Representation, Verification, and Visualization of Tarskian Interpretations for Typed First-order Logic

Anon One
Some where
Some place
Some country

Anon Two
Some where
Some place
Some country

Anon Three
Some where
Some place
Some country

Anon Four
Some where
Some place
Some country

Abstract

This paper describes a new format for representing Tarskian-style interpretations for formulae in typed first-order logic, using the TPTP TF0 language. It further describes a technique and an implemented tool for verifying models using this representation, and a tool for visualizing interpretations. The research contributes to the advancement of automated reasoning technology for model finding, which has several applications, including verification.

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 (Robinson and Voronkov 2001). The axioms and 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 (Sutcliffe et al. 2006). In this work typed first-order logic in the form of (Walther 1983; Schmidt-Schauss 1985; Cohn 1987), whose expressive power is sufficient for a wide range of topics (Sutcliffe 2017), is used. (This work is also applicable to untyped first-order logic, and can also be generalized to higher-order logics.) In the last two decades the converse task of disproving conjectures has become increasingly important. This process depends on finding an *interpretation*, i.e., a structure that maps terms to domain elements and formulae to truth values. An interpretation that maps a formula to *true* is a *model* of the formula. A conjecture is disproved by finding an interpretation that is a model of the axioms, but maps the conjecture to *false*. A salient application area that harnesses this form of ATP is verification (D'Silva, Kroening, and Weissenbacher 2008), 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 (Schulz et al. 2017), and finding a solution to a problem that is coded as a model finding problem (Winker 1982). This work describes a (new) format for representing interpretations using a TPTP language - Sections 2 and 3.

In addition to ATP systems that produce interpretations (typically models), e.g., Paradox (Claessen and Sörensson 2003), Vampire (Kovacs and Voronkov 2013), and Nitpick (Blanchette and Nipkow 2010), there is a need for tools that support examination and use of interpretations. This paper considers the tasks of verifying models and visualizing interpretations, and describes new tools for these tasks - Sections 4 and 5.

Related Work: In (Sutcliffe et al. 2006) a TPTP format for interpretations with finite domains was defined, and has been adopted by some ATP systems, e.g., Paradox and Vampire. The SMT-LIB standard (Barrett, Fontaine, and Tinelli 2017) defines a format for model output, and commands to inspect models. SAT solvers generally output models as specified by the SAT competitions (Järvisalo et al. 2012), in a simple format similar to the DIMACS input format (Babic 1993). Some individual model finding systems have defined their own formats for models, e.g., the output formats of Nitpick and Z3 (de Moura and Bjørner 2008).

Related work on model verification and interpretation visualization is sparse. In personal communications with members of the Vampire team it was revealed that Vampire can internally verify finite models in TPTP format by using the model formulae to evaluate the given formulae. This approach is limited to finite models. In personal communications with the developer of Paradox he explained his approach, which is to use a trusted model finder to show that the model formulae and the given formulae are together satisfiable. This shows that the model formulae are consistent with the given formulae, but does not verify the model – as the developer said, it is "poor-man's model verifier!".

For interpretation visualization, the Mace4 model finder (McCune 2003a) outputs textual information about the models it finds, including the interpretation of constants as integers, and tables for the function and predicate symbols' interpretations. The tables are naturally limited to symbols of arity up to two (which is just fine for algebras, where Mace4 is often applied). The only graphical visualization tool that has been found is described in (Schlyter 2013), which provided (past tense – it is no longer available) a visualization of finite first-order interpretations as produced by Paradox. The visualization had some nice features, e.g., showing functions as constructor functions, and reducing the visual clutter

when displaying relations with properties such as symmetry, transitivity, etc. In other ways that work was quite different from the visualization described in this work.

2 The TPTP World and Languages

The TPTP World (Sutcliffe 2017) is a well established infrastructure that supports research, development, and deployment of ATP systems. The TPTP World includes the TPTP problem library, the TSTP solution library, standards for writing ATP problems and reporting ATP solutions, tools and services for processing ATP problems and solutions, and it supports the CADE ATP System Competition (CASC). Various parts of the TPTP World have been deployed in both academia and industry. The web page `https://www.tptp.org` provides access to all components.

The TPTP language (Sutcliffe 2022) is one of the keys to the success of the TPTP World. The language is used for writing both problems and solutions. Originally the TPTP World supported only first-order clause normal form (CNF). Over the years full first-order form (FOF), typed first-order form (TFF), typed extended first-order form (TXF), typed higher-order form (THF), and non-classical forms (NTF), have been added. All the typed forms include constructs for arithmetic. TF0 (Sutcliffe et al. 2012), the monomorphic subform of TFF, is used in this work (see Section 2.1).

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

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

The *language*s supported are `cnf` (clause normal form), `fof` (first-order form), `tff` (typed first-order form), and `thf` (typed higher-order form). The *role*, e.g., `axiom`, `lemma`, `conjecture`, defines the use of the formula in an ATP system. In a *formula*, terms and atoms follow Prolog conventions – functions and predicates start with a lowercase letter or are `'`single quoted`'`, and variables start with an uppercase letter. The language also supports interpreted symbols, which either start with a `$`, e.g., the truth constants `$true` and `$false`, or are composed of non-alphabetic characters, e.g., integer/rational/real numbers such as 27, 43/92, -99.66. The basic logical connectives are `!`, `?`, `~`, `|`, `&`, `=>`, `<=`, `<=>`, and `<~>`, for $\forall$, $\exists$, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\Leftarrow$, $\Leftrightarrow$, and $\oplus$ respectively. Equality and inequality are expressed as the infix operators = and `!=`. The *source* and *useful_info* are optional. Annotated formulae (using TF0) can be seen in Figures 1-5.

2.1 The TF0 Language

TF0 is a typed first-order language. The TF0 types are (i) the predefined types `$i` for individuals and `$o` for booleans; (ii) the predefined arithmetic types `$int`, `$rat`, and `$real`; (iii) user-defined types declared to be of the kind `$tType`. Every symbol is declared with a type signature: (i) individual types for variables; (ii) function types from non-boolean argument types to a non-boolean result type; (iii) predicate types from non-boolean argument types to a boolean result. The equality predicates = and `!=` are ad hoc polymorphic over all types. Arithmetic predicates and functions are ad hoc polymorphic over the arithmetic types. Figures 1 and 2 are examples of problems in TF0. Their associated (counter)models are discussed in Section 3.

```
%----------------------------------------------------------
tff(man_type,type,           man: $tType ).
tff(grade_type,type,         grade: $tType ).
tff(john_decl,type,          john: man ).
tff(a_decl,type,             a: grade ).
tff(f_decl,type,             f: grade ).
tff(grade_of_decl,type,      grade_of: man > grade ).
tff(created_equal_decl,type,
    created_equal: ( man * man ) > $o ).

tff(all_created_equal,axiom,
    ! [H1: man,H2: man] : created_equal(H1,H2) ).

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

tff(someone_got_an_a,axiom,
    ? [H: man] : grade_of(H) = a ).

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

tff(equality_lost,conjecture,
    ! [H1: man,H2: man] :
      ( created_equal(H1,H2) <=> ( H1 = H2 ) ) ).
%----------------------------------------------------------
```

Figure 1: A TF0 problem (with a finite countermodel)
`https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Finite.p`

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

Figure 2: A TF0 problem (with an infinite model)
`https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Infinite.p`

3 Interpretations

A Tarskian-style interpretation (Tarski and Vaught 1956) of formulae in typed first-order logic consists of a non-empty domain of unequal elements for each type used in the formulae (just one domain for untyped logic), and interpretations of the function and predicate symbols with respect to the domains (Hunter 1996). An interpretation can normally interpret all expressions that can be written in the language of the formulae, but in some circumstances an interpretation can interpret only (at least) the given formulae; such an interpretation is a *partial interpretation*.

The domains of an interpretation may be finite or infinite. Interpretations with only finite domains are called *finite interpretations*, and interpretations with one of more infinite domains are called *infinite interpretations*. Finite domains are commonly explicitly enumerated, but can also take other

forms, e.g., the finite Herbrand Universe of a Herbrand interpretation (Herbrand 1930). Infinite domains can take several forms, including being implicitly specified (e.g., some set of algebraic numbers, such as the integers), explicitly generated (e.g., terms representing Peano numbers), and the infinite Herbrand Universe of a Herbrand interpretation.

In addition to Tarskian-style interpretations that provide explicit symbol interpretation, a Herbrand interpretation can also be embodied in a saturation (Bachmair et al. 2001). While the domain of a saturation is known to be the Herbrand Universe, there is no explicit symbol interpretation that can be used constructively by users. Saturations are thus a less useful form of interpretation.

The notions of interpretations, models, partial interpretations, finite interpretations, Herbrand interpretations, etc., are captured in the SZS ontologies (Sutcliffe 2008), as updated at `https://www.tptp.org/cgi-bin/SeeTPTP?Category=Documents&File=SZSOntology`

3.1 Representing Interpretations in TF0

As noted in Section 1, a TPTP format for interpretations with finite domains has previously been defined. Recently the need for a format for interpretations with infinite domains, and for a format for Kripke interpretations (Kripke 1963), led to the development of a new TPTP format for interpretations. The underlying principle is unchanged: interpretations are represented as formulae.

The new format uses an *interpretation formula*. Examples of interpretation formulae can be seen in Figures 3 and 4, illustrating the components described next. An interpretation formula is a conjunction of:

- a conjunction of domain specifications for the types in the given formulae; each specification is a conjunction of (each specification includes a *type-promotion* function to convert domain elements to terms, to make the interpretation formula well-typed):
  - the domain type, by a formula that makes the type-promotion function a surjection (unless it is unnecessary because the type is defined and is the same as the type in the given formulae, e.g., both are `$int`);
  - the domain elements (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; if the domain is infinite an existentially quantified formula that captures an infinite disjunction of equalities is used, e.g., for terms representing Peano numbers as the domain elements:
    $$\forall I{:}peano\ ((I = zero) \vee \exists P{:}peano\ (I = s(P)));$$
  - specification of the distinctness of the domain elements (unless implicit from their defined type);
  - a formula making the type-promotion function an injection, which with the surjectivity makes it a bijection.
- interpretation of the function symbols, as equalities whose left-hand sides are formed from symbols applied to type-promoted domain elements, and whose right-hand sides are type-promoted domain elements;
- interpretation of the predicate symbols, as literals formed from symbols applied to type-promoted domain elements; positive literals are *true* and negative literals are *false*.

The interpretation formula is preceded by the necessary type declarations:

- the types in the given formulae (except defined types, e.g., `$int`);
- the types of the domains (except defined types);
- the types of type-promotion functions;
- the types of the domain elements.

This representation is also directly usable for untyped first-order logic, where all terms in the given formulae and the interpretation formula are of the same type – "individuals". This obviates the need for type considerations, in particular type-promotion functions are not needed.

Figure 3 is a TF0 interpretation with finite domains – it is a countermodel for the problem in Figure 1. The comments show which parts of the formula specify what aspects of the interpretation. Figure 4 is a TF0 interpretation with an infinite domain – it is a model for the problem in Figure 2. Note: • 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 for an infinite number of argument tuples; • the interpretations of function and predicate symbols is not given for argument tuples with negative integers, i.e., this is an example of a partial interpretation.

4 Model Verification

ATP systems are complex pieces of software, implementing complex calculi, with the end goal being a sound implementation of a sound inference system whose output correctly corroborates the result obtained. The reality is that the complexity leads to incorrectness, and as such verification of ATP systems' outputs is necessary; for theorem proving this means verifying the proof output (Sutcliffe 2006), and for model finding this means verifying the model output. In the context of this work the model verification applies to the type declarations and the interpretation formula that represent the model found by the ATP system, and has (at least) the following aspects:

1. Are the type declarations and interpretation formula syntactically well-formed and semantically well-typed?
2. Is the interpretation formula satisfiable?
3. Does the interpretation formula correctly represent the interpretation found by the ATP system?
4. Is the interpretation represented by the interpretation formula a model for the given formulae?

These questions are answered as follows:

1. This can be confirmed using standard parsing and type checking tools, e.g., (Van Gelder and Sutcliffe 2006; Horozal and Rabe 2015).
2. This can be empirically confirmed using a trusted model finder (in the same way the GDV derivation verifier (Sutcliffe 2006) uses the Otter system (McCune 2003b) as a trusted theorem prover). Confirming that the interpretation formula is satisfiable is almost certainly much easier

```
%----------------------------------------------------------
tff(man_type,type,           man: $tType ).
tff(grade_type,type,         grade: $tType ).
tff(john_decl,type,          john: man ).
tff(a_decl,type,             a: grade ).
tff(f_decl,type,             f: grade ).
tff(grade_of_decl,type,      grade_of: man > grade ).
tff(created_equal_decl,type,
    created_equal: ( man * man ) > $o ).

%----Types of the domains
tff(d_man_type,type,         d_man: $tType).
tff(d_grade_type,type,       d_grade: $tType).
%----Types of the promotion functions
tff(d2man_decl,type,         d2man: d_man > man ).
tff(d2grade_decl,type,       d2grade: d_grade > grade ).
%----Types of the domain elements
tff(d_john_decl,type,        d_john: d_man ).
tff(d_gotA_decl,type,        d_gotA: d_man ).
tff(d_a_decl,type,           d_a: d_grade ).
tff(d_f_decl,type,           d_f: d_grade ).

tff(equality_lost,interpretation,
%----The domain for man is d_man
    ( ( ! [M: man] : ? [DM: d_man] : M = d2man(DM)
%----The d_man elements are d_john and d_gotA
      & ! [DM: d_man] : ( DM = d_john | DM = d_gotA )
      & $distinct(d_john,d_gotA)
%----The type-promoter is a bijection
      & ! [DM1: d_man,DM2: d_man] :
          ( d2man(DM1) = d2man(DM2) => DM1 = DM2 )
%----The domain for grade is d_grade
      & ! [G: grade] : ? [DG: d_grade] : G = d2grade(DG)
%----The d_grade elements are d_a and d_f
      & ! [DG: d_grade]: ( DG = d_a | DG = d_f )
      & $distinct(d_a,d_f)
%----The type-promoter is a bijection
      & ! [DG1: d_grade,DG2: d_grade] :
          ( d2grade(DG1) = d2grade(DG2) => DG1 = DG2 ) )
%----Interpret terms via the type-promoted domain
    & ( a = d2grade(d_a)
      & f = d2grade(d_f)
      & john = d2man(d_john)
      & grade_of(d2man(d_john)) = d2grade(d_f)
      & grade_of(d2man(d_gotA)) = d2grade(d_a) )
%----Interpret atoms as true of false
    & ( created_equal(d2man(d_john),d2man(d_john))
      & created_equal(d2man(d_john),d2man(d_gotA))
      & created_equal(d2man(d_gotA),d2man(d_john))
      & created_equal(d2man(d_gotA),d2man(d_gotA)) )
    ) ).
%----------------------------------------------------------
```

Figure 3: A TF0 interpretation with a finite domain
https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Finite.s

than finding the model itself, so the system used to check the satisfiability can be weaker and more trusted than the system that found the model.

3. This cannot be confirmed, as that representation is internal to the ATP system that found the model.
4. In this work a "semantic" approach is taken, in which the given formulae are proved from the interpretation formula using a trusted theorem prover; the interpretation formula is supplied as an axiom, and the given formulae as the conjecture to be proved. This approach relies on the proof below, which shows that if a given formula is a logical consequence of the interpretation formula, then the interpretation represented by the interpretation formula is a model of the given formula.

   Figure 5 shows the verification problem for the problem in Figure 2 and its model in Figure 4. An implementation is available online as the GMV tool in the SystemOnTSTP (Sutcliffe 2007) web interface `https://www.tptp.org/cgi-bin/SystemOnTSTP`. The tool input is the concatenation of the interpretation and modelled formula, delimited by `% SZS output` start and end lines. An example that can be used as the "URL to fetch from" in the web interface is `https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/FOF_Finite.sp.GMV.p`

```
%----------------------------------------------------------
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
    & ! [I1: $int,I2: $int] :
        ( int2person(I1) = int2person(I2)
       => I1 = I2 )
%----Mapping people to 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) ) ) ).
%----------------------------------------------------------
```

Figure 4: A TF0 interpretation with an infinite domain
https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Infinite.s

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

Figure 5: The TF0 verification problem for Figures 2 and 4
https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Infinite.s.p

The proof of soundness is given here for a finite interpretation in untyped first-order logic, where (as explained in Section 3.1) there is no need for type considerations. The proof for typed first-order logic follows exactly the same pattern, but is technically complicated due to the introduction of types and type promotion functions. The extension to infinite domains is quite simple after that, following Section 3.1.

Proof

Let Σ be an untyped first-order language:

- V_Σ - The variable symbols, starting in uppercase.
- F_Σ - The function symbols with arity, in the form f/n.
- P_Σ - The predicate symbols with arity, in the form p/n.

The formulae over Σ, $\mathcal{F}(\Sigma)$, are defined as usual.

Let I be an interpretation for Σ:

- D_I - A finite set of unequal domain elements.
- F_I - For each $f/n \in F_\Sigma$, a mapping $f_I : D_I^n \mapsto D_I$.
- R_I - For each $p/n \in P_\Sigma$, a mapping $p_I : D_I^n \mapsto \{true, false\}$.

Recalling Section 3.1, an interpretation is represented by an *interpretation formula*, φ. Let:

- D_φ be a set of fresh terms d_φ, one for each $d_I \in D_I$
- $D_{\varphi\mapsto I}$ be the corresponding mapping from D_φ to D_I
- Σ_φ be the untyped first-order language:
  - $V_{\Sigma_\varphi} = V_\Sigma$
  - $F_{\Sigma_\varphi} = F_\Sigma \cup D_\varphi$
  - $P_{\Sigma_\varphi} = P_\Sigma$
- $\varphi \in \mathcal{F}(\Sigma_\varphi) = D_\varphi^\vee \wedge D_\varphi^{\neq} \wedge F_\varphi^\wedge \wedge R_\varphi^\wedge$, where:

$$D_\varphi^\vee = \forall X \bigvee_{d_\varphi \in D_\varphi} (X = d_\varphi)$$

$$D_\varphi^{\neq} = \bigwedge_{\substack{\{d_\varphi, e_\varphi\} \subseteq D_\varphi \\ d_\varphi \neq e_\varphi}} (d_\varphi \neq e_\varphi)$$

$$F_\varphi^\wedge = \bigwedge_{\substack{(\overline{d_{I,i}} \mapsto d_I) \in f_I,\ f_I \in F_I \\ D_{\varphi\mapsto I}(d_{\varphi,i}) = d_{I,i} \\ D_{\varphi\mapsto I}(d_\varphi) = d_I}} (f(\overline{d_{\varphi,i}}) = d_\varphi)$$

$$R_\varphi^\wedge = \bigwedge_{\substack{(\overline{d_{I,i}} \mapsto true) \in p_I\ p_I \in R_I \\ D_{\varphi\mapsto I}(d_{\varphi,i}) = d_{I,i}}} p(\overline{d_{\varphi,i}}) \wedge \bigwedge_{\substack{(\overline{d_{I,i}} \mapsto false) \in p_I\ p_I \in R_I \\ D_{\varphi\mapsto I}(d_{\varphi,i}) = d_{I,i}}} \neg p(\overline{d_{\varphi,i}})$$

Let I_φ be an interpretation for Σ_φ:

- $D_{I_\varphi} = D_I$
- $F_{I_\varphi} = F_I \cup D_{\varphi\mapsto I}$
- $R_{I_\varphi} = R_I \circ D_{\varphi\mapsto I}$

Lemma. $I_\varphi \vdash \varphi$

Proof. To prove $I_\varphi \vdash \varphi$, prove $I_\varphi \vdash D_\varphi^\vee$, $I_\varphi \vdash D_\varphi^{\neq}$, $I_\varphi \vdash F_\varphi^\wedge$ and $I_\varphi \vdash R_\varphi^\wedge$:

- For every $d_{I_\varphi} \in D_{I_\varphi}$, or equivalently $d_I \in D_I$:
  - There is a $d_\varphi \in D_\varphi$ such that $D_{\varphi\mapsto I}(d_\varphi) = d_I$
  - $(X = d_\varphi) \in D_\varphi^\vee$
  - With X set to d_I
    $I_\varphi \vdash (d_I = d_\varphi)$ iff
    $d_I = F_{I_\varphi}(d_\varphi)$ iff
    $d_I = D_{\varphi\mapsto I}(d_\varphi)$
    which is *true* from the selection of d_φ

  For every $d_{I_\varphi} \in D_{I_\varphi}$, with X set to d_{I_φ}, a disjunct in $D_\varphi^\vee$ is *true*, i.e., $I_\varphi \vdash D_\varphi^\vee$
- For every $(d_\varphi \neq e_\varphi)$ in $D_\varphi^{\neq}$:
  - $I_\varphi \vdash (d_\varphi \neq e_\varphi)$ iff
    $F_{I_\varphi}(d_\varphi) \neq F_{I_\varphi}(e_\varphi)$ iff
    $D_{\varphi\mapsto I}(d_\varphi) \neq D_{\varphi\mapsto I}(e_\varphi)$ iff
    $d_I \neq e_I$
    which is *true* from the definition of D_I
  - Every inequality in $D_{\varphi_I}^{\neq}$ is *true*

  $D_\varphi^{\neq}$ is *true*, i.e., $I_\varphi \vdash D_\varphi^{\neq}$
- For every $(f(\overline{d_{\varphi,i}}) = d_\varphi)$ in $F_\varphi^\wedge$:
  - $I_\varphi \vdash (f(\overline{d_{\varphi,i}}) = d_\varphi)$ iff
    $F_{I_\varphi}(f(\overline{d_{\varphi,i}})) = F_{I_\varphi}(d_\varphi)$ iff
    $f_I(D_{\varphi\mapsto I}(\overline{d_{\varphi,i}})) = D_{\varphi\mapsto I}(d_\varphi)$ iff
    $f_I(\overline{d_{I,i}}) = d_I$
    which is *true* from the use of F_I in $F_\varphi^\wedge$
  - Every equality in $F_\varphi^\wedge$ is *true*

  $F_\varphi^\wedge$ is *true*, i.e., $I_\varphi \vdash F_\varphi^\wedge$
- For every (positive) $p(\overline{d_{\varphi,i}})$ in $R_\varphi^\wedge$:
  - $I_\varphi \vdash p(\overline{d_{\varphi,i}})$ iff
    $R_{I_\varphi}(p(\overline{d_{\varphi,i}}))$ iff
    $R_I(D_{\varphi\mapsto I}(\overline{d_{\varphi,i}}))$ iff
    $p_I(\overline{d_{I,i}})$
    which is *true* from the use of R_I in $R_\varphi^\wedge$
  - Every (positive) $p(\overline{d_{\varphi,i}})$ in $R_\varphi^\wedge$ is *true*
  - Analogously, every (negative) $\neg p(\overline{d_{\varphi,i}})$ in $R_\varphi^\wedge$ is *false*

  $R_\varphi^\wedge$ is *true*, i.e., $I_\varphi \vdash R_\varphi^\wedge$

□

Theorem. *Let $\Phi \in \mathcal{F}(\Sigma)$ be a closed formula, and let I be an interpretation for Σ. If $\varphi \models \Phi$ then $I \vdash \Phi$.*

Proof.

- If $\varphi \models \Phi$ then $I_\varphi \vdash \Phi$
  because every model of φ is a model of Φ, and I_φ is a model of φ by the **Lemma**.
- $I_\varphi \vdash \Phi$ iff $I \vdash \Phi$
  because Φ contains no symbols from D_φ, and I_φ is the same as I with respect to all other symbols.
- Thus if $\varphi \models \Phi$ then $I \vdash \Phi$.

□

5 Interpretation Visualization

Proof visualization is well-established, with several tools available, e.g., Evonne (Alrabbaa et al. 2022), ProofTree (Tews 2017), Treehehe (Battel 2018), and the Interactive Derivation Viewer (Trac, Puzis, and Sutcliffe 2007) (IDV) – a tool for visualization of TPTP format proofs. Interpretation visualization, however, has (to the knowledge of the authors) had minimal attention, as noted in Section 1. Visualization of interpretations is useful in areas such as teaching logic, debugging ATP systems, and understanding of a model.

A visualization for TF0 interpretations has been designed in this work, and an initial implementation is available as the IIV tool in SystemOnTSTP. IIV is built on top of IDV, and has benefited from the mature state of IDV. The implementation is "initial" because it is fully automated for only finite TF0 and FOF interpretations; for infinite interpretations different components of the interpretation formula currently have to be manually extracted into separate annotated formulae, to mimic a derivation that IDV can render.

Figure 6 is the visualization of the finite countermodel in Figure 3, modified so that `john` is not created equal to the person who got an `A`. The top row of inverted triangles are the types in the given formulae, while the bottom row of inverted triangles are the types of the domains. The inverted houses are the function and predicate symbols, and the successive rows of ovals are the successive domain element arguments used to specify the symbols' interpretations. Finally, the row of houses and boxes are the interpretations of the symbols applied to those arguments; houses for domain elements and boxes for truth values. Paths from leaf type nodes to root type nodes show the interpretation of symbols and the domain elements. For example, in Figure 6 the type of `grade_of` is `grade`, and `grade_of(d_john)` is interpreted as `d_f`, which is of type `d_grade` in the interpretation formula.

IIV has interactive features: In Figure 6 the cursor is hovering over the lower `d_john` node on the path from `created_equal` to `$true`, showing that `created_equal(d_john,d_john)` is interpreted as `$true`. The nodes above are increasingly darker red up to the `$o` node that is the result type of `created_equal`, and increasingly darker blue down to the `$o` node that is the type of `$true`. This highlighting provides easy focus on the interpretation of chosen symbols. This visualization is available in IIV using `https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Finite.s` as the "URL to fetch from".

Figure 7 is the visualization of the infinite model in Figure 4. Here (universally quantified) variables are used to represent an infinite number of domain elements, and builtin arithmetic predicates are used to compute symbols' mappings. The cursor is hovering over the `X:$int` node, showing how `child_of(X)` is interpreted as `$sum(X,1)`. This visualization is available in IIV using the IIV format file `https://raw.githubusercontent.com/GeoffsPapers/ModelVerification/main/TFF_Integer.s.IIV` as the "URL to fetch from" - this file was manually extracted from the infinite model in Figure 4.

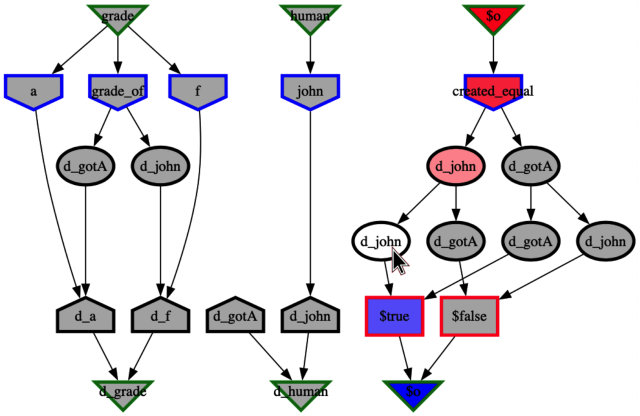

Figure 6: Visualization of the interpretation in Figure 3

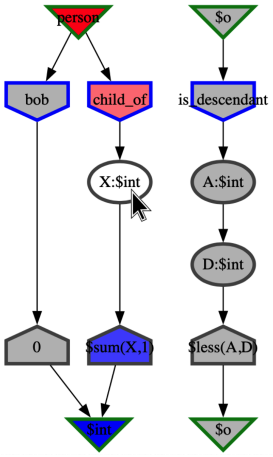

Figure 7: Visualization of the interpretation in Figure 4

6 Conclusion

This paper describes the new TPTP format for representing Tarskian-style interpretations for formulae in typed first-order logic, using the TPTP TF0 language. It further describes a technique and an implemented tool for verifying models using this representation, and a tool for visualizing interpretations. The research contributes to the advancement of automated reasoning technology for model finding, which has several applications, including verification.

Currently this work is being extended to Kripke interpretations for formula in non-classical typed first-order logic (Steen et al. 2022), using the TPTP TX0 language (Sutcliffe 2022). The tool to translate interpretation formulae to the format required for input to the IIV tool is being extended to infinite interpretations. Further inspiration might also lead to improvements to IIV's visualizations, especially for more complex infinite interpretations.

References

Alrabbaa, C.; Baader, F.; Borgwardt, S.; Dachselt, R.; Koopmann, P.; and Méndez, J. 2022. Evonne: Interactive Proof Visualization for Description Logics (System Description). In Blanchette, J.; Kovacs, L.; and Pattinson, D., eds., *Proceedings of the 11th International Joint Conference on Automated Reasoning*, number 13385 in Lecture Notes in Artificial Intelligence, 271–280.

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

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

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

Battel, C. 2018. Treehehe: An interactive visualization of proof trees. https://github.com/seachel/treehehe.

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

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

Cohn, A. 1987. A More Expressive Formulation of Many Sorted Logic. *Journal of Automated Reasoning* 3(2):113–200.

de Moura, L., and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In Ramakrishnan, C., and Rehof, J., eds., *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 4963 in Lecture Notes in Artificial Intelligence, 337–340. Springer-Verlag.

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

Herbrand, J. 1930. 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.

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

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

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

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

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

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

McCune, W. 2003b. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA.

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

Schlyter, C. 2013. Visualization of a Finite First Order Logic Model. Master's thesis, Department of Computer Science and Engineering, University of Gothenburg, Göteborg, Sweden.

Schmidt-Schauss, M. 1985. A Many-Sorted Calculus with Polymorphic Functions Based on Resolution and Paramodulation. In A., J., ed., *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 1162–1168. IJCAI Organization.

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

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

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

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

Sutcliffe, G. 2006. Semantic Derivation Verification: Techniques and Implementation. *International Journal on Artificial Intelligence Tools* 15(6):1053–1070.

Sutcliffe, G. 2007. TPTP, TSTP, CASC, etc. In Diekert, V.; Volkov, M.; and Voronkov, A., eds., *Proceedings of the 2nd*

International Symposium on Computer Science in Russia, number 4649 in Lecture Notes in Computer Science, 6–22. Springer-Verlag.

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

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

Sutcliffe, G. 2022. The Logic Languages of the TPTP World. *Logic Journal of the IGPL* https://doi.org/10.1093/jigpal/jzac068.

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

Tews, H. 2017. Proof tree visualization for proof general. http://askra.de/software/prooftree/.

Trac, S.; Puzis, Y.; and Sutcliffe, G. 2007. An Interactive Derivation Viewer. In Autexier, S., and Benzmüller, C., eds., *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers*, volume 174 of *Electronic Notes in Theoretical Computer Science*, 109–123.

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

Walther, C. 1983. A Many-Sorted Calculus Based on Resolution and Paramodulation. In A., B., ed., *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 882–891.

Winker, S. 1982. 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.