EPiC Series in Computing

Volume 40, 2016, Pages 106–121

IWIL-2015. 11th International Workshop on the Implementation of Logics

The Thousands of Models for Theorem Provers (TMTP) Model Library - First Steps

Geoff Sutcliffe[1] and Stephan Schulz[2]

[1] University of Miami, USA
[2] DHBW Stuttgart, Germany

Abstract

The TPTP World is a well established infrastructure that supports research, development, and deployment of Automated Theorem Proving (ATP) systems for classical logics. 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). This work describes a new component of the TPTP World - the Thousands of Models for Theorem Provers (TMTP) Model Library. This is a corpus of models for identified sets of axioms in the TPTP, along with functions for interpreting formulae wrt models, interfaces for visualizing interpretations, tools for verifying models, etc. The TMTP supports the development of semantically guided theorem proving ATP systems, provides models that can be used to guide axioms selection in large theories, and provides insights into the semantic structure of axiomatizations.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development of automatic techniques and computer programs for checking whether a conjecture is a theorem of some axioms, and for checking the consistency of a set of formulae. The TPTP World [21] is a well established infrastructure that supports research, development, and deployment of ATP systems for classical logics. The TPTP world includes the TPTP problem library [20], the TSTP solution library [21], standards for writing ATP problems and reporting ATP solutions [24, 19], tools and services for processing ATP problems and solutions [21], and it supports the CADE ATP System Competition (CASC) [22]. The TPTP world infrastructure has been deployed in a range of applications, in both academia and industry. The web page http://www.tptp.org provides access to all components.

The Thousands of Problems for Theorem Provers (TPTP) problem library is the original core component of the TPTP world, and is commonly referred to as "the TPTP". The TPTP problem library supplies the ATP community with a comprehensive library of the test problems that are available today, in order to provide an overview and a simple, unambiguous reference mechanism, to support the testing and evaluation of ATP systems, and to help ensure that performance results accurately reflect capabilities of the ATP systems being considered. The

Thousands of Solutions from Theorem Provers (TSTP) solution library is the "flip side" of the TPTP – a corpus of ATP systems' solutions to TPTP problems. A major use of the TSTP is for ATP system developers to examine solutions to problems, and thus understand how they can be solved, leading to improvements to their own systems. The TPTP language is one of the keys to the success of the TPTP world. The language is used for writing both TPTP problems and TSTP solutions, which enables convenient communication between different systems and researchers. In conjunction with the TPTP language, the TPTP world uses the SZS[1] ontologies to record what is known or has been established about a TPTP problem or solution, and to describe sets of formulae. The TPTP world includes tools, programming libraries, and online services that are used to support the application and deployment of ATP systems. One of the most used services is SystemOnTPTP [17], which is an online service that allows an ATP problem or solution to be easily and quickly submitted in various ways to a range of ATP systems and tools. An important tool is GDV [18], which verifies TPTP format derivations. A very useful tool for human users is IDV [26], which provides an interactive interface for viewing TPTP format derivations.

This work describes a new component of the TPTP world - the Thousands of Models for Theorem Provers (TMTP) Model Library.[2] This is a library of models for axiomatizations built from axiom sets in the TPTP. The library is supported by functions for efficiently interpreting ground terms and closed formulae wrt interpretations, and tools for examining and processing interpretations. The components parallel those already in the TPTP world for ATP problems and solutions. Details of the TMTP's components are provided in this paper, but in summary . . . The TMTP model library is similar to the TPTP problem library and TSTP solution library. The TPTP language is used for writing the models (they are one type of solution), and the SZS ontology is used to describe the various kinds of interpretations. The SystemOnTMTP service allows an interpretation to be submitted to various tools for evaluating formulae wrt the interpretation, and for examining and processing the interpretation. The IMV tool provides an interactive interface for viewing interpretations, and the GMV tool verifies that an interpretation is a model for a given set of formulae. The web page http://www.tptp.org/TMTP provides access to the TMTP.

The TMTP provides support for the development and execution of semantically guided ATP systems, in the style of SLM [6], SGLD [16], and SCOTT [15], which use one or more preselected interpretations to guide their search. (This is in contrast to ATP systems that use models that are computed or updated during their execution, e.g., iProver [10], Satallax [5], and CVC4 [3].) An implementation of semantic resolution [14] is planned. It is noteworthy that the dates of the publications cited here are rather old, and it is the first author's opinion that the potential for semantically guided ATP has not been fully exploited. This viewpoint was also expressed in [7]. The TMTP provides support for semantically guided techniques for axiom selection in large theories, in the style of SRASS [23]. The TMTP provides a basis for empirical research into the semantics of axiomatizations, hopefully leading to insights that benefit ATP research and development in general. Finally, while the TMTP does provide examples of solutions to satisfiable and countersatisfiable ATP problems[3], it is not intended to be a comprehensive repository of justifications for ATP systems' claims of satisfiability or countersatisfiability – that is the purpose of the TSTP solution library, which might provide justifications other than models, e.g., a claim of countersatisfiability can be justified by a proof showing that a conjecture is a countertheorem of the axioms.

[1]SZS is an acronym from the initials of the original authors' family names [25].

[2]Not all of these components have been completely implemented at the time of writing.

[3]The SZS ontology supplies the definitions of status values such as "theorem", "countertheorem", "satisfiable", "countersatisfiable".

The remainder of this paper is organized as follows: This section ends with definitions for the terminology used in this paper. Section 2 explains how the models in the TMTP are collected, and describes formats for writing various kinds of interpretations using the TPTP language. Section 3 describes some tools that have been developed for examining and processing interpretations. Section 4 concludes, and outlines plans for further development of the TMTP.

Terminology: A (logical) *language* is defined in the usual way [2], with variables, functions, and predicates. An *interpretation* for a language $\mathcal{L}$ is a structure that has a domain $\mathbb{D}$, and can interpret all ground terms $\mathcal{T}$ in $\mathcal{L}$ as elements of $\mathbb{D}$, and all closed formula $\mathcal{F}$ in $\mathcal{L}$ as either *true* or $false$. A *model* of a set $\mathcal{S}$ of closed formulae is an interpretation that interprets all the formulae in $\mathcal{S}$ as *true*. An interpretation is *complete* in the sense that it can interpret all ground terms and closed formulae in $\mathcal{L}$. A *partial interpretation* for $\mathcal{L}$ can interpret some (not necessarily all, but possibly all in which case it is complete) ground terms and closed formulae in $\mathcal{L}$. A *strictly partial interpretation* for $\mathcal{L}$ cannot interpret all ground terms and closed formulae in $\mathcal{L}$. A strictly partial interpretation can be complete for ground terms or closed formulae, but not for both. A strictly partial interpretation is *complete for a set* $\mathcal{S}$ of ground terms and closed formulae if it can interpret all the elements in the set. A *strictly partial model* of a set $\mathcal{S}$ of closed formulae is a strictly partial interpretation that is complete for $\mathcal{S}$, and interprets all the formulae in $\mathcal{S}$ as *true*.

2 Collecting Models

TMTP models (both complete and strictly partial) of a language $\mathcal{L}$ are expressed using the TPTP language, as a set of TPTP formulae. Four kinds of interpretations are currently defined for the TMTP: Herbrand interpretations, finite interpretations, integer interpretations, and real interpretations. These types have been categorized in the SZS dataform ontology, along with the notions of complete, partial, and strictly partial interpretations. The relevant section of the ontology is shown in Figure 1.[4] Examples of ontology values and their three-letter acronyms include "Interpretation (Int)" at the top of the hierarchy, "Herbrand Strictly partial Model (HSM)" in the right branch, and "Integer Partial Interpretation (IPI)" at bottom middle of the left branch. Full details of each possible value are given in Appendix A. The lines in the ontology can be followed up the hierarchy as "isa" links, e.g., an Integer Partial Interpretation (IPI) isa Domain Partial Interpretation (DPI) isa Partial Interpretation (PIn) isa Interpretation (Int). The classification of an interpretation into the ontology can be partially automated, e.g., if a finite domain is used then the interpretation is somewhere on the lefthand branch. More precise placement, e.g., specifying that the interpretation is complete, partial, or strictly partial, normally requires information from the ATP system that produced the interpretation. This information is important for some uses of interpretations, in particular for using deduction to interpret formulae wrt an interpretation (see Section 3.1).

2.1 Herbrand Interpretations

Herbrand interpretations are represented by sets of TPTP formulae that define the subset of the Herbrand base that is *true*, and a formula is evaluated wrt a Herbrand interpretation by trying to prove the formula is a theorem of the interpretation's formulae (see Section 3.1).

[4]The figure conflates the analogous trees for Interpretations and Models, and similarly the analogous subtrees for the Partial and Strictly partial cases. It could be expanded into a taxonomic tree. Each path from the "Interpretation" root to a leaf of this figure is one branch of the taxonomic tree.

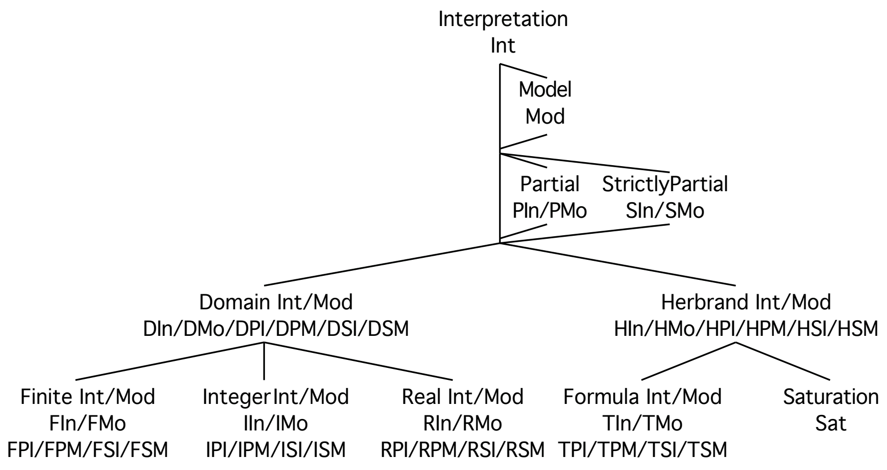


Figure 1: SZS Ontology for Interpretations

Figure 2 provides an example defining Herbrand model for PUZ001-3, produced by the iProver ATP system [10]. Examples of ATP systems that generate Herbrand models include iProver and Darwin [4].

```
%------ Negative definition of lives
fof(lives_defn,axiom,(
    ! [X0] : ( ~ lives(X0) <=> $false ) )).

%------ Positive definition of killed
fof(killed_defn,axiom,(
    ! [X0,X1] :
      ( killed(X0,X1)
    <=> ( X0 = agatha & X1 = agatha ) ) )).

%------ Positive definition of richer
fof(richer_defn,axiom,(
    ! [X0,X1] :
      ( richer(X0,X1)
    <=> ( X0 = butler & X1 = agatha ) ) )).

%------ Negative definition of hates
fof(hates_defn,axiom,(
    ! [X0,X1] :
      ( ~ hates(X0,X1)
    <=> ( ( X0 = agatha & X1 = butler )
        | ( X0 = butler & X1 = butler )
        | X0 = charles
        | ( X1 = butler & X0 != butler ) ) ) )).
```

Figure 2: A Herbrand Model for PUZ001-3

Saturations are a special case. The existence of a saturation (with respect to a complete calculus) guarantees the existence of at least one Herbrand model. However, the model is not, in general, uniquely defined on the logical level alone. For superposition-based calculi, the saturation defines a unique Herbrand model for a given literal selection strategy and term ordering. If the term ordering (and the induced ordering on literals and clauses) is not only terminating, but also has length-bounded descending chains (as in the case of the frequently used Knuth-Bendix ordering), Bachmair/Ganzinger style bottom-up model construction [1] can be used to interpret arbitrary ground atoms. However, the process is unlikely to be efficient in the general case, although it is plausible for, e.g., EPR. Using a saturation to interpret a formula by trying to prove the formula from the saturation (see Section 3.1) is possible, but cannot always succeed even in principle, since the model is not always uniquely defined. Moreover, it is necessary to use an ATP system that implements exactly the same ordering and redundancy criteria that were used to produce the saturation, i.e., in practice the ATP system that found the saturation in the first place.

Figure 3 provides an example saturation, for the TPTP problem PUZ001-3, produced by the E ATP system [13]. It is noteworthy that the saturation is expressed as a set of clauses in TPTP format, but it does not specify the ordering and redundancy criteria that were used. It will be necessary to extend the TPTP presentation to include this information. Examples of ATP systems that generate saturations include E, SPASS [27], and Vampire [11].

```
cnf(c_0_19,plain, ( hates(butler,butler) | killed(agatha,agatha) )).
cnf(c_0_20,plain, ( hates(butler,charles) | ~ killed(charles,agatha) )).
cnf(c_0_21,plain, ( hates(butler,X1) | richer(X1,agatha) | ~ lives(X1) )).
cnf(c_0_22,plain, ( hates(X1,X2) | ~ killed(X1,X2) )).
cnf(c_0_23,plain, ( ~ hates(butler,butler) | ~ hates(butler,charles) )).
cnf(c_0_24,plain, ( ~ hates(X1,agatha) | ~ hates(X1,butler) |
                    ~ hates(X1,charles) )).
cnf(c_0_25,plain, ( hates(butler,X1) | ~ hates(agatha,X1) )).
cnf(c_0_26,plain, ( ~ richer(X1,X2) | ~ killed(X1,X2) )).
cnf(c_0_27,plain, ( ~ hates(agatha,X1) | ~ hates(charles,X1) )).
cnf(c_0_28,plain, ( ~ hates(agatha,butler) )).
cnf(c_0_29,plain, ( ~ hates(charles,charles) )).
cnf(c_0_30,plain, ( hates(butler,agatha) )).
cnf(c_0_31,plain, ( hates(agatha,charles) )).
cnf(c_0_32,plain, ( hates(agatha,agatha) )).
cnf(c_0_33,plain, ( lives(charles) )).
cnf(c_0_34,plain, ( lives(butler) )).
cnf(c_0_35,plain, ( lives(agatha) )).
```

Figure 3: A Saturation for PUZ001-3

2.2 Finite Interpretations

Finite interpretations [24] are represented by FOF that specify the domain, define the interpretation of the functions, and define the interpretation of the predicates. The elements of the domain are specified in a formula of the form:

```
fof(fi_name,fi_domain,
    ! [X] : ( X = e1 | X = e2 | ... | X = en ) ).
```

where the e_i are all "distinct object"s, or all distinct integers, or all distinct constants. The use of "distinct object"s or integers for a domain is preferred over constants, because they are predefined to be unequal. If the domain elements are constants then their inequality must be explicitly stated in formulae of the form:

```
    fof(e_i_not_e_j,fi_domain,
        e_i != e_j ).
```

The interpretation of functors is written in the form:

```
    fof(fi_name,fi_functors,
        ( f(e_1,...,e_m) = e_r
        & f(e_1,...,e_p) = e_s
          ...  )  ).
```

specifying that, e.g., f(e_1,...,e_m) is interpreted as the domain element e_r. The interpretation of predicates is written in the form:

```
    fof(fi_name,fi_predicates,
        ( p(e_1,...,e_m)
        & ~ p(e_1,...,e_p)
          ...  )  ).
```

specifying that, e.g., p(e_1,...,e_m) is interpreted as *true* and p(e_1,...,e_p) is interpreted as *false*. Equality is interpreted naturally by the domain, with the understanding that identical elements are equal. For the interpretation of functors and predicates, universal quantifications can be used if all domain elements can be used in an argument position, e.g.,

```
    fof(fi_name,fi_functors,
        ( ! [X] : ( f(e_1,...,X) = e_r
        & f(e_1,...,e_p) = e_s
          ...  )  ).
```

specifies that the last argument position can be any domain element in the functions interpreted as e_r.

Figure 4 provides an example set of formulae that has a finite model, and Figure 5 provides an example finite model for the formulae, produced by the ATP system Paradox [8]. In the model the interpretation of the predicate p can be specified in two ways, one using a universal quantifier, and the other explicitly using the domain elements. The difference is relevant when interpreting formulae wrt an interpretation, as discussed in Section 3.1. Examples of ATP systems that generate finite models include Paradox, Mace4 [12] and DarwinFM [4].

```
%----About the constants
fof(a_not_b,   axiom, a != b ).

%----About the functions
fof(s_not_X,   axiom, ! [X] : s(X) != X ).
fof(f_b_a,     axiom, f(b) = a ).
fof(f_ss_X,    axiom, ! [X] : f(s(s(X))) = X ).

%----About the predicates
fof(p_a,       axiom, p(a) ).
```

Figure 4: Example Axiomatization that has a Finite Model

```
%----Model domain
fof(domain,fi_domain, ! [X] : ( X = "d1" | X = "d2" | X = "d3" ) ).

%----Constants
fof(a,   fi_functors, a = "d1" ).
fof(b,   fi_functors, b = "d2" ).

%----Total functions
fof(f,   fi_functors, f("d1") = "d3" & f("d2") = "d1" & f("d3") = "d2" ).
fof(s,   fi_functors, s("d1") = "d3" & s("d2") = "d1" & s("d3") = "d2" ).

%----Total predicates - Universal quantification
%---- fof(p, fi_predicates, ! [X1] : p(X1) <=> $true ).

%----Total predicates - Listed
fof(p, fi_predicates, p("d1") & p("d2") & p("d3") ).
```

Figure 5: A Finite Model for Figure 4

2.3 Integer and Real Interpretations

The domain of an integer interpretation is the integers, as defined by the `$int` type of the TPTP's TFF language. The interpretations are then represented by three types of TFF formulae: type declarations, formulae to define the interpretation of the functions, and formulae to define the interpretation of the predicates. The type declarations declare all constants to be of type `$int`, all functions to be from tuples of `$int` to `$int`, and all predicates to be from tuples of `$int` to `$o`. Thus every constant and function is interpreted as an element of the integer domain, the formulae that define the interpretation of the functions map tuples of domain elements to a domain element, and the formulae that define the interpretation of the predicates map tuples of domain elements to *true* or *false*.

Figure 6 provides an example set of formulae (modified from those in Figure 4) that does not have a finite model, and Figure 7 provides an example integer model for the formulae. Examples of ATP systems that generate integer models include CVC4 [3] and Z3 [9].

```
%----About the constants
fof(a_not_b,   axiom, a != b ).

%----About the functions
fof(bigger_s,  axiom, ! [X] : bigger(s(X),X) ).
fof(bigger_t,  axiom, ! [X,Y] : ( bigger(X,Y) => bigger(s(X),Y) ) ).
fof(s_not_X,   axiom, ! [X,Y] : ( bigger(X,Y) => X != Y ) ).
fof(f_b_a,     axiom, f(b) = a ).
fof(f_ss_X,    axiom, ! [X] : f(s(s(X))) = X ).

%----About the predicates
fof(p_a,       axiom, p(a) ).
```

Figure 6: Example Axiomatization that does not have a Finite Model

```
%----Model types
tff(a_type,    type,  a: $int ).
tff(b_type,    type,  b: $int ).
tff(s_type,    type,  s: $int > $int ).
tff(f_type,    type,  f: $int > $int ).
tff(b_type,    type,  bigger: ( $int * $int ) > $o ).
tff(p_type,    type,  p: $int > $o ).

%----Constants
tff(a_is_1,    axiom, a = 1 ).
tff(b_is_1,    axiom, b = 4 ).

%----Total functions
tff(s,         axiom, ! [X: $int] : s(X) = $product(X,2) ).
tff(f_s,       axiom, ! [X: $int] : f(X) = $quotient_t(X,4) ).

%----Total predicates
tff(bigger,    axiom, ! [X: $int,Y: $int] :
                        ( bigger(X,Y) <=> $greater(X,Y) ) ).
tff(p_natural, axiom, ! [X: $int] : ( p(X) <= $greatereq(X,1) )).
tff(not_p_more,axiom, ! [X: $int] : ( ~ p(X) <= $less(X,1) )).
```

Figure 7: An Integer Model for Figure 6

The domain of a real interpretation is the reals, as defined by the `$real` type of the TPTP's TFF language. The interpretations are then represented by three types of TFF formulae, analogous to integer interpretations, but using the `$real` type. Figure 8 provides an example real model for the formulae of Figure 6. Example of ATP systems that generate real models are CVC4 and Z3.

2.4 Building the TMTP

The TPTP problem library includes satisfiable *axiomatization problems* that consist of only `include` directives for TPTP axiom files (no problem-specific formulae), so that the axioms constitute an axiomatization of some recognized theory. For example, `PHI001^1` consists of

```
%----Axioms for Quantified Modal Logic KB.
include('Axioms/LCL016^0.ax').
include('Axioms/LCL016^1.ax').
%----Axioms about God
include('Axioms/PHI001^0.ax').
```

The TMTP is built by collecting solutions to axiomatization problems, i.e., their models from the TSTP solution library. As new (versions of) ATP systems are added to the TPTP world, they are run over all the problems in the TPTP, and their solutions are added to the TSTP. In the case of a new version of an ATP system, the old version's solutions are replaced by the new version's solutions in the TSTP. The new systems' models for axiomatization problems are new candidates for addition to the TMTP. Each such new model is checked against existing models in the TMTP, and if it is not a syntactic variant of an existing model it is copied into

```
%----Model types
tff(a_type,    type,  a: $real ).
tff(b_type,    type,  b: $real ).
tff(s_type,    type,  s: $real > $real ).
tff(f_type,    type,  f: $real > $real ).
tff(b_type,    type,  bigger: ( $real * $real ) > $o ).
tff(p_type,    type,  p: $real > $o ).

%----Constants
tff(a_is_1,    axiom, a = 1.0 ).
tff(b_is_1_4,  axiom, b = 0.25 ).

%----Total functions
tff(s,         axiom, ! [X: $real] : s(X) = $quotient(X,2) ).
tff(f_s,       axiom, ! [X: $real] : f(X) = $product(X,4) ).

%----Total predicates
tff(bigger,    axiom, ! [X: $real,Y: $real] :
                        ( bigger(X,Y) <=> $greater(X,Y) ) ).
tff(p_natural, axiom, ! [X: $real] : ( p(X) <= $greatereq(X,1) )).
tff(not_p_more,axiom, ! [X: $real] : ( ~ p(X) <= $less(X,1) )).
```

Figure 8: A Real Model for Figure 6

the TMTP. In this way multiple models of the TPTP axiomatization are collected. At the time of writing, many more potential axiomatization problems have been identified for addition to the TPTP, and subsequently their models will be added to the TMTP.

It is important to ensure that non-trivial models are produced and included in the TMTP. For example, the axiomatization of the natural numbers $\{int(zero), \forall X(int(X) \Rightarrow int(succ(X)))\}$ has a trivial interpretation with a single domain element $d1$, with $zero$ and $succ(d1)$ both mapping to $d1$, and $int(d1)$ mapping to $true$. The intended integer interpretation that should (also) be in the TMTP would have the normal interpretation of $succ$ as the successor function.

The TMTP has a naming scheme similar to that used for problems in the TPTP. The model naming scheme is *DDDNNNFV.MMM-SZS*. *DDD* is the TPTP domain acronym (`ALG`, `PUZ`, `SET`, etc.), *NNN* is the abstract problem number, *F* is the logical form (`^` for THF, `=` for TFF with arithmetic, `_` for TFF without arithmetic, `+` for FOF, and `-` for CNF), and *V* is the problem version number, so that *DDDNNNFV* is the name of a TPTP axiomatization problem. *MMM* is the model number for that axiomatization, and *SZS* is the SZS dataform (`FMo`, `Sat`, etc.). Thus an example TMTP model name is `KRS176+1.005-FMo`. An extension of `.m` is added to create the model file name.

Each model file consists of a header containing information about the ATP system that built the model, the computing resources used to build the model, statistics about the model, and user comments. The formulae that define the model follow below the header. Figure 9 shows an example TMTP model file (with the bulk for the defining formulae omitted).

```
%------------------------------------------------------------------------------
% File       : Paradox---4.0
% Problem    : KRS176+1 : TPTP v6.2.0. Released v4.0.0.
% Transform  : none
% Format     : tptp:short
% Command    : paradox --no-progress --time %d --tstp --model %s

% Computer   : n189.star.cs.uiowa.edu
% Model      : x86_64 x86_64
% CPU        : Intel(R) Xeon(R) CPU E5-2609 0 2.40GHz
% Memory     : 32286.75MB
% OS         : Linux 2.6.32-573.1.1.el6.x86_64
% CPULimit   : 300s
% DateTime   : Wed Aug  5 05:31:13 EDT 2015

% Result     : Satisfiable 0.01s
% Output     : FiniteModel 0.01s
% Verified   :
% Statistics : Number of formulae       :   63 (  63 expanded)
%              Number of leaves         :   63 (  63 expanded)
%              Depth                    :    0
%              Number of atoms          :  149 ( 149 expanded)
%              Number of equality atoms :  125 ( 125 expanded)
%              Maximal formula depth    :    9 (   2 average)
%              Maximal term depth       :    2 (   1 average)

% Comments   :
%------------------------------------------------------------------------------
% domain size is 2
fof(domain,fi_domain,(
    ! [X] : ( X = "1" | X = "2" ) )).

fof(cax,fi_functors,(
    cax = "2" )).

    ...

fof(model,fi_predicates,
    ( ( model("1","1") <=> $false )
    & ( model("1","2") <=> $false )
    & ( model("2","1") <=> $false )
    & ( model("2","2") <=> $false ) )).

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

Figure 9: A TMTP Model File

3 Examining and Processing the TMTP Models

3.1 Interpreting Closed Formulae wrt an Interpretation

A key capability required for using interpretations (and thus also the TMTP models) is efficiently interpreting closed formulae wrt the interpretations. Direct interpreting formulae according to the semantic rules of the logic often provides this, but the feasibility depends on the nature of the interpretation. For finite interpretations directly interpreting formulae is relatively easy, by instantiating quantified variables with domain elements, using the function definitions to interpret functors applied to domain elements, and using the predicate definitions to interpret predicates applied to domain elements. If universal quantification is used in the predicate definitions, e.g., as in the commented out definition in Figure 5, this can be used to short-circuit the interpretation process. For example, the atom `p(a)` could be immediately interpreted as *true* without having to first interpret `a` as `"d1"`. For Herbrand, integer, and real interpretations, direct evaluation of anything other than ground terms and atoms seems tricky (which makes a nice research problem).

An alternative to directly interpreting formulae is to rely on deduction, using the interpretation as axioms, and the formula to be interpreted (or its negation) as a conjecture. This works because, in this setting, logical consequence and directly interpreting formulae coincide in many cases. If $\mathcal{I}$ is a partial interpretation for $\mathcal{L}$, then if $\mathcal{F}$ (in $\mathcal{L}$) is a theorem of (the formulae that define) $\mathcal{I}$ then $\mathcal{I}$ interprets $\mathcal{F}$ as $true$. If $\mathcal{F}$ is a countertheorem of $\mathcal{I}$ (or equivalently, ~$\mathcal{F}$ is a theorem of $\mathcal{I}$), then $\mathcal{I}$ interprets $\mathcal{F}$ as $false$. Further, if $\mathcal{I}$ is an interpretation (i.e., is complete), and $\mathcal{F}$ is countersatisfiable wrt $\mathcal{I}$ (or equivalently ~$\mathcal{F}$ is satisfiable with $\mathcal{I}$), then $\mathcal{F}$ is necessarily a countertheorem of $\mathcal{I}$, and $\mathcal{I}$ interprets $\mathcal{F}$ as $false$. Finally, if $\mathcal{I}$ is an interpretation, and $\mathcal{F}$ is satisfiable wrt $\mathcal{I}$ (or equivalently ~$\mathcal{F}$ is countersatisfiable with $\mathcal{I}$), then ~$\mathcal{F}$ is necessarily a countertheorem of $\mathcal{I}$, and $\mathcal{F}$ is interpreted as $true$. Note that if $\mathcal{I}$ is a strictly partial interpretation for $\mathcal{L}$ and $\mathcal{F}$ is countersatisfiable wrt $\mathcal{I}$, then nothing can be concluded about the interpretation of $\mathcal{F}$. An example to illustrate this is as follows: Let $\mathcal{L} = [\{a/0, b/0, c/0\}, \{p/1\}]$, and let $\mathcal{I} = \{a \neq b, p(c)\}$. $\mathcal{I}$ is a strictly partial interpretation of $\mathcal{L}$, e.g., it does not interpret the closed formula $\mathcal{F} \equiv b \neq c$. $\mathcal{F}$ is countersatisfiable wrt $\mathcal{I}$, but $\mathcal{F}$ is not a countertheorem of $\mathcal{I}$. These observations provide a simple, but possibly inefficient, way of interpreting a formula wrt a TMTP format interpretation – use an ATP system to determine the status of the formula wrt the model (theorem, countersatisfiable, countertheorem, satisfiable), and hence determine the interpretation of the formula wrt the model ($true$, $false$, $false$, $true$, respectively). An efficient implementation of this idea would try to prove both $\mathcal{F}$ and ~$\mathcal{F}$ from the interpretation in parallel.

3.2 Viewing, Verifying, and Examining Models

The Interactive Model Viewer (IMV) tool provides an interactive interface for viewing interpretations. IMV aims to provide insights into the structures and features of models, and hence the semantics of axiomatizations. For example, it might be observed that certain domain elements are more or less often the range value of certain functions, or that some argument of a function or predicate does not affect the interpretation.

At the time of writing IMV was still in the design phase, considering different possible visualizations of the different types of interpretations. For finite interpretations for untyped first-order languages, one possible visualization is to provide a term tree for each function and predicate, for each resultant domain value and $true/false$. For example, for the function $f/2$ and the predicate $p/3$, and the finite domain $\mathbb{D} = \{d1, d2, d3\}$, Figure 10 shows the term tree for

f resulting in $d1$, and the term tree for p resulting in `true`. Thus one can see that, e.g., $\forall D \in \mathbb{D}\ f(d1, D)$ and $f(d2, d3)$ map to $d1$. Similarly, $p(d1, d1, d1)$, $p(d1, d2, d2)$, $\forall D \in \mathbb{D}\ p(d3, D, d1)$, and $\forall D \in \mathbb{D}\ p(d3, D, d3)$ map to $true$. A possible insight is that the second argument of f does not affect the interpretation when the first argument is interpreted as $d1$. Extensions of this visualization are being considered for other types of interpretations (Herbrand, integer domains, etc.), and more expressive languages (typed, higher-order, etc.).

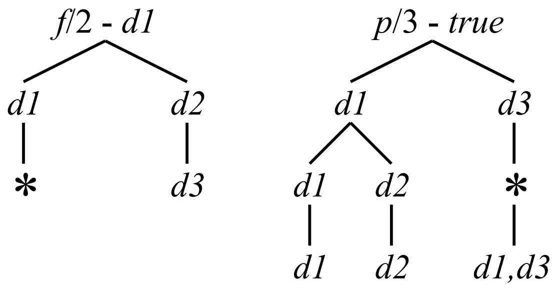


Figure 10: Example Term Trees

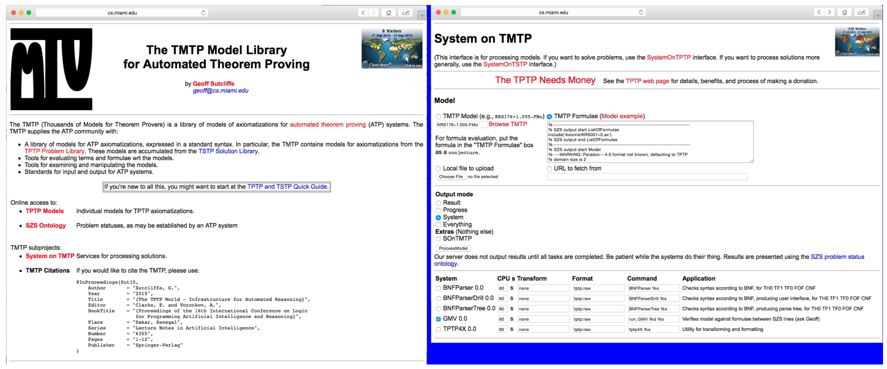
The GMV[5] tool verifies that an interpretation is a model for a given set of formulae. This is done by checking that each formula in the set is interpreted as $true$ in the model, using the techniques discussed in Section 3.1. If an ATP system is used to interpret the formulae wrt the model, it must be a trusted ATP system, and not the one that produced the model. A second approach, used for detecting that an interpretation is not a model of a set of formulae, is to conjoin the model with the set and check for unsatisfiability. If the conjoined set is unsatisfiable, then the interpretation is not a model of the set.

Another tool planned for examining interpretations is a *relationship tester*. This tool will check if two interpretations are syntactic variants of each other (as used when building the TMTP - see Section 2.4), compare the sizes of interpretations in terms of the number of $true/false$ Herbrand base elements, and check whether or not an interpretation makes a superset of Herbrand base elements $true/false$ compared to another interpretation (*interpretation subsumption*). Note that the syntactic-variant test is not the same as an equivalence test – two syntactically distinct interpretations can make the same Herbrand base elements $true$, e.g., they might be different types of interpretations, or might be the same type of interpretation but defined by different sets of formulae.

3.3 The TMTP Online

The TMTP has an online presence, starting at `http://www.tptp.org/TMTP`. The home page provides a linked hierarchy for browsing the TMTP models, and links to other relevant components, including the SystemOnTMTP web interface. SystemOnTMTP allows a model to be submitted to various tools, including parsers for models, evaluation of formulae wrt a model, and GMV. More tools, e.g., IMV, will be added as time goes by. Figure 11 shows the home page and the SystemOnTMTP page.

[5]That's "Geoff's Model Verifier".


cs.miami.edu

The TMTP Model Library for Automated Theorem Proving

by Geoff Sutcliffe
geoff@cs.miami.edu

The TMTP (Thousands of Models for Theorem Provers) is a library of models of axiomatizations for automated theorem proving (ATP) systems. The TMTP supplies the ATP community with:

- A library of models for ATP axiomatizations, expressed in a standard syntax. In particular, the TMTP contains models for axiomatizations from the TPTP Problem Library. These models are accumulated from the TSTP Solution Library.
- Tools for evaluating terms and formulae wrt the models.
- Tools for examining and manipulating the models.
- Standards for input and output for ATP systems.

If you're new to all this, you might want to start at the TPTP and TSTP Quick Guide.

Online access to:
- **TPTP Models** Individual models for TPTP axiomatizations.
- **SZS Ontology** Problem statuses, as may be established by an ATP system

TMTP subprojects:
- **System on TMTP** Services for processing solutions.
- **TMTP Citations** If you would like to cite the TMTP, please use:

```
@InProceedings{Sut10,
    Author       = "Sutcliffe, G.",
    Year         = "2010",
    Title        = "{The TPTP World - Infrastructure for Automated Reasoning}",
    Editor       = "Clarke, E. and Voronkov, A.",
    BookTitle    = "{Proceedings of the 16th International Conference on Logic
                    for Programming Artificial Intelligence and Reasoning}",
    Place        = "Dakar, Senegal",
    Series       = "Lecture Notes in Artificial Intelligence",
    Number       = "6355",
    Pages        = "1-12",
    Publisher    = "Springer-Verlag"
}
```

cs.miami.edu

System on TMTP

(This interface is for processing models. If you want to solve problems, use the SystemOnTPTP interface. If you want to process solutions more generally, use the SystemOnTSTP interface.)

The TPTP Needs Money See the TPTP web page for details, benefits, and process of making a donation.

Model

TMTP Model (e.g., `KRS176+1.005-FMo`) [✓ TMTP Formulae] (Model example)

KRS176+1.005-FMo Browse TMTP

For formula evaluation, put the formula in the "TMTP Formulae" box as a `conjecture`.

```
%------------------------------------------------------------------------------
% SZS output start ListOfFormulae
include('Axioms/KRS001+0.ax').
% SZS output end ListOfFormulae
%------------------------------------------------------------------------------
% SZS output start Model
%----WARNING: Paradox---4.0 format not known, defaulting to TPTP
% domain size is 2
```

Local file to upload
Choose File no file selected

URL to fetch from

Output mode
Result
Progress
[✓ System]
Everything
Extras (Nothing else)
SOnTMTP

ProcessModel

Our server does not output results until all tasks are completed. Be patient while the systems do their thing. Results are presented using the SZS problem status ontology.

System	CPU s	Transform	Format	Command	Application
BNFParser 0.0	60 s	none	tptp:raw	BNFParser %s	Checks syntax according to BNF, for TH0 TF1 TF0 FOF CNF
BNFParserDrill 0.0	60 s	none	tptp:raw	BNFParserDrill %s	Checks syntax according to BNF, producing user interface, for TH0 TF1 TF0 FOF CNF
BNFParserTree 0.0	60 s	none	tptp:raw	BNFParserTree %s	Checks syntax according to BNF, producing parse tree, for TH0 TF1 TF0 FOF CNF
✓ GMV 0.0	60 s	none	tptp:raw	run_GMV %d %s	Verifies model against formulae between SZS lines (ask Geoff)
TPTP4X 0.0	60 s	none	tptp:raw	tptp4X %s	Utility for transforming and formatting

Figure 11: The TMTP and SystemOnTMTP Web pages

4 Conclusion

This paper has described the beginnings of the TMTP Model Library, a new component of the TPTP world. The TMTP includes standards for writing interpretations, a library of TPTP format models for TPTP axiomatization problems, and tools for examining and using interpretations.

Future work on the TMTP includes adding more axiomatization problems to the TPTP problem library so that their models are added to the TMTP, researching ways to efficiently interpret formulae wrt an interpretation, implementing the IMV model viewer, extending the GMV verifier, and implementing an interpretation relationship tester.

When the TMTP and associated tools are in place, they will be available as the basis for the development of semantically guided ATP systems, including the planned implementation of semantic resolution.

References

[1] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.

[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] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 171–177. Springer-Verlag, 2011.

[4] P. Baumgartner, A. Fuchs, and C. Tinelli. Darwin - A Theorem Prover for the Model Evolution Calculus. In G. Sutcliffe, S. Schulz, and T. Tammet, editors, *Proceedings of the Workshop on Empirically Successful First Order Reasoning, 2nd International Joint Conference on Automated Reasoning*, 2004.

[5] C.E. Brown. Satallax: An Automated Higher-Order Prover (System Description). In B. Gramlich, D. Miller, and U. Sattler, editors, *Proceedings of the 6th International Joint Conference on Automated Reasoning*, number 7364 in Lecture Notes in Artificial Intelligence, pages 111–117, 2012.

[6] F.M. Brown. SLM. Technical Report Internal Memo #72, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, United Kingdom, 1974.

[7] A. Bundy. Personal Correspondence with Geoff Sutcliffe. Available as PDF from Geoff Sutcliffe, 1987.

[8] 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.

[9] L. de Moura and N. Bjorner. 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.

[10] K. Korovin and C. Sticksel. iProver-Eq - An Instantiation-Based Theorem Prover with Equality. In J. Giesl and R. Haehnle, editors, *Proceedings of the 5th International Joint Conference on Automated Reasoning*, number 6173 in Lecture Notes in Artificial Intelligence, pages 196–202, 2010.

[11] 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.

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

[13] S. Schulz. System Description: E 1.8. 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 477–483. Springer-Verlag, 2013.

[14] J.R. Slagle. Automatic Theorem Proving with Renamable and Semantic Resolution. *Journal of the ACM*, 14(4):687–697, 1967.

[15] J.K. Slaney, E. Lusk, and W.W. McCune. SCOTT: Semantically Constrained Otter, System Description. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 764–768. Springer-Verlag, 1994.

[16] G. Sutcliffe. The Semantically Guided Linear Deduction System. In D. Kapur, editor, *Proceedings of the 11th International Conference on Automated Deduction*, number 607 in Lecture Notes in Artificial Intelligence, pages 677–680. Springer-Verlag, 1992.

[17] G. Sutcliffe. SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 406–410. Springer-Verlag, 2000.

[18] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[19] 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.

[20] 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.

[21] 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.

[22] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, page To appear, 2015.

[23] 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.

[24] 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, 2006.

[25] G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, number 112 in Frontiers in Artificial Intelligence and Applications, pages 201–215. IOS Press, 2004.

[26] S. Trac, Y. Puzis, and G. Sutcliffe. An Interactive Derivation Viewer. In S. Autexier and C. Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning*, volume 174 of *Electronic Notes in Theoretical Computer Science*, pages 109–123, 2006.

[27] C. Weidenbach, A. Fietzke, R. Kumar, M. Suda, P. Wischnewski, and D. Dimova. SPASS Version 3.5. In R. Schmidt, editor, *Proceedings of the 22nd International Conference on Automated Deduction*, number 5663 in Lecture Notes in Artificial Intelligence, pages 140–145. Springer-Verlag, 2009.

A SZS Ontology for Interpretations

The list below provides details for each of the nodes in Figure 1. Each entry gives the full "OneWord" ontology values, its three-letter acronym, and a brief description.

- Interpretation (Int): An interpretation.
- Model (Mod): A model.
- PartialInterpretation (Pin): A partial interpretation.
- PartialModel (PMo): A partial model.
- StrictlyPartialInterpretation (SIn): A strictly partial interpretation.
- StrictlyPartialModel (SMo): A strictly partial model.
- DomainInterpretation (DIn): An interpretation whose domain is not the Herbrand universe.
- DomainModel (DMo): A model whose domain is not the Herbrand universe.
- DomainPartialInterpretation (DPI): A domain interpretation that is partial.
- DomainPartialModel (DPM): A domain model that is partial.
- DomainStrictlyPartialInterpretation (DSI): A domain interpretation that is strictly partial.
- DomainStrictlyPartialModel (DSM): A domain model that is strictly partial.
- FiniteInterpretation: A domain interpretation with a finite domain.
- FiniteModel (FMo): A domain model with a finite domain.
- FinitePartialInterpretation (FPI): A domain partial interpretation with a finite domain.
- FinitePartialModel (FPM): A domain partial model with a finite domain.
- FiniteStrictlyPartialInterpretation (FSI): A domain strictly partial interpretation with a finite domain.
- FiniteStrictlyPartialModel (FSM): A domain strictly partial model with a finite domain.
- IntegerInterpretation: An integer domain interpretation.
- IntegerModel (FMo): An integer domain model.
- IntegerPartialInterpretation (FPI): An integer domain partial interpretation.
- IntegerPartialModel (FPM): An integer domain partial model.
- IntegerStrictlyPartialInterpretation (FSI): An integer domain strictly partial interpretation.
- IntegerStrictlyPartialModel (FSM): An integer domain strictly partial model.
- RealInterpretation: A real domain interpretation.
- RealModel (FMo): A real domain model.
- RealPartialInterpretation (FPI): A real domain partial interpretation.
- RealPartialModel (FPM): A real domain partial model.
- RealStrictlyPartialInterpretation (FSI): A real domain strictly partial interpretation.

- RealStrictlyPartialModel (FSM): A real domain strictly partial model.
- HerbrandInterpretation (HIn): A Herbrand interpretation.
- HerbrandModel (HMo): A Herbrand model.
- FormulaInterpretation (TIn): A Herbrand interpretation defined by a set of TPTP formulae.
- FormulaModel (TMo): A Herbrand model defined by a set of TPTP formulae.
- FormulaPartialInterpretation (TPI): A Herbrand partial interpretation defined by a set of TPTP formulae.
- FormulaPartialModel (TMo): A Herbrand partial model defined by a set of TPTP formulae.
- FormulaStrictlyPartialInterpretation (TSI): A Herbrand strictly partial interpretation defined by a set of TPTP formulae.
- FormulaStrictlyPartialModel (TSM): A Herbrand strictly partial model defined by a set of TPTP formulae.
- Saturation (Sat): A Herbrand model expressed as a saturating set of formulae.