TPTP, TSTP, CASC, etc.

Geoff Sutcliffe

University of Miami, USA
geoff@cs.miami.edu

Abstract. This paper gives an overview of activities and products that stem from the Thousands of Problems for Theorem Provers (TPTP) problem library for Automated Theorem Proving (ATP) systems. These include the TPTP itself, the Thousands of Solutions from Theorem Provers (TSTP) solution library, the CADE ATP System Competition (CASC), tools such as my semantic Derivation Verifier (GDV) and the Interactive Derivation Viewer (IDV), meta-ATP systems such as the Smart Selective Competition Parallelism (SSCPA) system and the Semantic Relevance Axiom Selection System (SRASS), and applications in various domains.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of systems that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. The dual discipline, automated model finding, develops computer programs that establish that a set of statements is consistent, and in this work we consider automated model finding to be part of ATP. These capabilities lie at the heart of many important computational tasks. For example, formal methods for software and hardware design and verification [Lam05, Das06], the analysis of network security protocols [AB04, Mit05], solving hard problems in mathematics [SFS95, McC97], and inference for the semantic web [FS005]. ATP has been highly successful when the problem is expressed in classical first order logic, so that a refutation or model of the clause normal form of the problem can be obtained. There are some well known high performance ATP systems that search for a refutation or model of a set of clauses, e.g., Darwin/DarwinFM [BFT06], E/EP [Sch02], Mace [McC03], Paradox [CS03], SPASS [WBH^{+}02], Vampire [RV02], and Waldmeister [Hil03]. Throughout this paper (until Section 8 that describes future plans) all discussion is in terms of ATP for classical first order logic.

2 TPTP

The TPTP (Thousands of Problems for Theorem Provers) problem library [SS98] is a well known standard set of test problems for ATP systems. The TPTP supplies a comprehensive library of the ATP 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 the capabilities of the ATP system being considered. The problems in the TPTP are collected from various sources. The two principal initial sources were existing electronic problem collections and the ATP literature. Since then many people and organizations have contributed to the TPTP. Users of ATP systems find that contributing samples of their problems to the TPTP provides exposure to ATP system developers, who then improve their systems' performance on the problems, which completes a cycle to provide the users with more effective tools.

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 current TPTP (v3.3.0) has thirty domains, in the fields of logic, mathematics, computer science, science and engineering, and social sciences, with domains ranging from combinatory logic to Smullyanesque puzzles. Each problem has a unique name that reflects its domain and encoding. Each problem file has a header section that contains information for the user, such as references, the problem rating (see Section 2.1), the problem status (see Section 2.2), etc. The logical formulae are wrapped with annotations that provide a unique name for each formula in the problem, a user role (axiom, conjecture, etc), and auxiliary user information. The logical formulae themselves use a consistent and easily understood notation. The syntax (see Section 2.3) shares many features with Prolog, a language that is widely known in the ATP community. Indeed, with a few operator definitions, units of TPTP data can be read in Prolog using a single `read`/1 call, and written with a single `writeq`/1 call.

A key to the initial and ongoing success of the TPTP is the TPTP2X utility. The most important feature of TPTP2X is the conversion of TPTP format problems to formats used by existing ATP systems. This functionality provides a very low entry barrier to using the TPTP with existing ATP systems that cannot read the TPTP format.

The availability of the TPTP has provided a stable basis for the meaningful evaluation of ATP systems, and published results can be readily compared with new results to determine progress in the field. Although other test problems do exist and are sometimes used, the TPTP is now the de facto standard for testing first order ATP systems.

2.1 Ratings

An important feature of the TPTP is the problem ratings [SS01]. The ratings provide an accurate measure of how difficult the problems are for state-of-the-art ATP systems. To rate problems, the performance of contemporary ATP systems on the problems is analyzed. The performance data comes from the TSTP, described in Section 3. The unbiased problems of the TPTP are divided into Specialist Problem Classes (SPCs) - syntactically identifiable classes of problems for which certain ATP techniques or systems have been observed to be especially well suited. Rating is done separately for each SPC, to provide a rating that compares “apples with apples”. A partial order between 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 union of the problems solved by the non-subsumed systems defines the state-of-the-art - all the problems that are solved by any 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 difficult; problems that are unsolved get a rating of 1.00.

The analysis done for problem ratings also provides ratings for the ATP systems. The fraction of the difficult unbiased problems that a system solves is the rating for that system. 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.

2.2 SZS

In order to use ATP systems’ results as input to other tools, it is necessary that the results correctly and precisely specify what has been established. The SZS ontology [SZS04] provides a fine grained ontology of result and output forms that are used to specify what has been established about a given ATP problem. The ontology also recommends the precise way in which the ontology values should be reported in the output from systems and tools. Figure 1 shows an extract from the top of the result ontology (the full ontology is available as part of the TPTP distribution). Each value has a full name and a three letter acronym that is useful for tables of data.

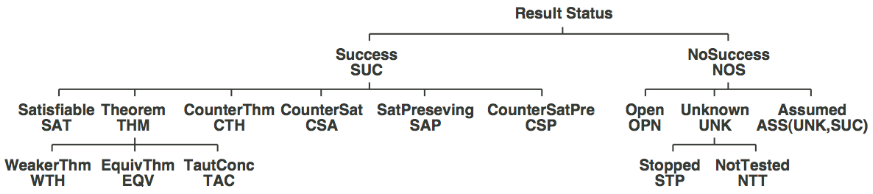


Fig. 1. SZS Ontology

At the top level the result ontology splits into two. The `Success` part catalogs semantic relationships between the axioms and conjecture (or its negation) of a problem. For example, if all models of the axioms are models of the conjecture then the status is `Theorem` with code `THM`, if some models of the axioms are models of the negation of the conjecture then the status is `CounterSatisfiable` with code `CSA`, and if there is a bijection between the models of the axioms and the models of the conjecture (as in a Skolemization step) then the status is `SatisfiabilityBijection` with code `SAB`. The `NoSuccess` part of the result ontology catalogs reasons that a system or tool could be not successful. For example, if a system stopped because the CPU time limit ran out then the status is `Timeout` with code `TMO`, and if a system has not attempted a problem but might in the future then the status is `NotTestedYet` with code `NTY`.

The output ontology catalogs forms of output from ATP systems and tools. For example, a prover might output a `CNFRefutation` with code `CRf`, and a model finder might output a `FiniteModel` with code `FMo`.

2.3 BNF

One of the keys to the success of the TPTP and related projects is their consistent use of the TPTP language, which enables convenient communication between different systems and researchers. TPTP v3.0.0 introduced a new version of the TPTP language [SSCVG06]. 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.

A principal goal of the development of the language grammar was to make it easy to translate the BNF into `lex`/`yacc`/`flex`/`bison` input, so that construction of parsers (in languages other than Prolog) can be a reasonably easy task [VGS06]. To this end the language definition uses a modified BNF meta-language that separates syntactic, semantic, lexical, and character-macro rules. The separation of syntax from semantics eases the task of building a syntactic analyzer. At the same time, the semantic rules provide the detail necessary for semantic checking.

The latest release of the grammar provides further structuring that allows users to build a parser for chosen components according to their need, including the FOF and CNF core, the TFF extension, extensions for theories (e.g., arithmetic), the THF core, and various THF extensions (see Section 8 regarding the TFF and THF languages).

3 TSTP

The TSTP (Thousands of Solutions from Theorem Provers) solution library [SutRL], the "flip side" of the TPTP, is becoming known as a resource for contemporary ATP systems' solutions. In particular, the TSTP contains solutions to problems from the TPTP. One 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.

A key development from the old (pre-v3.0.0) TPTP language to the new one was the addition of features for writing ATP solutions [SSCVG06], in a format consistent with ATP problems (see Sections 2 and 2.3). This enables output from ATP systems to be seamlessly used as input to further systems or tools. The features were designed for writing derivations, but their flexibility makes it possible to write a range of DAG structures. Additionally, there are features of the language that make it possible to conveniently specify finite interpretations.

At the time of writing this paper, the TSTP contains the results of running 44 ATP systems and system variants on all the problems in the TPTP. The results are classified according to the TPTP problem domains, then by TPTP problem. Each result file has a header section that contains information for the user, such as the system command line, information about the computer used, the SZS result and output status (see Section 2.2), and statistics about the solution. The output logical formulae use the TPTP language. Additional information that specifies a derivation's DAG structure, and details of inference steps, is used to annotate each formula, for use by tools such as GDV (see Section 5.4) and IDV (see Section 5.5).

4 CASC

In order to stimulate ATP system development, and to expose ATP systems to interested researchers, CASC (the CADE ATP System Competition) [SS06] is held at each CADE 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. Fulfillment of these objectives provides stimulus and insight for the development of more powerful ATP systems, leading to increased and more effective usage.

The design of CASC is linked to the problem and system rating scheme described in Section 2.1. The divisions and problem categories of CASC are similar to the SPCs used in the rating scheme. The problem ratings make it possible to select appropriately difficult problems for CASC, to differentiate between the systems. The rating scheme provides the principles for the CASC rating scheme, which provides a realistic and stable ranking of the systems. Table 1 lists the division winners over the years.

Through successive refinement of the competition design, CASC has been of significant benefit to the development of ATP [Nie02]. CASC has had two main effects on ATP system development. 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 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, the examination of the input to detect the domain structure of the problem, the use of machine learning techniques to optimize the choice of strategy, and the use of strategy scheduling. Other effects of CASC include increased interest in the production and verification of ATP system output, the development and refinement of FOF to CNF converters,

Table 1. CASC division winners

	FOF	CNF	SAT	EPR	UEQ
J3	Vampire 8.1	Vampire 8.1	Paradox 1.3	Darwin 1.3	Waldmeister 806
20	Vampire 8.0	Vampire 8.0	Paradox 1.3	DCTP 10.21p	Waldmeister 704
J2	Vampire 7.0	Vampire 7.0	Gandalf c-2.6-SAT	DCTP 10.21p	Waldmeister 704
19	Vampire 5.0	Vampire 6.0	Gandalf c-2.6-SAT	DCTP 1.3-EPR	Waldmeister 702
18	Vampire 5.0	Vampire 5.0	Gandalf c-2.5-SAT	E-SETHEO csp02	Waldmeister 702
JC	E-SETHEO csp01	Vampire 2.0	GandalfSat 1.0	E-SETHEO csp01	Waldmeister 601
17	VampireFOF 1.0	E 0.6	GandalfSat 1.0		Waldmeister 600
16	SPASS 1.00T	Vampire 0.0	OtterMACE 437		Waldmeister 799
15	SPASS 1.0.0a	Gandalf c-1.1	SPASS 1.0.0a		Waldmeister 798
14	SPASS 0.77	Gandalf	SPASS 0.77		Waldmeister
14		E-SETHEO			Otter 3.0.4z

systems that are robust in terms of installation and execution, and improved engineering and data structures in ATP systems.

5 Tools

The TPTP and TSTP are supported by a suite of tools for preparing and solving problems in TPTP format, and for processing solutions in TPTP format. Five of these tools are described in this section, along with WWW interfaces that provide global access to the TPTP, the TSTP, and the tools.

5.1 SystemOnTPTP

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

The implementation of SystemOnTPTP uses several subsidiary tools to preprocess the input, control the execution of the chosen ATP system(s), and postprocess the output. On the input side TPTP2X (see Section 2) is used to prepare the input for processing.[1] A strict resource limiting program called `TreeLimitedRun` is used to limit the CPU time and memory used. `TreeLimitedRun` monitors processes more tightly than is possible with standard operating system calls. (`TreeLimitedRun` is also used in CASC (see Section 4).) Finally a program called `X2tptp` converts an ATP system's output to TPTP format, if requested by the user.

5.2 Prophet

Prophet uses the syntax of an axiom formula F_a to gauge the potential for the axiom to contribute to a proof of a conjecture F_c, in the context of a set S of

[1] In some situations a faster, recently developed, alternative called TPTP4X is used.

axioms and the conjecture. First, the contextual direct relevance between all formulae in the set is measured by

$$\frac{\sum_{s\in(sym(F_a)\cap sym(F_c))}\left(1-\frac{|\{f:f\in S,s\in sym(f)\}|}{|S|}\right)}{|sym(F_a)\cup sym(F_c)|}$$

Next, the contextual path relevance of every path $F_a = F_1 \cdot F_2 \cdot \ldots \cdot F_n = F_c$ from F_a to F_c is calculated as the smallest contextual direct relevance in the path, divided by the length of the path. Finally, the contextual indirect relevance between F_a and F_c is taken as the maximal contextual path relevance over all paths connecting F_a to F_c.

Contextual indirect relevance can be used as a heuristic for selecting formulae to use in an ATP system – this is done in the SRASS system described in Section 6.2. A upgraded version of Prophet, based on deeper information retrieval concepts [Sah06], is being developed.

5.3 AGInT

AGInT (Automatic Generation of Interesting Theorems) [PGS06] is a tool that discovers interesting theorems of a given set of axioms. AGInT uses a deductive approach to discovery - it uses an ATP system to generate CNF logical consequences of the axioms, filters the logical consequences to extract interesting theorems, and then computes an interestingness rating for each theorem. This basic process takes place in the context of an outer level control loop that regularly refocuses the generation of logical consequences, thus enabling AGInT to proceed deeply into the search space of logical consequences. The overall architecture of AGInT is shown in Figure 2. The final output from AGInT is an ordered list of the interesting theorems retained in the interesting theorems store.

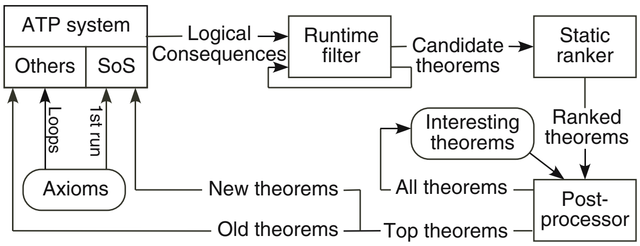


Fig. 2. AGInT Architecture

The runtime filter measures up to eight "interestingness" features of the formulae (some features are inappropriate in some situations): preprocessing detects and discards obvious tautologies, obviousness estimates the difficulty of proving a formula, weight estimates the effort required to read a formula, complexity estimates the effort required to understand a formula, surprisingness measures new relationships between function and predicate symbols in a formula, intensity measures how much a formula summarizes information from its leaf ancestors, adaptivity measures how tightly the universally quantified variables of a formula are constrained, and focus measures the extent to which a formula is making a positive or negative statement about the domain. Formulae that pass the majority of the runtime filters are passed to the static ranker. The static ranker combines the measures from the runtime filter with a measure of usefulness, which measures how much an interesting theorem has contributed to proofs of further interesting theorems. The scores are then normalized and averaged to produce an interestingness score.

AGInT has been evaluated in several domains and applications, ranging from puzzles to set theory. A particularly useful application has been in generating proof synopses in IDV, described in Section 5.5.

5.4 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. GDV (my Derivation Verifier) [Sut06] is a tool that uses structural and then semantic techniques to verify a derivation in TPTP format.

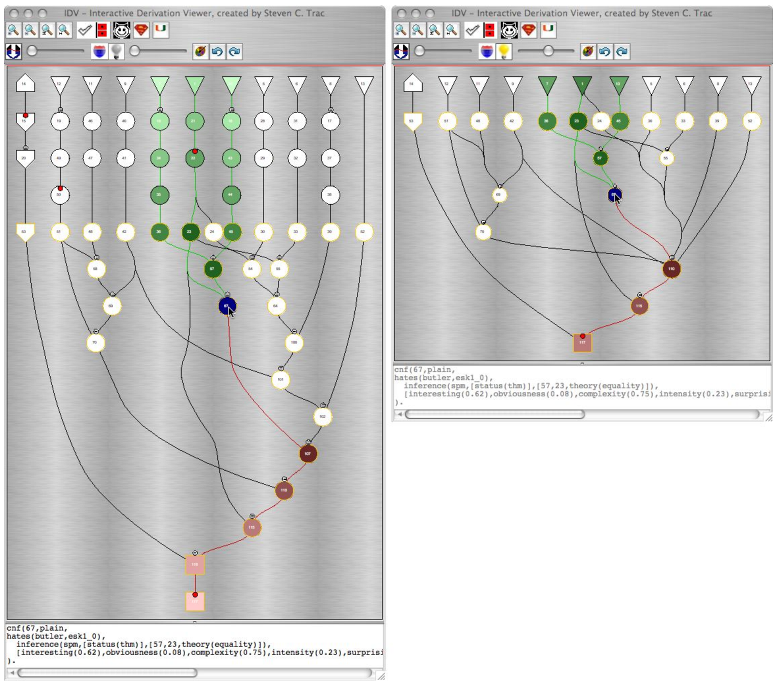
Structural verification checks that inferences have been done correctly in the context of the derivation. The structural checks include: checking that the specified parents of each inference step do exist, checking that the derivation is acyclic, checking that refutations end with a *false* formula, checking that assumptions are discharged, checking that split refutations are not mutually dependent, and checking that introduced symbols (e.g., in Skolemization) are distinct.

The core technique in semantic verification is to encode the expected semantic relationship between each inferred formula and its parent formulae into logical obligations, in the form of ATP problems. The obligations are then discharged by having trusted ATP systems solve the ATP problems. The required semantic relationship between an inferred formula and its parent formulae depends on the intent of the inference rule used. For example, deduction steps are verified by checking that the inferred formula is a logical consequence of its parent formulae. This intent is recorded as an SZS annotation to each inferred formula in TPTP format derivations (see Sections 2.2 and 3). GDV uses SystemOnTPTP to control the trusted ATP systems.

5.5 IDV

The proofs output by automated reasoning systems provide useful information to users, e.g., the proof structure, lemmas that may be useful in future proofs, which axioms are most used, etc. However, the proofs output by automated reasoning systems are often unsuitable for human consumption, due to, e.g., the conversion to CNF, use of proof by contradiction, and use of fine grained inference steps. IDV (Interactive Derivation Viewer) [TPS06] is a tool for graphical rendering of derivations in TPTP format. IDV provides an interactive interface that allows the user to quickly view various features of the derivation, and access various analysis facilities.

The lefthand side of Figure 3 shows the rendering of the derivation output by EP 0.99 for the TPTP problem `PUZ001+1`.[2] The IDV window is divided into three panes: the top control strip pane provides control buttons and sliders, the main middle pane shows the rendered DAG, and the bottom pane gives the text of the annotated formula for the node pointed to by the mouse.


IDV - Interactive Derivation Viewer, created by Steven C. Trac
IDV - Interactive Derivation Viewer, created by Steven C. Trac

```
cnf(67,plain,
hates(butler,esk1_0),
  inference(spm,[status(thm)],[57,23,theory(equality)]),
  [interesting(0.62),obviousness(0.08),complexity(0.75),intensity(0.23),surprisi
).
```

```
cnf(67,plain,
hates(butler,esk1_0),
  inference(spm,[status(thm)],[57,23,theory(equality)]),
  [interesting(0.62),obviousness(0.08),complexity(0.75),intensity(0.23),surprisi
).
```

Fig. 3. EP's Proof by Refutation of `PUZ001+1`

The rendering of the derivation DAG uses shape and color to visually provide information about the derivation. The shape of a node gives the user role of the formula, the color indicates FOF or CNF, and tags 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, 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 just that node and it's parents, and shift-control-clicking on a node opens a new IDV window with that node and its ancestors.

[2] `PUZ001+1` is the "Aunt Agatha" problem, a scenario in which one of the three people who live in the mansion killed Aunt Agatha. The goal is to prove that Aunt Agatha killed herself.

The buttons and sliders in the control strip 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 described in Section 5.6. A particularly novel feature of IDV is its ability to provide a synopsis of a derivation by identifying interesting lemmas within a derivation, and hiding less interesting intermediate formulae. This is implemented by calling AGInT 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 Figure 3.

5.6 WWW

All of the data and tools described in the preceding sections, and a few more besides, are accessible via a suite of online interfaces. The TPTP and TSTP interfaces 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 SystemB4TPTP, SystemOnTPTP [Sut00], and SystemOnTSTP interfaces provide access to an online service that uses the SystemOnTPTP utility to run selected ATP systems and tools on input specified by the user. The service can also be accessed directly via `http POST` requests. The input 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.

In addition to using the SystemOnTPTP utility, the SystemB4TPTP interface provides access to tools to convert from other input formats to TPTP format. The SystemOnTPTP interface additionally provides system reports, and recommendations for systems to use on a given problem, based on the system ratings (see Section 2.1). The SystemOnTPTP interface also has direct access to SSCPA (described in Section 6.1) to run multiple systems in competition parallel.

6 Meta-ATP

One of the benefits of having the common TPTP format for data, and the SZS ontology for status (see Section 2), is that it becomes easily possible to seamlessly integrate ATP systems and tools into more complex and effective reasoning systems. Two examples of such systems are described in this section.

6.1 SSCPA

One approach to developing more powerful ATP systems is the use of parallelism. SSCPA (Smart Selective Competition Parallelism ATP) [SS99] is an uncooperative multiple calculus competition parallelism ATP system. SSCPA runs multiple sequential ATP systems concurrently, in an SMP environment. This approach is motivated by the observation, e.g., in CASC, that no individual ATP system performs well on all problems. There is convincing evidence that the specialization of ATP systems is due to the deduction techniques used in relation to the syntactic characteristics of the problems. SSCPA uses the syntactic characteristics of a given problem to classify it into one of the specialist problem classes described in Section 2.1. SSCPA uses the system ratings (also described in Section 2.1) to determine which of the available ATP systems perform well for that class. A selection of the recommended systems are then run concurrently under the control of SystemOnTPTP (see Section 5.1), in one of several user selectable modes.

SSCPA was evaluated by entering it into the demonstration division of CASC-16 (see Section 4), running under the same conditions as the regular competition entries. In the mixed CNF and satisfiable CNF divisions SSCPA solved more problems than the competition winner. SSCPA also performed reasonably well in the first-order and unit equality divisions.

6.2 SRASS

In recent years the ability of ATP systems to reason over large theories – theories in which there are many functors and predicates, many axioms of which typically only a few are required for the proof of a theorem, and many theorems to be proved from the same set of axioms – has become more important. Large theory problems are becoming more prevalent as large knowledge bases, e.g., ontologies and large mathematical knowledge bases, are translated into forms suitable for automated reasoning [Qua92, Urb07, RPG05], and mechanical generation of ATP problems becomes more common, e.g., [DFS04, MP06]. SRASS (Semantic Relevance Axiom Selection System) [SP07] is a system for selecting necessary axioms, from a large set also containing superfluous axioms, to obtain a proof of a conjecture.

The basic algorithm of SRASS is to start with a set containing the negation of the conjecture to be proved, then repeatedly find a model of the set and augment the set with an axiom that is $false$ in the model, until no model exists. In this state the conjecture is a logical consequence of the selected axioms. SRASS augments this basic algorithm with extensions that improve the implemented performance. One of the keys to the success of SRASS is the use of Prophet (see Section 5.2) to measure the relevance of the axioms to the conjecture, to determine the order in which axioms are considered.

The implementation of SRASS uses a range of conventional ATP systems to implement the various tests and evaluations required: a theorem prover (currently E/EP 0.99) to test for (counter)theoremhood, test for unsatisfiability, and to find explicit proofs; a finite model builder (currently DarwinFM 1.3g) to test for (counter)satisfiability and build models; and a saturating system (currently SPASS 2.2) to further test for (counter)satisfiability in cases where the finite model builder fails and it is necessary only to establish the existence of a model. The various ATP systems are run under the control of SystemOnTPTP (see Section 5.1).

SRASS was tested in a conservative configuration on several problem sets from the TPTP – problems in logical calculi, problems in set theory, and problems in software verification, all of which are known to have superfluous axioms. In summary, of the 71 problems that were difficult enough to be eligible for comparative testing, SRASS solves 54 while the underlying ATP system (E/EP 0.99) solves only 39 without the benefit of axiom selection. SRASS was also tested in a less conservative configuration on the MPTP Challenge problems [US06]. In the bushy division of the challenge SRASS solves 171 of the 252 problems, compared to E/EP's 141, and in the chainy division (in which the problems have many more superfluous axioms) SRASS solves 127, compared to E/EP's 91.

7 Applications

The TPTP language and tools have been adopted for a range of user applications. The users employ ATP systems as embedded components of some larger process. By using the TPTP framework the users are not distracted by idiosyncrasies of automated reasoning, and can focus on their application. This section describes three such applications.

7.1 NASA

Research scientists in the Robust Software Engineering Group of the Intelligent Systems Division of NASA Ames have developed, implemented, and evaluated a certification approach that uses Hoare-style techniques to formally demonstrate the safety of aerospace programs that are automatically generated from high-level specifications [DF03, DFS04]. The focus is on automated – as opposed to interactive or (the auto-modes of) tactic-based – systems, since the aim is to have a fully automated push-button tool.

In this work the code generator was extended so that it simultaneously generates code and detailed annotations, e.g., loop invariants, regarding safety conditions. A verification condition generator processes the annotated code, and produces a set of safety obligations in the form of TPTP format problems that are provable if and only if the code is safe. The obligation problems are discharged using SSCPA (see Section 6.1), selecting up to three ATP systems, to produce TPTP format proofs that serve as safety certificates for authorities like the FAA. The derivations are verified by GDV (see Section 5.4). The individual derivations from GDV, which verify each inference step, provide explicit evidence that none of the individual tool components yield incorrect results and, hence, that the certificates are valid.

7.2 MPTP

The goal of the MPTP project [Urb07] is to make the large formal Mizar Mathematical Library (MML) [Rud92] available to current ATP systems (and vice versa), and to boost the development of domain-based, knowledge-based, and generally AI-based ATP methods. The MPTP converts Mizar format problems to a extended TPTP language that adds term-dependent sorts and abstract (Fraenkel) terms to the TPTP syntax. Problems in the extended language are transformed to standard TPTP format using relativization of sorts and deanonymization of abstract terms. Finding proofs for these problems provides cross verification of the underlying Mizar proofs. It is interesting that some of the ATP proofs correspond to shorter Mizar proofs of the original theorems, and therefore are likely to be used for MML refactoring.

Mizar proofs are also exported, as TPTP format derivations, allowing a number of ATP experiments and use of TPTP tools. An example of this is the combination of the Mizar WWW view with IDV (see Section 5.5) [UTSP07]. This allows a user to view and interact with Mizar level proofs, and to export these to IDV and beyond to view and interact with the corresponding first-order form.

7.3 SUMO and Cyc

In recent years there has been a growing interest in translating large ontological knowledge bases into first-order logic, so that ATP systems can be used to reason over the knowledge. Two examples of this are the translation of the Suggested Upper Merged Ontology (SUMO) [NP01] and of Cyc [MJWD06].

The translation of SUMO [PS07] requires dealing with some apparently and some truly second order constructs, adding guards to impose sort constraints, and converting from SUMO's SUO-KIF language to the TPTP language. Testing on a suite of reasoning tasks provided feedback on what choices in the translation process provide the most easily solved first-order problems. The translation has been integrated into the Sigma ontology development environment [Pea03], with access to IDV (see Section 5.5) for displaying derivations.

The FOLification of Cyc [RPG05] translated about 90% of ResearchCyc into first-order logic, in the TPTP format. As with the SUMO translation, special treatment of higher order constructs was necessary. The translation produced 1,253,117 axioms over 132,116 symbols (not including strings or numbers). This very large background theory presented practical difficulties for using ATP systems. With the exception of E/EP, none of the ATP systems tried were able to load more than 20% of the axioms without failing due to memory errors. This highlighted the need for reengineering of ATP systems, in order to cope with such large problems. A second translation of Cyc is now underway, in order to generate problems that can be added to the TPTP library as challenges to ATP systems.

8 Future

The TPTP and related projects are ongoing efforts, continuously aiming to extend the range and scope of TPTP compliant data and tools. Three main developments are planned for the near future.

Following discussions at the workshop on Empirically Successful Higher Order Logic (ESHOL) [BHS05], a typed higher-order TPTP syntax has been developed - the THF syntax. The THF syntax is divided into levels, starting with a simply typed Church lambda calculus core, and providing three layers of more complex constructs. This layered approach lowers the entry barrier for adopting the THF syntax, but also provides a rich language for ongoing development. With the syntax in place it is now planned to extend the TPTP and TSTP to include higher-order problems and solutions. The THF effort also resulted in completion of the typed first-order syntax (the TFF syntax), and the two are compatible.

Many applications of automated reasoning, including all those described in Section 7, require some simple reasoning over numbers. An extension of the TPTP language to provide interpreted arithmetic functors and predicates has been designed, aligned with the theory of integers in the Satisfiability Modulo Theories (SMT) library [RT06]. It is planned to extend the TPTP and TSTP to include problems and solutions that involve arithmetic.

The TPTP, TSTP, and tools, provide a stable environment for using ATP systems. While the ability to find solutions to ATP problems is useful directly, in many applications further features are necessary. Two such features are answer variables - the ability to extract an answer to a question that has been framed as a conjecture, and access to provenance information – information regarding information sources, assumptions, learned information, and answers, as an enabler for trust. Preliminary work on these and similar topics is now underway. Cyc's handling of answer variables [MJWD06] provides a starting point for determining the features and capabilities of answer variables in the TPTP, and issues of provenance information are being inspired by the work in the Inference Web [MPdS04].

9 Conclusion

This paper has given an overview of activities and products that stem from the TPTP problem library for ATP systems. The TPTP language, the SZS ontology, and the tools developed, provide an homogeneous environment for ongoing research, development, and application of automated reasoning. Contributions and feedback to improve the TPTP world are always welcome.

Acknowledgements. Many people have contributed to this work. Most salient are: Christian Suttner, the co-developer of the TPTP library and CASC; Stephan Schulz, who influenced the development of the new TPTP language; Allen Van Gelder who wrote the core of the BNF; the ARTists Yi Gao, Yury Puzis, and Steven Trac, who co-designed and implemented some of the tools; Petr Pudlak, who inspired SRASS, Bernd Fischer, Josef Urban, Adam Pease, and the Cyclists at Cycorp, who developed the applications described.

References

[AB04] Abadi, M., Blanchet, B.: Analyzing Security Protocols with Secrecy Types and Logic Programs. Journal of the ACM (2004)

[BFT06] Baumgartner, P., Fuchs, A., Tinelli, C.: Implementing the Model Evolution Calculus. International Journal on Artificial Intelligence Tools 15(1), 21–52 (2006)

[BHS05] Proceedings of the Workshop on Empirically Successful Higher-Order Logic. In: 12th International Conference on Logic for Programming Artificial Intelligence and Reasoning, vol. 0601042 of arXiv (2005)

[CS03] Claessen, K., Sorensson, N.: New Techniques that Improve MACE-style Finite Model Finding. In: Baumgartner, P., Fermueller, C. (eds.) Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications (2003)

[Das06] Das, M.: Formal Specifications on Industrial-Strength Code - From Myth to Reality. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, p. 1. Springer, Heidelberg (2006)

[DF03] Denney, E., Fischer, B.: Correctness of Source-level Safety Policies. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 894–913. Springer, Heidelberg (2003)

[DFS04] Denney, E., Fischer, B., Schumann, J.: Using Automated Theorem Provers to Certify Auto-generated Aerospace Software. In: Basin, D., Rusinowitch, M. (eds.) IJCAR 2004. LNCS (LNAI), vol. 3097, pp. 198–212. Springer, Heidelberg (2004)

[FS005] Fages, F., Soliman, S. (eds.): PPSWR 2005. LNCS, vol. 3703. Springer, Heidelberg (2005)

[Hil03] Hillenbrand, T.: Citius altius fortius: Lessons Learned from the Theorem Prover Waldmeister. In: Dahn, I., Vigneron, L. (eds.) Proceedings of the 4th International Workshop on First-Order Theorem Proving. Electronic Notes in Theoretical Computer Science, vol. 86.1 (2003)

[Lam05] Lam, W.: Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, Englewood Cliffs (2005)

[McC97] McCune, W.W.: Solution of the Robbins Problem. Journal of Automated Reasoning 19(3), 263–276 (1997)

[McC03] McCune, W.W.: Mace4 Reference Manual and Guide. Technical Report ANL/MCS-TM-264, Argonne National Laboratory, Argonne, USA (2003)

[Mit05] Mitchell, J.: Security Analysis of Network Protocols: Logical and Computational Methods. In: Barahona, P., Felty, A. (eds.) Proceedings of the 7th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, pp. 151–152 (2005)

[MJWD06] Matuszek, C., Cabral, J., Witbrock, M., DeOliveira, J.: An Introduction to the Syntax and Content of Cyc. In: Baral, 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)

[MP06] Meng, J., Paulson, L.: Translating Higher-Order Problems to First-Order Clauses. In: Sutcliffe, G., Schmidt, R., Schulz, S. (eds.) Proceedings of the FLoC'06 Workshop on Empirically Successful Computerized Reasoning, 3rd International Joint Conference on Automated Reasoning, CEUR Workshop Proceedings, vol. 192, pp. 70–80 (2006)

[MPdS04] McGuinness, D., Pinheiro da Silva, P.: Explaining Answers from the Semantic Web: The Inference Web Approach. Journal of Web Semantics 1(4), 397–413 (2004)

[Nie02] Nieuwenhuis, R.: The Impact of CASC in the Development of Automated Deduction Systems. AI Communications 15(2-3), 77–78 (2002)

[NP01] Niles, I., Pease, A.: Towards A Standard Upper Ontology. In: Welty, C., Smith, B. (eds.) Proceedings of the 2nd International Conference on Formal Ontology in Information Systems, pp. 2–9 (2001)

[Pea03] Pease, A.: The Sigma Ontology Development Environment. In: Giunchiglia, F., Gomez-Perez, A., Pease, A., Stuckenschmidt, H., Sure, Y., Willmott, S. (eds.) Proceedings of the IJCAI-03 Workshop on Ontologies and Distributed Systems, CEUR Workshop Proceedings, vol. 71 (2003)

[PGS06] 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 Press, Stanford, California, USA (2006)

[PS07] 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 (2007)

[Qua92] Quaife, A.: Automated Development of Fundamental Mathematical Theories. Kluwer Academic Publishers, Dordrecht (1992)

[RPG05] 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)

[RT06] Ranise, S., Tinelli, C.: The SMT-LIB Standard: Version 1.2. Technical Report Technical Report, Department of Computer Science, The University of Iowa, Iowa City, USA (2006)

[Rud92] Rudnicki, P.: An Overview of the Mizar Project. In: Proceedings of the 1992 Workshop on Types for Proofs and Programs, pp. 311–332 (1992)

[RV02] Riazanov, A., Voronkov, A.: The Design and Implementation of Vampire. AI Communications 15(2-3), 91–110 (2002)

[Sah06] Sahami, M.: Mining the Web to Determine Similarity Between Words, Objects, and Communities. In: Sutcliffe, G., Goebel, R. (eds.) Proceedings of the 19th International FLAIRS Conference, pp. 14–19. AAAI Press, Stanford, California, USA (2006)

[Sch02] Schulz, S.: E: A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)

[SFS95] Slaney, J.K., Fujita, M., Stickel, M.E.: Automated Reasoning and Exhaustive Search: Quasigroup Existence Problems. Computers and Mathematics with Applications 29(2), 115–132 (1995)

[SP07] Sutcliffe, G., Puzis, Y.: SRASS - a Semantic Relevance Axiom Selection System. In: Pfenning, F. (ed.) Proceedings of the 21st International Conference on Automated Deduction. LNCS (LNAI), vol. 4603, pp. 295–310. Springer, Heidelberg (2007)

[SS98] Sutcliffe, G., Suttner, C.B.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)

[SS99] 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 Press, Stanford, California, USA (1999)

[SS01] Sutcliffe, G., Suttner, C.B.: Evaluating General Purpose Automated Theorem Proving Systems. Artificial Intelligence 131(1-2), 39–54 (2001)

[SS06] Sutcliffe, G., Suttner, C.: The State of CASC. AI Communications 19(1), 35–48 (2006)

[SSCVG06] 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.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 67–81. Springer, Heidelberg (2006)

[Sut00] Sutcliffe, G.: SystemOnTPTP. In: McAllester, D. (ed.) Automated Deduction - CADE-17. LNCS, vol. 1831, pp. 406–410. Springer, Heidelberg (2000)

[Sut06] Sutcliffe, G.: Semantic Derivation Verification. International Journal on Artificial Intelligence Tools 15(6), 1053–1070 (2006)

[SutRL] Sutcliffe, G.: The TSTP Solution Library. `http://www.TPTP.org/TSTP`

[SZS04] Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In: Zhang, W., Sorge, V. (eds.) Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems. Frontiers in Artificial Intelligence and Applications, vol. 112, pp. 201–215. IOS Press, Amsterdam (2004)

[TPS06] Trac, S., Puzis, Y., Sutcliffe, G.: An Interactive Derivation Viewer. In: Autexier, S., Benzmüller, C. (eds.) Proceedings of the 7th Workshop on 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)

[Urb07] Urban, J.: MPTP 0.2: Design, Implementation, and Initial Experiments. Journal of Automated Reasoning 37(1-2), 21–43 (2007)

[US06] Urban, J., Sutcliffe, G.: The MPTP $100 Challenges (2006), `http://www.tptp.org/MPTPChallenge/`

[UTSP07] Urban, J., Trac, S., Sutcliffe, G., Puzis, Y.: Combining Mizar and TPTP Semantic Presentation Tools. In: Proceedings of the Mathematical User-Interfaces Workshop 2007 (2007)

[VGS06] Van Gelder, A., Sutcliffe, G.: Extending the TPTP Language to Higher-Order Logic with Automated Parser Generation. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 156–161. Springer, Heidelberg (2006)

[WBH$^+$02] Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topic, D.: SPASS Version 2.0. In: Voronkov, A. (ed.) Automated Deduction - CADE-18. LNCS (LNAI), vol. 2392. Springer, Heidelberg (2002)