

The TPTP Format for Interpretations

Geoff Sutcliffe¹, Alexander Steen², Pascal Fontaine³ and Lydia Kondylidou⁴

¹University of Miami, USA

²University of Greifswald, Germany

³Université de Liège, Belgium

⁴Ludwig-Maximilians-Universität München, Germany

Abstract

This paper describes the (new) TPTP format for representing interpretations. The sources and properties of interpretations, which influenced the design of the format, are discussed. The format for Tarskian, Herbrand, and Kripke interpretations is described. Tools for verification and visualization of the interpretations are described.

Keywords

TPTP World, Interpretations

1. Introduction

Historically, Automated Theorem Proving (ATP) has, as the name suggests, focused largely on the task of proving theorems from axioms – the derivation of conclusions that follow inevitably from known facts [1]. The axioms and the conjecture to be proved (and hence become a theorem) are written in an appropriately expressive logic, and the proofs are often similarly written in logic [2]. The converse task of disproving conjectures, proving that a conjecture is not a theorem of the axioms, is another facet of interest, e.g., [3, 4, 5, 6, 7, 8, 9]. This process depends on finding a *countermodel* for the conjecture, i.e., finding an *interpretation* (a structure that assigns meaning to the symbols of the language of the formulae, and consequently maps formulae to truth values) that is a *model* of the axioms (maps the axioms to *true*) but not a model for the conjecture (maps the conjecture to *false*). A salient application area that uses this form of ATP is verification [10], where a countermodel is used to pinpoint the reason why a proof obligation fails, and correspondingly points to the location of the fault in the system being verified. Other applications of model finding include checking the consistency of an axiomatization [11], operations research [12], commonsense reasoning [13], and solving model finding problems [3].

The TPTP World [14, 15] is the well established infrastructure that supports research, development, and deployment of ATP systems.¹ Various parts of the TPTP World have been deployed in a range of applications, in both academia and industry. The TPTP World includes the TPTP problem library [16], the TSTP solution library [17], tools and services for processing ATP problems and solutions [17], and it supports the CADE ATP System Competition (CASC) [18]. The TPTP language [19] is one of the keys to the success of the TPTP World. Originally the TPTP language supported only first-order clause normal form (CNF) [20]. Over the years full first-order form (FOF) [16], typed first-order form (TFF) [21, 22], typed extended first-order form (TXF) [23], typed higher-order form (THF) [24, 25], dependently typed higher-order form (DHF) [26], and non-classical forms (NXF, NHF) [27] have been added. The TPTP language is used for writing ATP problems, derivations, and, most relevant to this work, *interpretations* [2, 28].

A TPTP format for interpretations with finite domains has previously been defined [2], and has served the ATP community adequately for almost 20 years. The old format is output by several *model*

The 10th Workshop on Practical Aspects of Automated Reasoning, as part of the Federated Logic Conference (FLoC) 2026, Lisbon, Portugal, July 25, 2026

✉ geoff@cs.miami.edu (G. Sutcliffe); alexander.steen@uni-greifswald.de (A. Steen); Pascal.Fontaine@uliege.be (P. Fontaine); Lydia.Kondylidou@tcs.ifi.lmu.de (L. Kondylidou)

id 0000-0001-7116-9338 (G. Sutcliffe); 0000-0001-8781-9462 (A. Steen); 0000-0003-4700-6031 (P. Fontaine);

0009-0001-9875-2627 (L. Kondylidou)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹www.tptp.org

finder ATP systems, e.g., Paradox [29], FM-Darwin [30], Vampire [31]. The need for a format for interpretations with infinite domains, and for a format for Kripke interpretations [32], has led to the development of the (new current) TPTP format for interpretations. This work describes the format. The underlying principle is unchanged: interpretations are represented in formulae written in the TPTP language.

Related work: There are other concrete representations of interpretations in use: The SMT-LIB standard [33] defines a format for model output, and commands to inspect models. SAT solvers generally output models as specified by the SAT competitions [34], in a simple format similar to the DIMACS input format [35]. Some individual model finders have defined their own formats for models, e.g., the output formats of Nitpick [7] and Z3 [36].

This paper is organized as follows: Section 2 discusses the nature of interpretations, considering what is needed from interpretations, and the various forms that interpretations can take. Section 3 defines the format for Tarskian interpretations, and Section 4 does the same for Kripke interpretations. Section 5 concludes and discusses plans for future work.

Note well: This paper is self-contained, but refers to examples of problems and their interpretations in TPTP format that are in the appendices of the extended version of this paper [37]. They are referenced with a □ icon. It is highly recommended that readers of this paper have [37] available.

Declaration on Generative AI: We didn't use any.

2. About Interpretations

2.1. What do we Need?

The needs of applications that use model finding vary according to their use of the model. In the simplest case, applications need to know only that a model exists. Examples of such applications include checking the consistency of an axiomatization [38], use as a subroutine in more complex reasoning, e.g., for axiom selection [39, 40], and establishing the existence of a bug in a verification process². A key weakness of a model finder that claims to have found a model but does not output an explicit model is that it is necessary to trust the model finder.

In many applications it is necessary to have an explicit model, in some representation format that allows for analysis of the model. Applications that productively use an explicit model include finding inconsistencies in axiomatizations [11], identification of bugs in verification [41, 42], studying modal and description logics [43], solving problems encoded as a model finding problems [3], and evaluating formulae wrt the model [44]. Manual inspection of explicit models can also be useful, e.g., [45]. A key strength of a model finder that outputs an explicit model is that the model can be verified, i.e., it is not necessary to trust the model finder.

Given the innate desirability of obtaining explicit models of satisfiable formulae, the desirable properties of interpretation representation are as follows:

1. Interpretations must be for the symbols of the input formulae.
2. Evaluation of any formulae wrt an interpretation should be possible, because many uses of interpretations can be reduced to evaluation [46, §3.2]. The evaluation should be tractable, or in a tractable core. This corresponds to the *atom test* and *formula evaluation* postulates suggested by [47, 46], and might be considered to be the most important property of interpretation representation.

²Bill McCune claimed that establishing the existence of a bug without having an explicit model to help pinpoint the bug would be “frustrating”. And he should have known.

3. Verification of models should be possible by checking that the problem formulae evaluate to *true* wrt the model. The verification should be tractable, or in a tractable core.
4. Interpretations should be sufficiently (human) comprehensible to be useful in (human) applications.

2.2. What do we Have?

A *Tarskian interpretation* [48] consists of a non-empty domain of distinct elements (not the Herbrand Universe – see Herbrand interpretations below) for each type used in the formulae (just one domain for untyped logic), mappings for function symbols from tuples of domain elements to the domain, and mappings for predicate symbols from tuples of domain elements to $\{true, false\}$ [49, 50]. For higher-order logic only Henkin interpretations [51] are considered. An overview of some ways of building and representing Tarskian interpretations is provided by [46], and [52] provides a comprehensive list of approaches.

Two types of Tarskian interpretations are clear (and more might exist):

1. *Finite* Tarskian interpretations have only finite domains. The domain and symbol mappings can be explicitly enumerated. Finite models are typically produced by starting with domains of size one, and incrementing the sizes until a model is found. At each iteration the formulae are reduced to a decidable logic, e.g., propositional [29, 53] or function free [54] logic, and an ATP system for that logic is used to decide if there is a model. There are several ATP systems that produce finite Tarskian models, e.g., Paradox, FM-Darwin, and Vampire.
2. *Infinite* Tarskian interpretations have one or more infinite domains. Infinite domains can be explicitly generated, e.g., terms representing Peano numbers, or implicitly specified, e.g., some set of algebraic numbers such as the integers [55]. There are some ATP systems that produce infinite Tarskian models, e.g., cvc5 [56], FEST [9], and Z3.

A *Herbrand interpretation* [57] is a particular form of Tarskian interpretation. A Herbrand interpretation has the Herbrand universe as the domain, the mapping for function symbols is the “identity”, and the mapping for predicate symbols is from the Herbrand base to $\{true, false\}$. Every set of formulae induces its set of Herbrand models. Formulae in Skolem normal form induce Herbrand models with a Herbrand universe that includes any Skolem symbols. An empty set of formulae induces all interpretations, i.e., all interpretations are models of the set. Such sets of formulae can be the result of applying model preserving transformations to the problem formulae (even the input itself induces its set of Herbrand models), or be generated by an ATP system with the explicit intention of representing Herbrand models. They are important in the context of resolution and all similar modern calculi; saturations (see below) are a common case.

- Formulae that are intended to represent Herbrand interpretations are written *HI-formulae* in this paper, to avoid confusion. Examples include saturations (discussed separately in the next bullet point), Bachmair-Ganzinger models [58, 59], a disjunction of implicit generalisations (DIGs) [4], sets of constrained unit clauses [60, 61, 62], SGGs clause sequences [63], tableaux [64] and hyper-tableaux [65], eq-interpretations [52], and predicate definitions over the term algebra [66]. E-KRHyper [67] was an ATP system that produced hyper-tableaux, and iProver [68, 66] is an ATP system that outputs predicate definitions over the term algebra.
- *Saturations* [69, 70] are a special case of HI-formulae. A saturation is a fixed point for a set of clauses at which further application of a complete inference system does not generate any new clauses (up to redundancy). A saturation of a set of clauses determines the same set of Herbrand models as the clauses themselves. Saturations can be used as explicit models in equational theories [71]. Saturation-based ATP systems include E [72], Prover9 [73], Vampire, and Zipperposition [74].

While the domain of a Herbrand interpretation determined by Herbrand formulae is known to be the Herbrand universe, there might not be an explicit symbol interpretation that can be used constructively by users.

A *Kripke interpretation* [32] adds a layer of *possible worlds* over Tarskian interpretations. There can be a finite or infinite number of worlds. There is an *accessibility relation* between the worlds, which can be subject to the requirements of the logic being used, e.g., for modal logic **M** the accessibility relation must be reflexive [75]. Within each world there is a Tarskian interpretation. The Tarskian interpretations' domains might be *constant*, i.e., the same in all the worlds, but might also vary. The designation of symbols might be *rigid*, i.e., the same in all the worlds' Tarskian interpretations, but might also vary. The designation of ground terms might be *local*, i.e., from only the domain of the world, or might be global. Formulae can be evaluated wrt Kripke interpretations with a finite worlds and local terms by evaluating modal operators using Kripke semantics, and evaluating formulae within a world wrt the Tarskian interpretation in the world.

2.3. What does the TPTP format Provide?

The TPTP format represents interpretations in *interpretation-formulae* that have the new annotated formula role *interpretation*. The details are provided in Sections 3, 3.7 and 4. Interpretation-formulae are written using the TPTP syntax. As a brief reminder of the TPTP syntax, problems and solutions are built from *annotated formulae* of the form:

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

The *languages* 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`, `false`, the individual and boolean types `i`, `o`, 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 Π , Σ , choice (indefinite description), definite description, λ , $\forall$, $\exists$, function application, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$, and $\oplus$ respectively. Equality and inequality are expressed as the infix operators `=` and `!=`. The *source* and *useful_info* are optional.

The notions of interpretations, models, partial interpretations, finite interpretations, Herbrand interpretations, etc., are captured in the SZS ontologies [28].³ For this work the Success ontology was extended with a new value `ModelExtending` (MEX). The success ontology provides values for a pair $Ax-C$, where Ax is a set (conjunction) of axioms and C is a conjecture to be proved from Ax . MEX is defined as "Some interpretations are models of Ax , and some interpretations are models of C , and all models of C are conservative extensions of models of Ax (which means that all models of C are models of Ax)". This is used for transformations on satisfiable sets of formulae in which the models of the set are unchanged, or are changed only by adding new domain elements or mappings, so that the extended models are still models of the original formulae. Examples of such transformations are Skolemization, and adding logical consequences of a set into the set.

2.4. Do we Have what we Need?

Figure 1 provides an overview of the situation. The starting point is the set of Satisfiable formulae written in the language Σ . The satisfiable formulae might have been formed from $Ax \cup \{\sim C\}$. Going down leads to the Models of the satisfiable formulae. Going left from the Satisfiable formulae in Σ is the pathway taken by ATP systems that find a Tarskian/Kripke model, and output interpretation-formulae representing the model. The interpretation-formulae can be for a conservative extension Σ^+ of the language of the input formula, e.g., by the addition of Skolem symbols, defined symbols, etc. If the interpretation-formulae correctly represent a model of the satisfiable formulae, then the satisfiable formulae can be proved ($\models$) from the interpretation-formulae. The models of the interpretation-formulae are conservative extensions of a subset (by the definition of $\models$) of the models of the satisfiable formulae. Going right from the Satisfiable formulae in Σ is the pathway taken by ATP systems (for classical logics at least) that apply transformations to the satisfiable set, to produce formulae that determine Herbrand models of the satisfiable formulae. The satisfiable formulae can be proved ($\models$) from the sets that result from applying the transformations.

³Updated at tptp.org/UserDocs/SZSontology

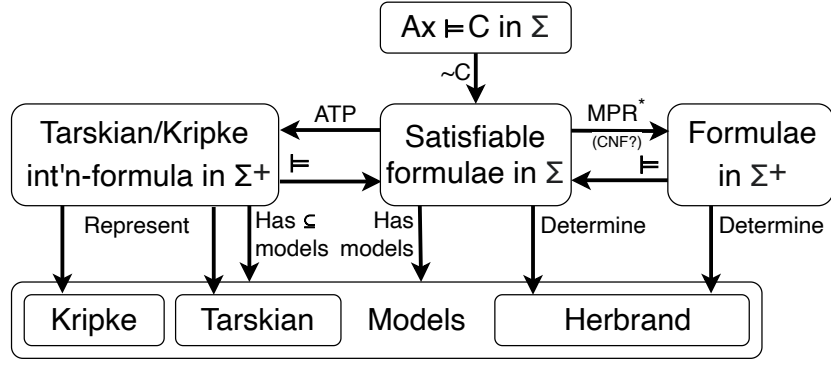


Figure 1: A Landscape of Model Building

2.4.1. Formula Evaluation

The second, most important, property of interpretation representation is to evaluate formulae wrt the interpretation. With interpretation-formulae this can be achieved as follows:

- For Tarskian interpretation-formulae with finite domains, direct evaluation can be used.
 - Interpret quantifiers using Tarskian semantics.
 - Interpret ground (grounded with domain elements) terms and atoms using the mappings.
 - Interpret formulae using the truth tables for connectives.

This approach is taken internally in Vampire to verify its finite models.

- For Tarskian interpretation-formulae, theorem proving can be used.
 - If a formula can be proved from the interpretation-formulae then it is *true* in the interpretation represented by the interpretation-formulae.
 - If a formula can be disproved from the interpretation-formulae then it is *false* in the interpretation represented by the interpretation-formulae.
 - In practice, a formula might be neither proved nor disproved within reasonable resource limits, so that nothing is known.
- For Herbrand interpretations determined by HI-formulae, theorem proving can be used:⁴
 - If a formula can be proved from the HI-formulae then it is *true* in all the Herbrand interpretations determined by the HI-formulae.
 - If a formula can be disproved from the HI-formulae then it is *false* in some of the Herbrand interpretations determined by the HI-formulae.
 - In practice, a formula might be neither proved nor disproved within reasonable resource limits, so that nothing is known.
- For Kripke interpretation-formulae, theorem proving can be used. This is described in Section 4.3.

2.4.2. Model Verification

The third property of interpretation representation is to verify a model of given formulae. Given ways to evaluate a formula wrt interpretation-formulae, models represented in interpretation-formulae can be checked. This has (at least) the following aspects:

1. Verify that the type declarations and interpretation-formulae are syntactically well-formed and well-typed.
2. Validate that the interpretation-formulae correctly represent the model found by the AT system.
3. Verify that the interpretation-formulae are satisfiable.
4. Verify that the interpretation represented by the interpretation-formulae is a model for the given formulae.

These steps can be (in)completed as follows:

1. This can be confirmed using standard parsing and type checking tools, e.g., [76, 77, 78].

⁴ We (well, at least the first author) believe that this verification technique works, and so do Stephan Schulz and Uwe Waldmann, but Andrei Voronkov and Christoph Weidenbach say they do not.

2. This cannot be confirmed without insight into the ATP system that found the model.
3. This can be confirmed using a trusted model finder. Hopefully, confirming that the interpretation-formulae are satisfiable is much easier than finding the model itself, so the ATP system used to check the satisfiability can be weaker and more trusted than the ATP system that found the model. In light of the next point, this model finding task can be made easier by combining the interpretation-formulae with the problem formulae (axioms and negated conjecture).
4. This can be confirmed using the theorem proving approach to verification, in which the problem formulae Φ are proved from the interpretation-formulae φ using a trusted theorem prover. The soundness of this approach is proved by showing that if Φ can be proved from φ then the interpretation I represented by φ is a model of Φ . This is done for finite Tarskian interpretations for FOF in [44]. We believe the proof in [44] lifts naturally to TFF and THF, but is technically more complicated due to the introduction of types. The extension to infinite domains is quite simple after that. For Kripke interpretation-formulae it is necessary to make changes that reduce the verification problem to TXF/THF (see Section 4.3).

Steps 2 to 4 are implemented in the AGMV model verifier [44], available in the SystemOnTSTP [79] web interface.⁵

3. The Format for Tarskian Interpretations

A simple Tarskian interpretation-formula is a conjunction of components (see [37, § A.2, B.2, B.4, D.2]):

- A domain for each type in the problem formulae.
- Interpretation of non-boolean symbols, as equalities whose left-hand sides are formed from symbols applied to domain elements, and whose right-hand sides are domain elements.
- Interpretation of boolean symbols, as literals formed from symbols applied to domain elements; positive literals are *true* and negative literals are *false*.

The TPTP format for Tarskian interpretation-formulae requires no changes to the TPTP language syntax (other than the addition of the `interpretation` role that is used for interpretation-formulae). At this stage only monomorphic types are supported, but the syntax is flexible enough to extend the format to polymorphic types.

Function and predicate symbols are interpreted by applying them directly to domain elements (*à la* [50, §5.3.4]), rather than using the presentation of interpretations that introduces a new interpretation function for each function/predicate symbol [50, §5.3.2]. This approach obviates the need for introducing interpretation functions, and makes interpretation of formulae using theorem proving simpler. However, applying the problem's function and predicate symbols to domain elements that are defined with different types to the problem's types requires using *type-promotion* bijections to keep interpretation-formulae well-typed (see Section 3.3).

[37, § A.2, B.2, C.2, C.3, D.2] illustrate the components of finite Tarskian interpretations in a single interpretation-formula, as explained in Sections 3.1, 3.2, and 3.3. [37, § A.4, A.5, B.6, B.7, D.4] illustrate Tarskian interpretations split over multiple interpretation-formulae, as explained in Section 3.5. [37, § B.8] illustrates a Tarskian interpretation that is compacted using quantification, as explained in Section 3.6. [37, § A.6, A.7] illustrate interpretation-formulae that determine Herbrand interpretations, as explained in Section 3.7. These examples illustrate the possibilities for writing Tarskian interpretations in the TPTP format, but are not intended to be exhaustive. In particular, interpretations of higher-order formulae have yet to be fleshed out.

3.1. Finite Interpretations without Types

[37, § A.1] shows a FOF problem that has a countermodel with a finite domain, which is shown in [37, § A.2].

The domain specification of a finite interpretation without types is a conjunction of:

- Specification of the domain elements as a universally quantified disjunction of equalities whose right-hand sides are the domain elements, e.g., in [37, § A.2]:

$$! [X] : (X = \text{"a"} \mid X = \text{"f"} \mid X = \text{"john"} \mid X = \text{"gotA"})$$
- Specification of the distinctness of the domain elements. Distinctness can be specified with pairwise inequalities, with the `$distinct` predicate, or by using "double quoted" terms as domain elements (different "double quoted" terms are known to be unequal (and often not used in problems, which makes them easy to identify as domain elements)), e.g., in [37, § A.2] "a" is known to be unequal to "f".

⁵www.tptp.org/cgi-bin/SystemOnTSTP

The function mappings are equalities between domain elements and functions applied to domain elements, e.g., in [37, □ A.2]:

```
grade_of("john") = "f"
```

The predicate mappings are literals, e.g., in [37, □ A.2]:

```
created_equal("john", "gotA")
```

If "double quoted" constants are already used in the formulae being interpreted, they are naturally interpreted as themselves. Note that for a complete interpretation all function and predicate mappings need to be provided, even if they make no sense in a typed sense, e.g., in [37, □ A.2]:

```
grade_of("f") = "a"
```

```
~created_equal("a", "john")
```

[37, □ A.3] shows the verification problem for the model.

3.2. Finite Interpretations reusing Problem Types

[37, □ B.1] shows a TFF problem that has a countermodel with finite domains, which is shown in [37, □ B.2].

Each domain specification is a conjunction of:

- Specification of the domain elements as a universally quantified disjunction of equalities whose right-hand sides are the domain elements, e.g., in [□: ??]:

```
! [DC: cat]: ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
```
- Specification of the distinctness of the domain elements, e.g., in [37, □ B.2]:

```
$distinct(d_garfield, d_arlene, d_nermal)
```

The function mappings are equalities between domain elements and functions applied to domain elements, e.g., in [37, □ B.2]:

```
loves(d_garfield) = d_garfield
```

The predicate mappings are literals, e.g., in [37, □ B.2]:

```
~owns(d_jon, d_nermal)
```

The interpretation-formulae are preceded by the necessary type declarations:

- The types in the problem.
- The types of the domains.
- The types of the domain elements.

[37, □ B.3] shows the verification problem for the model.

3.3. Finite Interpretations with Domain Types

Reusing the problem types for domain elements can get murky, e.g., when quantification over only the domain elements is intended. It is clearer to use new types for domain elements, and it becomes necessary when the domain elements are complex or of a defined type such as \$int (see Section 3.4). As noted in the introduction to Section 3, applying the problem's function and predicate symbols to domain elements that are defined with different types to the problem's types requires using type-promotion bijections to "convert" domain elements to terms of the corresponding problem type.

[37, □ B.1] shows a TFF problem that has a countermodel with finite domains, which is shown in [37, □ B.4]

[37, □ D.1] shows a THF problem that has a countermodel with finite domains, which is shown in [37, □ D.2].

Each domain specification is a conjunction of:

- Specification of the domain for each problem type, as a formula that makes the type-promotion function a surjection, e.g., in [37, □ B.4]:

```
! [H: human] : ? [DH: d_human] : H = d2human(DH)
! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
```

and in [37, □ D.2]:

```
! [B: beverage] : ? [DB: d_beverage] : ( B = ( d2beverage @ DB ) )
! [S: syrup] : ? [DS: d_syrup] : ( S = ( d2syrup @ DS ) )
```
- Specification of the domain elements as a universally quantified disjunction of equalities whose right-hand sides are the domain elements, e.g., in [37, □ B.4]:

```
! [DH: d_human] : ( DH = d_jon )
! [DC: d_cat]: ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
```

and in [37, □ D.2]:

```
! [DB: beverage] : ( DB = d_coffee )
! [DS: d_syrup] : ( DS = d_vanilla )
```

- Specification of the distinctness of the domain elements, e.g., in [37, § B.4]:
 $\$distinct(d_garfield, d_arlene, d_nermal)$
 If each domain has only one element this is unnecessary, as in [37, § D.2].
- A formula making the type-promotion function an injection, which together with the surjectivity makes it a bijection, e.g., in [37, § B.4]:
 $! [DH1: d_human, DH2: d_human] : (d2human(DH1) = d2human(DH2) \Rightarrow DH1 = DH2)$
 $! [DC1: d_cat, DC2: d_cat] : (d2cat(DC1) = d2cat(DC2) \Rightarrow DC1 = DC2)$
 and in [37, § D.2]:
 $! [DB1: d_beverage, DB2: d_beverage] : ((d2beverage @ DB1) = (d2beverage @ DB2)) \Rightarrow (DB1 = DB2))$
 $! [DS1: d_syrup, DS2: d_syrup] : ((d2syrup @ DS1) = (d2syrup @ DS2)) \Rightarrow (DS1 = DS2))$

The function mappings are equalities between type-promoted domain elements and functions applied to type-promoted domain elements, e.g., in [37, § B.4]:

```
jon = d2human(d_jon)
loves(d2cat(d_garfield)) = d2cat(d_garfield)
and in [37, § D.2]:
coffee = ( d2beverage @ d_coffee )
( heat @ ( d2beverage @ d_coffee ) ) = ( d2beverage @ d_coffee )
```

The predicate mappings are literals, e.g., in [37, § B.4]:

```
~ owns(d2human(d_jon), d2cat(d_nermal))
and in [37, § D.2]:
hot @ ( d2beverage @ d_coffee )
```

The mappings of higher-order function symbols are lambda abstractions, e.g., in [37, § D.2]:

```
mix = ( ^ [F: syrup > beverage] : ( d2beverage @ d_coffee ) )
heated_mix = ( ^ [F: syrup > beverage] : ( d2beverage @ d_coffee ) )
```

The lambda abstractions are convenient because the interpretation of symbols that take a function argument must specify the interpretation for every such function. This differs from ground domain elements that can simply be assigned a name. A function is entirely characterised by its values on all inputs, so specifying a function-type element necessarily involves defining its behaviour, which is expressed as a lambda term. For infinite base types such as $\$int$, the function space cannot be enumerated, providing an additional reason for using lambda abstractions, but the use of lambda abstractions is required regardless of cardinality.

The TFF and THF interpretation-formulae are preceded by the necessary type declarations:

- The types in the problem.
- The types of the domains.
- The types of the domain elements.
- The types of the type-promotion functions.

[37, § B.5, D.3] show the verification problems for the models. Sometimes higher-order formulae have models that can be expressed in TFF, which is preferable for presentation and readability. However, since the model and its verification problem must be in the same language, and verification obligations might require higher-order reasoning, both must be stated in THF.

3.4. Infinite Interpretations

Interpretations can have infinite domains formed from terms or defined types such as $\$int$. [37, § C.1] shows a TFF problem that has a model with an infinite domain. [37, § C.2] shows the model with an infinite term domain, and [37, § C.3] shows the model with an $\$int$ domain.

Each domain specification is a conjunction of:

- The domain for each problem type, as a formula that makes the type-promotion function a surjection, e.g., in [37, § C.2]:
 $! [P: person] : ? [I: peano] : (P = peano2person(I))$
 and in [37, § C.3]:
 $! [P: person] : ? [I: \$int] : P = int2person(I)$
 If the problem type and the domain type are the same defined type, e.g., both are $\$int$, this is unnecessary.
- Specification of the domain elements as an existentially quantified formula that captures an infinite disjunction of equalities, e.g., in [37, § C.2]:

- ! [I: peano] : (I = zero | ? [P: peano] : I = s(P))
If the domain is a defined type such as \$int this is unnecessary, as in [37, □ C.3].
- Specification of the distinctness of the domain elements, unless implicit from their type. e.g., in [37, □ C.2]:
(peano_less(I1,I2) => I1 != I2)
If the domain is a defined type such as \$int this is unnecessary, as in [37, □ C.3].
- A formula making the type-promotion function an injection, which together with the surjectivity makes it a bijection, e.g., in [37, □ C.2]:
! [I1: peano, I2: peano] : (peano2person(I1) = peano2person(I2) => I1 = I2)
and [□: ??]:
! [I1: \$int, I2: \$int] : (int2person(I1) = int2person(I2) => I1 = I2)

The function mappings are universally quantified equalities between type-promoted domain elements and functions applied to type-promoted domain elements, e.g., in [37, □ C.2]:

! [I: peano] : child_of(peano2person(I)) = peano2person(s(I))
and in [37, □ C.3]:

! [I: \$int] : child_of(int2person(I)) = int2person(\$sum(I,1))

The predicate mappings are universally quantified formulae, e.g., in [37, □ C.2]:

! [A: peano, D: peano] :
(is_descendant(peano2person(A), peano2person(D)) <=> peano_less(A,D))
and in [37, □ C.3]:

! [A: \$int, D: \$int] :
(is_descendant(int2person(A), int2person(D)) <=> peano_less(A,D))

The interpretation of function and predicate symbols is not given for argument tuples with negative integers.

The interpretation-formulae are preceded by the necessary type declarations:

- The types in the problem.
- The types of the domains.
- The types of the domain elements.
- The types of the type-promotion functions.

[37, □ C4, C5] show the verification problems for the models.

3.5. Interpretations with Multiple Interpretation-formulae

The interpretation-formulae described thus far have all the information about an interpretation in a single interpretation-formula. In some situations it is useful to split the various components into separate interpretation-formulae, e.g., separating the domains from the symbol mappings. The TPTP format offers splitting in a flexible way, at medium and fine grained levels, using annotated formula subroles.

At the medium grained level, the interpretation role can be extended with the subroles domains and mappings. Two (or more) interpretation-formulae are used, each containing the corresponding parts of the interpretation. [37, □ A.4, B.6] show examples of splitting the interpretation-formulae in the FOF and TFF interpretations [37, □ A.2, B.2] respectively, which separate the domain specifications from the symbol mappings. For example, in [37, □ B.6]:

tff(garfield_domains, interpretation-domains, *the domains*) .

contains the information about the two domains, and the interpretation-formula:

tff(garfield_mappings, interpretation-mappings, *the mappings*) .

contains the symbol mappings. Note that each role and role-subrole pair can be used multiple times according to need, e.g., in [37, □ A.4] there can be separate interpretation-mappings interpretation-formulae, one for functions and one for predicates.

At the fine grained level, interpretation-formulae can split the individual domains and symbol mappings, with the domains and mappings subroles given arguments to indicate which domain or symbol mapping is recorded. For domains the arguments of the subrole are the domain in the problem and the domain in the interpretation. For mappings the arguments are the symbol and its result domain type. [37, □ A.5, B.7] show examples of splitting the interpretation-formulae in [37, □ A.2, B.2] respectively. For example, in [37, □ B.7]:

tff(garfield_domain_human, interpretation-domains(human, d_human)

contains the d_human domain specification, and:

tff(garfield_domain_cat, interpretation-domains(cat, d_cat)

contains the d_cat domain specification.

The medium and fine grained splitting can be mixed. [37, □ D.4] shows an example of mixed splitting of [37, □ D.2]. For example, the domains are specified separately in [37, □ D.4]:

```

thf(hot_coffee_beverage, interpretation-domains(beverage, d_beverage),
    beverage domain) .
thf(hot_coffee_syrup, interpretation-domains(syrup, d_syrup), syrup domain) .
while all the symbol mappings are in:
thf(hot_coffee, interpretation-mappings, the mappings) .

```

3.6. Interpretations Compacted by Quantification

The full logical language can be used in interpretation-formulae to provide compaction. [37, § B.8] shows an example compacting the interpretation-formula `garfield_mapping_loves` in [37, § B.2]. For example, universal quantification can be used to map `loves` to `d_garfield` for all `d_cats` in [37, § B.8]:

```
! [DC: d_cat] : ( loves(d2cat(DC)) = d2cat(d_garfield) )
```

3.7. HI-formulae and Saturations

HI-formulae determine sets of Herbrand interpretations. Interpretation-formulae can be used for these, although the actual Herbrand interpretations are not easy to extract. To indicate that the interpretation-formulae are intended to determine Herbrand interpretations they are given the subrole `herbrand`. [37, § A.1] shows a problem that has a HI-formulae model formed from predicate definitions over the term algebra, which is shown in [37, § A.6]. [37, § A.1] also has a saturation, which is shown in [37, § A.7]. [37, § A.8, A.9] show the corresponding verification problems.

3.8. Visualization of Finite Tarskian Interpretations

Finite Tarskian interpretations with domain types, as described in Section 3.3, can be visualised in the Interactive Interpretation Viewer (IIV) [80]. Figure 2 shows the view of the model in [37, § B.4]. The cursor is hovering over the `d_nermal` node in the interpretation of the `owns` predicate applied to `d_jon` and `d_nermal`, which is interpreted as `$false`. The `$o` triangle above the `owns` inverted house shows that `owns` has a boolean result in the problem, and the `$o` triangle below `$false` box shows that the interpretation value is boolean. IIV is available in the SystemOnTSTP web interface.

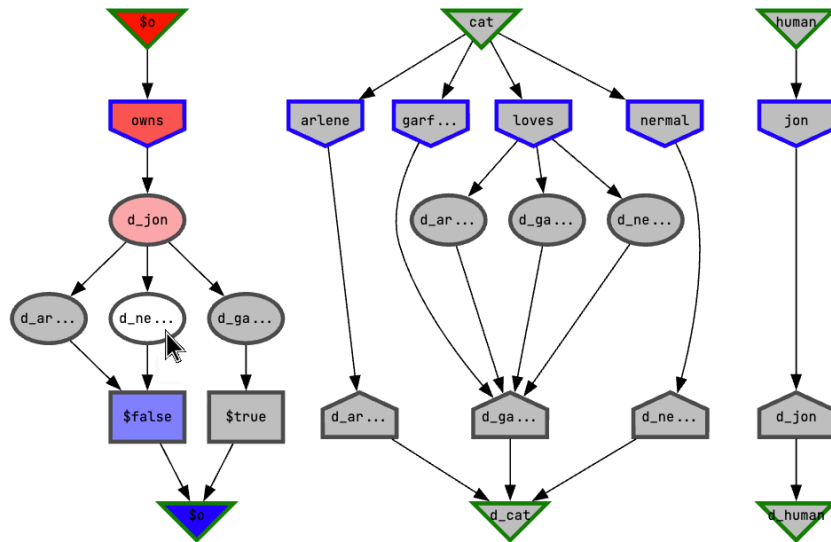


Figure 2: IIV view of [37, § B.4]

4. The Format for Kripke Interpretations

The TPTP format for Kripke interpretations also uses interpretation-formulae. As was noted in Section 2.2, Kripke interpretations can have either a finite or an infinite number of worlds, and the interpretations in a world can be Tarskian or Herbrand. The TPTP format for Kripke interpretation-formulae adds four defined symbols:

- A defined type `$world` for the worlds of the interpretation. Different constants of type `$world` are known to be unequal.
- A defined predicate `$accessible_world` of type $(\$world * \$world) > \$o$ to specify the accessibility relation between worlds.
- A defined constant `$local_world` of type `$world` to specify the world in which a local (the default) conjecture is to be proved.
- A defined predicate `$in_world` of type $(\$world * \$o) > \$o$ to specify the interpretations in the worlds.

A single Kripke interpretation-formula is a conjunction of:

- A specification of the worlds.
- Explication of the distinctness of the worlds (by definition different worlds are known to be distinct, but no ATP systems know that yet, so it's necessary to encode the distinctness explicitly using inequalities or the `$distinct` predicate).
- The accessibility relation.
- Specification of the local world if any.
- For each world, its Tarskian interpretation (also in the TPTP format for interpretations), augmented by existential subformulae that require the existence of the world's domain elements – these are necessary because the semantics is that of modal logic (see Sections 4.1 and 4.2 for examples).

The interpretation-formulae are preceded by the necessary type declarations:

- The types for the Tarskian interpretations in the worlds.
- The worlds declared of type `$world`, e.g., `w1: $world`.

The logic specification of the problem is included to specify that the interpretation is for formulae in that logic. This information is needed when processing an interpretation, e.g., in verification (see Section 2.4.2). 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).

[37, □ E.2, E.5] show the components of finite Kripke interpretations in a single interpretation-formula, as explained in Sections 4.1. [37, □ E.7, E.8] show Kripke interpretations split over multiple interpretation-formulae, as explained in Section 4.2. [37, □ E.9] shows a Kripke interpretations that is compacted using quantification, as explained in Section 4.2. These examples illustrate the possibilities for writing Kripke interpretations in the TPTP format, but are not intended to be exhaustive. In particular, Kripke interpretations with an infinite number of worlds are not considered.

4.1. Finite-Finite Kripke Interpretations

[37, □ E.1] shows a NXF problem that has a countermodel with finite worlds, each of which contains a finite Tarskian interpretation, which is shown in [37, □ E.2]. The problem has global axioms and a local conjecture. The countermodel has three worlds in [37, □ E.2]:

```
! [W: $world] : ( W = w1 | W = w2 | W = w3 )
$distinct(w1,w2,w3)
```

with the accessibility relation:

```
$accessible_world(w1,w1) & $accessible_world(w2,w2)
& $accessible_world(w1,w2) & $accessible_world(w2,w3)
& $accessible_world(w3,w1)
& ~$accessible_world(w1,w3) & ~$accessible_world(w2,w1)
& ~$accessible_world(w3,w2) & ~$accessible_world(w3,w3)
```

The conjecture is disproved in a local world in [37, □ E.2]:

```
$local_world = w1
```

The finite Tarskian interpretations for the worlds are provided in the format described in Section 3, e.g.,:

```
$in_world(w1, the Tarskian interpretation)
```

Note the augmentation of the Tarskian interpretations with subformulae such as:

? [CD: child_d] : CD = child_1

The quantification semantics is not classical, but instead that of modal logic, i.e., the quantification is over the domain elements that exist in the world. That subformula requires that the domain element `child_1` exists in the world. [37, □ E.3] shows the verification problem for the model.

[37, □ E.4] shows a NXF problem that has both local and global axioms. It has a countermodel with finite worlds each of which contains a finite Tarskian interpretation, shown in [37, □ E.5]. This example illustrates how a local axiom (with role `axiom-local`) is satisfied in only the local world (`w1`). [37, □ E.6] shows the verification problem for the model.

4.2. Kripke Interpretations Split and Compacted

Kripke interpretation-formulae can be split to separate the various components of the interpretation. The `interpretation` role can be extended with the subroles `worlds`, `domains`, and `mappings`. [37, □ E.7] shows a medium grained split of [37, □ E.2]. An interpretation-formula with the role `interpretation-worlds` contains information about the worlds of the interpretation, e.g., in [37, □ E.7] one interpretation-formula contains the information about the three worlds:

tff(people_worlds, interpretation-worlds, world information) .

Splitting can be done in a flexible way, using the fine grained splitting available for Tarskian interpretation-formulae. For example, the interpretation-formula in [37, □ E.8]:

tff(child_domains, interpretation-domains(child, child_d), child domain) .

contains the information about the domain `child_d`, the interpretation-formula:

tff(people_mappings, interpretation-mappings(charly, child_d) charly mapping) .

contains the mapping for the `charly` constant, and the interpretation-formula:

tff(rains_mappings, interpretation-mappings(rains, \$o), rains mapping) .

contains the mapping for the `rains` predicate.

In the same way that Tarskian interpretation-formulae can be compacted, Kripke interpretation-formulae can be compacted, e.g., by using universal quantification over the worlds to factor out common elements of the Tarskian interpretations in the worlds. For example, the interpretation-formula in [37, □ E.9]:

tff(people_domains, interpretation-domains,
! [W: \$world] : \$in_world(W, the domain specification) .

contains the information about the domains that is the same in all worlds in [37, □ E.2], and the interpretation-formula in [37, □ E.9]:

tff(charly_mappings, interpretation-mappings(charly, child_d),
! [W: \$world] : \$in_world(W, (charly = d2child(child_1))) .

contains the mapping for the `charly` constant that is the same in all the worlds in [37, □ E.2], and the interpretation-formula in [37, □ E.9]:

tff(rains_mappings, interpretation-mappings(rains, \$o),
! [W: \$world] : \$in_world(W, rains) .

contains the predicate mapping for `rains` that is the same in all the worlds in [37, □ E.2].

4.3. Verification of Kripke Models

Kripke models written as interpretation-formulae can be verified using the theorem proving approach described in Section 2.4.2.⁶ An NXF verification problem is built from the NXF problem formulae and the TXF interpretation-formulae. Although the verification problem includes the non-classical connectives in the problem formulae, it is not necessary to provide a full logic specification (as in [37, □ E.1, E.4] because the logical setting is captured in the interpretation-formulae. The verification problem is thus not in the same logic as the problem formulae, but rather some kind of hybrid logic [81], which is indicated by the special logic specification value `$$fom1Model`. [37, □ E.3] shows the NXF verification problem for the NXF problem [37, □ E.1] and its countermodel in [37, □ E.2]. [37, □ E.6] shows the NXF verification problem for the problem in [37, □ E.4] and its countermodel in [37, □ E.5].

In the NXF verification problem the Kripke interpretation-formulae are given the `axiom` role, and each of the problem axioms and the negated problem conjecture are given the `conjecture` role with a subrole of `local` or `global` according to their role in the problem. For example, the axiom in [37, □ E.1]:

tff(a1, axiom,
! [C: child] : ~(~quiet(C) & ? [A: adult] : sleepy(A))) .

is global (the default), so the NXF verification conjecture has the `global` subrole in [37, □ E.3]:

⁶To date we have done this for NXF problem formulae and TXF interpretation-formulae. Extension to NHF should be possible.

```
tff(a1, conjecture-global,
The conjecture of the problem is local (the default) in [37, □ E.1]:
tff(c, conjecture,
    {$possible} @ ( ~ rains ) ).
```

so the NXF verification (negated) conjecture has the local subrole in [37, □ E.3]:

```
tff(c, conjecture-local, verification obligation).
```

The NXF verification problem is embedded into TH0 using the ATFLET tool. ATFLET recognizes the special logic specification and the type and predicates, and combines the NXF conjectures into a single TH0 conjecture. The resultant TH0 problem is discharged using a trusted TH0 ATP system.

4.4. Visualization of Finite-Finite Kripke Interpretations

Finite-finite Kripke interpretations with domain types, as described in Section 4.1, can be visualised in the Interactive Interpretation Viewer (IIV). The left hand side of Figure 3 shows the worlds of the model in [37, □ E.2]. The cursor is hovering over the w1 world, and the worlds are coloured in lighter shades according to the number of accessibility relations that have to be traversed from the selected world. Clicking on a world displays the Tarskian interpretation in the world, shown in the right hand side of Figure 3. IIV is available in the SystemOnTSTP web interface.

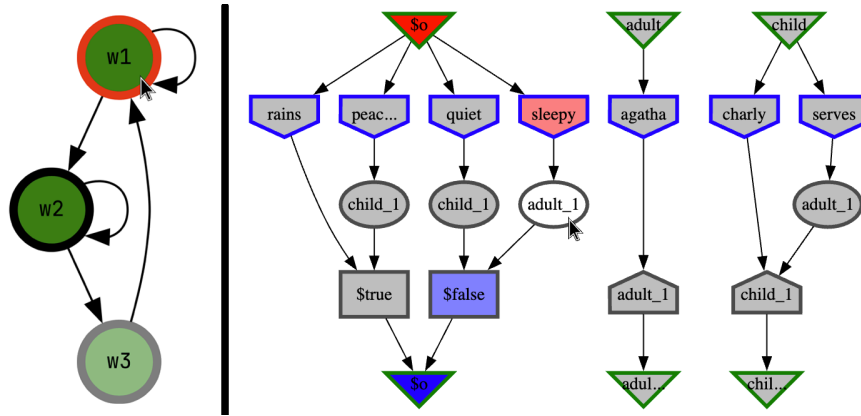


Figure 3: IIV view of the Kripke model in [37, □ E.2]

5. Conclusion

This paper describes the TPTP format for representing interpretations. It provides a background survey that helped ensure that the representation format is adequate for different types of interpretations, including Tarskian, Herbrand, and Kripke interpretations. Some tools that support processing the format have been noted.

The TPTP format provides numerous options and features, and it is expected that only the basic features will be adopted initially. For ATP systems that already output the old TPTP format for finite Tarskian interpretations, all that is needed is to replace the `fi_domain`, `fi_functors`, and `fi_predicates` roles by interpretation. To be more informative, `fi_domain` can be replaced by `interpretation-domains`, and `fi_functors` and `fi_predicates` can be replaced by `interpretationmappings`.

Section 2.1 lists four needs for interpretations:

1. Evaluability - the ability to evaluate a formula wrt an interpretation.
2. Tractability - the ability to evaluate a formula using some limited/reasonable amount of resources.

3. Verifiability - the ability to verify a model by evaluating all the formulae it claims to model as *true*.
4. Comprehensibility - be sufficiently comprehensible to humans and machines to be useful in applications.

In the light of the above (and hey, maybe there are more techniques than just those) we (well, at least the first author) claim:

- Finite interpretations represented by interpretation-formulae are typically evaluable, tractable, verifiable, and comprehensible.
- Infinite interpretations represented by interpretation-formulae are evaluable, often intractable, verifiable, and might not be comprehensible.
- HI-formulae (in interpretation-formulae) are evaluable, can be tractable, verifiable, and can be comprehensible.
- Saturations (in interpretation-formulae) are evaluable, often intractable, verifiable, and almost always incomprehensible.
- The verifiability, tractability, and comprehensibility of Kripke interpretations vary between individual cases.
- Use of the same language as the problem formulae for writing interpretation-formulae contributes to comprehensibility (but does not ensure it).

This work will be extended to deal with more non-classical languages. But for now we are pausing to allow ATP system developers to implement the format in their ATP systems.

References

- [1] A. Robinson, A. Voronkov, Handbook of Automated Reasoning, Elsevier Science, Amsterdam, The Netherlands, 2001.
- [2] 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, Berlin, Germany, 2006, pp. 67–81.
- [3] S. Winker, Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover Answering Two Open Questions, Journal of the ACM 29 (1982) 273–284.
- [4] J.-L. Lassez, K. Marriott, Explicit Representation of Terms Defined by Counter Examples, Journal of Automated Reasoning 3 (1987) 301–317.
- [5] W. McCune, Automatic Proofs and Counterexamples for some Ortholattice Identities, Information Processing Letters 65 (1998).
- [6] J. Zhang, Computer Search for Counterexamples to Wilkie’s Identity, in: R. Nieuwenhuis (Ed.), Proceedings of the 20th International Conference on Automated Deduction, number 3632 in Lecture Notes in Artificial Intelligence, Springer, Berlin, Germany, 2005, pp. 441–451.
- [7] 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, Berlin, Germany, 2010, pp. 131–146.
- [8] A. Reynolds, T. King, V. Kuncak, Solving Quantified Linear Arithmetic by Counterexample-guided Instantiation, Formal Methods in System Design 51 (2017) 500–532.
- [9] N. Elad, O. Padon, S. Shoham, An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification, in: D. Dreyer (Ed.), Proceedings of the ACM on Programming Languages, volume 8, ACM Press, 2024, pp. 970–1000.
- [10] V. D’Silva, D. Kroening, G. Weissenbacher, A Survey of Automated Techniques for Formal Software Verification, IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems 27 (2008) 1165–1178.
- [11] S. Schulz, G. Sutcliffe, J. Urban, A. Pease, Detecting Inconsistencies in Large First-Order Knowledge Bases, in: L. de Moura (Ed.), Proceedings of the 26th International Conference on Automated Deduction, number 10395 in Lecture Notes in Computer Science, Springer, Berlin, Germany, 2017, pp. 310–325.
- [12] J. Hooker, New Methods for Computing Inferences in First-order Logic, Annals of Operations Research 43 (1993) 477–492.
- [13] R. Khardon, D. Roth, Reasoning with Models, Artificial Intelligence 87 (1994) 187–213.
- [14] 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. doi:{10.1007/s10817-017-9407-7}.

- [15] G. Sutcliffe, Stepping Stones in the TPTP World, in: C. Benzmüller, M. Heule, R. Schmidt (Eds.), *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in *Lecture Notes in Artificial Intelligence*, 2024, pp. 30–50.
- [16] 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.
- [17] 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, Berlin, Germany, 2010, pp. 1–12.
- [18] G. Sutcliffe, The CADE ATP System Competition - CASC, *AI Magazine* 37 (2016) 99–101.
- [19] G. Sutcliffe, The Logic Languages of the TPTP World, *Logic Journal of the IGPL* 31 (2023) 1153–1169.
- [20] G. Sutcliffe, C. Suttner, The TPTP Problem Library: CNF Release v1.2.1, *Journal of Automated Reasoning* 21 (1998) 177–203.
- [21] 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, Berlin, Germany, 2012, pp. 406–419.
- [22] 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, Berlin, Germany, 2013, pp. 414–420.
- [23] 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.
- [24] G. Sutcliffe, C. Benzmüller, Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure, *Journal of Formalized Reasoning* 3 (2010) 1–27.
- [25] 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.
- [26] D. Ranalter, C. Kaliszyk, F. Rabe, G. Sutcliffe, The Dependently Typed Higher-Order Form for the TPTP World, in: R. Thiemann, C. Weidenbach (Eds.), *Proceedings of the 15th International Symposium on Frontiers of Combining Systems*, number 15979 in *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Germany, 2025, pp. 287–305. doi:{10.1007/978-3-032-04167-8_16}.
- [27] 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.
- [28] 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.
- [29] K. Claessen, N. Sörensson, New Techniques that Improve MACE-style Finite Model Finding, in: P. Baumgartner, C. Fermueller (Eds.), *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.
- [30] P. Baumgartner, A. Fuchs, H. de Nivelle, C. Tinelli, Computing Finite Models by Reduction to Function-Free Clause Logic, in: W. Ahrendt, P. Baumgartner, H. de Nivelle (Eds.), *Proceedings of the 3rd Workshop on Disproving - Non-Theorems, Non-Validity, Non-Provability*, 3rd International Joint Conference on Automated Reasoning, 2006.
- [31] L. Kovács, A. Voronkov, First-Order Theorem Proving and Vampire, in: N. Sharygina, H. Veith (Eds.), *Proceedings of the 25th International Conference on Computer Aided Verification*, number 8044 in *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Germany, 2013, pp. 1–35.
- [32] S. Kripke, Semantical Considerations on Modal Logic, *Acta Philosophica Fennica* 16 (1963) 83–94.
- [33] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.6, 2017. <https://smtlib.cs.uiowa.edu>.
- [34] M. Järvisalo, D. Le Berre, O. Roussel, L. Simon, The International SAT Solver Competitions, *AI Magazine* 33 (2012) 89–92.
- [35] D. Babic, Satisfiability Suggested Format, 1993. <https://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf>.
- [36] L. de Moura, N. Bjørner, Z3: An Efficient SMT Solver, in: C. Ramakrishnan, J. Rehof (Eds.), *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 4963 in *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Germany, 2008, pp. 337–340.

- [37] G. Sutcliffe, A. Steen, P. Fontaine, L. Kondylidou, The TPTP Format for Interpretations, 2026. 10.48550/arXiv.2406.06108.
- [38] V. Chaudhri, D. Inlezan, Representing States in a Biology Textbook, in: L. Leora Morgenstern, T. Patkos, R. Sloan (Eds.), Proceedings of 12th International Symposium on Logical Formalizations of Commonsense Reasoning, AAAI Press, Palo Alto, USA, 2015.
- [39] G. Sutcliffe, Y. Puzis, SRASS - a Semantic Relevance Axiom Selection System, in: F. Pfenning (Ed.), Proceedings of the 21st International Conference on Automated Deduction, number 4603 in Lecture Notes in Artificial Intelligence, Springer, Berlin, Germany, 2007, pp. 295–310.
- [40] P. Pudlak, Semantic Selection of Premises for Automated Theorem Proving, in: J. Urban, G. Sutcliffe, S. Schulz (Eds.), Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories, number 257 in CEUR Workshop Proceedings, 2007, pp. 27–44.
- [41] E. Clarke, E. Emerson, Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic, in: D. Kozen (Ed.), Proceedings of the Workshop on Logics of Programs, number 131 in Lecture Notes in Computer Science, Springer, Berlin, Germany, 1982, pp. 52–71.
- [42] J. Queille, J. Sifakis, Specification and Verification of Concurrent Systems in CESAR, in: M. Dezani-Ciancaglini, U. Montanari (Eds.), Proceedings of the 5th International Symposium on Programming, number 137 in Lecture Notes in Computer Science, Springer, Berlin, Germany, 1982, p. 337–351.
- [43] R. Schmidt, U. Hustadt, First-Order Resolution Methods for Modal Logics, in: A. Voronkov, C. Weidenbach (Eds.), Programming Logics, Essays in Memory of Harald Ganzinger, number 7797 in Lecture Notes in Computer Science, Springer, Berlin, Germany, 2013, pp. 345–391.
- [44] 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, Manchester, United Kingdom, 2023, pp. 369–385.
- [45] M. Emmer, Z. Khasidashvili, K. Korovin, A. Voronkov, Encoding Industrial Hardware Verification Problems into Effectively Propositional Logic, in: R. Bloem, N. Sharygina (Eds.), Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design, IEEE Press, 2010, pp. 137–144.
- [46] R. Caferra, A. Leitsch, N. Peltier, Automated Model Building, number 31 in Applied Logic Series, Kluwer Academic Publishers, 2004.
- [47] C. Fermüller, A. Leitsch, Hyperresolution and Automated Model Building, Journal of Logic and Computation 6 (1996) 173–203.
- [48] A. Tarski, R. Vaught, Arithmetical Extensions of Relational Systems, Compositio Mathematica 13 (1956) 81–102.
- [49] G. Hunter, Metalogic: An Introduction to the Metatheory of Standard First Order Logic, University of California Press, 1996.
- [50] J. Gallier, Logic for Computer Science - Foundations of Automatic Theorem Proving, Dover Publications, Mineola, USA, 2015.
- [51] L. Henkin, Completeness in the Theory of Types, Journal of Symbolic Logic 15 (1950) 81–91.
- [52] N. Peltier, A Calculus Combining Resolution and Enumeration for Building Finite Models, Journal of Symbolic Computation 36 (2003) 49–77.
- [53] W. McCune, Mace4 Reference Manual and Guide, Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA, 2003.
- [54] P. Baumgartner, A. Fuchs, H. de Nivelle, C. Tinelli, Computing Finite Models by Reduction to Function-Free Clause Logic, Journal of Applied Logic 7 (2009) 58–74.
- [55] P. Baumgartner, J. Bax, Proving Infinite Satisfiability, 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, Berlin, Germany, 2013, pp. 86–95.
- [56] H. Barbosa, C. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A Versatile and Industrial-Strength SMT Solver, in: D. Fisman, G. Rosu (Eds.), Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, number 13243 in Lecture Notes in Computer Science, Springer, Berlin, Germany, 2022, pp. 415–442.
- [57] J. Herbrand, Recherches sur la Théorie de la Démonstration, Travaux de la Société des Sciences et des Lettres de Varsovie, Class III, Sciences Mathématiques et Physiques 33 (1930).
- [58] L. Bachmair, H. Ganzinger, Rewrite-based Equational Theorem Proving with Selection and Simplification, Journal of Logic and Computation 4 (1994) 217–247.
- [59] C. Lynch, Constructing Bachmair-Ganzinger Models, in: A. Voronkov, C. Weidenbach (Eds.), Programming Logics - Essays in Memory of Harald Ganzinger, number 7797 in Lecture Notes in Computer Science,

- Springer, Berlin, Germany, 2013, pp. 285–301.
- [60] R. Caferra, N. Zabel, A Method for Simultaneous Search for Refutations and Models by Equational Constraint Solving, *Journal of Symbolic Computation* 13 (1992) 613–641.
 - [61] R. Caferra, N. Peltier, Extending Semantic Resolution via Automated Model Building: Applications, in: C. Mellish (Ed.), *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Burlington, USA, 1995, pp. 328–334.
 - [62] R. Caferra, N. Peltier, Model Building and Interactive Theory Discovery, in: P. Baumgartner, R. Hähnle, J. Possega (Eds.), *Proceedings of the 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, number 918 in *Lecture Notes in Computer Science*, 1995, pp. 154–168.
 - [63] M. Bonacina, D. Plaisted, Semantically-guided Goal-sensitive Reasoning: Model Representation, *Journal of Automated Reasoning* 56 (2016) 113–141.
 - [64] R. Hähnle, Tableaux and Related Methods, in: A. Robinson, A. Voronkov (Eds.), *Handbook of Automated Reasoning*, Elsevier Science, Amsterdam, The Netherlands, 2001, pp. 100–178.
 - [65] P. Baumgartner, U. Furbach, I. Niemelä, Hyper Tableaux, in: J. Alferes, L. Pereira, E. Orłowska (Eds.), *Proceedings of JELIA’96: European Workshop on Logic in Artificial Intelligence*, number 1126 in *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Germany, 1996, pp. 1–17.
 - [66] C. Stickse, K. Korovin, A Note on Model Representation and Proof Extraction in the First-order Instantiation-based Calculus Inst-Gen, in: R. Schmidt, F. Papacchini (Eds.), *Proceedings of the 19th Automated Reasoning Workshop*, 2012, pp. 11–12.
 - [67] B. Pelzer, C. Wernhard, System Description: E-KRHyper, in: F. Pfenning (Ed.), *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in *Lecture Notes in Artificial Intelligence*, Springer, Berlin, Germany, 2007, pp. 508–513.
 - [68] K. Korovin, iProver - An Instantiation-based Theorem Prover for First-order Logic (System Description), in: P. Baumgartner, A. Armando, G. Dowek (Eds.), *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in *Lecture Notes in Artificial Intelligence*, 2008, pp. 292–298.
 - [69] L. Bachmair, H. Ganzinger, D. McAllester, C. Lynch, Resolution Theorem Proving, in: A. Robinson, A. Voronkov (Eds.), *Handbook of Automated Reasoning*, Elsevier Science, Amsterdam, The Netherlands, 2001, pp. 19–99.
 - [70] N. Peltier, Model Building with Ordered Resolution: Extracting Models from Saturated Clause Sets, *Journal of Symbolic Computation* 36 (2003) 5–48.
 - [71] M. Janota, M. Rawson, S. Schulz, Case Study: Saturations as Explicit Models in Equational Theories, 2026. 10.48550/arXiv.2602.16324.
 - [72] 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, Berlin, Germany, 2019, pp. 495–507.
 - [73] W. McCune, Prover9, ??? <http://www.cs.unm.edu/~mccune/prover9/>.
 - [74] P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, V. Nummelin, S. Tourret, Making Higher-order Superposition Work, in: A. Platzer, G. Sutcliffe (Eds.), *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 2021, pp. 415–432.
 - [75] J. Garson, Modal Logic, in: E. Zalta (Ed.), *Stanford Encyclopedia of Philosophy*, Stanford University Press, Stanford, USA, 2018.
 - [76] 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, Berlin, Germany, 2006, pp. 156–161.
 - [77] F. Horozal, F. Rabe, Formal Logic Definitions for Interchange Languages, 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, Berlin, Germany, 2015, pp. 171–186.
 - [78] A. Steen, Scala TPTP Parser v1.7.4, 2026. doi:{10.5281/zenodo.5578872}.
 - [79] 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, Berlin, Germany, 2007, pp. 6–22.
 - [80] 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}.
 - [81] T. Bräuner, Hybrid Logic and its Proof-Theory, number 37 in *Applied Logic Series*, Springer, Berlin, Germany, 2011.