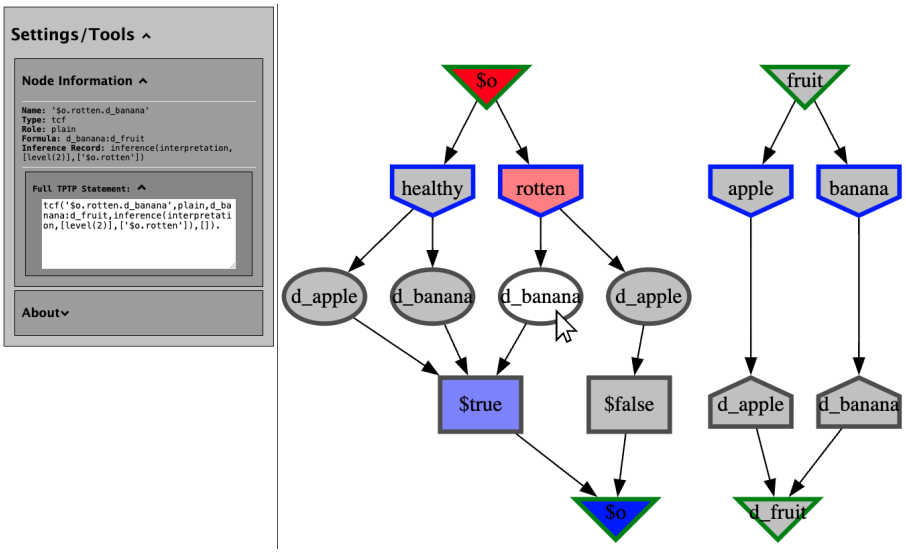

Figure 7: IIV view of the interpretation of world `w1` in Figure 5

6.4. Online

The TPTP World has three online interfaces that provide access to ATP systems and tools [79, 80]: SystemB4TPTP[8] for preparing formulae (often problems) for systems and tools, SystemOnTPTP[9] for submitting formulae to ATP systems, and SystemOnTSTP[10] for processing solutions (often derivations or models) from systems and tools. SystemB4TPTP includes access to TPTP4X, the BNF-based parsers, the KSP-to-TPTP translator, and NTFLET (see Section 6.1). SystemOnTPTP includes the ATP systems KSP, nanoCoP-M, MleanCoP, and Leo-III. SystemOnTSTP includes GDV, AGMV, IDV, IIV, and the `InterpretByATP` tool for evaluating a formula wrt an interpretation (including Kripke interpretations).

The TPTP2T tool for listing problems and solutions with specified syntactic and semantic characteristics is available in a separate interface[11]. It can be used, e.g., to list NX0 problems that are theorems, contain equality, and are easy to prove.

7. Conclusion

This paper describes the latest extensions to the TPTP World, providing languages, problems, solutions, and infrastructure for non-classical logics. The topics covered are the TPTP languages for writing non-classical problems and solutions, non-classical problems for the TPTP problem library, solutions to such problems in the TSTP solution library, and ATP systems and tools for non-classical logics. These are the keys steps towards releasing TPTP v9.0.0, with normal modal logic problems.

Ongoing and future work includes:

- Collecting lots more problems in non-classical logics.
- Working with ATP system developers to upgrade their systems to natively read problems written in NXF and NHF, and to produce proofs and models in TPTP format.
- Optimizing the NTFLET embeddings to automatically recognize language fragments for which TXF embedding is possible.
- Producing a complete interactive viewer for Kripke interpretations written in TPTP-format.

[8] `tptp.org/cgi-bin/SystemB4TPTP`
[9] `tptp.org/cgi-bin/SystemOnTPTP`
[10] `tptp.org/cgi-bin/SystemOnTSTP`
[11] `tptp.org/cgi-bin/TPTP2T`

- Standardizing the embedding of further specific non-classical logics in NXF/NHF (including choosing connective names, and allowed parameters of the logic specification).
- A non-classical division of CASC.

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 TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0, Journal of Automated Reasoning 43 (2009) 337–362.

[3] G. Sutcliffe, The TPTP World - Infrastructure for Automated Reasoning, in: E. Clarke, A. Voronkov (Eds.), Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, number 6355 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2010, pp. 1–12.

[4] G. Sutcliffe, S. Schulz, K. Claessen, A. Van Gelder, Using the TPTP Language for Writing Derivations and Finite Interpretations, 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, 2006, pp. 67–81.

[5] G. Sutcliffe, The SZS Ontologies for Automated Reasoning Software, in: G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, S. Schulz (Eds.), Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, number 418 in CEUR Workshop Proceedings, 2008, pp. 38–49.

[6] G. Sutcliffe, The CADE ATP System Competition - CASC, AI Magazine 37 (2016) 99–101.

[7] A. Steen, C. Benzmüller, What are Non-classical Logics and Why Do We Need Them? An Extended Interview with Dov Gabbay and Leon van der Torre, Künstliche Intelligenz (2024). DOI: 10.1007/s13218-023-00824-7.

[8] L. Giordano, V. Gliozzi, N. Olivetti, G. Pozzato, C. Schwind, Non-classical Logics for Knowledge Representation and Reasoning, Intelligenza Artificiale 5 (2011) 127–131.

[9] A. Liberman, A. Achen, R. Rendsvig, Dynamic Term-modal Logics for First-order Epistemic Planning, Artificial Intelligence 286 (2020). DOI: 10.1016/j.artint.2020.103305.

[10] C. Cachin, D. Lehnherr, T. Studer, Modal and Justification Logics for Multi-agent Systems, in: A. Herzig, J. Luo, P. Pardo (Eds.), Proceedings of the 5th International Conference on Logic and Argumentation, number 14156 in Lecture Notes in Computer Science, Springer-Verlag, 2023, pp. 3–8.

[11] C. Benzmüller, X. Parent, L. van der Torre, Designing Normative Theories for Ethical and Legal Reasoning: LogiKEy Framework, Methodology, and Tool Support, Artificial Intelligence 287 (2020) Article 103348.

[12] C. Benzmüller, B. Woltzenlogel Paleo, The Inconsistency in Gödel's Ontological Argument: A Success Story for AI in Metaphysics, in: S. Kambhampati (Ed.), Proceedings of the 25th International Joint Conference on Artificial Intelligence, AAAI Press, 2016, pp. 936–942.

[13] J. van Benthem, Towards a Computational Semantics, in: P. Gärdenfors (Ed.), Generalized Quantifiers, volume 31 of *Studies in Linguistics and Philosophy*, 1987. DOI: 10.1007/978-94-009-3381-1_2.

[14] A. Kratzer, What 'must' and 'can' Must and Can Mean, Linguistics and Philosophy 1 (1977) 337–355.

[15] R. Bryant, A Methodology for Hardware Verification Based on Logic Simulation, Journal of the Association for Computing Machinery 38 (1991) 299–328.

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

[17] L. Goble, The Blackwell Guide to Philosophical Logic, Wiley-Blackwell, 2001.

[18] M. Fitting, R. Mendelsohn, First-Order Modal Logic, Kluwer, 1998.
[19] G. Schurz, Alethic Modal Logics and Semantics, in: D. Jacquette (Ed.), A Companion to Philosophical Logic, Wiley, 2006, pp. 442–477.
[20] R. Hilpinen, Deontic Logic: Introductory and Systematic Readings, D. Reidel, 1971.
[21] H. van Ditmarsch, J. Halpern, W. van der Hoek, B. Kooi, Handbook of Epistemic Logic, College Publications, 2015.
[22] J. Hintikka, Knowledge and Belief - An Introduction to the Logic of the Two Notions, Texts in Philosophy, Cornell University Press, 1962.
[23] V. Goranko, A. Rumberg, Temporal Logic, in: E. Zalta (Ed.), Stanford Encyclopedia of Philosophy, Stanford University, 2022.
[24] P. Blackburn, J. van Benthem, F. Wolther, Handbook of Modal Logic, number 3 in Studies in Logic and Practical Reasoning, Elsevier Science, 2006.
[25] A. Steen, D. Fuenmayor, T. Gleißner, G. Sutcliffe, C. Benzmüller, Automated Reasoning in Non-classical Logics in the TPTP World, in: B. Konev, C. Schon, A. Steen (Eds.), Proceedings of the 8th Workshop on Practical Aspects of Automated Reasoning, number 3201 in CEUR Workshop Proceedings, 2022, p. Online.
[26] G. Sutcliffe, C. Suttner, The TPTP Problem Library: CNF Release v1.2.1, Journal of Automated Reasoning 21 (1998) 177–203.
[27] 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.
[28] 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.
[29] G. Sutcliffe, C. Benzmüller, Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure, Journal of Formalized Reasoning 3 (2010) 1–27.
[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] 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.
[32] R. Smullyan, What is the Name of This Book? The Riddle of Dracula and Other Logical Puzzles, Prentice-Hall, 1978.
[33] M. Baldoni, Normal Multimodal Logics: Automatic Deduction and Logic Programming Extensions, Ph.D. thesis, Universita degli studi di Torino, Torino, Italy, 1998.
[34] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.
[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] J. Garson, Modal Logic, in: E. Zalta (Ed.), Stanford Encyclopedia of Philosophy, Stanford University, 2018.
[37] C. Menzel, Alethic Modalities, in: R. Audi (Ed.), The Cambridge Dictionary of Philosophy, Cambridge University Press, 2015, p. 22.
[38] G. Forbes, Modern Logic. A Text in Elementary Symbolic Logic, Oxford University Press, 1994.
[39] R. Girle, Modal Logics and Philosophy, Acumen Publishers, 2000.

[40] T. Sider, Logic for Philosophy, Oxford University Press, 2010.
[41] R. Reiter, What Should a Database Know?, Journal of Logic Programming 14 (1992) 127–153.
[42] L. Fariñas del Cerro, A. Herzig, D. Longin, O. Rifi, Belief Reconstruction in Cooperative Dialogues, in: F. Giunchiglia (Ed.), Proceedings of the 8th International Conference on Artificial Intelligence: Methodology, SYstems, and Applications, number 1480 in Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 254–266.
[43] M. Stone, Towards a Computational Account of Knowledge, Action and Inference in Instructions, Journal of Language and Computation 1 (2000) 231–246.
[44] 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.
[45] C. Benzmüller, B. Woltzenlogel Paleo, Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers, in: T. Schaub (Ed.), Proceedings of the 21st European Conference on Artificial Intelligence, 2014, pp. 93–98.
[46] M. Mishra, A. Ravishankar Sarma, Tolerating Inconsistencies: A Study of Logic of Moral Conflicts, Bulletin of the Section of Logic 51 (2022) 177–195.
[47] A. Steen, G. Sutcliffe, P. Fontaine, J. McKeown, Representation, Verification, and Visualization of Tarskian Interpretations for Typed First-order Logic, in: R. Piskac, A. Voronkov (Eds.), Proceedings of 24th International Conference on Logic for Programming Artificial Intelligence and Reasoning, number 94 in EPiC Series in Computing, EasyChair Publications, 2023, pp. 369–385.
[48] S. Kripke, Semantical Considerations on Modal Logic, Acta Philosophica Fennica 16 (1963) 83–94.
[49] S. Schulz, S. Cruanes, P. Vukmirović, Faster, Higher, Stronger: E 2.3, in: P. Fontaine (Ed.), Proceedings of the 27th International Conference on Automated Deduction, number 11716 in Lecture Notes in Computer Science, Springer-Verlag, 2019, pp. 495–507.
[50] A. Steen, Extensional Paramodulation for Higher-order Logic and its Effective Implementation Leo-III, Ph.D. thesis, Free University of Berlin, Berlin, Germany, 2018.
[51] A. Steen, C. Benzmüller, Extensional Higher-Order Paramodulation in Leo-III, Journal of Automated Reasoning 65 (2021) 775–807.
[52] J. Blanchette, T. Nipkow, Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder, in: M. Kaufmann, L. Paulson (Eds.), Proceedings of the 1st International Conference on Interactive Theorem Proving, number 6172 in Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 131–146.
[53] A. Steen, tptp-utils v1.1, 2021. DOI: 10.5281/zenodo.5877564.
[54] A. Steen, Scala TPTP Parser v1.5, 2021. DOI: 10.5281/zenodo.5578872.
[55] A. Steen, An extensible logic embedding tool for lightweight non-classical reasoning (short paper), in: B. Konev, C. Schon, A. Steen (Eds.), Proceedings of the 8th Workshop on Practical Aspects of Automated Reasoning, number 3201 in CEUR Workshop Proceedings, 2022, p. Online.
[56] C. Benzmüller, L. Paulson, Quantified Multimodal Logics in Simple Type Theory, Logica Universalis 7 (2013) 7–20.
[57] C. Benzmüller, T. Raths, 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.
[58] 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.

[59] 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.
[60] C. Areces, B. ten Cate, Hybrid Logics, in: P. Blackburn, J. van Benthem, F. Wolter (Eds.), Handbook of Modal Logic, number 3 in Studies in Logic and Practical Reasoning, North-Holland, 2007, pp. 821–868.
[61] H. van Ditmarsch, W. van de Hoek, B. Kooi, Dynamic Epistemic Logic, Springer, 2007.
[62] E. Pacuit, Dynamic Epistemic Logic I: Modeling Knowledge and Belief, Philosophy Compass 8 (2013) 798–814.
[63] J. Carmo, A. Jones, Completeness and Decidability Results for a Logic of Contrary-to-duty Conditionals, Journal of Logic and Computation 23 (2013).
[64] L. Åqvist, Deontic Logic, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, volume 8, Springer, 2002, pp. 147–264.
[65] C. Nalon, U. Hustadt, C. Dixon, KSP: Architecture, Refinements, Strategies and Experiments, Journal of Automated Reasoning 64 (2020) 461–484.
[66] 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.
[67] 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.
[68] 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.
[69] 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.
[70] 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.
[71] I. Horrocks, A. Voronkov, Reasoning Support for Expressive Ontology Languages Using a Theorem Prover, in: J. Dix, S. Hegner (Eds.), Proceedings of the 4th International Symposium on Foundations of Information and Knowledge Systems, number 3861 in Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 201–218.
[72] R. Schmidt, U. Hustadt, The Axiomatic Translation Principle for Modal Logic, ACM Transactions on Compututational Logic 8 (2007) 19.
[73] M. Schneidner, G. Sutcliffe, Reasoning in the OWL 2 Full Ontology Language using First-Order Automated Theorem Proving, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), Proceedings of the 23rd International Conference on Automated Deduction, number 6803 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2011, pp. 461–475.
[74] C. Eisenhofer, R. Alassaf, M. Rawson, L. Kovács, Non-Classical Logics in Satisfiability Modulo Theories, in: D. Ramanayake, J. Urban (Eds.), Proceedings of the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, number 14278 in Lecture Notes in Computer Science, 2023, pp. 24–36.
[75] A. Steen, G. Sutcliffe, T. Scholl, C. Benzmüller, Solving Modal Logic Problems by Translation

to Higher-order Logic, in: A. Herzig, J. Luo, P. Pardo (Eds.), Proceedings of the 5th International Conference on Logic and Argumentation, number 14156 in Lecture Notes in Computer Science, Springer, 2023, pp. 25–43. (Best paper award).

[76] G. Sutcliffe, Semantic Derivation Verification: Techniques and Implementation, International Journal on Artificial Intelligence Tools 15 (2006) 1053–1070.

[77] S. Trac, Y. Puzis, G. Sutcliffe, An Interactive Derivation Viewer, in: S. Autexier, C. Benzmüller (Eds.), Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, volume 174 of *Electronic Notes in Theoretical Computer Science*, 2007, pp. 109–123.

[78] J. McKeown, G. Sutcliffe, An Interactive Interpretation Viewer for Typed First-order Logic, in: A. Ae Chun, M. Franklin (Eds.), Proceedings of the 36th International FLAIRS Conference, 2023. DOI: 10.32473/flairs.36.133073.

[79] G. Sutcliffe, SystemOnTPTP, in: D. McAllester (Ed.), Proceedings of the 17th International Conference on Automated Deduction, number 1831 in Lecture Notes in Artificial Intelligence, Springer-Verlag, 2000, pp. 406–410.

[80] 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.