An Interactive Interpretation Viewer for Typed First-order Logic

Jack McKeown
University of Miami
USA

Geoff Sutcliffe
University of Miami
USA

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 (to become a theorem) are written in an appropriately expressive logic, and the proofs are often similarly written in logic (Sutcliffe et al. 2006). In the last two decades the converse task of disproving conjectures has become increasingly important. This is achieved by finding an *interpretation* – a structure that maps terms to domain elements and formulae to truth values, that is a *model* of the axioms – it maps the axioms to *true*, and a *countermodel* of the conjecture – it maps the conjecture to *false* (or equivalently, it is a model of the negated conjecture). A salient application area that harnesses this form of ATP is verification (D'Silva, Kroening, and Weissenbacher 2008). This work describes an interactive interpretation viewer for finite interpretations written in the (new) TPTP format for interpretations (Steen et al. 2022), for formulae in typed first-order logic.

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 language (Sutcliffe 2022) is used for writing both problems and solutions (derivations and interpretations). 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 *languages* supported are `cnf` (clause normal form), `tcf` (typed clause normal form), `fof` (first-order form), `tff` (typed first-order form), and `thf` (typed higher-order form). The *role*, e.g., `type`, `axiom`, `conjecture`, defines the use of the formula in an ATP system. The *formula* follows Prolog conventions, and can additionally include interpreted symbols that start with a `$`, e.g., `$true` and its boolean type `$o`. The 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. Figure 1 is an example of a problem (not a theorem!) in monomorphic typed first-order form (TF0).


```
%------------------------------------------------------------------------------
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(three_cats,axiom,
    $distinct(garfield,arlene,nermal) ).

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

Figure 1: A TF0 problem

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 (just one domain for untyped logic), and interpretations of the function and predicate symbols with respect to the domains (Gallier 2015). 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). This work deals with only enumerated finite domains. The TPTP representation of an interpretation uses an *interpretation formula*, preceded by the necessary type declarations. The interpretation formula is a conjunction providing details of the domains - their types and elements, and the interpretation of the symbols applied to domain elements. Type-promotion functions are used to convert domain elements to terms, to make the interpretation formula well-typed. The representation is also usable for untyped first-order logic, where all terms in both the given and interpretation formulae are of the same type – "individuals"., which obviates the need for type considerations, in particular type-promotion functions are not needed.

Figure 2 is a TF0 interpretation with finite domains – it is a countermodel for the problem in Figure 1. The type declarations have been omitted, and can be found in the URL provided.

```
%------------------------------------------------------------------------
tff(garfield,interpretation,
%----The domain for 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 )
%----The domain for cat
      & ! [C: cat] : ? [DC: d_cat] : C = d2cat(DC)
      & ! [DC: d_cat]: ( DC = d_garfield | DC = d_arlene )
      & $distinct(d_garfield,d_arlene,d_nermal)
      & ! [DC1: d_cat,DC2: d_cat] :
          ( d2cat(DC1) = d2cat(DC2) => DC1 = DC2 ) )
%----Interpret terms and atoms
    & ( 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)) ) ) ).
%------------------------------------------------------------------------
```

Figure 2: A TF0 countermodel for the problem in Figure 1
https://raw.githubusercontent.com/GeoffsPapers/IIVPoster/main/TFF_Finite.s

4 Interpretation Visualization

Proof visualization is well-established, with several tools available, including the Interactive Derivation Viewer (IDV) – a tool for visualizing TPTP format proofs (Trac, Puzis, and Sutcliffe 2007). Interpretation visualization, however, has (to the knowledge of the authors) had minimal attention, with (Schlyter 2013) being the only tool found (past tense – it is no longer available). Visualization of interpretations is useful in areas such as teaching logic, debugging ATP systems, and understanding a model. A visualization for TF0 interpretations has been designed in this work, and it is available as the IIV tool in the SystemOnTSTP web interface[1].

Figure 3 is the visualization of the interpretation in Figure 2. The top row of inverted triangles are the types in the problem formulae, while the bottom row of inverted triangles are the types of the domains in the interpretation formula. The inverted houses are the function and predicate symbols, and the successive rows of ovals are the successive domain element arguments used in the symbols' interpretations. Finally, the row of houses and boxes are the interpretations of the symbols applied to those arguments; houses for functions and boxes for predicates. Paths from leaf to root nodes show the interpretations of the symbols applied to the domain elements. For example, in the given formulae the type of `loves` is `cat`, and `loves(d_arlene)` is interpreted as `d_garfield`, which is of type `d_cat` in the interpretation formula.

IIV provides some interactive features: Figure 3 shows the situation with the cursor hovering over the `d_garfield` node on the path from `owns` to `$true`, showing that `owns(d_jon,d_garfield)` is interpreted as `$true`. The nodes above are increasingly darker red (grey if printed) up to the type node `$o` that is the result type of `owns`, and increasingly darker blue down to the type node `$o` that is the type of `$true`. This highlighting provides easy focus on the interpretations of chosen symbols, e.g., it is easy to highlight what symbol applications are interpreted as `$true` or `$false` by hovering over the corresponding truth value node, or how a specific function symbol is interpreted, e.g, by hovering over the `loves` node.

This visualization is available in IIV using the URL provided with Figure 2 as the "URL to fetch from", selecting `IIV 0.0` as the "System", and clicking the "Process Solution" button.

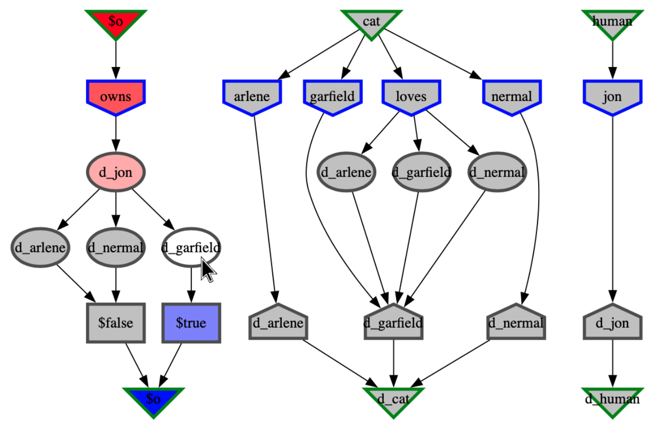

Figure 3: Visualization of the interpretation in Figure 2

5 IIV Implementation

IIV is implemented on top of IDV, and has benefited from the mature state of IDV. Different terms in an interpretation, e.g., problem types, problem symbols, domain elements, domain types, are extracted into annotated formulae with different languages and roles that IDV renders differently: types are extracted into `fof` annotated formulae with the role `axiom` that are rendered as inverted triangles with a green outline; problem symbols are extracted into `thf negated_conjectures` that are rendered as inverted houses with a blue outline; domain element arguments for problem symbols are extracted into `tcf plains` that are rendered as ellipses with a grey outline; domain element interpretations for function symbols are extracted into `tcf conjectures` that are rendered as houses with a grey outline; truth value interpretations for predicate symbols are extracted into `tcf conjectures` that are rendered as boxes (as a special case) with a grey outline. A "derivation" is formed by setting the "inference" parents of each node appropriately. In order to separate nodes for different kinds of terms into different layers, a `level` term is added to the source field of each derivation annotated formula, which is used by IDV only when rendering interpretations.

A Prolog program is used to extract the components of the interpretation formulae into the separate derivation annotated formulae. ANTLR is used to generate a JavaScript parser for the TPTP language. The parser is used in the browser, along with custom JavaScript, to create a graph in the Graphviz DOT language (Ellson et al. 2002) from the TPTP input (in this case, representing an interpretation). The resulting DOT text is then rendered using the "d3-graphviz"[2] library, which uses WebAssembly for speed.

6 Future Work

Further inspiration might lead to improvements to IDV, especially for infinite interpretations and Kripke interpretations (Kripke 1963) for non-classical logics.

References

D'Silva, V.; Kroening, D.; and Weissenbacher, G. 2008. A Survey of Automated Techniques for Formal Software Verification. *IEEE*

[1] https://www.tptp.org/cgi-bin/SystemOnTSTP

[2] https://github.com/magjac/d3-graphviz

Transactions on Computer-aided Design of Integrated Circuits and Systems 27(7):1165–1178.

Ellson, J.; Gansner, E.; Koutsofios, L.; North, S.; and Woodhull, G. 2002. Graphviz - Open Source Graph Drawing Tools. In Mutzel, P.; Jünger, M.; and Leipert, S., eds., *Proceedings of the 9th International Symposium on Graph Drawing*, number 2265 in Lecture Notes in Computer Science, 483–484. Springer-Verlag.

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

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.

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

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.

Steen, A.; Sutcliffe, G.; Fontaine, P.; ; and McKeown, J. 2022. Representation, Verification, and Visualization of Tarskian Interpretations for Typed First-order Logic. In Keshtkar, F., and Bell, E., eds., *Proceedings of the 36th International FLAIRS Conference*, Submitted. Florida Online Journals.

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. 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.

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.