TSTP Data-Exchange Formats for Automated Theorem Proving Tools

Geoff Sutcliffe[1], Jürgen Zimmer[2], Stephan Schulz[3]
[1]*University of Miami,* [2]*Universität des Saarlandes,*
[3]Technische Universität München and RISC/Linz

Abstract. This paper describes two data-exchange formats for Automated Theorem Proving (ATP) tools. First, a language for writing the problems that are input to ATP systems, and for writing the solutions that are output from ATP systems, is described. Second, a hierarchy of values for specifying the logical status of an ATP problem, as may be established by an ATP system, is described. These data-exchange formats will support application and research in ATP, and will facilitate communication between ATP tools in distributed and embedded environments.

1 Introduction

Automated Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the conjecture) is a logical consequence of a set of statements (the axioms). 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. ATP systems are used in a wide variety of domains: problems in mathematics have been solved, e.g., [SFS95, McC97], software and hardware have been designed and verified, e.g., [WSF02, CHM02], and applications to the internet seem possible [HS01]. 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., E [Sch02], Gandalf [Tam97], MACE [McC01], Paradox [CS03], SPASS [WBH$^+$02], and Vampire [RV02]. This paper describes two data-exchange formats for ATP tools for classical first order logic. First, a language for writing the problems that are input to ATP systems, and for writing the solutions that are output from ATP systems, is described. Second, a hierarchy of values for specifying the logical status of an ATP problem, as may be established by an ATP system, is described. The general designs are suitable for logics other than classical first order logic, and it would be desirable to adapt or extend these formats to provide for communication between first order ATP systems and systems for other logics, e.g., Coq [CoqRL], HOL [HOLRL], ACL2 [ACLRL].

The success of ATP systems is in large part attributable to progress in ATP research. Due to the semi-decidable nature of reasoning in first order logic, advances in ATP research are in part reliant on empirical analysis of system performance. This reliance drives a need for problem and solution libraries that can form the basis for analysis and comparisons. This need is met by the TPTP (Thousands of Problems for Theorem Provers) problem library [SS98]

and the TSTP (Thousands of Solutions from Theorem Provers) solution library [SutRL]. The TPTP is a comprehensive collection of the ATP test problems that are available today, along with documentation and utilities for accessing the library. The TSTP is the "flip side" of the TPTP, being a comprehensive collection of solutions to TPTP problems, produced by currently available ATP systems. When the TPTP was initiated in 1992, it was decided to write the formulae in a syntax that could be read directly in Prolog, so as to provide a very low entry barrier for use. The syntax was designed for problems in clause normal form (CNF), and a separate syntax for problems in "natural" first order form (FOF) was introduced in 1997. When the TSTP was initiated in 2002, it was necessary to design a language for writing solutions to ATP problems. The TSTP syntax extends the TPTP syntaxes, so as to be suitable for writing both ATP problems and their solutions. The TSTP syntax is described in Section 2.

As ATP systems move into real application areas, they are typically embedded as just one component in a larger system, including tools for proof analysis, transformation, presentation, and verification. In this environment, the output from one tool is often used as input to another. It is therefore desirable to have a common syntax for problem input and solution output, so that system components can communicate directly without the need for middleware to translate between their various input and output formats. The TSTP syntax meets these needs. In addition to being able to communicate formulae directly between tools, it is also necessary to be able to communicate precisely what has been established regarding the logical status of the formulae. The TSTP status hierarchy, described in Section 3, is an adequate hierarchy of such status values. In general, communication standards are crucial for the interoperability of distributed reasoning agents, and are similarly important for the description of deduction components as online services. Both multi-agent communication and service descriptions depend on commonly agreed ontologies and languages to encode reasoning problems and the outputs of reasoning systems.

The TSTP data-exchange formats are highly pragmatic and easy to adopt. The ease with which the new formats can be installed into existing and new ATP tools is illustrated by the fact that input and output in TSTP format was added to the equational theorem prover E in just a few days, and that a number of tools have already been built around the formats. Some sample applications are described in Section 4.

2 The TSTP Problem and Solution Language

Several different syntaxes exist for writing the first order logic problems that are input to ATP systems. Some of these syntaxes were originally developed for particular ATP systems; the Otter syntax [McC94], and the LOP syntax used in SETHEO [MIL+97] are examples. These system specific syntaxes often assume or support features specific to the system, and are therefore not consistently suitable for general use. Other syntaxes have been designed for general purpose use. The Knowledge Interchange Format (KIF) [GF92] is a logically comprehensive language for the representation of knowledge, and has declarative semantics. KIF provides expressive power beyond that required for ATP. The "DFG" syntax [HKW96] was designed as a common exchange format for logic problems used by members of the German DFG-Schwerpunktprogramm Deduction. DFG has a prefix-style grammar that is neither particularly easy for programs to parse nor for humans to read and write. Moreover, despite its aim, it is not very widely used. The Common Logic (CL) syntax [CL-RL] is a

framework for a family of logic-based languages. It does not specify any concrete syntax, but rather specifies an abstract syntax that can be specialized to a concrete language. CL grew out of work on the KIF language, and at this time the design of CL has not been completed. The OMDoc [Koh00], OpenMath [CC99], and MathML [CC99] languages specify XML based syntaxes for writing mathematical notions. These languages are quite expressive, but require a large amount of mark-up for quite simple content. For humans, reading and writing problems in these languages is difficult without specialized software. The syntax currently used for problems in the TPTP problem library is widely used in the ATP community. It has a simple syntax, but has the weakness that the CNF syntax is not a sub-syntax of the FOF syntax (while CNF formulae are a subset of FOF formulae).

All of the syntaxes described above were designed for writing the *input* to logical reasoning tools. There appear to be no consistently used general purpose formats for writing the *output* from reasoning tools. The PCL format [DS94, DS96] was designed for and is limited to the unit equality fragment of first order logic - it is used for the output from the E, DISCOUNT [DKS97] and Waldmeister [LH02] ATP systems. Although many of the above input formats could be (and in some cases are) used for writing the formulae output by reasoning tools, none were designed or contain features specific for comprehensively capturing output information. To provide seamless communication between reasoning tools it is necessary that the output from one tool should immediately be suitable as input for other tools.

The goal of designing a language for communication between reasoning tools cannot ignore the element of human readability. Human users, who are not necessarily ATP experts, are often the source of the initial input to, and the destination of the final output from, ATP tools. Human data formats are often semi-formal, and translation is required to and from a machine usable format, e.g., [Fie01, MRS01]. In an ideal world it would not be necessary for humans to look at the machine usable input and output, nor at intermediate data being passed between tools in a component based system. However, in reality, close examination of the input, output, and intermediate data of ATP tools is necessary for debugging, understanding of system behavior, and development of ideas [WP99]. It is thus necessary that the machine usable format be also human readable, at least to ATP developers and experts. It may not be possible to design a language that perfectly suits all the needs, but certainly a syntax that is designed for only machine processing, and ignores human readability, will not meet the needs of the ATP developers and experts upon whom external human users rely for technical support. One way around this issue is to provide two languages, one for humans and one for machines. This approach, however, requires extra software support, and the necessary translations may hide relevant details and may introduce errors.

The TSTP syntax is a comprehensive syntax, suitable for writing the input and output of ATP tools. The TSTP syntax was designed with the following aims and constraints:

- It must be able to completely express the problems that are input to ATP systems.
- It must be able to capture sufficient details of ATP systems' outputs to allow various forms of post processing, e.g., various styles of proof verification and presentation.
- The same syntax must be used for both input and output.
- It must be possible to annotate formulae with arbitrary information.
- It must be easy for humans to read and write.

  - It must be easy to write using a plain text editor.
  - It must be compact.
- It must be easy for programs to parse.
- It should be backward compatible with the TPTP syntax, but a single syntax must be used for CNF and FOF formulae.
- It must be extensible, to allow for expression of new types of formulae.
- It need have only local context and semantics, i.e., the syntax need not support universal denotation, as in, e.g., the semantic web.

The TSTP syntax is defined in BNF, and is available from the TSTP web site [SutRL]. The four top level building blocks of TSTP files are *annotated formulae*, *non-formula* parts, *include* directives, and *comments*. The TSTP syntax does not provide a structure to store information about the ATP tool and computing environment used to create the formulae and other data. Such information is expected to be stored as comments, as in done in the TSTP solution library - see Section 4.1.

An annotated TSTP formula has the following structure (where items in <> brackets are placeholders for specific values, and items in [] brackets are optional):

```
<language>(<name>,<type>,
    <formula>[,
    <source>[,
    <useful info>]]).
```

An example of a FOF input formula in TSTP syntax is:

```
fof(subclass_defn,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)]).
```

An example of a CNF output formula in TSTP syntax is:

```
cnf(140,derived,
    ( equal_sets(a,aUa)
    | member(member_of(a,aUa),a) ),
    inference(unit_del,[status(thm)],
              [inference(hyper,[status(thm)],[5,16]),11]),
    [iquote('hyper,5,16.1,unit_del,11')]).
```

The <language> field names the language of the <formula>. Initially full first order form (the fof <language>) and clause normal form (the cnf <language>) have been defined. New types of formulae are possible by specifying the new <language> name and defining the <formula> syntax (it is not intended that users should extend the syntax,

but rather to provide a mechanism for the syntax designers to do so). The `<type>` indicates the role of input formulae, with values such as `axiom`, `definition`, `assumption`, `conjecture`. For derived formulae a `-derived` tag is appended to these values.

The `<formula>` syntax for both `fof` and `cnf` is the FOF syntax of the TPTP, with CNF formulae expressed as FOF disjunctions with the universal quantifiers omitted. The TPTP FOF syntax was very carefully designed, following a survey of notation used for first order logic. It has features that make it easy for humans to read and write, e.g., it uses only characters available on a standard keyboard, and uses short notations for connectives, e.g., `!` rather than a word such as `forall` for universal quantification. It is easy for programs to parse, and a parser is easily constructed using a parser generator (such as `bison`) that accepts the language definition in BNF. The syntax for atoms is that of Prolog: atoms and terms are written in prefix notation, predicates and functors start with lower case letters, and variables start with upper case letters. The binary connectives are `&`, `|`, `=>`, `<=`, `<=>`, `~&`, and `~|`, for conjunction, disjunction, implication, reverse implication, equivalence, negated conjunction, and negated disjunction respectively. The only unary connective is `~` for negation. The universal and existential quantifiers are `!` and `?` respectively, with the quantified variables following in `[]` brackets. Negation has higher precedence than quantification, which in turn has higher precedence than the binary connectives. No precedence is specified between the binary connectives, but all are defined to be left associative.

The `<source>` contains informations about either an external source, such as a file or human creator, or a derivation, for formulae derived from other formula. Derivations of formulae are described in `inference`/3 terms, of the form:

```
inference(<rule name>,<useful info>,
    [<parent info>,<parent info>, ...])
```

Each `<parent info>` is either the name of another annotated formula, another inference term, or a `theory`/1 term. A theory term is used when axioms of some theory are encoded into the rule that inferred the formula, e.g., equality axioms are encoded into paramodulation, and a term such as `theory(equality)` may be used. In the future a selection of defined `<rule name>`s will be explicitly supported in the TSTP syntax, and the `<parent info>` will be annotated sufficiently for deterministic reproduction of the inference steps.

All `<useful info>` fields are lists of terms, and are used for various annotations. In inference terms, one use of the `<useful info>` is to capture the status of the formula, as described in Section 3. The `<useful info>` at the end of an annotated formula can be used to record task specific information, e.g., in the Axiom Selector tool described in Section 4.3 it is used to record the relevance of each axiom to the conjecture

The non-formula part of the syntax provides an abstract definition of how non-logical information, e.g., hints for ATP systems, must be written. The syntax does not specify any concrete forms for non-logical information, which is system specific and beyond the scope of a general purpose syntax. However, the abstract syntax ensures that parsing any non-logical information will be possible using the same tokenizer and abstract parser as used for other parts of annotated formulae.

The include directives are analogous to a C compiler's `#include` directives. An include directive may include an entire file at that point, or may specify the `<name>`s of the formulae in the file that are to be included, thus providing a more fine grained include mechanism. TSTP comments extend from the `%` character to the end of the line. There is no block comment facility.

Parsing tools written in C are available for the TSTP syntax, and the `tptp2X` utility distributed with the TPTP is compatible with the TSTP syntax.

3 The TSTP Status Hierarchy

The output from current ATP systems varies widely in quantity, quality, and meaning. At the low end of the scale, systems that search for a refutation of a set of clauses may output only an assurance that a refutation exists (the wonderful "yes" output). At the high end of the scale a system may output a natural deduction proof of a problem expressed in FOF, e.g., [Mei00]. In some cases the output is misleading, e.g., when a CNF based system claims that a FOF input problem is "unsatisfiable" it typically means that the negated CNF of the problem is unsatisfiable.

The hierarchy of status values described below provides a precise and reasonably fine grained set of status values that can be used to specify the logical status of an ATP problem, as may be established by an ATP system. The hierarchy was based on initial work [AKR00], done to establish communication protocols for systems on the MathWeb Software Bus [ZK02]. The hierarchy is shown in Figure 1.

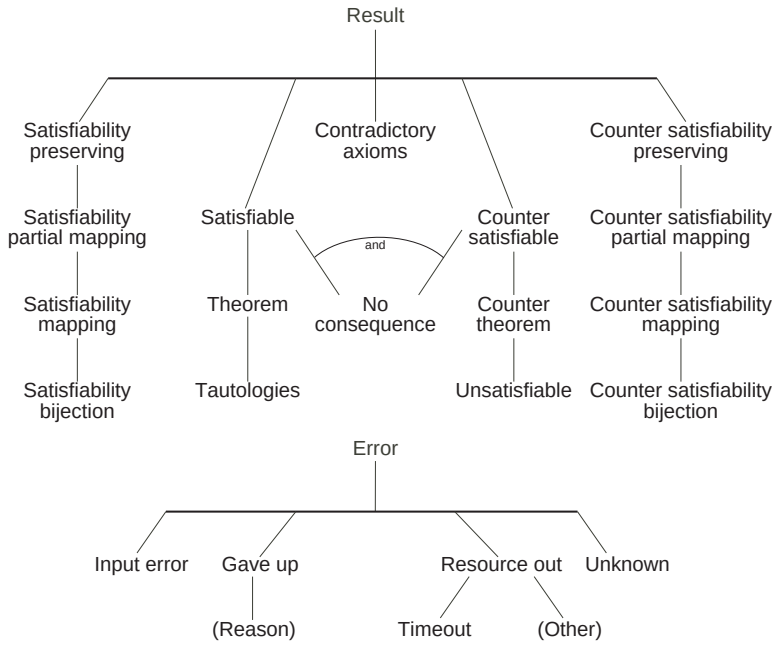

Figure 1: Status Hierarchy for ATP Systems' Outputs

The hierarchy assumes that the input F to the ATP system is of the form $Ax \Rightarrow C$, where Ax is a set of formulae, C is a single formula, Ax and C have no free variables, and $\Rightarrow$ is the standard first order implication. An empty Ax, i.e., F is a monolithic formula (a particular example is a set of clauses), is the same as Ax being $\{true\}$. An empty C, e.g., when testing the satisfiability of a set of axioms, is the same as C being $true$. By showing that F is valid, an ATP system shows that the conjecture C is a theorem (a logical consequence) of the axioms Ax, i.e., $Ax \models C$, where $\models$ is the standard first order entailment indicating that every model of Ax is a model of C. If F is not valid there are several other possible relationships between

Ax and C, as shown in the hierarchy and enumerated below. Associated with each possible status are possible outputs from an ATP system.

1. Tautologies
   Every interpretation is a model of Ax and a model of C
   - Shows: F is valid; $\sim F$ is unsatisfiable; C is a tautology
   - Outputs: Assurance; Proof of F; Refutation of $\sim F$

2. Theorem
   Every model of Ax (and there are some) is a model of C, but not case Tautologies
   - Shows: F is valid; C is a theorem of Ax
   - Outputs: Assurance; Proof of C from Ax; Refutation of $Ax \cup \{\sim C\}$; Refutation of the clause normal form of $Ax \cup \{\sim C\}$

3. Satisfiable
   Some models of Ax (and there are some) are models of C
   - Shows: F is satisfiable; $\sim F$ is not valid; C is not a theorem of Ax
   - Outputs: Assurance; Model; Saturation

4. SatisfiabilityBijection
   There is a bijection between the models of Ax (and there are some) and models of C
   - Examples: Skolemization; Pseudo-splitting [RV01]
   - Shows: Nothing about F
   - Outputs: Assurance

5. SatisfiabilityMapping
   There is a mapping from the models of Ax (and there are some) to models of C
   - Shows: Nothing about F
   - Outputs: Assurance

6. SatisfiabilityPartialMapping
   There is a partial mapping from the models of Ax (and there are some) to models of C
   - Example: `Ax = {p | q}, C = p & r`
   - Shows: Nothing about F
   - Outputs: Assurance; Pairs of models; Pairs of saturations

7. SatisfiabilityPreserving
   If there exists a model of Ax then there exists a model of C
   - Shows: Nothing about F
   - Outputs: Assurance

8. ContradictoryAxioms
   There are no models of Ax
   - Shows: F is valid; Anything is a theorem of Ax
   - Outputs: Assurance; Refutation of Ax; Refutation of the clause normal form of Ax

9. NoConsequence
   Some models of Ax (and there are some) are models of C, and some are models of $\sim C$.
   - Shows: F is not valid; F is satisfiable; $\sim F$ is not valid; $\sim F$ is satisfiable; C is not a theorem of Ax
   - Outputs: Assurance; Pair of models; Pair of saturations

10. CounterSatisfiabilityPreserving
    If there exists a model of Ax then there exists a model of $\sim C$
    - Shows: Nothing about F
    - Outputs: Assurance

11. CounterSatisfiabilityPartialMapping
    There is a partial mapping from the models of Ax (and there are some) to models of $\sim C$
    - Shows: Nothing about F
    - Outputs: Assurance; Pairs of models

12. CounterSatisfiabilityMapping
    There is a mapping from the models of Ax (and there are some) to models of $\sim C$
    - Shows: Nothing about F
    - Outputs: Assurance

13. CounterSatisfiabilityBijection
    There is a bijection between the models of Ax (and there are some) and models of $\sim C$
    - Shows: Nothing about F
    - Outputs: Assurance

14. CounterSatisfiable
    Some models of Ax (and there are some) are models of $\sim C$
    - Shows: F is not valid; $\sim F$ is satisfiable; C is not a theorem of Ax
    - Outputs: Assurance; Model; Saturation

15. CounterTheorem
    Every model of Ax (and there are some) is a model of $\sim C$, but not Unsatisfiable
    - Shows: F is not valid; $\sim F$ is valid; $\sim C$ is a theorem of Ax; C cannot be made into a theorem by extending Ax;
    - Outputs: Assurance; Proof of $\sim C$ from Ax; Refutation of $Ax \cup \{C\}$; Refutation of the clause normal form of $Ax \cup \{C\}$

16. Unsatisfiable
    Every interpretation is a model of Ax and a model of $\sim C$
    - Shows: F is unsatisfiable; $\sim F$ is valid; $\sim C$ is a tautology
    - Outputs: Assurance; Refutation of F; Proof of $\sim F$

4 Applications

The TSTP data-exchange formats have been adopted in several automated reasoning projects. These range from core ATP system development to multi-agent systems and theorem proving services on the semantic web. Some of these applications are described here, illustrating the benefits of adopting the TSTP formats.

4.1 Problem and Solution Libraries

The first use of the TSTP syntax has been in the TSTP solution library. The TSTP is a library of solutions to test problems for ATP systems, in particular, solutions to TPTP problems. The TSTP is built by running all contemporary ATP systems on all TPTP problems, and translating their system specific output to the TSTP syntax and a TSTP status value.[1] Each TSTP file contains: information about the problem; information about the system, it's input format, and the command line options used; information about the computing environment used to produce the output; the result status produced by the system, with the CPU time taken; the type of output produced by the system, with the CPU time taken; the original output from the ATP system; the TSTP format of the derivation output; statistical information about successful derivations; and access to a graphical view of the derivation. The TSTP is continuously updated as new TPTP versions are released and new (versions of) ATP systems become available. The TSTP is available online at [SutRL].

The MPTP project [Urb03], which will make all the theorems of the Mizar project [Rud92] available as first order ATP problems, will also use the TSTP formats. The problems extracted from the Mizar library include more than just the logical formulae, e.g., they include sort information. This type of information may be encoded as non-logical information, as abstractly supported by the TSTP language.

4.2 Developer Support

Common formats for input and output make the life of a system developer a lot easier. Implementing a high performance stand-alone reasoning system already is a daunting task. However, in addition to the main reasoning engine, there are a multitude of useful tools that help with analyzing the system behavior, verifying proofs, etc. In the case of the equational theorem prover E, in addition to the main prover the distribution contains:

- A problem analysis tool that determines certain numeric and discrete features of proof tasks.
- A proof analysis program that determines which clauses in the proof are the hardest to select for conventional heuristics.
- A wrapper program combining the inference engine and the analysis tool.
- An example generator that analyses proof searches, determines good and bad search decisions (represented by clauses), and stores them in a knowledge base that can be used to guide future searches [Sch01].

[1] The development of translator software is ongoing, and it is hoped it will become unnecessary as ATP system developers adopt the TSTP syntax as their native output format, as has already happened with E.

- A lemma detection tool that tries to find *interesting* facts, based on syntactic criteria of clauses and the structure of the search tree.
- A proof checker that verifies the proof, using one of a selection of other provers to establish the validity of each inference.
- A proof presentation program that extracts a proof from a full listing of all inferences.

In principle, much of this infrastructure can be reused by other ATP systems. However, the development of most of these tools predates the TSTP formats, and they were developed using various ad-hoc formalisms. The original native input syntax of E is E-LOP, which is a variant of the LOP syntax developed for SETHEO [MIL+97]. E-LOP is used for input and output of plain clauses. As an alternative, the TPTP syntax was added for both input and output. The original native output syntax of E is PCL2 language, an extension of PCL to full clausal logic. The exit status of E was denoted by somewhat ambiguous output strings, e.g. "`# Proof found!`" to denote unsatisfiability of the input clause set. All of these formats are specialized towards E, are only partially documented, and hence are unlikely to be used by other developers. While the richness of the tool set is somewhat unusual, the described state of I/O is rather typical - in fact, many existing systems do not even support the de-facto standard of TPTP, but only proprietary syntaxes.

The more generally useful parts of the E distribution are now migrating to the TSTP data-exchange formats. In particular, E version 0.81 can read TSTP input, can produce both raw and extracted proof protocols in TSTP syntax, and supports the relevant parts of the TSTP status hierarchy. As a result, much of the infrastructure of E can now be reused by other ATP systems that adopt the TSTP formats (and conversely, other interesting tools based on the TSTP formats can be used with E). This process has already started. For example, a project at the Charles University aims to automatically learn some field of mathematics by solving problems [Pud03]. When a proof is found by an ATP system, the proof is analyzed so that useful lemmas or theorems can be extracted, remembered, and used again. The project has adopted the TSTP abstract syntax because of its linkages to different ATP systems and related developer tools. The project also provides translation of the TSTP syntax to and from XML and Java object trees.

4.3 Automated Reasoning Tools

The ART (Automated Reasoning Tools) projects at the University of Miami are making extensive use of the TSTP syntax, in order to communicate data between the various components. One focus of the ART projects is to solve hard problems through selective use of axioms and lemmas. Figure 2 shows the components of the ART projects relevant to this goal. All tools and modules communicate in TSTP syntax.

The Lemma Creator aims to discover interesting lemmas and theorems by examining the logical consequences of a set of axioms, as generated by an ATP system [SGC03]. The lemma creator currently uses the E prover [Sch02] to generate logical consequences because it can output formulae in TSTP syntax. The RunTime and Ranker modules filter out boring logical consequences, leaving only interesting ones as the final output.

The Axiom Selector and Lemma Builder tools both use the Relevance module and the SystemOnTPTP module. The Relevance module provides a heuristic measure of the relevance of a formulae to a given conjecture. SystemOnTPTP [Sut00] controls the execution of

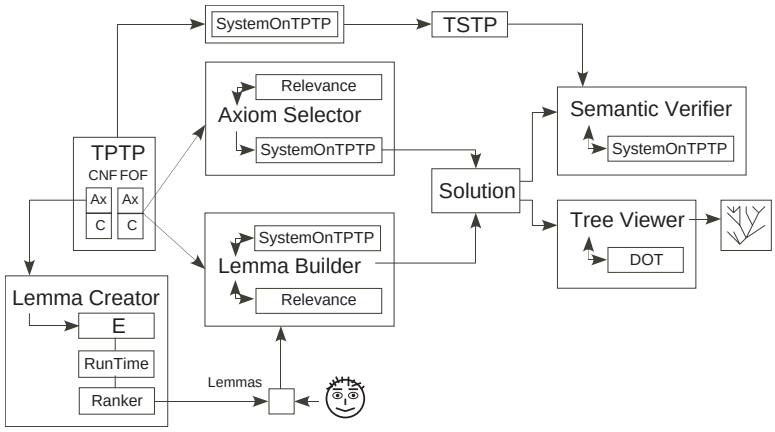

Figure 2: ART Projects for Hard Problems

a named ATP system. SystemOnTPTP accepts a problem in TSTP format, uses the tptp2X utility to convert the problem to the format of the ATP system, runs the ATP system, and uses the X2tstp utility to convert the system's output to TSTP format.

The Axiom Selector uses the Relevance module to measure the relevance of the axioms to the conjecture in a problem, and uses these measures to select subsets of the axioms to form *axiom reduced* problems that are submited to SystemOnTPTP. This is a more intelligent version of the RedAx system [SD03]. The Lemma Builder measures the relevance of lemmas that may assist in finding a proof of a conjecture, and incrementally builds a proof by proving the conjecture and the lemmas from the axioms and preceding lemmas. The component subproofs are done using SystemOnTPTP.

Given a proof or refutation in TSTP format, the Semantic Verifier does semantic and structural verification of the solution, and the Tree Viewer generates an annotated GIF image of the solution tree. As data is passed between the various ATP tools, extra information may be added to the `<useful info>` field of each annotated formula, e.g., the relevance module adds a `relevance`/1 term.

4.4 Reasoning Agents and Semantic Web Services

A framework for distributed computation and deduction agents is currently being developed by the second author [Zim03]. A precondition for multi-agent communication is that all agents must agree on a common ontology. Such an ontology for deduction systems, which includes the status hierarchy of Section 3, is being developed. The TSTP formats are being used to encode first order problems and the results of ATP systems.

One of the applications of the agent framework is the dynamic combination of ATP tools. For example, given a conjecture in TSTP FOF format, one possible query is to find a Natural Deduction (ND) proof for the conjecture. The brokering mechanism of the framework tries to come up with a suitable combination of ATP tools to answer the query. Figure 3 shows a combination of a specialized clausifying service, which produces the equivalent TSTP CNF problem, with the theorem prover Otter and the proof transformation tool Tramp [Mei00].

Such a combination of services heavily relies on a common syntax for first order ATP problems and proofs. However, tools like Tramp also rely on an agreed calculus, i.e., a lim-

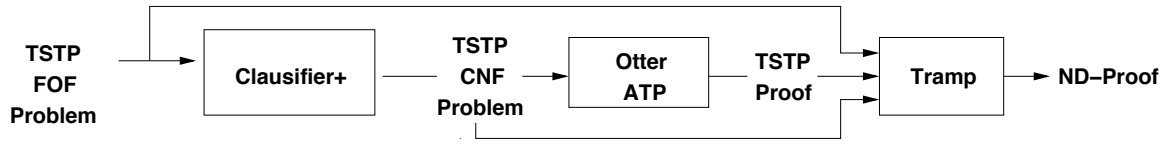

Figure 3: A clausifier, the prover Otter, and the proof transformation tool Tramp, communicating in TSTP format

ited set of inference rules occurring in a TSTP proof. Therefore the definition of a standard TSTP proof that allows only the system-independent rules of the resolution calculus, such as binary resolution, paramodulation, hyperresolution, and factoring, is currently being developed. Standard TSTP proofs would also allow for independent proof checkers.

The idea of globally accepted ontologies is also required for the Semantic Web [BLHL01] initiative, in which web resources, such as web pages and web services, are annotated with semantic markup. Together with the research projects MONET [ConRL] and MathBroker [SCRL], descriptions of web services for computation and deduction systems are being specified. The underlying ontologies will incorporate the TSTP formats to describe ATP services.

5 Conclusion

Two data-exchange formats for ATP tools have been presented. The ability of ATP tools to interface directly with other components in a larger system has an important influence on usability and uptake. The TSTP formats facilitate direct and correct communication between ATP tools.

The formats presented in this paper are finding immediate application. A reason why the formats can be put to use immediately is that they are highly pragmatic. They are sufficient to be immediately useful, and have not become embroiled in any attempt to encompass highly abstract and global ambitions. As the TSTP formats are used, their strengths and any weaknesses will be revealed, possibly leading to improvements. As with the TPTP syntax, which is now commonly used in the ATP community, these formats are most likely to succeed if there is sufficient successful real usage.

In order to assist the ATP community to use the TSTP syntax for input to ATP systems, TPTP v3.0.0 will be distributed in both the TPTP syntax and in the TSTP syntax. To encourage adoption of the TSTP syntax and the output statuses, the TSTP library will be distributed in the TSTP syntax, and a suite of TSTP postprocessing tools (which use the TSTP syntax and status values) will be made freely available. Future versions of the MathWeb Software Bus will provide full support for the TSTP language.

Future work includes designing formats for the output from model generation programs such as MACE [McC01] and FINDER [Sla94], and building a hierarchy of possible outputs (outputs such as “assurance”, “refutation”, “proof”) from ATP systems.

References

[ACLRL] ACL2. http://www.cs.utexas.edu/users/moore/acl2/, URL.

[AKR00] A. Armando, M. Kohlhase, and S. Ranise. Communication Protocols for Mathematical Services based on KQML and OMRS. In M. Kerber and M. Kohlhase, editors, *Proceedings of the Calculemus Symposium 2000*, 2000.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[CC99] O. Caprotti and D. Carlisle. OpenMath and MathML: Semantic Mark Up for Mathematics. *ACM Crossroads*, 6(2), 1999.

[CHM02] K. Claessen, R. Hähnle, and J. Mårtensson. Verification of Hardware Systems with First-Order Logic. In G. Sutcliffe, J. Pelletier, and C. Suttner, editors, *Proceedings of the CADE-18 Workshop - Problem and Problem Sets for ATP*, number 02/10 in Department of Computer Science, University of Copenhagen, Technical Report, 2002.

[CL-RL] Common Logic Standard. http://cl.tamu.edu/, URL.

[ConRL] The MONET Consortium. The MONET Project. http://monet.nag.co.uk/cocoon/monet, URL.

[CoqRL] The Coq Proof Assistant. http://pauillac.inria.fr/coq, URL.

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

[DKS97] J. Denzinger, M. Kronenburg, and S. Schulz. DISCOUNT: A Distributed and Learning Equational Prover. *Journal of Automated Reasoning*, 18(2):189–198, 1997.

[DS94] J. Denzinger and S. Schulz. Recording, Analyzing and Presenting Distributed Deduction Processes. In H. Hong, editor, *Proceedings of the 1st International Symposium on Parallel Symbolic Computation*, number 5 in Lecture Notes Series on Computing, pages 114–123. World Scientific Publishing, 1994.

[DS96] J. Denzinger and S. Schulz. Recording and Analysing Knowledge-Based Distributed Deduction Processes. *Journal of Symbolic Computation*, 21:523–541, 1996.

[Fie01] A. Fiedler. P.rex: An Interactive Proof Explainer. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 416–420. Springer-Verlag, 2001.

[GF92] M.R. Genesereth and R.E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, 1992.

[HKW96] R. Hähnle, M. Kerber, and C. Weidenbach. Common Syntax of the DFG-Schwerpunktprogramm Deduction. Technical Report TR 10/96, Fakultät für Informatik, Universät Karlsruhe, Karlsruhe, Germany, 1996.

[HOLRL] Automated Reasoning Group HOL Page. http://www.cl.cam.ac.uk/Research/HVG/HOL/, URL.

[HS01] I. Horrocks and U. Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* , pages 199–204. Morgan Kaufmann, 2001.

[Koh00] M. Kohlhase. OMDOC: Towards an Internet Standard for the Administration, Distribution, and Teaching of Mathematical Knowledge. In J.A. Campbell and E. Roanes-Lozano, editors, *Proceedings of the Artificial Intelligence and Symbolic Computation Conference, 2000*, number 1930 in Lecture Notes in Computer Science, pages 32–52. Springer-Verlag, 2000.

[LH02] B. Loechner and T. Hillenbrand. A Phytography of Waldmeister. *Journal of AI Communications*, 15(2/3):127–133, 2002.

[McC94] W.W. McCune. Otter 3.0 Reference Manual and Guide. Technical Report ANL-94/6, Argonne National Laboratory, Argonne, USA, 1994.

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

[McC01] W.W. McCune. MACE 2.0 Reference Manual and Guide. Technical Report ANL/MCS-TM-249, Argonne National Laboratory, Argonne, USA, 2001.

[Mei00] A. Meier. System Description: TRAMP - Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 460–464. Springer-Verlag, 2000.

[MIL+97] M. Moser, O. Ibens, R. Letz, J. Steinbach, C. Goller, J. Schumann, and K. Mayr. SETHEO and E-SETHEO: The CADE-13 Systems. *Journal of Automated Reasoning*, 18(2):237–246, 1997.

[MRS01] D. McMath, M. Rosenfeld, and R. Sommer. A Computer Environment for Writing Ordinary Mathematical Proofs. In R. Nieuwenhuis and A. Voronkov, editors, *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 2250 in Lecture Notes in Artificial Intelligence, pages 507–516. Springer-Verlag, 2001.

[Pud03] P. Pudlak. Email to G. Sutcliffe. 2003.

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

[RV01] A. Riazanov and A. Voronkov. Splitting without Backtracking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* , pages 611–617. Morgan Kaufmann, 2001.

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

[Sch01] S. Schulz. Learning Search Control Knowledge for Equational Theorem Proving. In F. Baader, G. Brewka, and T. Eiter, editors, *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence*, number 2174 in Lecture Notes in Artificial Intelligence, pages 320–334. Springer-Verlag, 2001.

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

[SCRL] W. Schreiner and O. Caprotti. The MathBroker Project. http://poseidon.risc.uni-linz.ac.at:8080, URL.

[SD03] G. Sutcliffe and A. Dvorsky. Proving Harder Theorems by Axiom Reduction. In I. Russell and S. Haller, editors, *Proceedings of the 16th Florida Artificial Intelligence Research Symposium*, pages 108–112. AAAI Press, 2003.

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

[SGC03] G. Sutcliffe, Y. Gao, and S. Colton. A Grand Challenge of Theorem Discovery. In J. Gow, T. Walsh, S. Colton, and V. Sorge, editors, *Proceedings of the Workshop on Challenges and Novel Applications for Automated Reasoning, 19th International Conference on Automated Reasoning*, pages 1–11, 2003.

[Sla94] J.K. Slaney. FINDER: Finite Domain Enumerator, System Description. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, number 814 in Lecture Notes in Artificial Intelligence, pages 798–801. Springer-Verlag, 1994.

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

[Sut00] G. Sutcliffe. SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 406–410. Springer-Verlag, 2000.

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

[Tam97] T. Tammet. Gandalf. *Journal of Automated Reasoning*, 18(2):199–204, 1997.

[Urb03] J. Urban. Translating Mizar for First Order Theorem Provers. In A. Asperti, B. Buchberger, and J.H. Davenport, editors, *Proceedings of the 2nd International Conference on Mathematical Knowledge Management*, number 2594 in Lecture Notes in Computer Science, pages 203–215. Springer-Verlag, 2003.

[WBH+02] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS Version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 275–279. Springer-Verlag, 2002.

[WP99] L. Wos and G. Pieper. *A Fascinating Country in the World of Computing: Your Guide to Automated Reasoning*. World Scientific, 1999.

[WSF02] M. Whalen, J. Schumann, and B. Fischer. Synthesizing Certified Code. In L.H. Eriksson and P Lindsay, editors, *Proceedings of the International Symposium of Formal Methods Europe (FME 2002: Formal Methods - Getting IT Right)*, number 2391 in Lecture Notes in Computer Science, pages 431–450. Springer-Verlag, 2002.

[Zim03] J. Zimmer. A New Framework for Reasoning Agents. In V. Sorge, S. Colton, M. Fisher, and J. Gow, editors, *Proceedings of the Workshop on Agents and Automated Reasoning, 18th International Joint Conference on Artificial Intelligence*, pages 58–64, 2003.

[ZK02] J. Zimmer and M. Kohlhase. System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 139–143. Springer-Verlag, 2002.