The New TPTP Format for Interpretations

Geoff Sutcliffe[1], Alexander Steen[2], and Pascal Fontaine[3]

[1] University of Miami, Miami, USA
geoff@cs.miami.edu
[2] University of Greifswald, Greifswald, Germany
alexander.steen@uni-greifswald.de
[3] University of Liège, Liège, Belgium
Pascal.Fontaine@uliege.be

Abstract

This paper describes the new TPTP format for representing interpretations. It provides a background survey that helped us ensure that the representation format is adequate for different types of interpretations: Tarskian, Herbrand, and Kripke interpretations. The needs of applications that use models are considered. The syntax and semantics of the new format is expounded in detail, with multiple examples. Verification of models is discussed. Some tools that support processing the new format are noted. The properties of interpretations represented in the new format are discussed.

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 [47]. 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 [67]. In the last two decades the converse task of disproving conjectures, proving that a conjecture is not a theorem of the axioms, has become increasingly important. 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 [22], 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 [50], operations research [30], commonsense reasoning [35], and solving model finding problems [72].

The TPTP World [61] (www.tptp.org) is a well established infrastructure that supports research, development, and deployment of ATP systems. 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 [58], the TSTP solution library [59], tools and services for processing ATP problems and solutions [59], and it supports the CADE ATP System Competition (CASC) [60]. The TPTP language [62] is one of the keys to the success of the TPTP World. Originally the TPTP language supported only first-order clause normal form (CNF) [68]. Over the years full first-order form (FOF) [58], typed first-order form (TFF) [66, 10], typed extended first-order form (TXF) [64], typed higher-order form (THF) [63, 34], and non-classical forms (NXF, NHF) [52]. Most relevant to this work, the TPTP languages are used for writing ATP problems, derivations, and *interpretations* [67, 57].

A TPTP format for interpretations with finite domains has previously been defined [67], and has served the ATP community adequately for almost 20 years. The old format is output by several ATP systems, e.g., Paradox [19], FM-Darwin [6], Vampire [37]. Recently the need for a format for interpretations with infinite domains, and for a format for Kripke interpretations [38], has led to the development of the new TPTP format for interpretations. This work describes the new 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 [4] defines a format for model output, and commands to inspect models. SAT solvers generally output models as specified by the SAT competitions [33], in a simple format similar to the DIMACS input format [1]. Some individual model finding systems have defined their own formats for models, e.g., the output formats of Nitpick [9] and Z3 [21].

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 new format for Tarskian interpretations, and Section 5 does the same for Kripke interpretations. Section 6 concludes and discusses plans for future work.

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 [18], use as a subroutine in more complex reasoning, e.g., for axiom selection [65, 45], and establishing the existence of a bug in a verification process[1]. A key weakness of a model finder system 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 [50], identification of bugs in verification [20, 46], studying modal and description logics [48], solving problems encoded as a model finding problems [72], and evaluating formulae wrt the model [53]. Manual inspection of explicit models can also be useful, e.g., [24]. A key advantage of having an explicit model output is that the model can be verified, i.e., the model finder does not need to be trusted,

Given the innate desirability of obtaining explicit models of satisfiable formulae, desirable properties of interpretation representation can be considered.

- An interpretation should be for the symbols of the input formulae.
- Evaluation of any formulae wrt the interpretation should be possible, because many uses of interpretations can be reduced to evaluation [14, §3.2]. The evaluation should be tractable, or in a tractabe core. This corresponds to the *atom test* and *formula evaluation* postulates suggested by [25, 14].

[1] Bill 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.

- 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 tractabe core.
- Interpretations should be sufficiently (human) comprehensible to be useful in (human) applications.

2.2 What do we Have?

A *Tarskian interpretation* [69] consists of a non-empty domain of distinct elements (not the Herbrand Universe – see Herbrand interpretation 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 $\{true, false\}$ [32, 26]. An overview of some ways of building and representing Tarskian interpretations is provided by [14], and [42] provides a comprehensive list of approaches. A *complete* interpretation can be used to evaluate all formulae that can be written in the language of the problem formulae, but a *partial* interpretation can be used to evaluate the problem formulae but not necessarily all possible formulae [13]. Two types of Tarskian interpretations are clear (and more might exist) ...

- *Finite* Tarskian interpretations have only finite domains. The domain and symbol mappings can be 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 [19, 41] or function free [7] logic, and an ATP system for that logics 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.
- *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 [5]. There are some ATP systems that produce infinite Tarskian models, e.g., cvc5 [3], FEST [23], and Z3.

A *Herbrand interpretation* [29] 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 determines its set of Herbrand models. Formulae in Skolem normal form determine Herbrand models with a Herbrand universe that includes any Skolem symbols. An empty set of formulae determines 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 determines 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), a disjunction of implicit generalisations (DIGs) [39], sets of constrained unit clauses [17, 15, 16], SGGS clause sequences [11], tableaux [28] and hyper-tableaux [8], eq-interpretations [42], and predicate definitions over the term algebra [55]. E-KRHyper [44] was an ATP systems that produced hyper-tableaux, and iProver [36, 55] is an ATP system that outputs predicate definitions over the term algebra.
- *Saturations* [2, 43] 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. Saturation-based ATP systems include E [49], Prover9 [40], Vampire, and Zipperposition [71].

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* [38] 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 [27]. Within each world there is a Tarskian interpretation, each possibly having different domains. If *local terms* are assumed, the designation of a ground term in a world may be chosen from only the domain in that world. If *global terms* are assumed, the designation of a ground term in a world may be chosen from the domain in any world. Quantification is over the domains in the world of evaluation (*actualist quantification*), but the domains in the worlds can be chosen to coincide to simplify matters (*possibilist quantification*). Formulae can be evaluated wrt Kripke interpretations with finite worlds and local terms ...

- Evaluate modal operators using Kripke semantics.
- Evaluate formulae within a world wrt the Tarskian interpretation in the world.

The notions of interpretations, models, partial interpretations, finite interpretations, Herbrand interpretations, etc., are captured in the SZS ontologies [57], as updated at `www.tptp.org/cgi-bin/SeeTPTP?Category=Documents&File=SZSOntology`. For this work the ontology was extended with a new value *Model Preserving* (MPR), 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.

2.3 Do we Have what we Need?

The new TPTP format represents interpretations in *interpretation-formulae* – the details are provided in Sections 3, 4 and 5. Interpretation-formulae are written in an extension of the language of the formulae being interpreted, extending the vocabulary but still using TPTP syntax.

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 $\mathsf{Ax} \cup \{\sim\mathsf{C}\}$. 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 of the language of the input formula, e.g., by the addition of Skolem symbols, defined symbols, etc. (hence Σ^+). 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 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 model

preserving transformations (MPRs) to the satisfiable set, to produce formulae that determine Herbrand models of the satisfiable formulae. The satisfiable formulae can be proved from the sets that result from applying MPRs

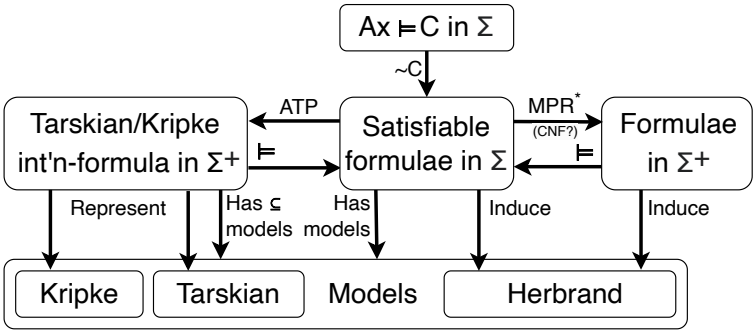

Figure 1: A Landscape of Model Building

To evaluate a formula wrt interpretation-formulae ...

- For Kripke interpretation-formulae, theorem proving can be used. This is described in Section 5.3.
- 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 boolean formulae using the truth tables for connectives.

  This approach is taken internally in Vampire.
- 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 ...
  - 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 all 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.

  I believe that this verification technique works, and so do Stephan Schulz and Uwe Waldmann, but Andrei Voronkov and Christoph Weidenbach say they do not. I'm looking for help on this.

2.4 Model Verification

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. Validate that the interpretation-formulae correctly represent the model found by the ATP system.
2. Verify that the type declarations and interpretation-formulae are syntactically well-formed and well-typed.
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 cannot be confirmed without insight into the ATP system that found the model.
2. This can be confirmed using standard parsing and type checking tools, e.g., [70, 31, 51].
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, the 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 untyped first-order logic in [53]. We believe the proof in [53] 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 5.3).

Steps 2 to 4 are implemented in the AGMV model verifier [53], available in the SystemOnTSTP [56] web interface `www.tptp.org/cgi-bin/SystemOnTSTP`.

3 The New Format for Tarskian Interpretations

A simple Tarskian interpretation-formula is a conjunction of components (see simple FOF and TFF examples in Appendices A.2, B.2, and B.4) ...

- A domain for each of the types 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*.

Note how function and predicate symbols are interpreted by applying them directly to domain elements (*à la* [26, §5.3.4]), rather than using the presentation of interpretations that

introduces a new interpretation function for each function/predicate symbol [26, §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 3.2).

Examples of single Tarskian interpretation-formulae are in Appendices A.2, B.4, C.2, C.3 and D.2, illustrating the components described below. Tarskian interpretations split over multiple interpretation-formulae, as in Appendices A.4, A.5, B.5, B.6, and D.4, are explained in Section 3.3. Tarskian interpretations that are compacted using quantification, as in B.7 and D.5, are explained in Section 3.4. Examples of interpretation-formulae that determine Herbrand interpretations, as in Appendices A.7, and A.6, are explained in Section 4.

3.1 FOF Tarskian Interpretations

FOF interpretation-formulae have a single untyped domain whose elements are new (not appearing in the problem) ground terms. The mappings apply the function and predicate symbols to the domain elements. Appendix A.1 shows a FOF problem that has a countermodel with a finite domain, which is shown in Appendix A.2. Appendix A.3 shows the verification problem for the model.

The distinctness of the domain elements must be made explicit, either with pairwise inequalities, or 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 Appendix A.2 ...

```
! [X] : ( X = "a" | X = "f" | X = "john" | X = "gotA")
```

The function mappings are equalities between functions applied to domain elements, e.g., in Appendix A.2 ...

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

and the predicate mappings are atoms, e.g., in Appendix A.2 ...

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

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 Appendix A.2 ...

```
grade_of("f") = "a"
~ created_equal("a","john")
```

3.2 TFF and THF Tarskian Interpretations

TFF and THF interpretation-formulae have a domain for each type in the problem. The domain types can reuse problem types, can be new user types, or can be defined types such as `$int`. If the domain types are different from the problem types then the function and predicate symbols from the problem cannot be applied directly to domain elements in the interpretation-formulae, because the domain elements are of the wrong type wrt the type declarations of the symbols. In this situation type-promotion bijections are used to "convert" domain elements to terms of the corresponding problem type. If the problem types are reused as the domain types (see [26, §5.3.4]) then type-promotion is not necessary.

Each domain specification is a conjunction of (following along in Appendices B.4, C.2, and C.3 might help here) ...

- The domain for each problem type, as a formula that makes the type-promotion function a surjection (unless it is unnecessary because the problem type and the domain type are the same defined type, e.g., both are `$int`), e.g., in Appendix B.4 ...

  ```
  ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
  ```

- Definition of the elements of the domain (unless implicit from their defined type). If the domain is finite this is a universally quantified disjunction of equalities whose right-hand sides are the terms, e.g., in Appendix B.4 ...
  ```
  ! [DC: d_cat]: ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
  ```
  If the domain is infinite this is an existentially quantified formula that captures an infinite disjunction of equalities, e.g., in Appendix C.3 ...
  ```
  ! [I: peano] : ( I = zero | ? [P: peano] : I = s(P) )
  ```
- Specification of the distinctness of the domain elements (unless implicit from their defined type), e.g., in Appendix B.4 ...
  ```
  $distinct(d_garfield,d_arlene,d_nermal)
  ```
- If type-promotion functions are needed, a formula making the type-promotion functions injections, which together with the surjectivity makes them bijections, e.g., in Appendix B.4 ...
  ```
  ! [DC1: d_cat,DC2: d_cat] :
     ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 )
  ```

The interpretation of symbols applies the corresponding type-promotion function to each domain element, e.g., in Appendix B.4 ...

```
owns(d2human(d_jon),d2cat(d_garfield))
```

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

- The types in the problem (except defined types).
- The types of the domains (except defined types).
- The types of the type-promotion functions.
- The types of the domain elements.

Examples:

- Appendix B.1 shows a TFF problem that has a countermodel with finite domains. The countermodel using separate domain types and type-promotion functions is shown in Appendix B.2, The comments show which parts of the interpretation-formula specify which part of the interpretation. Appendix B.3 shows the verification problem for that model. The countermodel that reuses the problem types as domain types is shown in in Appendix B.4.
- Appendix C.1 shows a TFF problem that has a model with an integer domain, which is shown in Appendix C.2. Note that • the defined type `$int` is the domain type for the formula type `person`, so that there is no specification of the domain elements and their distinctness; • universal quantification is used for the interpretation of function and predicate symbols to cover an infinite number of argument tuples; • the interpretation of function and predicate symbols is not given for argument tuples with negative integers, i.e., this is an example of a partial interpretation. Appendix C.4 shows the verification problem for the model.
- The problem in Appendix C.1 also has a model with an infinite term domain, which is shown in Appendix C.3. Appendix C.5 shows the verification problem for the model.
- Appendix D.1 shows a THF problem that has a countermodel with finite domains, which shown in Appendix D.2. Appendix D.3 shows the verification problem for the model.

3.3 Tarskian Interpretations Split over Multiple Formulae

The interpretation-formulae described thus far put all the information about an interpretation in a single interpretation-formula. In some situations it is useful to separate the various components, e.g., the domains could be separated from the symbol mappings. The new 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 `domain` and `mapping`. Two (or more) interpretation-formulae are used, each containing the corresponding parts of the interpretation. Appendices A.4 and B.5 show examples of splitting the interpretation-formulae in Appendices A.2 and B.4 respectively, which separate the domain specifications from the symbol

mappings, e.g., in Appendix B.4 the annotated formula ...

```
tff(garfield_domains,interpretation-domain,
```

contains the information about the two domains, and the annotated formula ...

```
tff(garfield_mappings,interpretation-mapping,
```

contains the symbol mappings. Note that each role and role-subrole pair can be used multiple times according to need, e.g., in Appendix A.4 there are two `interpretation-mapping` annotated 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 `domain` and `mapping` subroles given arguments to indicate which domain or symbol mapping is recorded. For `domain`s the arguments of the subrole are the domain in the problem and the domain in the interpretation. For `mapping`s the arguments are the symbol and its result domain type. Appendices A.5 and B.6 show examples of splitting the interpretation-formulae in Appendices A.2 and B.4 respectively, e.g., in Appendix B.4 the annotated formula ...

```
tff(garfield_domain_human,interpretation-domain(human,d_human),
```

contains the `d_human` domain specification, and ...

```
tff(garfield_domain_cat,interpretation-domain(cat,d_cat),
```

contains the `d_cat` domain specification.

The medium and fine grained splitting can be mixed. Appendix D.4 shows an example of mixed splitting of Appendix D.2, where the domains are specified separately in the annotated formulae ...

```
thf(hot_coffee_beverage,interpretation-domain(beverage,d_beverage),
```

and ...

```
thf(hot_coffee_syrup,interpretation-domain(syrup,d_syrup),
```

while all the symbol mappings are in ...

```
thf(hot_coffee,interpretation-mapping,
```

3.4 Tarskian Interpretations Compacted using Quantification

The full logical language can be used in interpretation-formulae to provide compaction. Appendices B.7 and D.5 show examples compacting the interpretation-formulae in Appendices A.2 and B.4 respectively. In Appendix B.7 note how universal quantification is used to map `loves` to `d_garfield` for all `d_cats` ...

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

and in Appendix D.5 universal quantification is used to say that for all functions of type `beverage > syrup` and all `syrups`, applying the function to `d_coffee` and the syrup results in `d_coffee` ...

```
! [F: beverage > syrup > beverage,S: syrup] :
  ( ( F @ ( d2beverage @ d_coffee ) @ ( d2syrup @ S ) )
  = ( d2beverage @ d_coffee ) )
```

4 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`. The problem in Appendix A.1 has a HI-formulae model formed from predicate definitions over the term algebra, which is shown in Appendix A.6. The problem in Appendix A.1 also has a saturation, which is shown in Appendix A.7. Appendices A.8 and A.9 show the corresponding verification problems.

5 Kripke Interpretations

The new 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. *I don't really have a firm handle on the cases with an infinite number of worlds, but I have some tentative examples that show they can be represented in the new format.*[2] *For now the format is explained with examples that have a finite number of worlds and finite Tarskian interpretations in the worlds.* 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 hybrid 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) > $o` is used to specify the accessibility relation between worlds.
- A defined constant `$local_world` of type `$world > $o` is used to specify the world in which a local (the default) conjecture is to be proved.
- A defined predicate `$in_world` of type `($world * $o) > $o` is used 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 world are known to be distinct, but no ATP systems know that yet).
- The accessibility relation.
- Specification of the local world if any.
- For each world, its Tarskian interpretation (also in the new TPTP format for interpretations), augmented by existential subfomulae that require the existence of the world's domain elements – these are necessary because the semantics is that of modal logic (see Sections 5.1 and 5.2 for examples).

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

- The types for the 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). 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).

5.1 Finite-Finite Kripke Interpretations

Appendix 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 Appendix E.2. The problem has global axioms and a local conjecture. The countermodel was found by embedding the problem into THF using the NTFLET tool [54, 51], running Nitpick on the THF problem, and converting Nitpick's output to the TPTP format by hand. There are two worlds ...

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

with a complete accessibility relation (meeting the requirements of modal logic **M**) ...

```
$accessible_world(w1,w1)
$accessible_world(w2,w2)
$accessible_world(w1,w2)
$accessible_world(w2,w1)
```

[2] www.tptp.org/TPTP/Proposals/InterpretationsModels.shtml

The conjecture is disproved in a local world ...

```
$local_world = w2
```

The finite Tarskian interpretations for the worlds are provided in the format described in Section 3 ...

```
$in_world(w1,the Tarskian interpretation)
$in_world(w2,the Tarskian interpretation)
```

Note the augmentation of the Tarskian interpretations with subformulae such as `?[DP:d_person]:(DP=d_alex)`. The quantification semantics is not classical, but instead that of modal logic, i.e., the quantification is over the domain elements that exist in `w1`, i.e., that subformulae requires that the domain element `d_lex` exists in `w1`. Appendix E.3 shows the verification problem for the model.

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

5.2 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 `world` and `in_world`. Appendix E.7 shows a medium grained split of Appendix E.2. An annotated formula with the role `interpretation-world` contains information about the worlds of the interpretation, e.g., in Appendix E.7 the annotated formula ...

```
tff(leo_workers_worlds,interpretation-world,
```

contains the information about the two worlds. Annotated formulae with the role `interpretation-in_world` provide information about the interpretation in the world named as the first argument of the `in_world` subrole. The second argument of the `in_world` subrole tells what information is provided, using the roles and subroles for Tarskian/Herbrand interpretation-formulae, e.g., in Appendix E.7 the annotated formula ...

```
tff(leo_workers_w1_domain,interpretation-in_world(w1,interpretation-domain),
```

contains the information about the domains in world `w1`, and the annotated formula ...

```
tff(leo_workers_w1_mappings,interpretation-in_world(w1,interpretation-mapping),
```

contains the information about the mappings in world `w1`.

Splitting can be done in a flexible way, using the fine-grained splitting available for Tarskian interpretation-formulae. Appendix E.8 shows a more fine grained split of Appendix E.2, in which the annotated formula ...

```
tff(leo_workers_w1_person,
  interpretation-in_world(w1,interpretation-domain(person,d_person)),
```

contains the information about the domain `d_person` in world `w1`, and the annotated formula ...

```
tff(leo_workers_w2_gets_rich,
  interpretation-in_world(w2,interpretation-mapping(gets_rich,$o)),
```

contains the information about the mappings for the `gets_rich` predicate in world `w2`.

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. Appendix E.9 shows a compacted version of Appendix E.2, in which the annotated formula ...

```
tff(leo_workers_W_product,
  interpretation-in_world(W,interpretation-domain),
```

contains the information about the domains that are the same in all worlds, and the annotated formula ...

```
tff(leo_workers_W_work_hard,
  interpretation-in_world(W,interpretation-mapping(work_hard,$o)),
```

contains the mappings for the predicate symbol `work_hard` that are the same in all worlds.

5.3 Kripke Model Verification

Kripke models written as interpretation-formulae can be verified using the theorem proving approach described in Section 2.4.[3] 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 the problems in Appendices E.1 and 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 [12], which is indicated by the special logic specification value `$$fomlModel`. Appendix E.3 shows the NXF verification problem for the problem in Appendix E.1 and its countermodel in Appendix E.2. Appendix E.6 shows the NXF verification problem for the problem in Appendix E.4 and its countermodel in Appendix E.5.

In the NXF verification problem the Kripke interpretation-formulae is 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, e.g., in Appendix E.1 the axiom of the problem ...

```
tff(not_all_get_rich,axiom,
  ~ ? [P: person] : ( {$necessary} @ (gets_rich(P)) ) ).
```

is global (the default), so the NXF verification conjecture in Appendix E.3 has the `global` subrole ...

```
tff(not_all_get_rich,conjecture-global,
  ~ ? [P: person] : ( {$necessary} @ (gets_rich(P)) ) ).
```

The conjecture of the problem is local (the default) ...

```
tff(only_alex_gets_rich,conjecture,
  {$possible} @ (gets_rich(alex) & ~ gets_rich(chris)) ).
```

so the NXF verification (negated) conjecture has the `local` subrole ...

```
tff(only_alex_gets_rich,conjecture-local,
  ~ ( {$possible} @ ( gets_rich(alex) & ~ gets_rich(chris) ) ) ).
```

The NXF verification problem is embedded into TH0 using the NTFLET tool. NTFLET recognizes the special logic specification and the new 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.

6 Conclusion

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

The new TPTP format provides very many 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 by `interpretation-domain`, and `fi_functors` and `fi_predicates` can be replaced by `interpretation-mapping`.

Section 2.1 listed four needs for interpretations:

- Evaluability - the ability to evaluate a formula wrt an interpretation.
- Tractability - the ability to evaluate a formula using some limited/reasonable amount of resources.
- Verifiability - the ability to verify a model by evaluating all the formulae it claims to model as *true*.
- Comprehensibility - be understandable, typically by a human but possibly by a machine.

[3] To date we have done this for NXF problem formulae and TXF interpretation-formulae. Extension to NHF might be possible.

In the light of the above (and hey, maybe there are more techniques than just those) I 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 varies 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 react to the format, which might lead to improvements, and hopefully will lead to them implementing the format in the ATP systems.

References

[1] D. Babic. Satisfiability Suggested Format. https://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf, 1993.

[2] L. Bachmair, H. Ganzinger, D. McAllester, and C. Lynch. Resolution Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier Science, 2001.

[3] 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, and Y. Zohar. cvc5: A Versatile and Industrial-Strength SMT Solver. In D. Fisman and G. Rosu, editors, *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, pages 415–442. Springer, 2022.

[4] C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. https://smtlib.cs.uiowa.edu, 2017.

[5] P. Baumgartner and J. Bax. Proving Infinite Satisfiability. In K. McMillan, A. Middeldorp, and A. Voronkov, editors, *Proceedings of the 19th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 8312 in Lecture Notes in Computer Science, pages 86–95. Springer-Verlag, 2013.

[6] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. In W. Ahrendt, P. Baumgartner, and H. de Nivelle, editors, *Proceedings of the 3rd Workshop on Disproving - Non-Theorems, Non-Validity, Non-Provability, 3rd International Joint Conference on Automated Reasoning*, 2006.

[7] P. Baumgartner, A. Fuchs, H. de Nivelle, and C. Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. *Journal of Applied Logic*, 7(1):58–74, 2009.

[8] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In J. Alferes, L. Pereira, and E. Orlowska, editors, *Proceedings of JELIA'96: European Workshop on Logic in Artificial Intelligence*, number 1126 in Lecture Notes in Artificial Intelligence, pages 1–17. Springer-Verlag, 1996.

[9] J. Blanchette and T. Nipkow. Nitpick: A Counterexample Generator for Higher-Order Logic Based on a Relational Model Finder. In M. Kaufmann and L. Paulson, editors, *Proceedings of the 1st International Conference on Interactive Theorem Proving*, number 6172 in Lecture Notes in Computer Science, pages 131–146. Springer-Verlag, 2010.

[10] J. Blanchette and A. Paskevich. TFF1: The TPTP Typed First-order Form with Rank-1 Polymorphism. In M.P. Bonacina, editor, *Proceedings of the 24th International Conference on Automated Deduction*, number 7898 in Lecture Notes in Artificial Intelligence, pages 414–420. Springer-Verlag, 2013.

[11] M.P. Bonacina and D. Plaisted. Semantically-guided Goal-sensitive Reasoning: Model Representation. *Journal of Automated Reasoning*, 56(2):113–141, 2016.

[12] T. Braüner. *Hybrid Logic and its Proof-Theory*. Number 37 in Applied Logic Series. Springer, 2011.

[13] M. Bromberger, S. Schwarz, and C. Weidenbach. Exploring Partial Models with SCL. In R. Piskac and A. Voronkov, editors, *Proceedings of the 24th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 94 in EPiC Series in Computing, pages 48–72. EasyChair Publications, 2023.

[14] R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building*. Number 31 in Applied Logic Series. Kluwer Academic Publishers, 2004.

[15] R. Caferra and N. Peltier. Extending Semantic Resolution via Automated Model Building: Applications. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 328–334. Morgan Kaufmann, 1995.

[16] R. Caferra and N. Peltier. Model Building and Interactive Theory Discovery. In P. Baumgartner, R. Hähnle, and J. Possega, editors, *Proceedings of the 4th International Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, number 918 in Lecture Notes in Computer Science, pages 154–168, 1995.

[17] R. Caferra and N. Zabel. A Method for Simultaneous Search for Refutations and Models by Equational Constraint Solving. *Journal of Symbolic Computation*, 13(6):613–641, 1992.

[18] V. Chaudhri and D. Inclezan. Representing States in a Biology Textbook. In L. Leora Morgenstern, T. Patkos, and R. Sloan, editors, *Proceedings of 12th International Symposium on Logical Formalizations of Commonsense Reasoning*. AAAI Press, 2015.

[19] K. Claessen and N. Sörensson. New Techniques that Improve MACE-style Finite Model Finding. In P. Baumgartner and C. Fermueller, editors, *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[20] E. Clarke and E. Emerson. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. In D. Kozen, editor, *Proceedings of the Workshop on Logics of Programs*, number 131 in Lecture Notes in Computer Science, pages 52–71. Springer-Verlag, 1982.

[21] L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In C. Ramakrishnan and J. Rehof, editors, *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, pages 337–340. Springer-Verlag, 2008.

[22] V. D'Silva, D. Kroening, and G. Weissenbacher. A Survey of Automated Techniques for Formal Software Verification. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.

[23] N. Elad, O. Padon, and S. Shoham. An Infinite Needle in a Finite Haystack: Finding Infinite Counter-Models in Deductive Verification. In D. Dreyer, editor, *Proceedings of the ACM on Programming Languages*, volume 8, pages 970–1000. ACM Press, 2024.

[24] M. Emmer, Z. Khasidashvili, K. Korovin, and A. Voronkov. Encoding Industrial Hardware Verification Problems into Effectively Propositional Logic. In R. Bloem and N. Sharygina, editors, *Proceedings of the 10th International Conference on Formal Methods in Computer-Aided Design*, pages 137–144. IEEE Press, 2010.

[25] C. Fermüller and A. Leitsch. Hyperresolution and Automated Model Building. *Journal of Logic and Computation*, 6(2):173–203, 1996.

[26] J. Gallier. *Logic for Computer Science - Foundations of Automatic Theorem Proving*. Dover Publications, 2015.

[27] J. Garson. Modal Logic. In E. Zalta, editor, *Stanford Encyclopedia of Philosophy*. Stanford University, 2018.

[28] R. Hähnle. Tableaux and Related Methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 100–178. Elsevier Science, 2001.

[29] 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.

[30] J.N. Hooker. New Methods for Computing Inferences in First-order Logic. *Annals of Operations Research*, 43:477–492, 1993.

[31] F. Horozal and F. Rabe. Formal Logic Definitions for Interchange Languages. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Proceedings of the International Conference on Intelligent Computer Mathematics*, number 9150 in Lecture Notes in Computer Science, pages 171–186. Springer-Verlag, 2015.

[32] G. Hunter. *Metalogic: An Introduction to the Metatheory of Standard First Order Logic*. University of California Press, 1996.

[33] M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon. The International SAT Solver Competitions. *AI Magazine*, 33(1):89–92, 2012.

[34] C. Kaliszyk, G. Sutcliffe, and F. Rabe. TH1: The TPTP Typed Higher-Order Form with Rank-1 Polymorphism. In P. Fontaine, S. Schulz, and J. Urban, editors, *Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning*, number 1635 in CEUR Workshop Proceedings, pages 41–55, 2016.

[35] R. Khardon and D. Roth. Reasoning with Models. *Artificial Intelligence*, 87(1-2):187–213, 1994.

[36] K. Korovin. iProver - An Instantiation-based Theorem Prover for First-order Logic (System Description). In P. Baumgartner, A. Armando, and G. Dowek, editors, *Proceedings of the 4th International Joint Conference on Automated Reasoning*, number 5195 in Lecture Notes in Artificial Intelligence, pages 292–298, 2008.

[37] L. Kovacs and A. Voronkov. First-Order Theorem Proving and Vampire. In N. Sharygina and H. Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification*, number 8044 in Lecture Notes in Artificial Intelligence, pages 1–35. Springer-Verlag, 2013.

[38] S. Kripke. Semantical Considerations on Modal Logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[39] J-L. Lassez and K. Marriott. Explicit Representation of Terms Defined by Counter Examples. *Journal of Automated Reasoning*, 3:301–317, 9187.

[40] W.W. McCune. Prover9. http://www.cs.unm.edu/ mccune/prover9/.

[41] W.W. McCune. Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA, 2003.

[42] N. Peltier. A Calculus Combining Resolution and Enumeration for Building Finite Models. *Journal of Symbolic Computation*, 36(1-2):49–77, 2003.

[43] N. Peltier. Model Building with Ordered Resolution: Extracting Models from Saturated Clause Sets. *Journal of Symbolic Computation*, 36(1-2):5–48, 2003.

[44] B. Pelzer and C. Wernhard. System Description: E-KRHyper. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 508–513. Springer-Verlag, 2007.

[45] P. Pudlak. Semantic Selection of Premises for Automated Theorem Proving. In J. Urban, G. Sutcliffe, and S. Schulz, editors, *Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories*, number 257 in CEUR Workshop Proceedings, pages 27–44, 2007.

[46] J. Queille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, number 137 in Lecture Notes in Computer Science, page 337–351. Springer-Verlag, 1982.

[47] A. Robinson and A. Voronkov. *Handbook of Automated Reasoning*. Elsevier Science, 2001.

[48] R. Schmidt and U. Hustadt. First-Order Resolution Methods for Modal Logics. In A. Voronkov and C. Weidenbach, editors, *Programming Logics, Essays in Memory of Harald Ganzinger*, number 7797 in Lecture Notes in Computer Science, pages 345–391. Springer-Verlag, 2013.

[49] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, Higher, Stronger: E 2.3. In P. Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[50] S. Schulz, G. Sutcliffe, J. Urban, and A. Pease. Detecting Inconsistencies in Large First-Order Knowledge Bases. In L. de Moura, editor, *Proceedings of the 26th International Conference on Automated Deduction*, number 10395 in Lecture Notes in Computer Science, pages 310–325. Springer-Verlag, 2017.

[51] A. Steen. tptp-utils v1.2.3, 2023. DOI: 10.5281/zenodo.7740507.

[52] A. Steen, D. Fuenmayor, T. Gleißner, G. Sutcliffe, and C. Benzmüller. Automated Reasoning in Non-classical Logics in the TPTP World. In B. Konev, C. Schon, and A. Steen, editors, *Proceedings of the 8th Workshop on Practical Aspects of Automated Reasoning*, number 3201 in CEUR Workshop Proceedings, page Online, 2022.

[53] A. Steen, G. Sutcliffe, P. Fontaine, and J. McKeown. Representation, Verification, and Visualization of Tarskian Interpretations for Typed First-order Logic. In R. Piskac and A. Voronkov, editors, *Proceedings of 24th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 94 in EPiC Series in Computing, pages 369–385. EasyChair Publications, 2023.

[54] Alexander Steen. An extensible logic embedding tool for lightweight non-classical reasoning (short paper). In B. Konev, C. Schon, and A. Steen, editors, *Proceedings of the 8th Workshop on Practical Aspects of Automated Reasoning*, number 3201 in CEUR Workshop Proceedings, page Online, 2022.

[55] C. Sticksel and K. Korovin. A Note on Model Representation and Proof Extraction in the First-order Instantiation-based Calculus Inst-Gen. In R. Schmidt and F. Papacchini, editors, *Proceedings of the 19th Automated Reasoning Workshop*, pages 11–12, 2012.

[56] G. Sutcliffe. TPTP, TSTP, CASC, etc. In V. Diekert, M. Volkov, and A. Voronkov, editors, *Proceedings of the 2nd International Symposium on Computer Science in Russia*, number 4649 in Lecture Notes in Computer Science, pages 6–22. Springer-Verlag, 2007.

[57] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *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, pages 38–49, 2008.

[58] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[59] G. Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In E. Clarke and A. Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.

[60] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[61] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[62] G. Sutcliffe. The Logic Languages of the TPTP World. *Logic Journal of the IGPL*, 31(6):1153–1169, 2023.

[63] G. Sutcliffe and C. Benzmüller. Automated Reasoning in Higher-Order Logic using the TPTP THF Infrastructure. *Journal of Formalized Reasoning*, 3(1):1–27, 2010.

[64] G. Sutcliffe and E. Kotelnikov. TFX: The TPTP Extended Typed First-order Form. In B. Konev, J. Urban, and S. Schulz, editors, *Proceedings of the 6th Workshop on Practical Aspects of Automated Reasoning*, number 2162 in CEUR Workshop Proceedings, pages 72–87, 2018.

[65] G. Sutcliffe and Y. Puzis. SRASS - a Semantic Relevance Axiom Selection System. In F. Pfenning, editor, *Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 295–310. Springer-Verlag, 2007.

[66] G. Sutcliffe, S. Schulz, K. Claessen, and P. Baumgartner. The TPTP Typed First-order Form with Arithmetic. In N. Bjørner and A. Voronkov, editors, *Proceedings of the 18th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 7180 in Lecture Notes in Artificial Intelligence, pages 406–419. Springer-Verlag, 2012.

[67] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81. Springer, 2006.

[68] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[69] A. Tarski and R. Vaught. Arithmetical Extensions of Relational Systems. *Compositio Mathematica*, 13:81–102, 1956.

[70] A. Van Gelder and G. Sutcliffe. Extending the TPTP Language to Higher-Order Logic with Automated Parser Generation. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 156–161. Springer-Verlag, 2006.

[71] P. Vukmirović, A. Bentkamp, J. Blanchette, S. Cruanes, V. Nummelin, and S. Tourret. Making Higher-order Superposition Work. In A. Platzer and G. Sutcliffe, editors, *Proceedings of the 28th International Conference on Automated Deduction*, number 12699 in Lecture Notes in Computer Science, pages 415–432. Springer-Verlag, 2021.

[72] S. Winker. Generation and Verification of Finite Models and Counterexamples Using an Automated Theorem Prover Answering Two Open Questions. *Journal of the ACM*, 29(2):273–284, 1982.

A FOF

A.1 FOF Problem with a Finite Countermodel (Online)

```
%------------------------------------------------------------------------------
%----All (hu)men are created equal. John is a human. John got an F grade.
%----There is someone (a human) who got an A grade. An A grade is not
%----equal to an F grade. Grades are not human. Therefore, it is not the
%----case being created equal is the same as really being equal.

fof(all_created_equal,axiom,
    ! [H1,H2] :
      ( ( human(H1)
        & human(H2) )
     => created_equal(H1,H2) ) ).

fof(john,axiom,
    human(john) ).

fof(john_failed,axiom,
    grade_of(john) = f ).

fof(someone_got_an_a,axiom,
    ? [H] :
      ( human(H)
      & grade_of(H) = a ) ).

fof(distinct_grades,axiom,
    a != f ).

fof(grades_not_human,axiom,
    ! [G] : ~ human(grade_of(G)) ).

fof(equality_lost,conjecture,
    ! [H1,H2] :
      ( ( human(H1)
        & human(H2)
        & created_equal(H1,H2) )
    <=> H1 = H2 ) ).
%------------------------------------------------------------------------------
```

A.2 FOF Finite Model for A.1, Coarse Grained (Online)

```
%--------------------------------------------------------------------------
fof(equality_lost,interpretation,
    ( ! [X] : ( X = "a" | X = "f" | X = "john" | X = "gotA")
    & ( a = "a"
      & f = "f"
      & john = "john"
      & grade_of("a") = "a"
      & grade_of("f") = "a"
      & grade_of("john") = "f"
      & grade_of("gotA") = "a" )
    & ( ~ human("a")
      & ~ human("f")
      & human("john")
      & human("gotA")
      & ~ created_equal("a","a")
      & ~ created_equal("a","f")
      & ~ created_equal("a","john")
      & ~ created_equal("a","gotA")
      & ~ created_equal("f","a")
      & ~ created_equal("f","f")
      & ~ created_equal("f","john")
      & ~ created_equal("f","gotA")
      & ~ created_equal("john","a")
      & ~ created_equal("john","f")
      & created_equal("john","john")
      & created_equal("john","gotA")
      & ~ created_equal("gotA","a")
      & ~ created_equal("gotA","f")
      & created_equal("gotA","john")
      & created_equal("gotA","gotA") ) ) ).
%--------------------------------------------------------------------------
```

A.3 FOF Verification Problem for A.1 and A.2 (Online)

```
%------------------------------------------------------------------------------
fof(equality_lost,axiom,
    ( ! [X] : ( X = "a" | X = "f" | X = "john" | X = "gotA" )
    & a = "a"
    & f = "f"
    & john = "john"
    & grade_of("a") = "a"
    & grade_of("f") = "a"
    & grade_of("john") = "f"
    & grade_of("gotA") = "a"
    & ~ human("a")
    & ~ human("f")
    & human("john")
    & human("gotA")
    & ~ created_equal("a","a")
    & ~ created_equal("a","f")
    & ~ created_equal("a","john")
    & ~ created_equal("a","gotA")
    & ~ created_equal("f","a")
    & ~ created_equal("f","f")
    & ~ created_equal("f","john")
    & ~ created_equal("f","gotA")
    & ~ created_equal("john","a")
    & ~ created_equal("john","f")
    & created_equal("john","john")
    & created_equal("john","gotA")
    & ~ created_equal("gotA","a")
    & ~ created_equal("gotA","f")
    & created_equal("gotA","john")
    & created_equal("gotA","gotA") ) ).

fof(prove_formulae,conjecture,
    ( ! [H1,H2] :
        ( ( human(H1)
          & human(H2) )
       => created_equal(H1,H2) )
    & human(john)
    & grade_of(john) = f
    & ? [H] :
        ( human(H)
        & grade_of(H) = a )
    & a != f
    & ! [G] : ~ human(grade_of(G))
    & ~ ! [H1,H2] :
          ( ( human(H1)
            & human(H2)
            & created_equal(H1,H2) )
          <=> H1 = H2 ) ) ).
%------------------------------------------------------------------------------
```

A.4 FOF Finite Model for A.1, Medium Grained (Online)

```
%--------------------------------------------------------------------------
fof(equality_lost_domain,interpretation-domain,
    ! [X] : ( X = "a" | X = "f" | X = "john" | X = "gotA") ).

fof(equality_lost_term_mappings,interpretation-mapping,
    ( a = "a"
    & f = "f"
    & john = "john"
    & grade_of("a") = "a"
    & grade_of("f") = "a"
    & grade_of("john") = "f"
    & grade_of("gotA") = "a" ) ).

fof(equality_lost_predicate_mappings,interpretation-mapping,
    ( ~ human("a")
    & ~ human("f")
    & human("john")
    & human("gotA") ).
    & ~ created_equal("a","john")
    & ~ created_equal("a","gotA")
    & ~ created_equal("f","john")
    & ~ created_equal("f","gotA")
    & ~ created_equal("john","a")
    & ~ created_equal("john","f")
    & ~ created_equal("gotA","a")
    & ~ created_equal("gotA","f")
    & ~ created_equal("a","a")
    & ~ created_equal("a","f")
    & ~ created_equal("f","a")
    & ~ created_equal("f","f")
    & created_equal("john","john")
    & created_equal("john","gotA")
    & created_equal("gotA","john")
    & created_equal("gotA","gotA") ).
%--------------------------------------------------------------------------
```

A.5 FOF Finite Model for A.1, Fine Grained (Online)

```
%----------------------------------------------------------------------------
fof(equality_lost_domain,interpretation-domain($i,$i),
    ! [X] : ( X = "a" | X = "f" | X = "john" | X = "gotA") ).

fof(equality_lost_a,interpretation-mapping(a,$i),
    a = "a" ).

fof(equality_lost_f,interpretation-mapping(f,$i),
    f = "f" ).

fof(equality_lost_john,interpretation-mapping(john,$i),
    john = "john" ).

fof(equality_lost_grade_of,interpretation-mapping(grade_of,$i),
    ( grade_of("a") = "a"
    & grade_of("f") = "a"
    & grade_of("john") = "f"
    & grade_of("gotA") = "a" ) ).

fof(equality_lost_human,interpretation-mapping(human,$o),
    ( ~ human("a")
    & ~ human("f")
    & human("john")
    & human("gotA") ).

fof(equality_lost_created_equal,interpretation-mapping(created_equal,$o),
    ( ~ created_equal("a","john")
    & ~ created_equal("a","gotA")
    & ~ created_equal("f","john")
    & ~ created_equal("f","gotA")
    & ~ created_equal("john","a")
    & ~ created_equal("john","f")
    & ~ created_equal("gotA","a")
    & ~ created_equal("gotA","f") ).
    & ~ created_equal("a","a")
    & ~ created_equal("a","f")
    & ~ created_equal("f","a")
    & ~ created_equal("f","f")
    & created_equal("john","john")
    & created_equal("john","gotA")
    & created_equal("gotA","john")
    & created_equal("gotA","gotA") ).
%----------------------------------------------------------------------------
```

A.6 FOF Formula Model for A.1 (Online)

```
%----------------------------------------------------------------------------
fof(created_equal,interpretation-herbrand,
    ! [X0,X1] : ( created_equal(X0,X1) <=> $true ) ).

fof(human,interpretation-herbrand,
    ! [X0] : ( human(X0) <=> ( X0 != "f" & X0 != "a" ) ) ).

fof(john,interpretation-herbrand,
    ! [X0] : ( X0 = john <=> X0 = "john" ) ).

fof(grade_of,interpretation-herbrand,
    ! [X0,X1] :
      ( X0 = grade_of(X1)
    <=> ( ( X0 = "f" & X1 != "gotA" )
        | ( X0 = "a" & X1 = "gotA" ) ) ) ).

fof(f,interpretation-herbrand,
    ! [X0] : ( X0 = f <=> X0 = "f" ) ).

fof(a,interpretation-herbrand,
    ! [X0] : ( X0 = a <=> X0 = "a" ) ).
%----------------------------------------------------------------------------
```

A.7 FOF Saturation for A.1 (Online)

```
%------------------------------------------------------------------------------
cnf(c_0_15,interpretation-herbrand,
    ( created_equal(X1,X2) | ~ human(X1) | ~ human(X2) ) ).

cnf(c_0_16,interpretation-herbrand, ~ human(grade_of(X1)) ).

cnf(c_0_17,interpretation-herbrand, grade_of(esk3_0) = a ).

cnf(c_0_18,interpretation-herbrand, grade_of(john) = f ).

cnf(c_0_20,interpretation-herbrand,
    ( esk1_0 != esk2_0 | ~ human(esk1_0) | ~ human(esk2_0) ) ).

cnf(c_0_21,interpretation-herbrand,
    ( created_equal(esk1_0,esk2_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_22,interpretation-herbrand, ( human(esk2_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_23,interpretation-herbrand, ( human(esk1_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_24,interpretation-herbrand, ~ human(a) ).

cnf(c_0_25,interpretation-herbrand, ~ human(f) ).

cnf(c_0_26,interpretation-herbrand, a != f ).

cnf(c_0_27,interpretation-herbrand, human(esk3_0) ).

cnf(c_0_28,interpretation-herbrand, human(john) ).
%------------------------------------------------------------------------------
```

A.8 FOF Verification Problem for A.1 and A.6 (Online)

```
%--------------------------------------------------------------------------
fof(created_equal,axiom,
    ! [X0,X1] : ( created_equal(X0,X1) <=> $true ) ).

fof(human,axiom,
    ! [X0] : ( human(X0) <=> ( X0 != "f" & X0 != "a" ) ) ).

fof(john,axiom,
    ! [X0] : ( X0 = john <=> X0 = "john" ) ).

fof(grade_of,axiom,
    ! [X0,X1] :
      ( X0 = grade_of(X1)
    <=> ( ( X0 = "f" & X1 != "gotA" )
        | ( X0 = "a" & X1 = "gotA" ) ) ) ).

fof(f,axiom,
    ! [X0] : ( X0 = f <=> X0 = "f" ) ).

fof(a,axiom,
    ! [X0] : ( X0 = a <=> X0 = "a" ) ).

fof(prove_formulae,conjecture,
    ( ! [H1,H2] :
        ( ( human(H1)
          & human(H2) )
       => created_equal(H1,H2) )
    & human(john)
    & grade_of(john) = f
    & ? [H] :
        ( human(H)
        & grade_of(H) = a )
    & a != f
    & ! [G] : ~ human(grade_of(G))
    & ~ ! [H1,H2] :
          ( ( human(H1)
            & human(H2)
            & created_equal(H1,H2) )
        <=> H1 = H2 ) ) ).
%--------------------------------------------------------------------------
```

A.9 FOF Verification Problem for A.1 and A.7 (Online)

```
%----------------------------------------------------------------------------
cnf(c_0_15,axiom, ( created_equal(X1,X2) | ~ human(X1) | ~ human(X2) ) ).

cnf(c_0_16,axiom, ~ human(grade_of(X1)) ).

cnf(c_0_17,axiom, grade_of(esk3_0) = a ).

cnf(c_0_18,axiom, grade_of(john) = f ).

cnf(c_0_20,axiom, ( esk1_0 != esk2_0 | ~ human(esk1_0) | ~ human(esk2_0) ) ).

cnf(c_0_21,axiom, ( created_equal(esk1_0,esk2_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_22,axiom, ( human(esk2_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_23,axiom, ( human(esk1_0) | esk1_0 = esk2_0 ) ).

cnf(c_0_24,axiom, ~ human(a) ).

cnf(c_0_25,axiom, ~ human(f) ).

cnf(c_0_26,axiom, a != f ).

cnf(c_0_27,axiom, human(esk3_0) ).

cnf(c_0_28,axiom, human(john) ).

fof(prove_formulae,conjecture,
    ( ! [H1,H2] :
        ( ( human(H1)
          & human(H2) )
       => created_equal(H1,H2) )
    & human(john)
    & grade_of(john) = f
    & ? [H] :
        ( human(H)
        & grade_of(H) = a )
    & a != f
    & ! [G] : ~ human(grade_of(G))
    & ~ ! [H1,H2] :
          ( ( human(H1)
            & human(H2)
            & created_equal(H1,H2) )
        <=> H1 = H2 ) ) ).
%----------------------------------------------------------------------------
```

B TF0, Finite Interpretations

B.1 TF0 Problem with a Finite Countermodel (Online)

```
%----------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

tff(only_jon,axiom, ! [H: human] : H = jon ).

tff(only_garfield_and_arlene_and_nermal,axiom,
    ! [C: cat] :
      ( C = garfield | C = arlene | C = nermal ) ).

tff(distinct_cats,axiom,
    ( garfield != arlene & arlene != nermal
    & nermal != garfield ) ).

tff(jon_owns_garfield_not_arlene,axiom,
    ( owns(jon,garfield) & ~ owns(jon,arlene) ) ).

tff(all_cats_love_garfield,axiom,
    ! [C: cat] : ( loves(C) = garfield ) ).

tff(jon_owns_garfields_lovers,conjecture,
    ! [C: cat] :
      ( ( loves(C) = garfield & C != arlene )
       => owns(jon,C) ) ).
%----------------------------------------------------------------------------
```

B.2 TF0 Finite Model for B.1, Separate Domain Types (Online)

```
%------------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

%----Types of the domains
tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
%----Types of the promotion functions
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
%----Types of the domain elements
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield,interpretation,
%----The domain for human is d_human
    ( ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
%----The d_human elements are {d_jon}
      & ! [DH: d_human] : ( DH = d_jon )
%----The type-promoter is a bijection
      & ! [DH1: d_human,DH2: d_human] :
          ( d2human(DH1) = d2human(DH2) => DH1 = DH2 )
%----The domain for cat is d_cat
      & ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
%----The d_cat elements are {d_garfield,d_arlene,d_nermal}
      & ! [DC: d_cat]:
          ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
      & $distinct(d_garfield,d_arlene,d_nermal)
%----The type-promoter is a bijection
      & ! [DC1: d_cat,DC2: d_cat] :
          ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) )
%----Interpret terms via the type-promoted domain
    & ( jon = d2human(d_jon)
      & garfield = d2cat(d_garfield)
      & arlene = d2cat(d_arlene)
      & nermal = d2cat(d_nermal)
      & loves(d2cat(d_garfield)) = d2cat(d_garfield)
      & loves(d2cat(d_arlene)) = d2cat(d_garfield)
      & loves(d2cat(d_nermal)) = d2cat(d_garfield) )
%----Interpret atoms as true or false
    & ( owns(d2human(d_jon),d2cat(d_garfield))
      & ~ owns(d2human(d_jon),d2cat(d_arlene))
      & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ) ).
```

```
%----------------------------------------------------------------------------
```

B.3 TF0 Verification Problem for B.1, Separate Domain Types (Online)

```
%----------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield,axiom,
    ( ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
      & ! [DH: d_human] : ( DH = d_jon )
      & ! [DH1: d_human,DH2: d_human] :
          ( d2human(DH1) = d2human(DH2) => DH1 = DH2 )
      & ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
      & ! [DC: d_cat]:
          ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
      & $distinct(d_garfield,d_arlene,d_nermal)
      & ! [DC1: d_cat,DC2: d_cat] :
          ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) )
    & ( jon = d2human(d_jon)
      & garfield = d2cat(d_garfield)
      & arlene = d2cat(d_arlene)
      & nermal = d2cat(d_nermal)
      & loves(d2cat(d_garfield)) = d2cat(d_garfield)
      & loves(d2cat(d_arlene)) = d2cat(d_garfield)
      & loves(d2cat(d_nermal)) = d2cat(d_garfield) )
    & ( owns(d2human(d_jon),d2cat(d_garfield))
      & ~ owns(d2human(d_jon),d2cat(d_arlene))
      & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ) ).

tff(verify,conjecture,
    ( ! [H: human] : H = jon
    & ! [C: cat] : ( C = garfield | C = arlene | C = nermal )
    & ( garfield != arlene & garfield != nermal & arlene != nermal )
    & ( owns(jon,garfield) & ~ owns(jon,arlene) )
    & ! [C: cat] : ( loves(C) = garfield )
    & ~ ! [C: cat] :
          ( ( loves(C) = garfield & C != arlene)
         => owns(jon,C) ) ) ).
```

```
%--------------------------------------------------------------------------
```

B.4 TF0 Finite Model for B.1, Reusing Problem Types (Online)

```
%------------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

%----Types of the domains
tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
%----Types of the promotion functions
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
%----Types of the domain elements
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield,interpretation,
%----The domain for human is d_human
    ( ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
%----The d_human elements are {d_jon}
      & ! [DH: d_human] : ( DH = d_jon )
%----The type-promoter is a bijection
      & ! [DH1: d_human,DH2: d_human] :
          ( d2human(DH1) = d2human(DH2) => DH1 = DH2 )
%----The domain for cat is d_cat
      & ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
%----The d_cat elements are {d_garfield,d_arlene,d_nermal}
      & ! [DC: d_cat]:
          ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
      & $distinct(d_garfield,d_arlene,d_nermal)
%----The type-promoter is a bijection
      & ! [DC1: d_cat,DC2: d_cat] :
          ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) )
%----Interpret terms via the type-promoted domain
    & ( jon = d2human(d_jon)
      & garfield = d2cat(d_garfield)
      & arlene = d2cat(d_arlene)
      & nermal = d2cat(d_nermal)
      & loves(d2cat(d_garfield)) = d2cat(d_garfield)
      & loves(d2cat(d_arlene)) = d2cat(d_garfield)
      & loves(d2cat(d_nermal)) = d2cat(d_garfield) )
%----Interpret atoms as true or false
    & ( owns(d2human(d_jon),d2cat(d_garfield))
      & ~ owns(d2human(d_jon),d2cat(d_arlene))
      & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ) ).
```

```
%--------------------------------------------------------------------------
```

B.5 TF0 Finite Model for B.1, Medium Grained (Online)

```
%----------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield_domains,interpretation-domain,
    ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
    & ! [DH: d_human] : ( DH = d_jon )
    & ! [DH1: d_human,DH2: d_human] :
        ( d2human(DH1) = d2human(DH2) => DH1 = DH2 )
    & ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
    & ! [DC: d_cat]:
        ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
    & $distinct(d_garfield,d_arlene,d_nermal)
    & ! [DC1: d_cat,DC2: d_cat] :
        ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) ) ).

tff(garfield_mappings,interpretation-mapping,
    ( ( jon = d2human(d_jon)
      & garfield = d2cat(d_garfield)
      & arlene = d2cat(d_arlene)
      & nermal = d2cat(d_nermal)
      & loves(d2cat(d_garfield)) = d2cat(d_garfield)
      & loves(d2cat(d_arlene)) = d2cat(d_garfield)
      & loves(d2cat(d_nermal)) = d2cat(d_garfield) )
    & ( owns(d2human(d_jon),d2cat(d_garfield))
      & ~ owns(d2human(d_jon),d2cat(d_arlene))
      & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ) ).
%----------------------------------------------------------------------------
```

B.6 TF0 Finite Model for B.1, Fine Grained (Online)

```
%--------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield_domain_human,interpretation-domain(human,d_human),
    ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
    & ! [DH: d_human] : ( DH = d_jon )
    & ! [DH1: d_human,DH2: d_human] :
        ( d2human(DH1) = d2human(DH2) => DH1 = DH2 ) ) ).

tff(garfield_domain_cat,interpretation-domain(cat,d_cat),
    ( ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
    & ! [DC: d_cat]:
        ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
    & $distinct(d_garfield,d_arlene,d_nermal)
    & ! [DC1: d_cat,DC2: d_cat] :
        ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) ) ).

tff(garfield_mapping_jon,interpretation-mapping(jon,d_human),
    jon = d2human(d_jon) ).

tff(garfield_mapping_garfield,interpretation-mapping(garfied,d_cat),
    garfield = d2cat(d_garfield) ).

tff(garfield_mapping_arlene,interpretation-mapping(arlene,d_cat),
    arlene = d2cat(d_arlene) ).

tff(garfield_mapping_nermal,interpretation-mapping(nermal,d_cat),
    nermal = d2cat(d_nermal) ).

tff(garfield_mapping_loves,interpretation-mapping(loves,d_cat),
    ( loves(d2cat(d_garfield)) = d2cat(d_garfield)
    & loves(d2cat(d_arlene)) = d2cat(d_garfield)
    & loves(d2cat(d_nermal)) = d2cat(d_garfield) ) ).

tff(garfield_mapping_owns,interpretation-mapping(owns,$o),
```

```
    ( owns(d2human(d_jon),d2cat(d_garfield))
    & ~ owns(d2human(d_jon),d2cat(d_arlene))
    & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ).
%----------------------------------------------------------------------------
```

B.7 TF0 Finite Model for B.1, Compacted (Online)

```
%----------------------------------------------------------------------------
tff(human_type,type,      human: $tType ).
tff(cat_type,type,        cat: $tType ).
tff(jon_decl,type,        jon: human ).
tff(garfield_decl,type,   garfield: cat ).
tff(arlene_decl,type,     arlene: cat ).
tff(nermal_decl,type,     nermal: cat ).
tff(loves_decl,type,      loves: cat > cat ).
tff(owns_decl,type,       owns: ( human * cat ) > $o ).

tff(d_human_type,type,    d_human: $tType ).
tff(d_cat_type,type,      d_cat: $tType ).
tff(d2human_decl,type,    d2human: d_human > human ).
tff(d2cat_decl,type,      d2cat: d_cat > cat ).
tff(d_jon_decl,type,      d_jon: d_human ).
tff(d_garfield_decl,type, d_garfield: d_cat ).
tff(d_arlene_decl,type,   d_arlene: d_cat ).
tff(d_nermal_decl,type,   d_nermal: d_cat ).

tff(garfield_domain_human,interpretation-domain(human,d_human),
    ( ! [H: human] : ? [DH: d_human] : H = d2human(DH)
    & ! [DH: d_human] : ( DH = d_jon )
    & ! [DH1: d_human,DH2: d_human] :
        ( d2human(DH1) = d2human(DH2) => DH1 = DH2 ) ) ).

tff(garfield_domain_cat,interpretation-domain(cat,d_cat),
    ( ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
    & ! [DC: d_cat]:
        ( DC = d_garfield | DC = d_arlene | DC = d_nermal )
    & $distinct(d_garfield,d_arlene,d_nermal)
    & ! [DC1: d_cat,DC2: d_cat] :
        ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) ) ).

tff(garfield_mapping_jon,interpretation-mapping(jon,d_human),
    jon = d2human(d_jon) ).

tff(garfield_mapping_cats,interpretation-mapping,
    ( garfield = d2cat(d_garfield)
    & arlene = d2cat(d_arlene)
    & nermal = d2cat(d_nermal) ) ).

tff(garfield_mapping_loves,interpretation-mapping(loves,d_cat),
    ! [DC: d_cat] : ( loves(d2cat(DC)) = d2cat(d_garfield) ) ).

tff(garfield_mapping_owns,interpretation-mapping(owns,$o),
    ( owns(d2human(d_jon),d2cat(d_garfield))
    & ~ owns(d2human(d_jon),d2cat(d_arlene))
    & ~ owns(d2human(d_jon),d2cat(d_nermal)) ) ).
%----------------------------------------------------------------------------
```

C TF0, Infinite Interpretations

C.1 TF0 Axioms with an Infinite Model (Online)

```
%----------------------------------------------------------------------------
tff(person_type,type,        person: $tType ).
tff(bob_decl,type,           bob: person ).
tff(child_of_decl,type,      child_of: person > person ).
tff(is_descendant_decl,type, is_descendant: (person * person) > $o ).

tff(descendents_different,axiom,
    ! [A: person,D: person] :
      ( is_descendant(A,D) => ( A != D ) ) ).

tff(descendent_transitive,axiom,
    ! [A: person,C: person,G: person] :
      ( ( is_descendant(A,C) & is_descendant(C,G) )
     => is_descendant(A,G) ) ).

tff(child_is_descendant,axiom,
    ! [P: person] : is_descendant(P,child_of(P)) ).

tff(all_have_child,axiom,
    ! [P: person] : ? [C: person] : C = child_of(P) ).
%----------------------------------------------------------------------------
```

C.2 TF0 Infinite Model for C.1, Integer Domain (Online)

```
%----------------------------------------------------------------------------
tff(person_type,type,        person: $tType ).
tff(bob_decl,type,           bob: person ).
tff(child_of_decl,type,      child_of: person > person ).
tff(is_descendant_decl,type, is_descendant: ( person * person ) > $o ).

tff(int2person_decl,type,    int2person: $int > person ).

tff(people,interpretation,
%----Domain for type person is the integers
    ( ( ! [P: person] : ? [I: $int] : int2person(I) = P
%----The type promoter is a bijection (injective and surjective)
      & ! [I1: $int,I2: $int] :
          ( int2person(I1) = int2person(I2) => I1 = I2 ) )
%----Mapping people to integers. Note that Bob's ancestors will be interpreted
%----as negative integers.
    & ( bob = int2person(0)
      & ! [I: $int] : child_of(int2person(I)) = int2person($sum(I,1)) )
%----Interpretation of descendancy
    & ! [A: $int,D: $int] :
        ( is_descendant(int2person(A),int2person(D)) <=> $less(A,D) ) ) ).
%----------------------------------------------------------------------------
```

C.3 TF0 Infinite Model for C.1, Term Domain (Online)

```
%----------------------------------------------------------------------------
tff(person_type,type,         person: $tType ).
tff(bob_decl,type,            bob: person ).
tff(child_of_decl,type,       child_of: person > person ).
tff(is_descendant_decl,type,  is_descendant: ( person * person ) > $o ).

tff(peano_type,type,          peano: $tType).
tff(zero_decl,type,           zero: peano ).
tff(s_decl,type,              s: peano > peano ).
tff(peano2person_decl,type,   peano2person: peano > person ).
tff(peano_less_decl,type,     peano_less: ( peano * peano ) > $o ).

tff(people,interpretation,
%----Domain for type person is the Peano numbers
    ( ( ! [P: person] : ? [I: peano] : ( P = peano2person(I) )
      & ! [I: peano] : ( I = zero | ? [P: peano] : I = s(P) )
%----The type promoter is a bijection (injective and surjective)
      & ! [I1: peano,I2: peano] :
          ( peano2person(I1) = peano2person(I2) => I1 = I2 )
%----Relationships between Peano numbers
      & ! [I1: peano,I2: peano,I3: peano] :
          ( peano_less(I1,s(I1))
          & ( ( peano_less(I1,I2) & peano_less(I2,I3) )
           => peano_less(I1,I3) )
          & ( peano_less(I1,I2)
           => I1 != I2 ) ) )
%----Mapping people to Peano numbers
    & ( bob = peano2person(zero)
      & ! [I: peano] :
          child_of(peano2person(I)) = peano2person(s(I)) )
%----Interpretation of descendancy
    & ( ! [A: peano,D: peano] :
          ( is_descendant(peano2person(A),peano2person(D))
        <=> peano_less(A,D) ) ) ) ).
%----------------------------------------------------------------------------
```

C.4 TF0 Verification Problem for C.1 and C.2 (Online)

```
%--------------------------------------------------------------------------
tff(person_type,type,        person: $tType ).
tff(bob_decl,type,           bob: person ).
tff(child_of_decl,type,      child_of: person > person ).
tff(is_descendant_decl,type, is_descendant: ( person * person ) > $o ).

tff(int2person_decl,type,    int2person: $int > person ).

tff(people,axiom,
    ( ( ! [P: person] : ? [I: $int] : int2person(I) = P
      & ! [I1: $int,I2: $int] :
          ( int2person(I1) = int2person(I2) => I1 = I2 ) )
    & ( bob = int2person(0)
      & ! [I: $int] : child_of(int2person(I)) = int2person($sum(I,1)) )
    & ! [A: $int,D: $int] :
        ( is_descendant(int2person(A),int2person(D)) <=> $less(A,D) ) ) ).

tff(prove_formulae,conjecture,
    ( ! [A: person,D: person] : ( is_descendant(A,D) => A != D )
    & ! [A: person,C: person,G: person] :
        ( ( is_descendant(A,C) & is_descendant(C,G) ) => is_descendant(A,G) )
    & ! [P: person] : is_descendant(P,child_of(P))
    & ! [P: person] : ? [C: person] : C = child_of(P) ) ).
%--------------------------------------------------------------------------
```

C.5 TF0 Verification Problem for C.1 and C.3 (Online)

```
%--------------------------------------------------------------------------
tff(person_type,type,         person: $tType ).
tff(bob_decl,type,            bob: person ).
tff(child_of_decl,type,       child_of: person > person ).
tff(is_descendant_decl,type,  is_descendant: ( person * person ) > $o ).

tff(peano_type,type,          peano: $tType).
tff(zero_decl,type,           zero: peano ).
tff(s_decl,type,              s: peano > peano ).
tff(peano2person_decl,type,peano2person: peano > person ).
tff(peano_less_decl,type,     peano_less: ( peano * peano ) > $o ).

tff(people,axiom,
    ( ( ! [P: person] : ? [I: peano] : ( P = peano2person(I) )
      & ! [I: peano] : ( I = zero | ? [P: peano] : I = s(P) )
      & ! [I1: peano,I2: peano] :
          ( peano2person(I1) = peano2person(I2) => I1 = I2 )
      & ! [I1: peano,I2: peano,I3: peano] :
          ( peano_less(I1,s(I1))
          & ( ( peano_less(I1,I2)
              & peano_less(I2,I3) )
           => peano_less(I1,I3) )
          & ( peano_less(I1,I2)
           => I1 != I2 ) ) )
    & ( bob = peano2person(zero)
      & ! [I: peano] :
          child_of(peano2person(I)) = peano2person(s(I)) )
    & ( ! [A: peano,D: peano] :
          ( is_descendant(peano2person(A),peano2person(D))
        <=> peano_less(A,D) ) ) ) ).

tff(prove_formulae,conjecture,
    ( ! [A: person,D: person] : ( is_descendant(A,D) => A != D )
    & ! [A: person,C: person,G: person] :
        ( ( is_descendant(A,C)
          & is_descendant(C,G) )
       => is_descendant(A,G) )
    & ! [P: person] : is_descendant(P,child_of(P))
    & ! [P: person] : ? [C: person] : C = child_of(P) ) ).
%--------------------------------------------------------------------------
```

D TH0, Finite Interpretations

D.1 TH0 Problem with a Finite Countermodel (Online)

```
%------------------------------------------------------------------------------
thf(beverage_decl,type,   beverage: $tType ).
thf(syrup_decl,type,      syrup: $tType ).
thf(coffee_type,type,     coffee: beverage ).
thf(mix_type,type,        mix: beverage > syrup > beverage ).
thf(heat_type,type,       heat: beverage > beverage ).
thf(heated_mix_type,type, heated_mix: beverage > syrup > beverage ).
thf(hot_type,type,        hot: beverage > $o ).

thf(heated_mix,axiom,
    ( heated_mix
    = ( ^ [B: beverage,S: syrup] : ( heat @ ( mix @ B @ S ) ) ) ) ).

thf(hot_mixture,axiom,
    ! [B: beverage,S: syrup] : ( hot @ ( heated_mix @ B @ S ) ) ).

thf(heated_coffee_mix,axiom,
    ! [S: syrup] : ( ( heated_mix @ coffee @ S ) = coffee ) ).

thf(hot_coffee,conjecture,
    ? [Mixture: syrup > beverage] :
      ~ ? [S: syrup] :
          ( ( ( Mixture @ S ) = coffee )
          & ( hot @ ( Mixture @ S ) ) ) ).
%------------------------------------------------------------------------------
```

D.2 TH0 Finite Model for D.1, Coarse Grained (Online)

```
%------------------------------------------------------------------------------
thf(beverage_decl,type,   beverage: $tType ).
thf(syrup_decl,type,      syrup: $tType ).
thf(coffee_type,type,     coffee: beverage ).
thf(mix_type,type,        mix: beverage > syrup > beverage ).
thf(heat_type,type,       heat: beverage > beverage ).
thf(heated_mix_type,type, heated_mix: beverage > syrup > beverage ).
thf(hot_type,type,        hot: beverage > $o ).

thf(d_beverage_decl,type, d_beverage: $tType ).
thf(d_syrup_decl,type,    d_syrup: $tType ).
thf(d2beverage_type,type, d2beverage: d_beverage > beverage ).
thf(d2syrup_type,type,    d2syrup: d_syrup > syrup ).
thf(d_coffee_type,type,   d_coffee: d_beverage ).
thf(d_date_type,type,     d_date: d_syrup ).

thf(hot_coffee,interpretation,
    ( ( ! [B: beverage] : ? [DB: d_beverage] : ( B = ( d2beverage @ DB ) )
      & ! [DB: d_beverage] : ( DB = d_coffee )
      & ! [DB1: d_beverage,DB2: d_beverage] :
          ( ( ( d2beverage @ DB1 ) = ( d2beverage @ DB2 ) ) => ( DB1 = DB2 ) )
      & ! [S: syrup] : ? [DS: d_syrup] : ( S = ( d2syrup @ DS ) )
      & ! [DS: d_syrup] : ( DS = d_date )
      & ! [DS1: d_syrup,DS2: d_syrup] :
          ( ( ( d2syrup @ DS1 ) = ( d2syrup @ DS2 ) ) => ( DS1 = DS2 ) ) )
    & ( ( ( mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
        = ( d2beverage @ d_coffee ) )
      & ( ( heat @ ( d2beverage @ d_coffee ) )
        = ( d2beverage @ d_coffee ) )
      & ( ( heated_mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
        = ( d2beverage @ d_coffee ) )
      & ( hot @ ( d2beverage @ d_coffee ) ) ) ) ).
%------------------------------------------------------------------------------
```

D.3 TH0 Verification Problem for D.1 (Online)

```
%--------------------------------------------------------------------------
thf(beverage_decl,type,   beverage: $tType ).
thf(syrup_decl,type,      syrup: $tType ).
thf(coffee_type,type,     coffee: beverage ).
thf(mix_type,type,        mix: beverage > syrup > beverage ).
thf(heat_type,type,       heat: beverage > beverage ).
thf(heated_mix_type,type, heated_mix: beverage > syrup > beverage ).
thf(hot_type,type,        hot: beverage > $o ).

thf(d_beverage_decl,type, d_beverage: $tType ).
thf(d_syrup_decl,type,    d_syrup: $tType ).
thf(d2beverage_type,type, d2beverage: d_beverage > beverage ).
thf(d2syrup_type,type,    d2syrup: d_syrup > syrup ).
thf(d_coffee_type,type,   d_coffee: d_beverage ).
thf(d_date_type,type,     d_date: d_syrup ).

thf(hot_coffee,axiom,
    ( ( ! [B: beverage] : ? [DB: d_beverage] : ( B = ( d2beverage @ DB ) )
      & ! [DB: d_beverage] : ( DB = d_coffee )
      & ! [DB1: d_beverage,DB2: d_beverage] :
          ( ( ( d2beverage @ DB1 ) = ( d2beverage @ DB2 ) ) => ( DB1 = DB2 ) )
      & ! [S: syrup] : ? [DS: d_syrup] : ( S = ( d2syrup @ DS ) )
      & ! [DS: d_syrup] : ( DS = d_date )
      & ! [DS1: d_syrup,DS2: d_syrup] :
          ( ( ( d2syrup @ DS1 ) = ( d2syrup @ DS2 ) ) => ( DS1 = DS2 ) ) )
    & ( ( ( mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
        = ( d2beverage @ d_coffee ) )
      & ( ( heat @ ( d2beverage @ d_coffee ) )
        = ( d2beverage @ d_coffee ) )
      & ( ( heated_mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
        = ( d2beverage @ d_coffee ) )
      & ( hot @ ( d2beverage @ d_coffee ) ) ) ) ).

thf(verify,conjecture,
    ( ( heated_mix
      = ( ^ [B: beverage,S: syrup] : ( heat @ ( mix @ B @ S ) ) ) )
    & ! [B: beverage,S: syrup] : ( hot @ ( heated_mix @ B @ S ) )
    & ! [S: syrup] : ( ( heated_mix @ coffee @ S ) = coffee )
    & ~ ( ? [Mixture: syrup > beverage] :
            ~ ? [S: syrup] :
                ( ( ( Mixture @ S ) = coffee )
                & ( hot @ ( Mixture @ S ) ) ) ) ) ).
%--------------------------------------------------------------------------
```

D.4 TH0 Finite Model for D.1, Mixed Grained (Online)

```
%------------------------------------------------------------------------------
thf(beverage_decl,type,   beverage: $tType ).
thf(syrup_decl,type,      syrup: $tType ).
thf(coffee_type,type,     coffee: beverage ).
thf(mix_type,type,        mix: beverage > syrup > beverage ).
thf(heat_type,type,       heat: beverage > beverage ).
thf(heated_mix_type,type, heated_mix: beverage > syrup > beverage ).
thf(hot_type,type,        hot: beverage > $o ).

thf(d_beverage_decl,type, d_beverage: $tType ).
thf(d_syrup_decl,type,    d_syrup: $tType ).
thf(d2beverage_type,type, d2beverage: d_beverage > beverage ).
thf(d2syrup_type,type,    d2syrup: d_syrup > syrup ).
thf(d_coffee_type,type,   d_coffee: d_beverage ).
thf(d_date_type,type,     d_date: d_syrup ).

thf(hot_coffee_beverage,interpretation-domain(beverage,d_beverage),
    ( ! [B: beverage] : ? [DB: d_beverage] : ( B = ( d2beverage @ DB ) )
    & ! [DB: d_beverage] : ( DB = d_coffee )
    & ! [DB1: d_beverage,DB2: d_beverage] :
        ( ( ( d2beverage @ DB1 ) = ( d2beverage @ DB2 ) ) => ( DB1 = DB2 ) ) ) ).

thf(hot_coffee_syrup,interpretation-domain(syrup,d_syrup),
    ( ! [S: syrup] : ? [DS: d_syrup] : ( S = ( d2syrup @ DS ) )
    & ! [DS: d_syrup] : ( DS = d_date )
    & ! [DS1: d_syrup,DS2: d_syrup] :
        ( ( ( d2syrup @ DS1 ) = ( d2syrup @ DS2 ) ) => ( DS1 = DS2 ) ) ) ).

thf(hot_coffee,interpretation-mapping,
    ( ( ( mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
      = ( d2beverage @ d_coffee ) )
    & ( ( heat @ ( d2beverage @ d_coffee ) )
      = ( d2beverage @ d_coffee ) )
    & ( ( heated_mix @ ( d2beverage @ d_coffee ) @ ( d2syrup @ d_date ) )
      = ( d2beverage @ d_coffee ) )
    & ( hot @ ( d2beverage @ d_coffee ) ) ) ).
%------------------------------------------------------------------------------
```

D.5 TH0 Finite Model for D.1, Compacted (Online)

```
%--------------------------------------------------------------------------
thf(beverage_decl,type,   beverage: $tType ).
thf(syrup_decl,type,      syrup: $tType ).
thf(coffee_type,type,     coffee: beverage ).
thf(mix_type,type,        mix: beverage > syrup > beverage ).
thf(heat_type,type,       heat: beverage > beverage ).
thf(heated_mix_type,type, heated_mix: beverage > syrup > beverage ).
thf(hot_type,type,        hot: beverage > $o ).

thf(d_beverage_decl,type, d_beverage: $tType ).
thf(d_syrup_decl,type,    d_syrup: $tType ).
thf(d2beverage_type,type, d2beverage: d_beverage > beverage ).
thf(d2syrup_type,type,    d2syrup: d_syrup > syrup ).
thf(d_coffee_type,type,   d_coffee: d_beverage ).
thf(d_date_type,type,     d_date: d_syrup ).

thf(hot_coffee_beverage,interpretation-domain(beverage,d_beverage),
    ( ! [B: beverage] : ? [DB: d_beverage] : ( B = ( d2beverage @ DB ) )
    & ! [DB: d_beverage] : ( DB = d_coffee )
    & ! [DB1: d_beverage,DB2: d_beverage] :
        ( ( ( d2beverage @ DB1 ) = ( d2beverage @ DB2 ) ) => ( DB1 = DB2 ) ) ) ).

thf(hot_coffee_syrup,interpretation-domain(beverage,d_beverage),
    ( ! [S: syrup] : ? [DS: d_syrup] : ( S = ( d2syrup @ DS ) )
    & ! [DS: d_syrup] : ( DS = d_date )
    & ! [DS1: d_syrup,DS2: d_syrup] :
        ( ( ( d2syrup @ DS1 ) = ( d2syrup @ DS2 ) ) => ( DS1 = DS2 ) ) ) ).

thf(hot_coffee,interpretation-mapping,
    ( ! [F: beverage > syrup > beverage,S: syrup] :
        ( ( F @ ( d2beverage @ d_coffee ) @ ( d2syrup @ S ) )
        = ( d2beverage @ d_coffee ) )
    & ( ( heat @ ( d2beverage @ d_coffee ) )
      = ( d2beverage @ d_coffee ) )
    & ( hot @ ( d2beverage @ d_coffee ) ) ) ).
%--------------------------------------------------------------------------
```

E NXF, Finite Worlds with Finite Interpretations

E.1 NXF Problem with Global Axioms, with a Finite-Finite Countermodel (Online)

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

tff(person_decl,type,person: $tType).
tff(product_decl,type,product: $tType).
tff(alex_decl,type,alex: person).
tff(chris_decl,type,chris: person).
tff(leo_decl,type,leo: product).
tff(work_hard_decl,type,work_hard: (person * product) > $o).
tff(gets_rich_decl,type,gets_rich: person > $o).

%----If there is a product that a person works hard on, then
%----it's possible that the person will get rich.
tff(work_hard_to_get_rich,axiom,
    ! [P: person] :
      ( ? [R: product] : work_hard(P,R)
     => ( {$possible} @ (gets_rich(P)) ) ) ).

%----Nobody necessarily gets rich.
tff(not_all_get_rich,axiom,
    ~ ? [P: person] : ({$necessary} @ (gets_rich(P)) ) ).

%----Alex and Chris work hard on Leo-III.
tff(alex_works_on_leo,axiom,
    work_hard(alex,leo) ).

tff(chris_works_on_leo,axiom,
    work_hard(chris,leo) ).

%----Chris is not Alex
tff(chris_not_alex,axiom,
    chris != alex ).

%----It's possible that Alex gets rich but Chris does not.
tff(only_alex_gets_rich,conjecture,
    ( {$possible} @ (gets_rich(alex) & ~ gets_rich(chris)) ) ).
%------------------------------------------------------------------------------
```

E.2 TXF Finite-Finite Model for E.1 (Online)

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

%----Declarations to fool Vampire when processing this file directly
% tff('$world_type',type,$world: $tType).
% tff('$local_world_decl',type,$local_world: $world).
% tff('$accessible_world_decl',type,$accessible_world: ($world * $world) > $o).
% tff('$in_world_decl',type,$in_world: ($world * $o) > $o).

tff(person_decl,type,    person: $tType).
tff(product_decl,type,   product: $tType).
tff(alex_decl,type,      alex: person).
tff(chris_decl,type,     chris: person).
tff(leo_decl,type,       leo: product).
tff(work_hard_decl,type, work_hard: (person * product) > $o).
tff(gets_rich_decl,type, gets_rich: person > $o).

tff(d_person_type,type,  d_person: $tType).
tff(d2person_decl,type,  d2person: d_person > person ).
tff(d_alex_decl,type,    d_alex: d_person).
tff(d_chris_decl,type,   d_chris: d_person).
tff(d_product_type,type, d_product: $tType).
tff(d2product_decl,type, d2product: d_product > product ).
tff(d_leo_decl,type,     d_leo: d_product).

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

tff(leo_workers,interpretation,
    ( ( ! [W: $world] : ( W = w1 | W = w2 )
      & $distinct(w1,w2)
      & $local_world = w2
      & $accessible_world(w1,w1)     %----Logic is M
      & $accessible_world(w2,w2)
      & $accessible_world(w1,w2)
      & $accessible_world(w2,w1) )
    & $in_world(w1,
        ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
        & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( d2person(DP1) = d2person(DP2) => DP1 = DP2 )
        & ! [P: product] : ? [DP: d_product] : P = d2product(DP)
        & ! [DP: d_product] : DP = d_leo
```

```
        & ? [DP: d_product] : DP = d_leo
        & ! [DP1: d_product,DP2: d_product] :
            ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) )
      & ( alex = d2person(d_alex)
        & chris = d2person(d_chris)
        & leo = d2product(d_leo) )
      & ( work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo))
        & gets_rich(d2person(d_alex))
        & gets_rich(d2person(d_chris)) ) )
    & $in_world(w2,
        ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
        & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( d2person(DP1) = d2person(DP2) => DP1 = DP2 )
        & ! [P: product] : ? [DP: d_product] : P = d2product(DP)
        & ! [DP: d_product] : DP = d_leo
        & ? [DP: d_product] : DP = d_leo
        & ! [DP1: d_product,DP2: d_product] :
            ( d2product(DP1) = d2product(DP2) => DP1 = DP2 )
        & ( alex = d2person(d_alex)
          & chris = d2person(d_chris)
          & leo = d2product(d_leo) )
        & ( work_hard(d2person(d_alex),d2product(d_leo))
          & work_hard(d2person(d_chris),d2product(d_leo))
          & ~ gets_rich(d2person(d_alex))
          & ~ gets_rich(d2person(d_chris)) ) ) ) ) ).
%------------------------------------------------------------------------------
```

E.3 NXF Verification Problem for E.1 and E.2 (Online)

```
%----------------------------------------------------------------------------
tff(the_logic,logic,$$fomlModel).

tff(person_decl,type,    person: $tType ).
tff(product_decl,type,   product: $tType ).
tff(alex_decl,type,      alex: person ).
tff(chris_decl,type,     chris: person ).
tff(leo_decl,type,       leo: product ).
tff(work_hard_decl,type, work_hard: ( person * product ) > $o ).
tff(gets_rich_decl,type, gets_rich: person > $o ).

tff(d_person_type,type,  d_person: $tType ).
tff(d2person_decl,type,  d2person: d_person > person ).
tff(d_alex_decl,type,    d_alex: d_person ).
tff(d_chris_decl,type,   d_chris: d_person ).
tff(d_product_type,type, d_product: $tType ).
tff(d2product_decl,type, d2product: d_product > product ).
tff(d_leo_decl,type,     d_leo: d_product ).

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

tff(leo_workers,interpretation,
    ( ! [W: $world] :
        ( ( W = w1 ) | ( W = w2 ) )
    & $distinct(w1,w2)
    & ( $local_world = w2 )
    & $accessible_world(w1,w1)
    & $accessible_world(w2,w2)
    & $accessible_world(w1,w2)
    & $accessible_world(w2,w1)
    & $in_world(w1,
        ( ! [P: person] :
          ? [DP: d_person] : ( P = d2person(DP) )
        & ! [DP: d_person] :
            ( ( DP = d_alex ) | ( DP = d_chris ) )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( ( d2person(DP1) = d2person(DP2) )
           => ( DP1 = DP2 ) )
        & ! [P: product] :
          ? [DP: d_product] : ( P = d2product(DP) )
        & ! [DP: d_product] : ( DP = d_leo )
        & ? [DP: d_product] : ( DP = d_leo )
        & ! [DP1: d_product,DP2: d_product] :
            ( ( d2product(DP1) = d2product(DP2) )
           => ( DP1 = DP2 ) )
        & ( alex = d2person(d_alex) )
        & ( chris = d2person(d_chris) )
```

```
        & ( leo = d2product(d_leo) )
        & work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo))
        & gets_rich(d2person(d_alex))
        & gets_rich(d2person(d_chris)) ))
    & $in_world(w2,
        ( ! [P: person] :
          ? [DP: d_person] : ( P = d2person(DP) )
        & ! [DP: d_person] :
            ( ( DP = d_alex ) | ( DP = d_chris ) )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( ( d2person(DP1) = d2person(DP2) )
           => ( DP1 = DP2 ) )
        & ! [P: product] :
          ? [DP: d_product] : ( P = d2product(DP) )
        & ! [DP: d_product] : ( DP = d_leo )
        & ? [DP: d_product] : ( DP = d_leo )
        & ! [DP1: d_product,DP2: d_product] :
            ( ( d2product(DP1) = d2product(DP2) )
           => ( DP1 = DP2 ) )
        & ( alex = d2person(d_alex) )
        & ( chris = d2person(d_chris) )
        & ( leo = d2product(d_leo) )
        & work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo))
        & ~ gets_rich(d2person(d_alex))
        & ~ gets_rich(d2person(d_chris)) )) ) ).

tff(work_hard_to_get_rich,conjecture-global,
    ! [P: person] :
      ( ? [R: product] : work_hard(P,R)
     => ( {$possible} @ (gets_rich(P)) ) ) ).

tff(not_all_get_rich,conjecture-global,
    ~ ? [P: person] : ( {$necessary} @ (gets_rich(P)) ) ).

tff(alex_works_on_leo,conjecture-global,
    work_hard(alex,leo) ).

tff(chris_works_on_leo,conjecture-global,
    work_hard(chris,leo) ).

tff(only_alex_gets_rich,conjecture-local,
    ~ ( {$possible}
      @ (( gets_rich(alex)
         & ~ gets_rich(chris) )) ) ).
%----------------------------------------------------------------------------
```

E.4 NXF Problem with Global and Local Axioms, with a Finite-Finite Countermodel (Online)

```
%------------------------------------------------------------------------------
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).

tff(apple_not_banana,axiom,
    apple != banana ).

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

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

tff(rotten_banana_here,axiom-local,
    rotten(banana) ).

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

E.5 TXF Finite-Finite Model for E.4 (Online)

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

%----Declarations to fool Vampire when processing this file directly
% tff('$world_type',type,$world: $tType).
% tff('$local_world_decl',type,$local_world: $world).
% tff('$accessible_world_decl',type,$accessible_world: ($world * $world) > $o).
% tff('$in_world_decl',type,$in_world: ($world * $o) > $o).

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).

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,
    ( ( ! [W: $world] : ( W = w1 | W = w2 )
      & $local_world = w1
      & $accessible_world(w1,w1)     %----Logic is M
      & $accessible_world(w2,w2)
      & $accessible_world(w1,w2) )
    & $in_world(w1,
        ( ( ! [F: fruit] : ? [DF: d_fruit] : F = d2fruit(DF)
          & ! [DF: d_fruit] : ( DF = d_apple | DF = d_banana )
          & $distinct(d_apple,d_banana)
          & ? [DP: d_fruit] : ( DP = d_apple )
          & ? [DP: d_fruit] : ( DP = d_banana )
          & ! [DF1: d_fruit,DF2: d_fruit] :
              ( d2fruit(DF1) = d2fruit(DF2) => DF1 = DF2 ) )
        & ( apple = d2fruit(d_apple)
          & banana = d2fruit(d_banana) )
        & ( healthy(d2fruit(d_apple))
          & healthy(d2fruit(d_banana))
          & ~ rotten(d2fruit(d_apple))
          & rotten(d2fruit(d_banana)) ) ) )
    & $in_world(w2,
        ( ( ! [F: fruit] : ? [DF: d_fruit] : F = d2fruit(DF)
          & ! [DF: d_fruit] : ( DF = d_apple | DF = d_banana )
```

```
          & $distinct(d_apple,d_banana)
          & ? [DP: d_fruit] : ( DP = d_apple )
          & ? [DP: d_fruit] : ( DP = d_banana )
          & ! [DF1: d_fruit,DF2: d_fruit] :
              ( d2fruit(DF1) = d2fruit(DF2) => DF1 = DF2 ) )
        & ( apple = d2fruit(d_apple)
          & banana = d2fruit(d_banana) )
        & ( healthy(d2fruit(d_apple))
          & healthy(d2fruit(d_banana))
          & ~ rotten(d2fruit(d_apple))
          & ~ rotten(d2fruit(d_banana)) ) ) ) ) ).
%------------------------------------------------------------------------------
```

E.6 NXF Verification Problem for E.4 and E.5 (Online)

```
%----------------------------------------------------------------------------
tff(the_logic,logic,$$fomlModel).

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 ).

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,
    ( ! [W: $world] : ( ( W = w1 ) | ( W = w2 ) )
    & ( $local_world = w1 )
    & $accessible_world(w1,w1) & $accessible_world(w2,w2)
    & $accessible_world(w1,w2)
    & $in_world(w1,
        ( ! [F: fruit] :
          ? [DF: d_fruit] : ( F = d2fruit(DF) )
        & ! [DF: d_fruit] :
            ( ( DF = d_apple )
            | ( DF = d_banana ) )
        & $distinct(d_apple,d_banana)
        & ? [DP: d_fruit] : ( DP = d_apple )
        & ? [DP: d_fruit] : ( DP = d_banana )
        & ! [DF1: d_fruit,DF2: d_fruit] :
            ( ( d2fruit(DF1) = d2fruit(DF2) )
           => ( DF1 = DF2 ) )
        & ( apple = d2fruit(d_apple) )
        & ( banana = d2fruit(d_banana) )
        & healthy(d2fruit(d_apple))
        & healthy(d2fruit(d_banana))
        & ~ rotten(d2fruit(d_apple))
        & rotten(d2fruit(d_banana)) ))
    & $in_world(w2,
        ( ! [F: fruit] :
          ? [DF: d_fruit] : ( F = d2fruit(DF) )
        & ! [DF: d_fruit] :
            ( ( DF = d_apple )
            | ( DF = d_banana ) )
        & $distinct(d_apple,d_banana)
        & ? [DP: d_fruit] : ( DP = d_apple )
        & ? [DP: d_fruit] : ( DP = d_banana )
        & ! [DF1: d_fruit,DF2: d_fruit] :
            ( ( d2fruit(DF1) = d2fruit(DF2) )
```

```
           => ( DF1 = DF2 ) )
        & ( apple = d2fruit(d_apple) )
        & ( banana = d2fruit(d_banana) )
        & healthy(d2fruit(d_apple))
        & healthy(d2fruit(d_banana))
        & ~ rotten(d2fruit(d_apple))
        & ~ rotten(d2fruit(d_banana)) )) ) ).

tff(apple_not_banana,conjecture-global,
    apple != banana ).

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

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

tff(rotten_banana_here,conjecture-local,
    rotten(banana) ).

tff(not_true,conjecture-local,
    ~ ( {$necessary}
      @ (( healthy(apple)
         & ~ rotten(banana) )) ) ).
%----------------------------------------------------------------------------
```

E.7 TXF Finite-Finite Model for E.1, Medium Grained (Online)

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

tff(person_decl,type,    person: $tType).
tff(product_decl,type,   product: $tType).
tff(alex_decl,type,      alex: person).
tff(chris_decl,type,     chris: person).
tff(leo_decl,type,       leo: product).
tff(work_hard_decl,type, work_hard: (person * product) > $o).
tff(gets_rich_decl,type, gets_rich: person > $o).

tff(d_person_type,type,  d_person: $tType).
tff(d2person_decl,type,  d2person: d_person > person ).
tff(d_alex_decl,type,    d_alex: d_person).
tff(d_chris_decl,type,   d_chris: d_person).
tff(d_product_type,type, d_product: $tType).
tff(d2product_decl,type, d2product: d_product > product ).
tff(d_leo_decl,type,     d_leo: d_product).

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

tff(leo_workers_worlds,interpretation-world,
    ( ! [W: $world] : ( W = w1 | W = w2 )
    & $distinct(w1,w2)
    & $local_world = w2
    & $accessible_world(w1,w1)     %----Logic is M
    & $accessible_world(w2,w2)
    & $accessible_world(w1,w2)
    & $accessible_world(w2,w1) ) ).

tff(leo_workers_w1_domain,
    interpretation-in_world(w1,interpretation-domain),
    $in_world(w1,
      ( ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
        & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( d2person(DP1) = d2person(DP2) => DP1 = DP2 ) )
      & ( ! [P: product] : ? [DP: d_product] : P = d2product(DP)
        & ! [DP: d_product] : DP = d_leo
        & ? [DP: d_product] : DP = d_leo
        & ! [DP1: d_product,DP2: d_product] :
```

```
            ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) ) ) ) ).

tff(leo_workers_w2_domain,
    interpretation-in_world(w2,interpretation-domain),
    $in_world(w2,
      ( ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
        & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
        & $distinct(d_alex,d_chris)
        & ? [DP: d_person] : ( DP = d_alex )
        & ? [DP: d_person] : ( DP = d_chris )
        & ! [DP1: d_person,DP2: d_person] :
            ( d2person(DP1) = d2person(DP2) => DP1 = DP2 ) )
      & ( ! [P: product] : ? [DP: d_product] : P = d2product(DP)
        & ! [DP: d_product] : DP = d_leo
        & ? [DP: d_product] : DP = d_leo
        & ! [DP1: d_product,DP2: d_product] :
            ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) ) ) ) ).

tff(leo_workers_w1_mappings,
    interpretation-in_world(w1,interpretation-mapping),
    $in_world(w1,
      ( ( alex = d2person(d_alex)
        & chris = d2person(d_chris)
        & leo = d2product(d_leo) )
      & ( work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo))
        & gets_rich(d2person(d_alex))
        & gets_rich(d2person(d_chris)) ) ) ) ).

tff(leo_workers_w2_mappings,
    interpretation-in_world(w2,interpretation-mapping),
    $in_world(w2,
      ( ( alex = d2person(d_alex)
        & chris = d2person(d_chris)
        & leo = d2product(d_leo) )
      & ( work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo))
        & ~ gets_rich(d2person(d_alex))
        & ~ gets_rich(d2person(d_chris)) ) ) ) ).
%----------------------------------------------------------------------------
```

E.8 TXF Finite-Finite Model for E.1, Fine Grained (Online)

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

tff(person_decl,type,    person: $tType).
tff(product_decl,type,   product: $tType).
tff(alex_decl,type,      alex: person).
tff(chris_decl,type,     chris: person).
tff(leo_decl,type,       leo: product).
tff(work_hard_decl,type, work_hard: (person * product) > $o).
tff(gets_rich_decl,type, gets_rich: person > $o).

tff(d_person_type,type,  d_person: $tType).
tff(d2person_decl,type,  d2person: d_person > person ).
tff(d_alex_decl,type,    d_alex: d_person).
tff(d_chris_decl,type,   d_chris: d_person).
tff(d_product_type,type, d_product: $tType).
tff(d2product_decl,type, d2product: d_product > product ).
tff(d_leo_decl,type,     d_leo: d_product).

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

tff(leo_workers_worlds,interpretation-world,
    ( ! [W: $world] : ( W = w1 | W = w2 )
    & $distinct(w1,w2)
    & $local_world = w2
    & $accessible_world(w1,w1) & $accessible_world(w2,w2)
    & $accessible_world(w1,w2) & $accessible_world(w2,w1) ) ).

tff(leo_workers_w1_person,
    interpretation-in_world(w1,interpretation-domain(person,d_person)),
    $in_world(w1,
      ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
      & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
      & $distinct(d_alex,d_chris)
      & ? [DP: d_person] : ( DP = d_alex )
      & ? [DP: d_person] : ( DP = d_chris )
      & ! [DP1: d_person,DP2: d_person] :
          ( d2person(DP1) = d2person(DP2) => DP1 = DP2 ) ) ) ).

tff(leo_workers_w1_product,
    interpretation-in_world(w1,interpretation-domain(product,d_product)),
    $in_world(w1,
      ( ! [P: product] : ? [DP: d_product] : P = d2product(DP)
      & ! [DP: d_product] : DP = d_leo
      & ? [DP: d_product] : DP = d_leo
```

```
      & ! [DP1: d_product,DP2: d_product] :
          ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) ) ) ).

tff(leo_workers_w2_person,
    interpretation-in_world(w2,interpretation-domain(person,d_person)),
    $in_world(w2,
      ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
      & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
      & $distinct(d_alex,d_chris)
      & ? [DP: d_person] : ( DP = d_alex )
      & ? [DP: d_person] : ( DP = d_chris )
      & ! [DP1: d_person,DP2: d_person] :
          ( d2person(DP1) = d2person(DP2) => DP1 = DP2 ) ) ) ).

tff(leo_workers_w2_product,
    interpretation-in_world(w2,interpretation-domain(product,d_product)),
    $in_world(w2,
      ( ! [P: product] : ? [DP: d_product] : P = d2product(DP)
      & ! [DP: d_product] : DP = d_leo
      & ? [DP: d_product] : DP = d_leo
      & ! [DP1: d_product,DP2: d_product] :
          ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) ) ) ).

tff(leo_workers_w1_alex,
    interpretation-in_world(w1,interpretation-mapping(alex,d_person)),
    $in_world(w1, ( alex = d2person(d_alex) )) ).

tff(leo_workers_w2_alex,
    interpretation-in_world(w2,interpretation-mapping(alex,d_person)),
    $in_world(w2, ( alex = d2person(d_alex) )) ).

tff(leo_workers_w1_chris,
    interpretation-in_world(w1,interpretation-mapping(chris,d_person)),
    $in_world(w1, ( chris = d2person(d_chris) )) ).

tff(leo_workers_w2_chris,
    interpretation-in_world(w2,interpretation-mapping(chris,d_person)),
    $in_world(w2, ( chris = d2person(d_chris) )) ).

tff(leo_workers_w1_leo,
    interpretation-in_world(w1,interpretation-mapping(leo,d_product)),
    $in_world(w1, ( leo = d2product(d_leo) )) ).

tff(leo_workers_w2_leo,
    interpretation-in_world(w2,interpretation-mapping(leo,d_product)),
    $in_world(w2, ( leo = d2product(d_leo) )) ).

tff(leo_workers_w1_work_hard,
    interpretation-in_world(w1,interpretation-mapping(work_hard,$o)),
    $in_world(w1,
      ( work_hard(d2person(d_alex),d2product(d_leo))
      & work_hard(d2person(d_chris),d2product(d_leo)) )) ).
```

```
tff(leo_workers_w2_work_hard,
    interpretation-in_world(w2,interpretation-mapping(work_hard,$o)),
    $in_world(w2,
      ( work_hard(d2person(d_alex),d2product(d_leo))
      & work_hard(d2person(d_chris),d2product(d_leo)) )) ).

tff(leo_workers_w1_gets_rich,
    interpretation-in_world(w1,interpretation-mapping(gets_rich,$o)),
    $in_world(w1,
      ( gets_rich(d2person(d_alex)) & gets_rich(d2person(d_chris)) ) ) ).

tff(leo_workers_w2_gets_rich,
    interpretation-in_world(w2,interpretation-mapping(gets_rich,$o)),
    $in_world(w2,
      ( ~ gets_rich(d2person(d_alex)) & ~ gets_rich(d2person(d_chris)) ) ) ).
%----------------------------------------------------------------------------
```

E.9 TXF Finite-Finite Model for E.1, Compacted (Online)

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

tff(person_decl,type,    person: $tType).
tff(product_decl,type,   product: $tType).
tff(alex_decl,type,      alex: person).
tff(chris_decl,type,     chris: person).
tff(leo_decl,type,       leo: product).
tff(work_hard_decl,type, work_hard: (person * product) > $o).
tff(gets_rich_decl,type, gets_rich: person > $o).

tff(d_person_type,type,  d_person: $tType).
tff(d2person_decl,type,  d2person: d_person > person ).
tff(d_alex_decl,type,    d_alex: d_person).
tff(d_chris_decl,type,   d_chris: d_person).
tff(d_product_type,type, d_product: $tType).
tff(d2product_decl,type, d2product: d_product > product ).
tff(d_leo_decl,type,     d_leo: d_product).

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

tff(leo_workers_worlds,interpretation-world,
    ( ! [W: $world] : ( W = w1 | W = w2 )
    & $distinct(w1,w2)
    & $local_world = w2
    & $accessible_world(w1,w1) & $accessible_world(w2,w2)
    & $accessible_world(w1,w2) & $accessible_world(w2,w1) ) ).

tff(leo_workers_W_product,
    interpretation-in_world(W,interpretation-domain),
    ! [W: $world] :
      $in_world(W,
        ( ( ! [P: product] : ? [DP: d_product] : P = d2product(DP)
          & ! [DP: d_product] : DP = d_leo
          & ? [DP: d_product] : DP = d_leo
          & ! [DP1: d_product,DP2: d_product] :
              ( d2product(DP1) = d2product(DP2) => DP1 = DP2 ) )
        & ( ! [P: person] : ? [DP: d_person] : P = d2person(DP)
          & ! [DP: d_person] : ( DP = d_alex | DP = d_chris )
          & $distinct(d_alex,d_chris)
          & ? [DP: d_person] : ( DP = d_alex )
          & ? [DP: d_person] : ( DP = d_chris )
          & ! [DP1: d_person,DP2: d_person] :
              ( d2person(DP1) = d2person(DP2) => DP1 = DP2 ) ) ) ) ).
```

```
tff(leo_workers_W_alex_chris,
    interpretation-in_world(W,interpretation-mapping),
    ! [W: $world] :
      $in_world(W,
        ( ( alex = d2person(d_alex) ) & ( chris = d2person(d_chris) ) ) ).

tff(leo_workers_W_leo,
    interpretation-in_world(W,interpretation-mapping(leo,d_product)),
    ! [W: $world] :
      $in_world(W, ( leo = d2product(d_leo) ) ) ) ).

tff(leo_workers_W_work_hard,
    interpretation-in_world(W,interpretation-mapping(work_hard,$o)),
    ! [W: $world] :
      $in_world(W,
        ( work_hard(d2person(d_alex),d2product(d_leo))
        & work_hard(d2person(d_chris),d2product(d_leo)) ) ) ).

tff(leo_workers_w1_gets_rich,
    interpretation-in_world(w1,interpretation-mapping(gets_rich,$o)),
    $in_world(w1,
      ( gets_rich(d2person(d_alex)) & gets_rich(d2person(d_chris)) ) ) ).

tff(leo_workers_w2_gets_rich,
    interpretation-in_world(w2,interpretation-mapping(gets_rich,$o)),
    $in_world(w2,
      ( ~ gets_rich(d2person(d_alex)) & ~ gets_rich(d2person(d_chris)) ) ) ).
%----------------------------------------------------------------------------
```