J Autom Reasoning (2009) 43:337–362
DOI 10.1007/s10817-009-9143-8

The TPTP Problem Library and Associated Infrastructure

The FOF and CNF Parts, v3.5.0

Geoff Sutcliffe

Received: 24 December 2008 / Accepted: 6 July 2009 / Published online: 22 July 2009

Abstract This paper describes the First-Order Form (FOF) and Clause Normal Form (CNF) parts of the TPTP problem library, and the associated infrastructure. TPTP v3.5.0 was the last release containing only FOF and CNF problems, and thus serves as the exemplar. This paper summarizes the history and development of the TPTP, describes the structure and contents of the TPTP, and gives an overview of TPTP related projects and tools.

Keywords TPTP · ATP system evaluation

1 Introduction

The Thousands of Problems for Theorem Provers (TPTP) problem library is the de facto standard set of test problems for classical first-order Automated Theorem Proving (ATP) systems. The TPTP 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. The principal motivation for the TPTP is to support the testing and evaluation of ATP systems, to help ensure that performance results accurately reflect capabilities of the ATP systems being considered. A large, common, stable library of test problems is necessary for meaningful system evaluations (with statistically significant results), system comparisons, and repeatability of testing. Since its first release in 1993, many researchers have used the TPTP as an appropriate and convenient basis for ATP system evaluation. Over the years the TPTP has also increasingly been used as a conduit for ATP users to provide samples of their problems to ATP system developers—users have found that contributing samples of their problems to the TPTP exposes the problems to the

G. Sutcliffe (✉)
Department of Computer Science, University of Miami, Coral Gables, FL 33124, USA
e-mail: geoff@cs.miami.edu

developers, who then improve their systems' performances on the problems, which completes a cycle to provide the users with more effective tools.

One of the keys to the success of the TPTP and related projects is the consistent use of the TPTP language for writing problems and solutions, which enables convenient communication between different systems and researchers. The common adoption of the TPTP language and associated standards has made it easier to build complex reasoning systems and applications from TPTP compliant components, e.g., [9, 21, 41, 60, 62].

This paper describes the First-Order Form (FOF) and Clause Normal Form (CNF) parts of the TPTP, and the associated infrastructure. TPTP v3.5.0 was the last release containing only FOF and CNF problems (later releases also contain Typed Higher-order Form (THF) problems), and thus serves as the exemplar. The remainder of Section 1 gives guidelines for obtaining and using the TPTP. Section 2 summarizes the history and development of the TPTP (thus implicitly explaining the design decisions made), and provides summary statistics about release v3.5.0. Section 3 describes the structure and contents of the TPTP. Section 4 gives an overview of TPTP related projects and tools. Section 5 concludes, discussing current developments and directions, including the higher-order form and typed first-order form parts of the TPTP, to be released in 2009 and beyond.

This paper is a natural successor to the 1998 paper that described the CNF part of the TPTP, using release v1.2.1 as the exemplar [55]. In order to make this paper self-contained, some material from [55] is summarized in this paper. Some material from [45] is also expanded in this paper.

1.1 Obtaining and Using the TPTP

The TPTP is managed in the manner of a software product, in the sense that fixed releases are made. Each release of the TPTP is identified by a release number, in the form v*Version*.*Edition*.*Patch*. The *Version* number enumerates major new releases of the TPTP, in which important new features have been added. The *Edition* number is incremented each time new problems are added to the current version. The *Patch* level is incremented each time errors, found in the current edition, are corrected.

The TPTP may be downloaded from the TPTP web site, www.tptp.org. The web site also provides a quick-guide introduction to the TPTP, detailed documentation for the TPTP, online browsing of the TPTP problems and their solutions, an interface for pre-processing problems, an interface for running ATP systems on problems, an interface for post-processing solutions, and links to a range of related projects.

Appropriate use of the TPTP allows developers and users to meaningfully evaluate ATP systems. To that end, the following guidelines for using the TPTP and presenting results should be followed:

- The TPTP release number must be stated.
- Each problem must be referenced by its unambiguous TPTP name.
- The formulae should, as far as is possible, not be changed in any way. Any changes made (addition, removal, reordering, reformatting, etc.) must be explicitly noted.
- Any information given to the ATP system, other than the formulae, must be explicitly noted. All system switches and settings must be recorded. The header

information in problems may not be used by the ATP system without explicit notice.

Abiding by these conditions allows unambiguous identification of the problem, the formulae used, and further input to the ATP system. Published evaluations that abide by these conditions should be accompanied by the explicit statement "These results were obtained in compliance with the guidelines for use of the TPTP [48]". By making this clear statement, readers are assured that the evaluation was performed within the guidelines. Conversely, it will be clear when the guidelines may have been ignored.

2 The TPTP History and Development

The development of the TPTP started in mid-1992, as a collaboration between Geoff Sutcliffe at the University of Western Australia (James Cook University from 1993), and Christian Suttner at the Technische Universität München. A beta release, v0.5.0, of the TPTP was made to selected researchers on 15th April 1993. The first public release, v1.0.0, of the TPTP was made on Friday, 12th November 1993. The exemplar release of this paper, v3.5.0, was made on 13th August 2008, and is the seventeenth TPTP edition. This section describes the history of the TPTP, its growth, and provides overview statistics for v3.5.0.

2.1 Origins and Influences on the TPTP

Prior to the development of the TPTP a large number of interesting problems had accumulated in the ATP community. From early on researchers published particularly interesting individual problems, e.g., [6, 16, 18, 20, 25, 29]. Problems published in hardcopy form are, however, often not suitable for testing ATP systems, because they have to be transcribed to electronic form. This is a cumbersome, error-prone process, and is feasible for only small numbers of small problems. Researchers also collected problems in electronic form, e.g., in 1988 the Argonne National Laboratory produced a collection in Otter format [19], and the SPRFN ATP system [27] was distributed with a collection of problems. Although some of these early collections provided significant support to researchers, and formed the early core of the TPTP, none, with the possible exception of the ANL collection, were designed to serve as a common basis for ATP research. These early collections typically were built in the course of research into a particular ATP system. As a result there were several factors that limited their usefulness.

The goal for building the TPTP was to overcome the previous drawbacks, and to centralize the burden of problem collection and maintenance to one place. The TPTP tries to address all relevant issues. In particular, the TPTP is easy to discover and obtain, is easy to use, is comprehensive, is up-to-date, is independent of any particular ATP system, spans a diversity of subject matters, is large enough for statistically significant testing, is well structured and documented, has an unambiguous naming scheme, documents each problem, contains problems varying in difficulty, provides a difficulty rating for each problem, provides statistics for each problem and the library as a whole, provides a mechanism for adding new problems, provides a mechanism for correcting errors in existing problems, and provides guidelines for its use in

evaluating ATP systems. The development of the TPTP is an ongoing project, with the aim of providing all of the desired properties.

In the early years of the TPTP (the early 1990s) most ATP systems were based on the resolution inference rule, using the clause normal form of first-order logic. As such, the first seven editions of the TPTP contained only CNF problems. The problems collected for those first editions came principally from existing electronic problem collections and the ATP literature. As more ATP users became aware of the TPTP, more full first-order form problems became available, and from v2.0.0 (released June 1997) the TPTP included FOF problems. The trend towards FOF problems increased significantly from v3.1.0, due to the influence of contributions from "real world" users. Another effect linked to this influence was the addition of large FOF problems in v3.4.0 (released March 2008), based on encodings of large mathematical [61] and commonsense [24, 30, 40] knowledge bases. Data on this trend towards FOF is provided and discussed in Section 2.2.

The CADE ATP System Competition (CASC)—see Section 4.2, has had an organizational influence on the TPTP. CASC is organized annually at the CADE (or IJCAR, of which CADE is a constituent) conference. In the first few years of CASC (the late 1990s), developers increasingly tuned their systems to TPTP problems, in order to do well in CASC. This raised a concern that ATP systems were becoming over-tuned to the TPTP, rather than generally applicable. Thus, from the sixth CASC (in 2001) the TPTP edition used for the competition was not released until after the competition, thus making over-tuning disadvantageous [54]. This put the TPTP releases into a corresponding annual cycle, from v2.4.0 (released June 2001).

By 2002 there was significant common use of the TPTP, which generated a demand for a standard format for output from ATP systems. The common usage also highlighted the need to unify the original CNF and FOF languages (which were compatible but had different syntax forms). These influences resulted in the development [57] of the current TPTP language, which was designed to be suitable for expressing FOF and CNF problems, and also solutions from ATP systems—see Section 3.3. The language was introduced in v3.0.0 (released November 2004), and has since been used consistently in the TPTP and related projects. In particular, it is used for the Thousands of Solutions from Theorem Provers (TSTP) library—see Section 4.1. Associated with the new language was the development of the SZS result ontologies, which are used to specify the status of problems, the output from ATP systems, and the semantics of inference steps in derivations—see Section 3.5.

2.2 The Growth of the TPTP

All changes between TPTP releases are recorded in a history file that is distributed with each release. Table 1 lists the edition releases of the TPTP up to v3.5.0, the major changes, the numbers of problem domains, and the numbers of abstract problems and ATP problem versions—see Section 3.1 for an explanation of the TPTP domain structure, and Section 3.2 for information about TPTP problem versions. Figure 1 graphs the numbers of problems over time. The growing predominance of FOF problems, notable from v3.1.0 (released July 2005), suggests that CNF is no longer the primary language of ATP, with users preferring FOF for encoding their problems.

Table 2 gives some overview statistics for TPTP v3.5.0, with key characteristics. The problems are mostly known or believed to be theorems or unsatisfiable sets of

Table 1 Overview of TPTP editions

Release	Date	Changes	Doms	Abst	Probs	FOF	CNF
v0.5.0	15/04/93	Beta release		1,211	1,695	0	1,695
v1.0.0	12/11/93	First public release	23	1,577	2,295	0	2,295
v1.1.0	08/04/94	TPTP2X overhauled	25	1,897	2,652	0	2,652
v1.2.0	30/08/95	Problem generators introduced	25	2,044	2,752	0	2,752
v2.0.0	05/06/97	FOF and ratings introduced	28	2,268	3,277	217	3,060
v2.1.0	17/12/97	German distribution stopped	28	2,402	3,622	347	3,275
v2.2.0	11/02/99	TPTP2X upgraded	28	2,666	4,004	670	3,334
v2.3.0	16/11/99	Ratings made more accurate	28	2,813	4,229	671	3,558
v2.4.0	24/06/01	New domains and problems	30	3,624	5,882	1,463	4,419
v2.5.0	28/07/02	VerySimilarProblems listed	30	4,330	6,672	1,491	5,181
v2.6.0	20/08/03	Regular release	31	4,628	6,973	1,500	5,473
v2.7.0	15/08/04	Regular release	32	4,880	7,267	1,745	5,522
v3.0.0	11/11/04	New TPTP language	32	4,879	7,266	1,745	5,522
v3.1.0	25/07/05	Regular release	33	5,609	8,013	2,324	5,689
v3.2.0	19/07/06	Language BNF updated	35	6,270	8,984	2,724	6,260
v3.3.0	28/06/07	Regular release	35	6,831	9,894	3,644	6,250
v3.4.0	04/03/08	Very large problems added	35	7,113	11,279	5,029	6,250
v3.5.0	13/08/08	Last FOF and CNF release	35	7,134	11,395	5,049	6,346

formulae—see Section 4.1 for some statistics on the extent to which the problems have been solved by contemporary ATP systems. Equality occurs in the majority of problems, as it occurs naturally in many domains. However, it is typically mixed with other notions, so pure equality problems are quite uncommon, coming mostly from logic and mathematics domains. The majority of the CNF problems are Horn, which shows that reasoning systems specialized for Horn clauses are useful. Specialization for range restricted and unit equality problems is less useful on TPTP problems. There are very few propositional problems—these are of lesser interest to first-order ATP, and are collected by the SAT community [12].

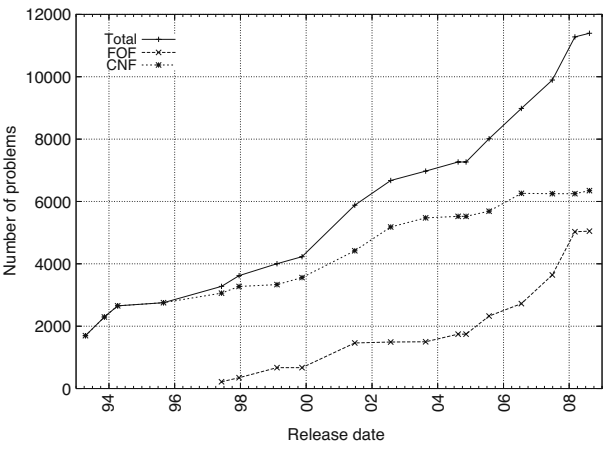

Fig. 1 The growth of the TPTP

Table 2 Statistics for TPTP v3.5.0

	Total	Percent	FOF	Percent	CNF	Percent
Number of problem domains	35		26		30	
Number of abstract problems	7,134		3,480		4,824	
Number of generic problems	85		1		84	
Number of problems	11,395		5,049	44%[a]	6,346	55[a]
Known theorems/unsatisfiable	8,993	78%[a]	4,393	87%[b]	4,600	72[c]
Known non-theorems/satisfiable	1,335	11%[a]	511	10%[b]	824	12[c]
Number of problems with equality	8,265	72%[a]	4,094	81%[b]	4,171	65[c]
Number of pure equality problems	1,515	13%[a]	211	4%[b]	1,034	20[c]
Number of CNF non-Horn	–		–		3,825	33[c]
Number of CNF range restricted	–		–		702	6[c]
Number of CNF unit equality	–		–		863	7[c]
Number of propositional problems	70	0%[a]	25	0%[b]	45	0[c]

[a]Percentage of the 11,395 problems
[b]Percentage of the 5,049 FOF problems
[c]Percentage of the 6,346 CNF problems

Tables 3 and 4 provide problem-level statistics over the FOF and CNF problems, respectively. Some of the FOF problems are very large—these come from an export of part of the Cyc knowledge base [15]. However, most FOF problems have less than 68 formulae and 270 atoms. The predominance of universally quantified variables

Table 3 Statistics over the FOF problems

	Min	Max	Avg	Median
Number of formulae	1	3,341,990	43,659	67
Percentage of unit formulae	0%	100%	24%	19%
Number of atoms	1	5,328,217	87,795	269
Percentage of equality atoms	0%	100%	15%	11%
...in problems with equality	0%	100%	19%	12%
Avg atoms per formula	1.0	1,736.0	27.8	3.4
Number of predicate symbols	1	204,678	3,395	17
Percentage of propositions	0%	97%	7%	0%
Maximal predicate arity	0	81	3	2
Number of functors	0	1,050,014	14,010	10
Percentage of constants	0%	100%	34%	20%
Maximal functor arity	0	10	3	2
Number of variables	0	972,236	22,399	122
...universally quantified	0	972,235	9,085	108
...existentially quantified	0	9,512	451	8
Number of connectives	0	2,065,419	48,154	177
Maximal formula depth	1	1,614	31	16
Average formula depth	1	1,614	12	5
Maximal term depth	1	40	4	4
Average term depth	1	4	1	1

Table 4 Statistics over the CNF problems

	Min	Max	Avg	Median
Number of clauses	1	3,240	179	34
Percentage of unit clauses	0%	100%	45%	34%
Percentage of non-Horn clauses	0%	99%	12%	7%
...in non-Horn problems	1%	99%	21%	11%
Percentage of range rest. clauses	0%	100%	69%	75%
Number of literals	1	127,126	833	82
Percentage of equality literals	0%	100%	29%	16%
...in problems with equality	0%	100%	45%	22%
Maximal clause size	1	60	6	5
Average clause size	1	40	2	2
Number of predicate symbols	1	1,817	23	4
Percentage of propositions	0%	100%	2%	0%
Maximal predicate arity	1	56	3	2
Number of functors	0	903	31	10
Percentage of constants	0%	100%	49%	50%
Maximal functor arity	0	18	2	2
Number of variables	0	120,428	713	66
Maximal term depth	0	85	4	4
Average term depth	0	8	1	1

shows that most problems encode information that applies to all things. The CNF problem statistics reflect smaller problem sizes, which stems from the fact that the very large problems are mechanically generated in FOF rather than CNF. The statistics show the syntactic diversity of the problems in the TPTP.

3 The TPTP Structure and Content

3.1 TPTP Domains

The problems in the TPTP are classified into *domains* that reflect the natural hierarchy of scientific domains, as presented in standard subject classification literature. The classification is based mainly on the Dewey Decimal Classification (DDC) [7] and the Mathematics Subject Classification (MSC) [37]. Six main fields are defined: logic, mathematics, computer science, science & engineering, social sciences, and other. Each field is subdivided into domains, each identified by a three-letter mnemonic, as shown in Table 5. The range of domains covered shows the semantic diversity of the problems in the TPTP.

3.2 TPTP Problem Versions

Each problem in the TPTP is a *version* of an underlying *abstract problem*, and is stored in a separate physical file. Each abstract problem is thus represented by its

Table 5 The domain structure of the TPTP

Field	Domain	Mnemonic	Problems	FOF	CNF
Logic	Combinatory logic	COL	239	0	239
	Logic calculi	LCL	702	136	566
	Henkin models	HEN	67	0	67
Mathematics	Set theory	SET	1,268	468	800
	Set theory continued	SEU	906	906	0
	Graph theory	GRA	33	32	1
	Algebras				
	Boolean algebra	BOO	140	1	139
	Robbins algebra	ROB	40	0	40
	Left distributive	LDA	23	0	23
	Lattices	LAT	707	397	310
	Groups	GRP	941	149	792
	Rings	RNG	104	0	104
	Fields	FLD	279	0	279
	Homological Alg	HAL	9	9	0
	General algebra	ALG	300	285	15
	Number theory	NUM	451	136	315
	Topology	TOP	131	107	24
	Analysis	ANA	91	0	91
	Geometry	GEO	489	236	253
	Category theory	CAT	130	68	62
Computer science	Computing theory	COM	21	7	14
	Knowledge representation	KRS	175	158	17
	Natural language processing	NLP	516	258	258
	Planning	PLA	46	6	40
	Agents	AGT	52	52	0
	Commonsense reasoning	CSR	442	442	0
	Software creation	SWC	846	423	423
	Software verification	SWV	630	288	342
Science & engg	Hardware creation	HWC	6	0	6
	Hardware verification	HWV	83	0	83
	Medicine	MED	10	10	0
Social sciences	Management	MGT	156	78	78
Other	Syntactic	SYN	1,208	366	842
	Puzzles	PUZ	126	24	102
	Miscellaneous	MSC	28	7	21

corresponding collection problem versions. Table 5 shows the numbers of problem versions in each domain. Each problem file is annotated with information that differentiates it from other versions. These annotations help users select problems according to their intent.

The most common source of different versions of an abstract problem is the use of different axiomatizations. In the TPTP, different axiomatizations are kept in separate axiom files, and are included in problems as appropriate, leading to different problem versions. An axiomatization is *standard* if it is a complete axiomatization of an established theory, and it has not had any lemmas added. An *incomplete* axiomatization is formed by removing axioms from a standard axiomatization, and a *redundant* axiomatization is formed by adding lemmas to a standard axiomatization.

An *especial* axiomatization does not capture any established theory, and is typically used in only a few problems.

Another source of different problem versions is *generic problems*, where particular instances of an abstract problem are formed according to some size parameters, e.g., the *N*-queens problem. Since v1.1.3 the TPTP has included generator files, which allow users to generate arbitrary size instances of generic problems. The generator files are used in conjunction with the TPTP2X utility—see Section 4.3.2. For convenience, the TPTP still contains a particular instance of each generic problem. The size chosen for each such instance is large enough to be non-trivial. The theoremhood/satisfiability of a generic problem can depend on the problem size.

The *theorem formulation* is a third source of different problem versions, where different representations of the abstract problem are used. Some formulations are noted to be *biased* towards or against a particular ATP technique or system.

3.3 The TPTP Language

TPTP v3.0.0 introduced the current TPTP language [51]. The language was designed to be suitable for writing both ATP problems and ATP solutions, to be flexible and extensible, and easily processed by both humans and computers. The syntax is Prolog-like, and, with a few operator definitions, units of TPTP data form Prolog terms. As a result, the development of reasoning software (for TPTP data) in Prolog has a low entry barrier. The use of a common language for writing problems and solutions enables output from ATP systems to be seamlessly passed on as input to further systems or tools.

A principal goal of the development of the language grammar was to make it easy to translate into `lex`/`yacc`/`flex`/`bison` input, so that construction of parsers (in languages other than Prolog) can be a reasonably easy task [63]. To this end the language definition uses a modified BNF meta-language that separates syntactic, semantic, lexical, and character-macro rules. Syntactic rules use the standard `::=` separator, semantic constraint rules use a `:==` separator, rules that produce tokens at the lexical level use a `::-` separator, and the bottom level character-macros are defined by regular expressions in rules using a `:::` separator. The separation of syntax from semantics eases the task of building a syntactic analyzer. At the same time, the semantic rules provide the details necessary for semantic checking.

The TPTP and TSTP tools—see Section 4.3—naturally process data written in the TPTP language, parsers written in C and Java are available, and the language BNF can be automatically converted into `lex`/`yacc`/`flex`/`bison` input. Many ATP system users and developers have adopted the TPTP language.

3.4 Problem and Axiom Files

Figure 2 shows an example of a TPTP problem file. All information that is not for use by ATP systems is formatted as comments, on lines starting with a `%`.

The first section of each TPTP problem file is a header that contains information for users. It is divided into four parts. The first part identifies and describes the problem. The second part provides information about occurrences of the problem in the literature and elsewhere. The third part gives the problem's SZS status—see Section 3.5, its difficulty rating—see Section 3.6, and a table of syntactic

```
%------------------------------------------------------------------------------
% File     : GRP194+1 : TPTP v3.5.0. Released v2.0.0.
% Domain   : Group Theory (Semigroups)
% Problem  : In semigroups, a surjective homomorphism maps the zero
% Version  : [Gol93] axioms.
% English  : If (F,*) and (H,+) are two semigroups, phi is a surjective
%            homomorphism from F to H, and id is a left zero for F,
%            then phi(id) is a left zero for H.

% Refs     : [Gol93] Goller (1993), Anwendung des Theorembeweisers SETHEO a
% Source   : [Gol93]
% Names    :

% Status   : Theorem
% Rating   : 0.15 v3.5.0, 0.16 v3.4.0, ... 0.33 v2.2.1, 0.00 v2.1.0
% Syntax   : Number of formulae    :    8 (   2 unit)
%            Number of atoms       :   21 (   4 equality)
%            Maximal formula depth :    8 (   4 average)
%            Number of connectives :   13 (   0 ~  ;   0  |;   6  &)
%                                         (   1 <=>;   6 =>;   0 <=)
%                                         (   0 <~>;   0 ~|;   0 ~&)
%            Number of predicates  :    3 (   0 propositional; 2-2 arity)
%            Number of functors    :    5 (   3 constant; 0-3 arity)
%            Number of variables   :   15 (   0 singleton;  14 !;   1 ?)
%            Maximal term depth    :    3 (   1 average)

% Comments :
%------------------------------------------------------------------------------
%----Include Semigroup axioms
include('Axioms/GRP007+0.ax').
%------------------------------------------------------------------------------
%----Definition of a homomorphism
fof(homomorphism1,axiom,
    ( ! [X] :
        ( group_member(X,f)
       => group_member(phi(X),h) ) )).

fof(homomorphism2,axiom,
    ( ! [X,Y] :
        ( ( group_member(X,f)
          & group_member(Y,f) )
       => multiply(h,phi(X),phi(Y)) = phi(multiply(f,X,Y)) ) )).

fof(surjective,axiom,
    ( ! [X] :
        ( group_member(X,h)
       => ? [Y] :
            ( group_member(Y,f)
            & phi(Y) = X ) ) )).

%----Definition of left zero
fof(left_zero,axiom,
    ( ! [G,X] :
        ( left_zero(G,X)
      <=> ( group_member(X,G)
          & ! [Y] :
              ( group_member(Y,G)
             => multiply(G,X,Y) = X ) ) ) )).

%----The conjecture
fof(left_zero_for_f,hypothesis,
    ( left_zero(f,f_left_zero) )).

fof(prove_left_zero_h,conjecture,
    ( left_zero(h,phi(f_left_zero)) )).
%------------------------------------------------------------------------------
```

Fig. 2 Example of a problem file (`GRP194+1.p`)

measurements made on the problem. The last part contains general information about the problem. TPTP axiom files have similar headers.

The second section of each TPTP problem and axiom file optionally contains `include` directives. These are typically used in problem files to include the contents of axiom files. An `include` directive may include an entire file, or may specify the names of the annotated formulae that are to be included from the file, thus providing a more finely grained include mechanism.

The last section of each TPTP problem and axiom file contains *annotated formulae*. An annotated formula has the form: *language*`(`*name*`,` *role*`,` *formula*`,` *source*`,` *useful_info*`)`. The *language*s supported in TPTP v3.5.0 are `fof` and `cnf`—formulae in FOF and CNF respectively. The *name* identifies the annotated formula within the file. The *role* gives the user semantics of the *formula*, e.g., `axiom`, `lemma`, `conjecture`, and hence defines its use in an ATP system.

The logical *formula*, in either FOF or CNF, uses a consistent and easily understood notation, and uses only standard ASCII characters. The syntax for atoms is that of Prolog: variables start with upper case letters, uninterpreted predicates and functors either start with lower case and contain alphanumerics and underscore, or are in 'single quotes', and uninterpreted atoms and terms are written in prefix notation, The language also supports interpreted predicates and functors, that are syntactically distinct from uninterpreted ones. These come in two varieties: *defined* predicates and functors, whose interpretation is specified by the TPTP language, and *system* predicates and functors, whose interpretation is ATP system specific. The defined predicates recognized so far are `$true` and `$false`, with the obvious interpretations, and infix `=` and `!=` for equality and inequality. The defined functors recognized so far are distinct objects written in "double quotes", and numbers. A distinct object is interpreted as the domain element in the double quotes. Numbers are interpreted as themselves (as domain elements). A consequence of the predefined interpretations is that all different distinct objects and numbers are known to be unequal, e.g., `"Apple" != "Microsoft"` and `1 != 2` are implicit axioms. Such implicit axioms may be built into an ATP system, e.g., [35]. System predicates and functors are used for interpreted predicates and functors that are available in particular ATP systems. System predicates and functors start with `$$`. These are not controlled by the TPTP language, so they must be used with caution.

The logical connectives are written using short notations, rather than words (which would look like, and thus be easy to confuse with, predicates and functors). The universal and existential quantifiers are `!` and `?` respectively, with the quantified variables following in `[]` brackets. The binary connectives are `&`, `|`, `=>`, `<=`, `<=>`, `<~>`, `~&`, and `~|`, for conjunction, disjunction, implication, reverse implication, equivalence, inequivalence, negated conjunction, and negated disjunction respectively. The only unary connective is `~` for negation. Negation has higher precedence than quantification, which in turn has higher precedence than the binary connectives. No precedence is specified between the binary connectives; brackets are used to ensure the correct association.

The *useful_info* field of an annotated formula is optional, and if it is not used then the *source* field becomes optional. Examples of annotated formulae with *source* and *useful_info* fields are given in Fig. 3. The *source* field is used to record where the annotated formula came from, and is most commonly a `file` record or an `inference` record. A `file` record stores the name of the file from which the

```
fof(formula_27,axiom,
    ! [X,Y] :
      ( subclass(X,Y) <=>
        ! [U] :
          ( member(U,X) => member(U,Y) )),
    file('SET005+0.ax',subclass_defn),
    [description('Definition of subclass'), relevance(0.9)]).

cnf(175,lemma,
    ( rsymProp(ib,sk_c3)
    | sk_c4 = sk_c3 ),
    inference(factor_simp,[status(thm)],[
        inference(para_into,[status(thm)],[96,78,theory(equality)])]),
    [iquote('para_into,96.2.1,78.1.1,factor_simp')]).
```

Fig. 3 Annotated formulae with *source* and *useful_info* fields

annotated formula was read, and optionally the name of the annotated formula as it occurs in that file—this may be different from the name of the annotated formula itself. An `inference` record stores information about an inferred formula: the inference rule name provided by the ATP system, a list of useful information about the inference, e.g., the semantic status of the formula with respect to its parents as an SZS value—see Section 3.5, and a list identifying the parents of the influence. Most commonly the list elements are the names of the parent annotated formulae, but can also be nested `inference` records, and `theory` records. A theory record is used when the axioms of some theory are built into the inference rule, e.g., equality axioms are built into paramodulation. The *useful_info* field of an annotated formula is a `[]`ed list of arbitrary useful information, e.g., for user applications, formatted as Prolog terms.

Regular comments in the TPTP language extend from a `%` character to the end of the line, or may be block comments within /∗ . . . ∗/ bracketing. System comments in the TPTP language are used for system specific annotations. They extend from a `%$$` sequence to the end of the line, or may be block comments within `/∗$$ . . . ∗/` bracketing. System comments look like regular comments, so normally they are discarded. However, a wily user of the language can store/extract information from the comment before discarding it. System comments should identify the system, followed by a :, e.g., `/∗$$Otter 3.3: Demodulator ∗/`.

3.5 The SZS Ontologies

When the TPTP is used for testing ATP systems, it is necessary to know what result is expected for each problem, for comparison with the output from the ATP systems. For derivation post-processing, e.g., verification—see Section 4.3.4, it is useful to know the semantic relationship between the parents and inferred formula of each inference. In a broader context, in order to seamlessly embed ATP systems and tools in more complex reasoning systems, it is necessary to correctly and precisely describe their input and output data. The SZS[1] ontologies [47] provide status and dataform

[1]Named after the authors of the first presentation of the ontology [57].

values that can be used for this. The SZS standard also specifies how ontology values should be reported in the output from ATP systems and tools, in order to facilitate easy processing. Figure 4 shows extracts from the ontologies. Each ontology value has a full name (some are abbreviated in Fig. 4) and a three letter acronym that is useful for presentation in tables of data. The full ontologies are available as part of the TPTP distribution, and are used consistently in the TPTP and related projects.

The `Success` ontology catalogs semantic relationships between the axioms and conjecture of a problem, or the parents and inferred formula of an inference. For example, if all models of the axioms are models of the conjecture then the status is `Theorem`, if some models of the axioms are models of the negation of the conjecture then the status is `CounterSatisfiable`, and if the axioms and conjecture are equisatisfiable (as in Skolemization) then the status is `EquiSatisfiable`. The `NoSuccess` ontology catalogs reasons that a system or tool has not been successful. For example, if a system stopped because the CPU time limit ran out then the status is `Timeout`, and if a system has not attempted a problem but might in the future then the status is `NotTriedYet`. The `LogicalData` ontology catalogs dataforms

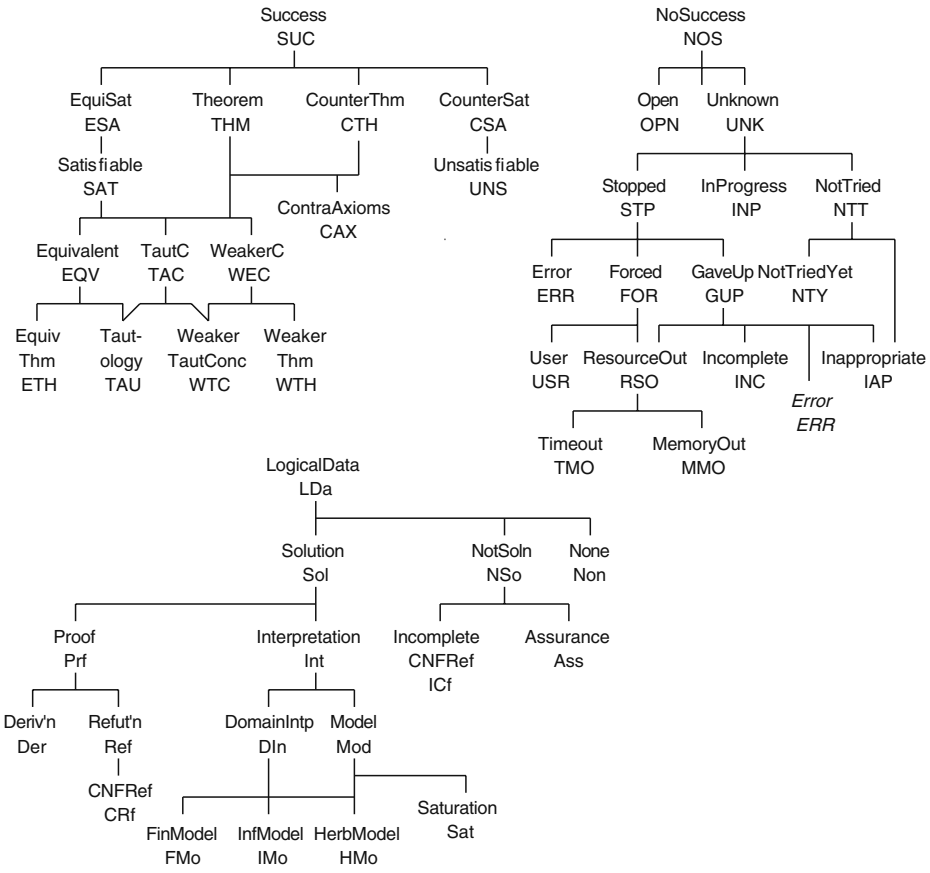

Fig. 4 The SZS result (success and no success) and dataform ontologies

of output from ATP systems and tools. For example, a prover might output a `CNFRefutation`, and a model finder might output a `FiniteModel`.

3.6 The TPTP Problem and System Rating Scheme

An important feature of the TPTP is the problem ratings [56]. The ratings provide a well-defined measure of how difficult the problems are for ATP systems, and how effective the ATP systems are for different types of problems. For rating, the TPTP problems are divided into Specialist Problem Classes (SPCs)—syntactically identifiable classes of problems for which certain ATP techniques have been observed to be especially well suited, and for which there are ATP systems that can, in principle, attempt to solve all the problems in the class. Examples of SPCs are "FOF theorems with some (not pure) equality", "CNF unsatisfiable unit equality problems", and "CNF satisfiable effectively propositional problems". Rating is done separately for each SPC, to provide ratings based on comparable problems, ATP techniques, and ATP systems. Problems that are known to be biased—see Section 3.2—are excluded from the calculations. For each SPC, performance data of contemporary ATP systems on the problems, taken from the TSTP—see Section 4.1, is analyzed. A partial order between ATP systems is determined according to whether or not a system solves a strict superset of the problems solved by another system. If a strict superset is solved, the first system is said to *subsume* the second system. The fraction of non-subsumed systems that fail on a problem is the difficulty rating for the problem. Problems that are solved by all non-subsumed systems get a rating of 0.00, and are considered to be *easy*; problems that are solved by just some of the non-subsumed systems get a rating between 0.00 and 1.00, and are considered to be *difficult*; problems that are *unsolved* get a rating of 1.00.

Figure 5 shows the problem ratings for TPTP v3.5.0, sorted into increasing order, for the FOF (leftmost plot) and CNF (middle plot) problems separately, and for all problems together (rightmost plot). The solid sequence of points along the "Problem number" bottom axis, with rating 0.00, come from easy problems. The solid sequences of points along the "Problem number" top axis, with rating 1.00, come from unsolved problems. There are 3,093 easy, 5,809 difficult, and 2,493 unsolved

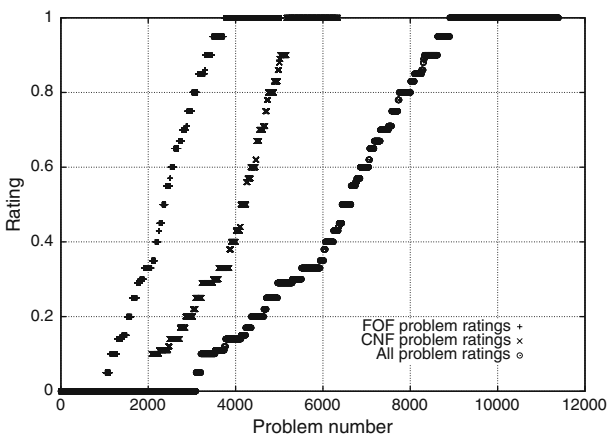

Fig. 5 TPTP v3.5.0 problem ratings

problems. Of these, 2,076, 3,089, and 1,181 are CNF problems, and 1,017, 2,720, and 1,312 are FOF problems, respectively. The plots show that the TPTP contains problems with well distributed difficulty, thus providing test problems that are easy enough for testing newly developed ATP systems, while also providing significant challenges for mature systems.

Over time, decreasing ratings for individual problems provide an indication of progress in the field [49]. While the individual problem ratings typically do show a downward trend, they can also increase when results from a new non-subsumed system are added to the calculations. This happens, for example, with the addition of results from a new system that implements a novel calculus, and is thus able to solve some previously unsolved problems, but is unable to solve some problems that have been solved by other systems. In this case the ratings of the newly solved problems drop down from 1.00, but the ratings of the solved problems increase. This effect has been noted in recent years, with the implementation of several new systems based on non-resolution/superposition calculi [46].

The analysis done for problem ratings also provides ratings for ATP systems, for each SPC. The rating of a system in an SPC is the fraction of the difficult problems that it solves. Systems that subsume all other systems get a rating of 1.00, and systems that solve only easy problems get a rating of 0.00.

4 TPTP Related Projects

4.1 The TSTP Solution Library

The Thousands of Solutions from Theorem Provers (TSTP) solution library [42], the "flip side" of the TPTP, is 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. At the time of writing this paper, the TSTP contained the results of running 44 ATP systems and system variants on all the problems in the TPTP that they can, in principle, attempt to solve (therefore, e.g., finite model finding systems are not run on problems that are known to be unsatisfiable). The ATP system versions used for building the TSTP are the most recent available, taken either from the systems' web sites, or from the most recent CASC—see Section 4.2.

The first section of each TSTP solution file is a header that contains information for users, analogous to the TPTP problem file header section. It contains: information about the TPTP problem, information about the ATP system, characteristics of the computer used, the SZS status and output dataform from the system, and statistics about the solution. The second section of each TSTP solution file contains the annotated formulae that make up the solution. A key feature of the TSTP is that solutions from many of the ATP systems are written in the TPTP language—the same language as used for TPTP problems. Formats for expressing derivations and finite interpretations have been defined [51], and formats for other forms of solutions, e.g., tableaux and natural deduction proofs, are under development. Some ATP systems write TPTP format output natively, while for others their output is translated to the TPTP format. The third section of each TSTP solution file contains the raw original output from the ATP system, formatted as TPTP comments.

Table 6 provides some statistics about the TSTP, illustrating mainly the large amount of data that is available. The "Problems" row gives the numbers of problems that are known to have that SZS status—the remaining problems have Unknown or Open status. The "Solved" row gives the numbers of "Problems" that have been solved by at least one ATP system, the "Solutions" row gives the numbers of "Problems" that have been solved with a solution (proof, refutation, model, etc.) output by at least one ATP system, and the "SuccessP" row is the ratio of the "Solution" and "Problems" rows. The "Attempts" row gives the numbers of ATP system runs on the "Problems", the "Solutions" row gives the numbers of runs with a solution output, and the "SuccessA" row is the ratio of the "Solutions" and "Attempts" rows. All the runs were done on 2.80 GHz computers with 1GB memory and running the Linux operating system, with a 600 s CPU limit. The key statistic is that the TSTP contains over 65,000 solutions to TPTP problems (of which over 58,000 are CNF refutations), which can be analyzed to understand and learn how ATP problems can be solved. Another observation is that there is plenty of room for improvement—there are 2,902 problems (out of the 11,395 problems—25%) in the TPTP that do not have any solutions in the TSTP.[2]

4.2 The CADE ATP System Competition

The CADE ATP System Competition (CASC) [53] is held annually at each CADE (or IJCAR, of which CADE is a constituent) conference. CASC evaluates the performance of sound, fully automatic ATP systems—it is the world championship for such systems. The primary purpose of CASC is a public evaluation of the relative capabilities of ATP systems. Additionally, CASC aims to stimulate ATP research in general, to stimulate ATP research towards autonomous systems, to motivate implementation of robust ATP systems, to provide an inspiring environment for personal interaction between ATP researchers, and to expose ATP systems within and beyond the ATP community.

The design and implementation of CASC is closely linked to the TPTP. The divisions and problem categories of CASC are similar to the SPCs used in the TPTP problem and system rating scheme. The divisions that have been used so far are: FOF—**F**irst-**O**rder **F**orm non-propositional theorems; CNF—**C**lause **N**ormal **F**orm unsatisfiable clause sets; SAT—**SAT**isfiable clause sets; EPR—**E**ffectively **PR**opositional (with a finite Herbrand universe) clause sets; UEQ—**U**nit **EQ**uality unsatisfiable clause sets; SEM—**SEM**antic (from a chosen TPTP domain) theorems; and LTB—**L**arge **T**heory (with very many axioms) theorems presented as a **B**atch. The problems used in CASC are taken from the TPTP problem library, and the TPTP edition used is not released until after the competition, so that new problems have not been seen by the entrants. The TPTP problem ratings are used to select appropriately difficult problems that differentiate between the systems in CASC.

[2] An observant reader might note that the numbers of problems solved are slightly different from the numbers with ratings less than 1.00 given in Section 3.6, e.g., here 8,915 are solved overall, while in Section 3.6 it is reported that there are 3,093 easy plus 5,809 difficult = 8,902 problems solved. This is because the ratings were computed in advance of CASC—see Section 4.2, at which point not all the systems entered into CASC were available. The TSTP data was finalized when all the CASC systems were available.

Table 6 TSTP statistics

	FOF (5,049 problems)					CNF (6,346 problems)			All 11,395
	THM	CSA	UNS	SAT	Total	UNS	SAT	Total	problems
Problems	4,254	424	139	87	4,094	4,600	824	5,424	9,518
Solved	3,181	363	137	70	3,751	4,377	787	5,164	8,915
Solution	2,812	362	135	70	3379	4,327	787	5,114	8493
SuccessP	0.66	0.85	0.97	0.80	0.83	0.94	0.96	0.94	0.89
Attempts	102,742	11,802	3,337	2,432	120,313	127,434	26,011	153,445	273,758
Solutions	20,248	1,859	1,040	404	23,551	38,163	3,881	42,044	65,595
SuccessA	0.20	0.16	0.31	0.17	0.20	0.30	0.15	0.27	0.24

The TPTP rating scheme provides the principles for the CASC rating scheme, which provides a realistic and stable ranking of the systems. Table 7 lists the CASC division winners over the years.

CASC has been a catalyst for impressive improvements in ATP, stimulating both theoretical and implementation advances [23]. First, new strategies and techniques have been developed to increase the range of problems that can be solved by individual systems, and second, the quality of implementations has improved. Possibly

Table 7 CASC division winners

	FOF	FNT	CNF	SAT
J4 (2008)	Vampire 10.0	MetaProver 1.0	Vampire 10.0	MetaProver 1.0
21 (2007)	Vampire 9.0	Paradox 2.2	Vampire 9.0	Paradox 1.3
J3 (2006)	Vampire 8.1		Vampire 8.1	Paradox 1.3
20 (2005)	Vampire 8.0		Vampire 8.0	Paradox 1.3
J2 (2004)	Vampire 7.0		Vampire 7.0	Gandalf c-2.6-SAT
19 (2003)	Vampire 5.0		Vampire 6.0	Gandalf c-2.6-SAT
18 (2002)	Vampire 5.0		Vampire 5.0	Gandalf c-2.5-SAT
JC (2001)	E-SETHEO csp01		Vampire 2.0	GandalfSat 1.0
17 (2000)	VampireFOF 1.0		E 0.6	GandalfSat 1.0
16 (1999)	SPASS 1.00T		Vampire 0.0	OtterMACE 437
15 (1998)	SPASS 1.0.0a		Gandalf c-1.1	SPASS 1.0.0a
14 (1997)	SPASS 0.77		Gandalf	SPASS 0.77
13 (1996)			E-SETHEO	
	EPR	UEQ	SEM	LTB
J4 (2008)	iProver 0.5	Waldmeister 806		SInE 0.3
21 (2007)	Darwin 1.3	Waldmeister 806		
J3 (2006)	Darwin 1.3	Waldmeister 806		
20 (2005)	DCTP 10.21p	Waldmeister 704		
J2 (2004)	DCTP 10.21p	Waldmeister 704		
19 (2003)	DCTP 1.3-EPR	Waldmeister 702		
18 (2002)	E-SETHEO csp02	Waldmeister 702		
JC (2001)	E-SETHEO csp01	Waldmeister 601		
17 (2000)		Waldmeister 600	E-SETHEO csp00	
16 (1999)		Waldmeister 799		
15 (1998)		Waldmeister 798		
14 (1997)		Waldmeister		
13 (1996)		Otter 3.0.4z		

the most important improvement has been in the selection of strategies according to the characteristics of the given problem—the "auto-mode"s now available in almost all ATP systems. There have been significant developments in this area, including a deeper understanding of what problem characteristics are important for what aspects of strategy selection [52], the examination of the input to detect the domain structure of the problem [14], the use of machine learning techniques to optimize the choice of strategy [34], and the use of strategy scheduling [39, 58, 64]. Other effects of CASC include increased interest in the production and verification of ATP system output [5, 8], the development and refinement of FOF to CNF converters, systems that are robust in terms of installation and execution, and improved data structures in ATP systems.

The positive effects of CASC on ATP system development have had reciprocal positive effects on the TPTP. Observers at the event have been encouraged to contribute their problems to the TPTP. The ATP systems entered into CASC are the most recent versions available, and after CASC they are run over the TPTP to build the TSTP solution library, which in turn provides updated problem and system ratings.

4.3 TPTP and TSTP Tools

The TPTP and TSTP are large data collections, and are necessarily supported by a suite of tools. Some of the tools were developed explicitly for users of the libraries, and some were initially developed to ease maintenance of the libraries, but have since also been made available to users. The most salient of these tools are discussed in this section.

4.3.1 The Problem and Solution Selection Tools, TPTP1T *and* TPTP2T

TPTP1T and TPTP2T are scripts that provide lists of TPTP problems and TSTP solutions that have user specified characteristics, e.g., for problems: FOF or CNF, no equality or some equality or pure equality, easy or difficult or unsolved; and for solutions: with particular results from particular ATP systems, with a solution using a certain fraction of the available axioms. Thus, for example, it is possible to find all FOF problems with some equality, that are difficult, and have been solved by Otter and not solved by Vampire. This is useful because it is not directly possible to find characteristic problems and solutions without such support. TPTP1T and TPTP2T provide user friendly options for specifying the required problem and solution characteristics. In addition to being part of the TPTP distribution, TPTP2T is available as an online service at www.tptp.org/cgi-bin/TPTP2T.[3]

4.3.2 The Problem and Solution Utilities, TPTP2X *and* TPTP4X

TPTP2X is a multi-functional utility for generating, transforming, and reformatting TPTP problem files. In particular, it controls the generation of problem files from

[3]Support for TPTP1T was discontinued after TPTP v3.5.0, as its functionality is subsumed by TPTP2T.

TPTP generator files, expands `include` directives to produce full versions of problem files, applies various transformations to TPTP problems, converts from the TPTP format to formats used by existing ATP systems, computes the statistics that appear in the header sections of problem files, and pretty prints TPTP problems and solutions. TPTP2X has been distributed with the TPTP from the very start, and was a key to the initial success of the TPTP. It allowed ATP developers to convert TPTP problems to suit existing parsers, etc., rather than the converse. As many ATP systems and tools now read and write TPTP format data natively, the critical value of TPTP2X as a format translator has diminished, and in place it is used for other manipulations of TPTP format data. As TPTP2X is written in Prolog it is easily modified and extended, and is thus useful as a platform for prototyping new ideas, new transformations, and new output formats.

With the development of the TSTP, and the addition of large FOF problems to the TPTP, it became necessary to provide faster processing of TPTP format data than is possible with TPTP2X. TPTP4X is a reimplementation of much of the TPTP2X functionality, in C. TPTP4X has additional capabilities for processing derivations in TPTP format, in particular for building and analyzing the DAG structures that represent derivations. For example, TPTP4X can extract all the lemmas that are derived from only axioms in a derivation.

4.3.3 The System and Tool Execution Harness, SystemOnTPTP

SystemOnTPTP is a utility that allows an ATP problem or solution in TPTP format to be easily and quickly submitted to ATP systems and tools. The utility uses a suite of currently available ATP systems and tools, whose properties (input format, command line, reporting of result status, etc.) are stored in a simple text database. The utility allows the input file to be selected from the TPTP library, or provided by the user. Multiple ATP systems and tools may be applied to each input file.

The implementation of SystemOnTPTP uses several subsidiary tools to pre-process the input, control the execution of the chosen ATP system(s), and post-process the output. On the input side, TPTP2X or TPTP4X is used to prepare the input for processing. A strict resource limiting program called `TreeLimitedRun` is used to limit the CPU time and memory used by the ATP systems and tools. `TreeLimitedRun` monitors processes more tightly than is possible with standard operating system calls. (`TreeLimitedRun` is also used in CASC.) Finally, a program called `X2tptp` converts an ATP system's output to TPTP format, if requested by the user.

4.3.4 The Semantic Derivation Verifier, GDV

ATP systems are complex pieces of software, and thus may have bugs that make them unsound or incomplete. While incompleteness is common (sometimes by design) and tolerable, when an ATP system is used in an application it is important, typically mission critical, that it be sound. Proof verification is thus an important post-processing task in ATP [8, 17, 36]. GDV [44] is a tool that uses structural and then semantic techniques to verify derivations in TPTP format.

Structural verification checks that inferences have been done correctly in the context of the derivation. The structural checks include: checking that the specified parents of each inference step do exist, checking that the derivation is acyclic,

checking that refutations end with a *false* formula, checking that assumptions have been discharged, checking that split refutations are not mutually dependent, and checking that introduced symbols (e.g., in Skolemization) are distinct.

Semantic verification checks the expected semantic relationship between the parents and inferred formula of an inference step. This is done by encoding the expectations into logical obligations in ATP problems, and then discharging the obligations by solving the problems with trusted ATP systems. The expected semantic relationship between the parents and inferred formula of an inference step depends on the intent of the inference rule used. For example, deduction steps expect the inferred formula to be a theorem of its parent formulae. The expected relationship is recorded as an SZS value in the `inference` record of an annotated formula. GDV uses SystemOnTPTP to execute the trusted ATP systems.

4.3.5 The Interactive Derivation Viewer, IDV

The derivations output by ATP systems provide useful information for users, e.g., the derivation structure, lemmas that may be used in future proofs, which axioms are most used, etc. However, the derivations are often unsuitable for human consumption, due to their text format, typically of very many fine grained inference steps. IDV [59] is a tool for graphical rendering and analysis of derivations in TPTP format. IDV provides an interactive interface that allows the user to quickly view features of the derivation, and access analysis facilities. These are best appreciated by using IDV through the online interface described in Section 4.3.6.

The lefthand side of Fig. 6 shows the rendering of the derivation output by the EP ATP system [34] for the TPTP problem `PUZ001+1`. The IDV window is divided into three panes: the top control pane contains control buttons and sliders, the middle main pane shows the rendered DAG, and the bottom text pane gives the text of the annotated formula for the node pointed to by the mouse.

The rendering of the derivation DAG uses shape and color to provide information about the derivation. The shape of a node shows the role of the formula, the outline color indicates FOF or CNF, and tags attached to a node indicate features of the inference step. The user can interact with the rendering in various ways. Mousing over a node highlights its ancestors and descendants in fading color intensities, clicking on a node pops up a window with details of the inference and with access to GDV to verify the inference, control-clicking on a node opens a new IDV window with that node and its parents, and shift-control-clicking on a node opens a new IDV window with that node and its ancestors.

The buttons and sliders in the control pane provide a range of manipulations on the rendering—zooming, hiding and displaying parts of the DAG according to various criteria, access to GDV for verification of the whole derivation, and access to the SystemOnTSTP interface. A particularly novel feature of IDV is its ability to provide a synopsis of a derivation by identifying interesting lemmas, and hiding less interesting intermediate formulae. This is implemented by calling the AGInT system [28] to evaluate the interestingness of each formula, and then using the interestingness slider to select an interestingness threshold to hide nodes for less interesting formulae. After extracting a synopsis it is possible to zoom in with the redraw button, rendering only the "interesting" nodes. A synopsis is shown on the righthand side of Fig. 6.

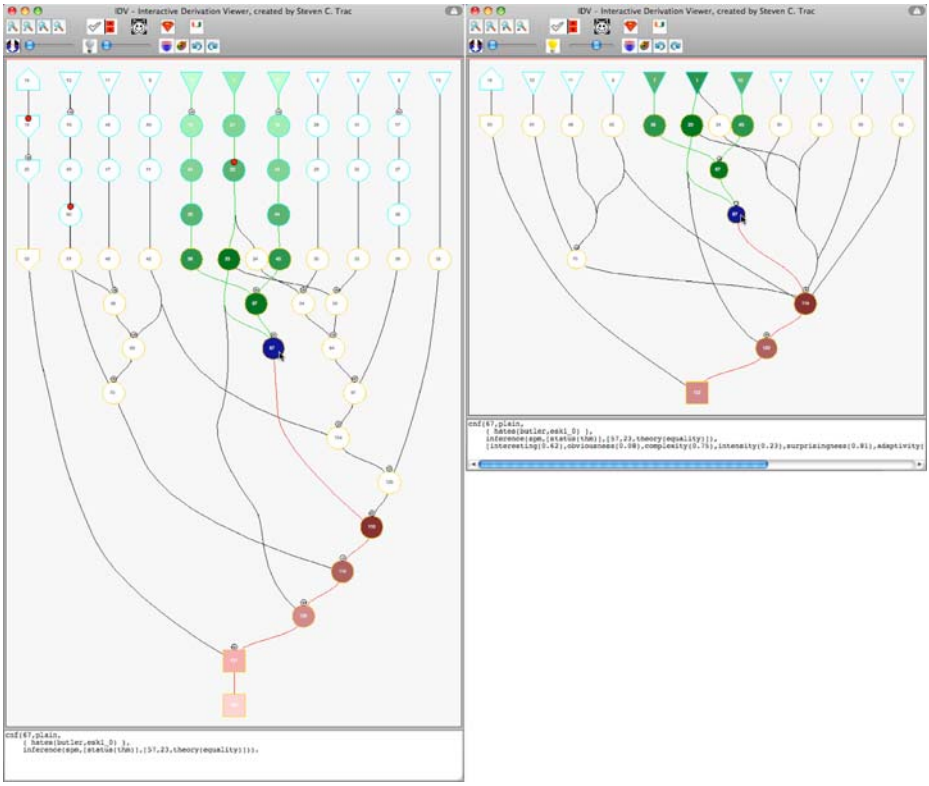

Fig. 6 EP's proof by refutation of `PUZ001+1`

4.3.6 Online Interfaces

The TPTP, TSTP, and many associated tools (including those described in the preceding subsections) are accessible online. The TPTP (www.tptp.org) and TSTP (www.tptp.org/TSTP) web pages provide access to all the problems, solutions, and documents related to the libraries, as well as subprojects and proposals for forthcoming extensions to the libraries. The SystemOnTPTP interface (www.tptp.org/cgi-bin/SystemOnTPTP) [43] provides online access to the SystemOnTPTP utility described in Section 4.3.3, and the SystemB4TPTP and SystemOnTSTP interfaces provide analogous access to tools for pre- and post-processing TPTP format data. The interfaces can be accessed using a web browser, or directly via `http POST` requests. The TPTP format data can be selected from the TPTP or TSTP library, or provided in TPTP format by the user as text, a file, or a URL source.

The SystemB4TPTP interface provides access to problem pre-processing tools, including TPTP2X and TPTP4X, conversion from FOF to CNF, and a range of parsers. It additionally provides a facility to convert a problem to TPTP format from some other formats. The SystemOnTPTP interface provides the core functionality of the SystemOnTPTP utility, to submit an ATP problem to ATP systems. It additionally provides system reports, and recommendations for which systems to use on a given problem, based on the SPCs and system ratings. The SystemOnTPTP interface

also has direct access to the SSCPA ATP meta-system [52], which runs multiple recommended ATP systems in competition parallel. Finally, the SystemOnTSTP interface provides access to derivation post-processing tools, including GDV and IDV.

5 Conclusion

The TPTP problem library provides the basis for meaningful empirical evaluation of ATP systems, and provides a support infrastructure that enables developers and users to take advantage of the data corpus. The TPTP has had significant effects on ATP research. The original goals for building the TPTP have been largely met - performance results based on the TPTP accurately reflect capabilities of the ATP systems being considered. The large scale testing enabled by the TPTP has led to the development of automatic result analysis, and effective approaches to ATP system tuning (e.g., [22, 33, 62]). The common TPTP format for data, and the SZS ontologies for status information, have eased the tasks of building ATP meta-systems (e.g., [39, 50, 52, 62]), and embedding ATP systems into more complex reasoning environments (e.g., [9, 21, 60]). The TPTP has inspired other automated reasoning and artificial intelligence communities to build their own test corpuses (e.g., [11, 31, 32]), and run competitions analogous to CASC (e.g., [2, 10]).

The next development in the TPTP is the addition of problems in Typed Higher-order Form (THF) [4]. The development of the higher-order part of the TPTP is taking a layered approach, with the first problems being written in the core THF0 fragment of the THF language. THF0 is based on Church's simple type theory, and provides the commonly used and accepted aspects of a higher-order logic language. The full THF language, and the extended THFX language, provide successively richer constructs in higher-order logic. TPTP v4.0.0, released in August 2009, contains THF problems. An interesting new possibility is easy comparison of higher-order and first-order versions of problems, and the evaluation of the relative benefits of the different encodings with respect to ATP systems for the logics. To date only the LEO II system [3] is known to accept the THF0 language, but export to the TPS [1], Twelf [26], and OmDoc [13] languages has made the problems accessible to other higher-order ATP systems. It is hoped that more higher-order systems and tools will adopt the THF language, making easy and direct communication between the systems and tools possible.

There are several developments on the TPTP horizon. The first is the Typed First-order Form (TFF) extension of the TPTP language, which adds a simple type system to the FOF language, compatible with the THF language. A proposal for this language has been established, and a prototype implementation is underway in the E ATP system [34]. The second, which depends on the first, is the specification of interpreted integer arithmetic functors and predicates in the TPTP language, to support problems that require arithmetic notions. The third is the specification of TPTP language features that allow a conjecture with outermost existentially quantified variables to be interpreted as a question, so that the variables' bindings in a proof can be returned as an answer to the question. A proposal for this language has been established, and is already implemented experimentally in the SNARK ATP system [38].

The keys to sustaining the value of the TPTP in the future are its continued growth, and continued adoption of the TPTP language, standards, and tools for

building automated reasoning software. ATP system developers and users are encouraged to build TPTP compliant ATP software, write ATP problems using the TPTP language, and to contribute ATP problems to the TPTP.

Acknowledgements Many people have contributed to this work. Most salient are: Christian Suttner, the co-developer of the TPTP library and CASC; Stephan Schulz and Koen Claessen who influenced the development of the TPTP language; Allen Van Gelder who wrote the core of the language BNF; and the ARTists Yi Gao, Yury Puzis, Alex Roederer, and Steven Trac, who co-designed and implemented some of the TPTP and TSTP tools.

References

1. Andrews, P.B., Bishop, M., Issar, S., Nesmith, D., Pfenning, F., Xi, H.: TPS: a theorem-proving system for classical type theory. J. Autom. Reason. **16**(3), 321–353 (1996)
2. Barrett, C., de Moura, L., Stump, A.: SMT-COMP: satisfiability modulo theories competition. In: Etessami, K., Rajamani, S. (eds.) Proceedings of the 17th International Conference on Computer Aided Verification. Lecture Notes in Computer Science, no. 3576, pp. 20–23. Springer, New York (2005)
3. Benzmüller, C., Paulson, L., Theiss, F., Fietzke, A.: LEO-II—a cooperative automatic theorem prover for higher-order logic. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) Proceedings of the 4th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 5195, pp. 162–170. Springer, New York (2008)
4. Benzmüller, C., Rabe, F., Sutcliffe, G.: THF0—the core TPTP language for classical higher-order logic. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) Proceedings of the 4th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 5195, pp. 491–506. Springer, New York (2008)
5. Bonichon, R., Delahaye, D., Doligez, D.: Zenon: an extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. Lecture Notes in Artificial Intelligence, no. 4790, pp. 151–165 (2007)
6. Boyer, R., Lusk, E.L., McCune, W.W., Overbeek, R., Stickel, M.E., Wos, L.: Set theory in first-order logic: clauses for Gödel's axioms. J. Autom. Reason. **2**(3), 287–327 (1986)
7. Comaromi, J.P., Beall, J., Matthews, W.E., New, G.R.: Dewey Decimal Classification and Relative Index, 20th edn. Forest, Albany (1989)
8. de Nivelle, H., Witkowski, P.: A small framework for proof checking. In: Schmidt, R., Konev, B., Schulz, S. (eds.) Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 4th International Joint Conference on Automated Reasoning. CEUR Workshop Proceedings, no. 373, pp. 81–93 (2008)
9. Denney, E., Fischer, B., Schumann, J.: Using automated theorem provers to certify auto-generated aerospace software. In: Rusinowitch, M., Basin, D. (eds.) Proceedings of the 2nd International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 3097, pp. 198–212 (2004)
10. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczynski, M.: The first answer set programming system competition. In: Baral, C., Bewka, G., Schlipf, J. (eds.) Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning. Lecture Notes in Artificial Intelligence, no. 4483, pp. 3–17. Springer, New York (2007)
11. Gent, I., Walsh, T.: CSPLib: a benchmark library for constraints. In: Jaffar, J. (ed.) Proceedings of the 5th International Conference on the Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, no. 1713, pp. 480–481. Springer, New York (1999)
12. Hoos, H., Stützle, T.: SATLIB: an online resource for research on SAT. In: Gent, I., van Maaren, H., Walsh, T. (eds.) Proceedings of the 3rd Workshop on the Satisfiability Problem (2001). http://www.satlib.org/
13. Kohlhase, M.: OMDoc—an open markup format for mathematical documents [version 1.2]. In: Lecture Notes in Artificial Intelligence, no. 4180. Springer, New York (2006)
14. Loechner, B., Hillenbrand, T.: A phytography of Waldmeister. J. AI Commun. **15**(2/3), 127–133 (2002)

15. Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J.: An introduction to the syntax and content of Cyc. In: Baralm C. (ed.) Proceedings of the 2006 AAAI Spring Symposium on Formalizing and Compiling Background Knowledge and Its Applications to Knowledge Representation and Question Answering, pp. 44–49 (2006)
16. McCharen, J.D., Overbeek, R.A., Wos, L.A.: Problems and experiments for and with automated theorem-proving programs. IEEE Trans. Comput. **C-25**(8), 773–782 (1976)
17. McCune, W., Shumsky-Matlin, O.: Ivy: a preprocessor and proof checker for first-order logic. In: Kaufmann, M., Manolios, P., Strother Moore, J. (eds.) Computer-Aided Reasoning: ACL2 Case Studies. Advances in Formal Methods, no. 4, pp. 265–282. Kluwer Academic, Dordrecht (2000)
18. McCune, W.W.: Single axioms for groups and Abelian groups with various operations. J. Autom. Reason. **10**(1), 1–13 (1993)
19. McCune, W.W.: Otter 3.3 reference manual. Technical report ANL/MSC-TM-263, Argonne National Laboratory, Argonne (2003)
20. McCune, W.W., Wos, L.: Experiments in automated deduction with condensed detachment. In: Kapur, D. (ed.) Proceedings of the 11th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, no. 607, pp. 209–223. Springer, New York (1992)
21. Meng, J., Quigley, C., Paulson, L.: Automation for interactive proof: first prototype. Inf. Comput. **204**(10), 1575–1596 (2006)
22. Newborn, M., Wang, Z.: Octopus: combining Learning and parallel search. J. Autom. Reason. **33**(2), 171–218 (2004)
23. Nieuwenhuis, R.: The impact of CASC in the development of automated deduction systems. AI Commun. **15**(2–3), 77–78 (2002)
24. Pease, A., Sutcliffe, G.: First order reasoning on a large ontology. In: Urban, J., Sutcliffe, G., Schulz, S. (eds.) Proceedings of the CADE-21 Workshop on Empirically Successful Automated Reasoning in Large Theories. CEUR Workshop Proceedings, no. 257, pp. 59–69 (2007)
25. Pelletier, F.J.: Seventy-five problems for testing automatic theorem provers. J. Autom. Reason. **2**(2), 191–216 (1986)
26. Pfenning, F., Schürmann, C.: System description: Twelf—a meta-logical framework for deductive systems. In: Ganzinger, H. (ed.) Proceedings of the 16th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, no. 1632, pp. 202–206. Springer, New York (1999)
27. Plaisted, D.A.: Non-Horn clause logic programming without contrapositives. J. Autom. Reason. **4**(3), 287–325 (1988)
28. Puzis, Y., Gao, Y., Sutcliffe, G.: Automated generation of interesting theorems. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference, pp. 49–54. AAAI, Menlo Park (2006)
29. Quaife, A.: Automated deduction in von Neumann-Bernays-Godel set theory. J. Autom. Reason. **8**(1), 91–147 (1992)
30. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchCyc: expressiveness and efficiency in a common sense knowledge base. In: Shvaiko, P. (ed.) Proceedings of the Workshop on Contexts and Ontologies: Theory, Practice and Applications (2005)
31. Ranise, S., Tinelli, C.: The SMT-LIB standard: version 1.2. Technical report, Department of Computer Science, The University of Iowa, Iowa City (2006)
32. Raths, T., Otten, J., Kreitz, C.: The ILTP problem library for intuitionistic logic—release v1.1. J. Autom. Reason. **38**(1–2), 261–271 (2007)
33. Schulz, S.: Learning search control knowledge for equational theorem proving. In: Baader, F., Brewka, G., Eiter, T. (eds.) Proceedings of the Joint German/Austrian Conference on Artificial Intelligence. Lecture Notes in Artificial Intelligence, no. 2174, pp. 320–334. Springer, New York (2001)
34. Schulz, S.: E: a brainiac theorem prover. AI Commun. **15**(2–3), 111–126 (2002)
35. Schulz, S., Bonacina, M.-P.: On handling distinct objects in the superposition calculus. In: Konev, B., Schulz, S. (eds.) Proceedings of the 5th International Workshop on the Implementation of Logics, pp. 66–77 (2005)
36. Siekmann, J.H., Benzmüller, C., Brezhnev, V., Cheikhrouhou, L., Fiedler, A., Franke, A., Horacek, H., Kohlhase, M., Meier, A., Melis, E., Moschner, M., Normann, I., Pollet, M., Sorge, V., Ullrich, C., Wirth, C.P., Zimmer, J.: Proof development with OMEGA. In: Voronkov, A. (ed.) Proceedings of the 18th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, no. 2392, pp. 143–148. Springer, New York (2002)
37. American Mathematical Society: Mathematical Subject Classification. American Mathematical Society, Providence (1992)

38. Stickel, M.E.: SNARK—SRI's New Automated Reasoning Kit. http://www.ai.sri.com/~stickel/snark.html
39. Streeter, M., Golovin, D., Smith, S.F.: Combining multiple heuristics online. In: Holte, R.C., Howe, A. (eds.) Proceedings of the 22nd Conference on Artificial Intelligence, pp. 1197–1203. AAAI, Menlo Park (2007)
40. Suchanek, F., Kasneci, G., Weikum, G.: YAGO: a core of semantic knowledge. In: Patel-Schneider, P., Shenoy, P. (eds.) Proceedings of the 16th International World Wide Web Conference, pp. 697–706 (2007)
41. Suda, M., Sutcliffe, G., Wischnewsk, P., Lamotte-Schubert, M.: External sources of axioms in automated theorem proving. In: Mertsching, B. (ed.) Proceedings of the 32nd Annual Conference on Artificial Intelligence (2009)
42. Sutcliffe, G.: The TSTP Solution Library. http://www.TPTP.org/TSTP
43. Sutcliffe, G.: SystemOnTPTP. In: McAllester, D. (ed.) Proceedings of the 17th International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, no. 1831, pp. 406–410. Springer, New York (2000)
44. Sutcliffe, G.: Semantic derivation verification. Int. J. Artif. Intell. Tools **15**(6), 1053–1070 (2006)
45. Sutcliffe, G.: TPTP, TSTP, CASC, etc. In: Diekert, V., Volkov, M., Voronkov, A. (eds.) Proceedings of the 2nd International Computer Science Symposium in Russia. Lecture Notes in Computer Science, no. 4649, pp. 7–23. Springer, New York (2007)
46. Sutcliffe, G.: The CADE-21 automated theorem proving system competition. AI Commun. **21**(1), 71–82 (2008)
47. Sutcliffe, G.: The SZS ontologies for automated reasoning software. In: Sutcliffe, G., Rudnicki, P., Schmidt, R., Konev, B., 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. CEUR Workshop Proceedings, no. 418, pp. 38–49 (2008)
48. Sutcliffe, G.: The TPTP problem library and associated infrastructure. The FOF and CNF Parts, v3.5.0. J. Autom. Reason. (2009). doi:10.1007/s10817-009-9143-8
49. Sutcliffe, G., Fuchs, M., Suttner, C.: Progress in automated theorem proving, 1997–1999. In: Hoos, H., Stützle, T. (eds.) Proceedings of the IJCAI'01 Workshop on Empirical Methods in Artificial Intelligence, pp. 53–60 (2001)
50. Sutcliffe, G., Puzis, Y.: SRASS—a semantic relevance axiom selection system. In: Pfenning, F. (ed.) Proceedings of the 21st International Conference on Automated Deduction. Lecture Notes in Artificial Intelligence, no. 4603, pp. 295–310. Springer, New York (2007)
51. Sutcliffe, G., Schulz, S., Claessen, K., Van Gelder, A.: Using the TPTP language for writing derivations and finite interpretations. In: Furbach, U., Shankar, N. (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 4130, pp. 67–81 (2006)
52. Sutcliffe, G., Seyfang, D.: Smart selective competition parallelism ATP. In: Kumar, A., Russell, I. (eds.) Proceedings of the 12th International FLAIRS Conference, pp. 341–345. AAAI, Menlo Park (1999)
53. Sutcliffe, G., Suttner, C.: The state of CASC. AI Commun. **19**(1), 35–48 (2006)
54. Sutcliffe, G., Suttner, C., Pelletier, F.J.: The IJCAR ATP system competition. J. Autom. Reason. **28**(3), 307–320 (2002)
55. Sutcliffe, G., Suttner, C.B.: The TPTP problem library: CNF release v1.2.1. J. Autom. Reason. **21**(2), 177–203 (1998)
56. Sutcliffe, G., Suttner, C.B.: Evaluating general purpose automated theorem proving systems. Artif. Intell. **131**(1–2), 39–54 (2001)
57. Sutcliffe, G., Zimmer, J., Schulz, S.: Communication formalisms for automated theorem proving tools. In: Sorge, V., Colton, S., Fisher, M., Gow, J. (eds.) Proceedings of the Workshop on Agents and Automated Reasoning, 18th International Joint Conference on Artificial Intelligence, pp. 52–57 (2003)
58. Tammet, T.: Gandalf. J. Autom. Reason. **18**(2), 199–204 (1997)
59. Trac, S., Puzis, Y., Sutcliffe, G.: An interactive derivation viewer. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, 3rd International Joint Conference on Automated Reasoning. Electronic Notes in Theoretical Computer Science, vol. 174, pp. 109–123 (2006)
60. Trac, S., Sutcliffe, G., Pease, A.: Integration of the TPTPWorld into SigmaKEE. In: Schmidt, R., Konev, B., Schulz, S. (eds.) Proceedings of the Workshop on Practical Aspects of Automated Reasoning, 4th International Joint Conference on Automated Reasoning. CEUR Workshop Proceedings, no. 373, pp. 103–114 (2008)

61. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. J. Autom. Reason. **37**(1–2), 21–43 (2006)
62. Urban, J., Sutcliffe, G., Pudlak, P., Vyskocil, J.: MaLARea SG1: machine learner for automated reasoning with semantic guidance. In: Baumgartner, P., Armando, A., Gilles, D. (eds.) Proceedings of the 4th International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 5195, pp. 441–456. Springer, New York (2008)
63. Van Gelder, A., Sutcliffe, G.: Extending the TPTP language to higher-order logic with automated parser generation. In: Furbach, U., Shankar, N. (eds.) Proceedings of the 3rd International Joint Conference on Automated Reasoning. Lecture Notes in Artificial Intelligence, no. 4130, pp. 156–161. Springer, New York (2006)
64. Wolf, A., Letz, R.: Strategy parallelism in automated theorem proving. Int. J. Pattern Recogn. Artif. Intell. **13**(2), 219–245 (1999)