Tipi: A TPTP-based theory development environment emphasizing proof dependencies

Jesse Alama*
Center for Artificial Intelligence
New University of Lisbon
Portugal
j.alama@fct.unl.pt, http://centria.di.fct.unl.pt/~alama/

Abstract

In some theory development tasks, a problem is satisfactorily solved once it is shown that a theorem (conjecture) is derivable from the background theory (premises). Depending on one's motivations, the details of the derivation of the conjecture from the premises may or may not be important. In some contexts, though, one wants more from theory development than simply derivability of the target theorems from the background theory. One may want to know which premises of the background theory were used in the course of a proof output by an automated theorem prover (when a proof is available), whether they are all, in suitable senses, necessary (and why), whether alternative proofs can be found, and so forth. The problem, then, is to support proof analysis in theory development; the tool described in this paper, Tipi, aims to provide precisely that.

1 Introduction

A characteristic feature of theorem proving problems arising in theory development is that we often do not know which premises of our background theory are needed for a proof until we find one. If we are working in a stable background theory in which the axioms are fixed, we naturally include all premises of the background theory because it a safe estimate (perhaps an overestimate) of what is needed to solve the problem. We may add lemmas on top of the background theory to help a theorem prover find a solution or to make our theory more comprehensible. Since computer-assisted theorem proving is beset on all sides by intractability, any path through a formal theory development task is constantly threatened by limitations both practical (time, memory, patience, willpower) and theoretical (undecidability of first-order validity). Finding even one solution (proof, model, etc.) is often no small feat, so declaring victory once the first solution is found is thus quite understandable and may be all that is wanted.

In some theory development tasks, though, we want to learn more about our problem beyond its solvability. This paper announces Tipi, a tool that helps us to go beyond mere solvability of a reasoning problem by providing support for answer such questions as:

- What premises of the problem were used in the solution?
- Do other automated reasoning systems derive the conclusion from the same premises?
- Are my premises consistent? Do they admit unintended models?

*Supported by the ESF research project *Dialogical Foundations of Semantics* within the ESF Eurocores program *LogICCC* (funded by the Portuguese Science Foundation, FCT LogICCC/0001/2007). Research for this paper was partially done while a visiting fellow at the Isaac Newton Institute for the Mathematical Sciences in the programme 'Semantics & Syntax'. The author wishes to thank Ed Zalta and Paul Oppenheimer for stimulating the development of Tipi and for being guinea pigs for it.

- What premises are truly needed for the conclusion? Can we find multiple sets of such premises? Is there a a "minimal" theory that derives the conclusion?

- Are my axioms independent of one another?

Let us loosely call the investigation of these and related questions *proof analysis*.

Tipi is useful for theory exploration both in the context of discovery and in the convex of justification. In the context of discovery, one crafts lemmas, adds or deletes axioms, changes existing axioms, modifies the problem statement, etc., with the aim of eventually showing that the theory adequate for one's purposes (it derives a certain conjecture, is satisfiable, is unsatisfiable, etc.). In the context of discovery, the set of one's axioms is in flux, and one needs tools to help ensure that the development is not veering too far off course into the unexpected countersatisfiability, admitting "nonsense" models, being inconsistent, etc. In the context of justification, after the initial work is done and a solution is found, one wants to know more about the relationship between the theory and the conjecture than simply that the latter is derivable from the former. What is the proof like? Are there other proofs? Tipi is designed to facilitate answering questions such as these.

The theorem provers and model finders that make up Tipi include E, Vampire, Prover9, Mace4, and Paradox. The system is extensible; adding suport for new automated reasonig systems is striaghtforward because we rely only on the SZS ontology to make judgments about theorem proving problems.

Tipi uses a variety of automated reasoning technology to carry out its analysis. It uses theorem provers and model finders and is based on the TPTP syntax for expressing reasoning problems [6] and the SZS problem status ontology [5] thereby can flexibly use of a variety of automated reasoning tools that support this syntax.

Going beyond solvability and demanding more of our solutions is obviously not a new idea. Our interests complement those of Wos and collaborators, who are also often interested not simply in derivability, but in finding proofs that have certain valuable properties, such as being optimal in various senses; see [10, 9]. Tipi emphasizes proof analysis at the level of sets of premises, whereas one could be interested in more fine grained information such as the number of symbols employed in a proof, whether short proofs are available, whether a theory is axiomatized by a single formula, etc. Such analysis tends to involve rather expert knowledge of particular problems and low-level tweaking of proof procedures. Tipi uses automated reasoning technology essentially always in "automatic mode".

The philosophical background of Tipi is a classic problem in the philosophy of logic known as the proof identity problem:

> *When are two proofs the same?*

Standard approaches to the proof identity problem work with natural deduction derivations or category theory. One well-known proposal is to identify "proof" with a natural deduction derivation, define a class of conversion operations on natural deduction derivations, and declare that two proofs are the same if one can be converted to the other. See [3] for a discussion of this approach to the proof identity problem. The inspiration for Tipi is to take on the proof identity problem with the assistance of modern automated reasoning tools. From this perspective, the TPTP library [6] can be seen as a useful resource on which to carry out experiments about "practical" proof identity. TPTP problems typically don't contain proofs in the usual sense of the term, but they do contain hints of proofs in the sense that they specify axioms and perhaps some intermediate lemmas.

One does not need to share the philosophical background (or even care about it) to start using Tipi, which in any case was designed to facilitate TPTP-based theory development.

Terminology In the following we sometimes equivocate on the term *theory*, understanding it sometimes in its mathematical sense as a set of formulas closed under logical consequence (and so is always infinite and has infinitely many axiomatizations), and sometimes in its practice sense, represented as a TPTP problem, which always has finitely many axioms. "TPTP theory" simply means an arbitrary (first-order) TPTP problem. Of course, from a TPTP theory T we can obtain a theory in the mathematical sense of the term by simply reading the formulas of T as logical formulas and closing T under logical consequence. From an arbitrary first-order theory in the mathematical sense of the term obviously one cannot extract a unique finite axiomatization and, worse, many theories of interest are not even finitely axiomatizable. Still, we may at times, for precision, need to understand "theory" in its mathematical sense, even though of course we shall always work with finite TPTP theories (problems).

Convention Some TPTP problems do not have a conjecture formula. Indeed, some TPTP problems are not theorem proving problems per se but are better understood as model finding problems (e.g., the intended SZS status is `Satisfiable`). For expositional convenience we shall restrict ourselves to TPTP problems whose intended interpretation is that a set of premises entails a single conclusion.

The structure of this paper is as follows. Section 2 describes some simple tools provided by Tipi to facilitate theory development. Section 3 discusses the problem of determining which premises are needed. Section 4 discusses two algorithms, one syntactic and the other semantic, for determining needed needed premises. Section 5 concentrates on independent sets of axioms. Section 6 discusses some simple model analysis tools provided by Tipi. Section 7 gives a sense of the experience so far with using Tipi on real-world proof analysis tasks. Section 8 says where can obtain Tipi and briefly discusses its implementation. Section 9 concludes and suggests further directions for proof analysis and dependencies.

2 Syntax analysis

When designing TPTP theories, one needs to be careful about the precise language (signature) that one employs. An all-too-familiar problem is typos: one intends to write `connected_to` but writes `conected_to` instead. One quick check that can help catch this kind of error is to look for unique occurrences of constants, functions, or predicates. A unique occurrence of a relation or function symbol is a sign (though by no means a necessary or sufficient condition) that the theory is likely to be trivially inadequate to the intended reasoning task because it will fail to be (un)satisfiable, or fail to derive the conjecture. Detecting such hapax legomena early in the theory development process can prevent wasted time "debugging" TPTP theories that appear to be adequate but which are actually flawed.

3 Needed premises

Once it is known that a conjecture c is derivable from a background theory T, one may want to know about the class of proofs that witness this derivability relation. Depending on which automated theorem prover (ATP) is used, there may not even be a derivation of c from T, but only the judgment that c is a theorem of T. If one does have a derivation (e.g., a resolution proof) d, one can push the investigation further:

- Which premises of T occur in d?

- Are all the premises occurring in d needed?

Various notions of "needed" are available. For lack of space we cannot give a complete survey of this intresting concept; see [1] for a more thorough discussion of the notion of "dependency" in the context of interactive theorem provers. One can distinguish whether a formula is needed for a *derivation*, or for a *conclusion*. In a Hilbert style calculus, the sequence d of formulas

$$\langle C, A, A \to B, B \rangle$$

is a derivation of D from the premises $X = \{C, A, A \to B\}$. All axioms of X do appear in d, so it is reasonable to assert that all of X is needed for d. But are all premises needed for the conclusion B of d? The formula C is not used as the premise of any application of a rule of inference (here, modus ponens is the only rule of inference). Thus one can simply delete the first term of d and obtain a derivation d' of B from $X - \{C\}$. In a plain resolution calculus, a derivation of the empty clause from a set $\mathcal{C}$ of clauses can have unused premises in the same sense as there can be unused premises of a Hilbert-style derivation. Still, there can be "irrelevant" literals in clauses of $\mathcal{C}$ whose deletion from $\mathcal{C}$ and from a refutation d of $\mathcal{C}$ yields a more focused proof.

Intuitively, any premise that is needed for a conclusion is also needed for any derivation of the conclusion (assuming sensible notions of soundness and completeness of the calculi in which derivations are carried out). However, a premise that is needed for a derivation need not be needed for its conclusion. Clearly, multiple proofs of a conclusion are often available, employing different sets of premises.

In the ATP context, we may even find that, if we keep trimming unused premises from a theory T that derives a conjecture c until no more trimming is possible (so that every premise is needed by the ATP to derive the conjecture), there may still be proper subsets of "minimal" premises that suffice. The examples below in Section 7 illustrate this.

4 Reproving

Given an ATP A, a background theory T and a conjecture c, assume that T does derive c and is witnessed by an A-derivation d. Define

$$T_0 := \{\varphi \in T \colon \varphi \text{ occurs in } d\}$$

as the set of premises of T occurring in d. Do we need all of T to derive c? If T_0 is a proper subset of T, then the answer is evidently "no". One simple method to investigate the question of which premises of T are needed to derive c is to simply repeat the invocation of A using successively weaker subtheories of T. Given T_k and an A-derivation d_k of c from T_k, define

$$T_{k+1} := \{\varphi \in T_k \colon \varphi \text{ occurs in } d_k\}$$

We are then after the fixed point of the sequence

$$T_0 \supset T_1 \supset T_2 \supset \ldots$$

We can view this discussion as the definition of a new proof procedure:

Definition 1 (Syntactic reproving). Given a background theory T, an ATP A, a conjecture c, use A to derive c from A. If this succeeds, extract the premises of T that were used by A to derive c; call this set T'. If $T' = T$, then stop and return d. If T' is a proper subset of T, then let $T := T'$ and repeat.

We call this proof procedure "syntactic" simply because we view the task of a proof finder as a syntactic one. The name is not ideal because an ATP may use manifestly semantic methods in the course of deriving c from T. The definition of syntactic reproving requires of A only that we can compute from a successful search for a derivation, which premises were used; we do not require a derivation from A, though in practice various ATPs do in fact emit derivations and from these we simply extract used premises.

If A is a complete ATP, then we can find a fixed point, provided we have unlimited resources; the existence of a fixed point follows from the assumption that $T_0 \vdash c$ and the fact that T_0 is finite.[1] Of course, we do place restrictions on our proof searches, so we often cannot determine that a proper subset of T suffices to derive c, even if there is such a subset.

It can happen that the syntactic reprove procedure applied with an ATP A, a theory T, and a conjecture c, terminates with a subtheory T' of T even though there is a proper subset T'' of T that also suffices and, further, A can verify that T'' suffices. The syntactic reprove procedure does not guarantee that the solution it finds is truly minimal. Some other proof procedure, then, is needed.

Definition 2 (Semantic reproving). Given a background theory T and ATPs A and B, a conjecture c, use A to derive c from A. If this succeeds, extract the premises of T that were used by A to derive c; call this set T'. Define

$$T^* := \{\varphi \in T' : T' - \{\varphi\} \nvdash c\}$$

Now use A to check whether $T^* \vdash c$. If this succeeds, return T^*.

The semantic reprove procedure takes two ATPs A and B as parameters. A is used for checking derivability, whereas B is used to check underivability. This proof procedure is called "semantic" because the task of constructing T^* is carried out in Tipi using a model finder (e.g., Paradox or Mace4), which solves the problem of showing that $X \nvdash \varphi$ by producing a model of $X \cup \{\neg\varphi\}$. As with "syntactic" in "syntactic reprove", the "semantic" in "semantic reprove" is not ideal because any ATP that can decide underivability judgments would work; whether B uses syntactic or semantic methods (or a combination thereof) to arrive at its solution is immaterial. Indeed, in principle, for B a theorem prover could be used. Even though Vampire and E are typically used to determine derivability, because of the properties of their search procedures, they can be used for determining underivability, though establishing underivability is not necessarily their strong suit and often a model finder can give an answer more efficiently to the problem of whether $X \vdash \varphi$.

5 Independence

In proof analysis a natural question is whether, in a set X of axioms, there is an axiom φ that depends on the others in the sense that φ can be derived from $X - \{\varphi\}$.

Definition 3 (Independent set of formulas). A set X of formulas is **independent** if for every formula φ in X it is not the case that $X - \{\varphi\} \vdash \varphi$.

Tipi provides a proof procedure for testing whether a set of formulas is independent. The algorithm for testing this is straightforward: given a finite set $X = \{\varphi_1, \ldots, \varphi_n\}$ of axioms whose independence we need to test, test successively

1. whether there is any $\varphi = \varphi_k$ in X such that for some $j_1 \neq k$, we have $\{x_{j_1}\} \vdash \varphi$

[1] There may even be multiple A-minimal subtheories of T that derive c, but the proof procedure under discussion will find only one of them.

2. whether there is any $\varphi = \varphi_k$ in X such that for some $j_1, j_2 \neq k$ we have $\{x_{j_1}, x_{j_2}\} \vdash \varphi$,
3. etc., for increasingly larger k (the upper bound is of course $n - 1$).

On the assumption that most set of axiom that arise in practice are not independent, Tipi employs a "fail fast" heuristic: if X is not independent, then we can likely find, for some axiom φ in X, a small subset X^* suffices to prove φ.

Other algorithms for testing independence are conceivable. It could be that the naive algorithm that is immediately suggested by the definition—enumerate the axioms, checking for each one whether it is derivable from the others—may be the best approach. Experience shows this is indeed an efficient algorithm if one really does have an independent set (obviously the iterative "fail fast" algorithm sketched requires $n(n-1) = O(n^2)$ calls to an ATP for a set of axioms of size n, whereas the obvious algorithm just makes n calls). A model finder can be used to facilitate this: if $X - \{\varphi\} \cup \{\neg\varphi\}$ is satisfiable, then φ is independent of X. If one is dealing with large sets of axioms, testing independence becomes prohibitively expensive, so one could employ a randomized algorithm: randomly choose an axiom φ and a proper subset T' of T that does not contain φ and test whether T' proves φ. Tipi implements all these algorithms for checking independence.

A typical application of independence checking first invokes one of the minimization algorithms described in Section 4. If there is a proper subset T' of T that suffices to derive c, then the independence of the full theory T is probably less interesting (and in any event requires more work to determine) than the independence of the sharper set T'.

Checking independence is related to the semantic reprove algorithm described in Section 4. If we are dealing with a theory that has a conjecture formula, then the two notions are not congruent, because the property of independence holds for a set of formulas without regard to whether they are coming from a theory that has a conjecture formula. If we are dealing with a theory T without a conjecture whose intended SZS status is `Unsatisfiable`, i.e., the theory should be shown to be unsatisfiable, then an axiom φ of a theory T gets included in the the set T^* of Definition 2 of semantic reproving if $T - \varphi$ is satisfiable. T is "semantically minimal" when no proper subtheory of T is unsatisfiable, i.e., every proper subtheory of T is satisfiable. Independence and semantic minimality thus coincide in the setting of theories without a conjecture formula with intended SZS status `Unsatisfiable`.

6 Model analysis

When developing formal theories, one's axioms, lemmas, and conjecture are typically in flux. One may request the assistance of an automated reasoning system to check simply whether one's premises are consistent. One might go further an ask whether, if one is dealing with a TPTP theory that has a conjecture, there the theory is satisfiable when the conjecture is taken as simply another axiom. An "acid test" for whether one is proceeding down the right path at all is whether one's problem is countersatisfiable.

Tipi provides tools for facilitating this kind of analysis. A single command is available that can check, given a theory

- whether the theory without the conjecture has a model
- whether the axioms of the theory together with the conjecture (if present) has a model
- whether the axioms of the theory together with the negation of the conjecture has a model.

edge_ends_are_vertices	For every edge, the head and tail of the edge are vertices
in_path_properties	If P is a path from vertices v_1 and v_2, and if v is in P, then (i) v is a vertex and (ii) there is an edge e of P such that v is either the head or the tail of e.
on_path_properties	If P is a path from vertices v_1 and v_2, and if e is on P, then e is an edge and and both the head and the tail of e are in P.

Table 1: Axioms appearing in the two minimal subtheories of GRA008+1

The second consistency check is useful to verify that one's whole problem (axioms together with the conjecture, considered as just another axiom) is sensible. It can happen that the axioms of a problem have very simple models, but adding the conjecture makes the models somewhat more complicated. If a set of axioms has a finite model but we cannot determine reasonably quickly that the axioms together with the conjecture have a finite model, then we can take such results as a sign that the conjecture may not be derivable from the axioms. (Of course, it is possible that that the set of axioms is finitely satisfiable but the set containing axioms and the conjecture is finitely unsatisfiable. One can use tools such as Infinox [2] to complement Tipi in such scenarios.)

7 Experience

Tipi has so far been used successfully to analyze a variety of theories occurring in diverse TPTP theory development tasks. It has proved quite useful for theory development tasks in computational metaphysics [4], which was the initial impetus for Tipi.

To get a sense of how one can apply Tipi, we now consider several applications of Tipi to problems coming from the large TPTP library of automated reasoning problems. In these examples we use E as our theorem prover and Paradox as our model finder.

Example 1 (GRA008+1). A problem in a first-order theory[2] about graphs has 17 premises. Syntactic reprove brings this down to 12. Progressing further with semantic repoving, we find that 8 of the 12 are needed (in the sense that for each of them, their deletion, while keeping the others, leads to countersatisfiability). Moreover, none of the other 4 is individually needed (the conjecture is still derivable from the 4 theories one obtains by deleting the 4). It turns out that there are two minimal theories that suffice to derive the conjecture. The relevant axioms are in Table 1 are:

The two minima consist of the 8 needed formulas together with

- {edge_ends_are_vertices}, and
- {in_path_properties, on_path_properties}.

Example 2 (REL001+1.p). A problem about relation algebra is to show that a certain term **0** is a least element. There are 13 premises[3]. Syntactic reprove gets this down to 10 (3 are not needed). Semantic reprove shows that 3 axioms are needed, and moreover the other 7 are are separately not needed; see Table 2.

[2] See GRA001+0.ax.
[3] See REL001-0.ax.

composition_identity	$x;\mathbf{1} = x$
converse_cancellativity	$\breve{x};\overline{x;\overline{y}} \vee \bar{y} = \bar{y}$
converse_idempotence	$\breve{\breve{x}} = x$
converse_multiplicativity	$x\check{;}y = \breve{x};\breve{y}$
maddux1_join_commutativity	$x \vee y = y \vee x$
maddux2_join_associativity	$x \vee (y \vee z) = (x \vee y) \vee z$
def_top	$\top = x \vee \bar{x}$

Table 2: Axioms appearing in some minimal subtheory of REL001+1

symmetry_of_glb	$x \vee y = y \vee x$
associativity_of_glb	$x \vee (y \vee z) = (x \vee y) \vee z$
idempotence_of_glb	$x \vee x = x$

Table 3: Axioms appearing in a minimal subtheory of GRP143-1

The three known needed premises are together too weak to derive the conclusion. Of the 128 subsets of the 7 separately not needed premises, we find two distinct minimal solutions:

- The singleton set {maddux2_join_associativity}
- The 5-element set

  {composition_identity, converse_cancellation,
  converse_idempotence, converse_multiplicativity,
  maddux1_join_commutativity}

Surprisingly, postulating only maddux2_join_commutativity suffices (in the presence of the 3 axioms known to be needed). If one does not postulate this, 5 other axioms are needed (again, in the presence of the 3 axioms known to be needed). Notice that we can do away with def_top.

Example 3 (GRP143-1). A problem about lattice-ordered group theory has 15 axioms[4]. The aim is to show that $a \vee (a \vee b) = a \vee b$, where $\vee$ is the greatest lower bound operator on group elements. Syntactic and semantic reprove reduce this to 4 axioms, all of which are needed; see Table 3.

For theories with large numbers of needed axioms, applying Tipi's tools can, understandably, be quite time consuming. On the other hand, one interesting application of the semantic reprove method is that it can allow one to tackle problems that, if approached directly, seem to be quite hard. Sometimes it seems that all that's needed to go from intractability to practical solvability is a hint that some axioms might not be needed for a solution.

8 Availability

Tipi is a collection of Perl modules and scripts and XSLT stylesheets. It can be downloaded from

[4]See GRP004-0.ax and GRP004-2.ax.

https://github.com/jessealama/tipi

At present Tipi relies on the GetSymbols and TPTP4X tools, which are part of the TPTP World distribution [7]. These are used to parse TPTP theory files; a standalone parser for the TPTP language is planned, which would eliminate the dependency on these additional tools.

9 Conclusion and future work

At the moment Tipi supports a handful of theorem provers and model finders. Supporting further systems is desirable; any automated reasoning system that provides some non-trivial support for the SZS ontology could, in principle, be added.

Tipi supports, at the moment, only first-order logic, and so covers only a part of the space of all TPTP theories. There seems to be no inherent obstacle to extending Tipi to support higher-order theories as well.

More systematic investigation for alternative proofs of a theorem could be carried out using Prover9's clause weight mechanism. One could have an alternative approach to the problem of generating multiple alternative proofs to the simple approach taken by Tipi.

When working with models of a theory under development that makes true some rather unusual or unexpected formulas, it can sometimes be difficult to pinpoint the difficulty with the theory that allows it to have such unusual models. One has to infer, by looking at the raw presentation of the model, what the strange properties are. We would like to implement a smarter, more interactive approach to the task of investigating models that can help pinpoint theory problems. A related tool is McCune's clausefilter.

The problem of finding minimal subtheories sufficient to derive a conjecture, checking independence of sets of axioms, etc., clearly requires much more effort than simply deriving the conjecture. Tipi thus understandably can take a lot of time to answer some of the questions put to it. Some of this inefficiency seems unavoidable, but it is reasonable to expect that further experience with Tipi could lead to new insights into the problem of finding theory minima, determining independence, etc.

The proof procedures defined by Tipi naturally suggest extensions to the SZS ontology [5]. One can imagine SZS statuses such as

- `IndependentAxioms`: The set of axioms is independent.
- `DependentAxioms`: The set of axioms of dependent.
- `MinimalPremises`: No proper subset of the axioms suffices to derive the conjecture.
- `NonMinimalPremises`: A proper subset of the axioms suffices to derive the conjecture.
- `UniqueMinimum`: There is a unique subset S of the axioms such that S derives the conjecture and every proper subset of S fails to derive the conjecture.
- `MultipleIncomparableMinima`: There are at least two proper subsets S_1 and S_2 of the axioms suffices to derive the conjecture, with neither $S_1 \subseteq S_2$ nor $S_2 \subseteq S_1$.

Tipi itself already can be seen as supporting these (currently unavailable) SZS statuses. One could even annotate the statistics for many problems in the TPTP library by listing the number of possible solutions (minimal subtheories of the original theory) they admit, or the number of premises that are actually needed.

References

[1] Jesse Alama, Lionel Mamane, and Josef Urban. Dependencies in formal mathematics. *CoRR*, abs/1109.3687, 2011.

[2] Koen Claessen and Ann Lilliesträm. Automated inference of finite unsatisfiability. In R. A. Schmidt, editor, *CADE 2009*, volume 5633 of *Lecture Notes in Artificial Intelligence*, pages 388–403, 2009.

[3] K. Došen. Identity of proofs based on normalization and generality. *Bulletin of Symbolic Logic*, pages 477–503, 2003.

[4] B. Fitelson and E.N. Zalta. Steps toward a computational metaphysics. *Journal of Philosophical Logic*, 36(2):227–247, 2007.

[5] G. Sutcliffe. The SZS ontologies for automated reasoning software. In *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, volume 418, pages 38–49. CEUR Workshop Proceedings, 2008.

[6] G. Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[7] G. Sutcliffe. The TPTP World–Infrastructure for automated reasoning. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 1–12. Springer, 2010.

[8] A. Tarski. What is elementary geometry? *Studies in Logic and the Foundations of Mathematics*, 27:16–29, 1959.

[9] L. Wos. The flowering of automated reasoning. In D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 204–227. Springer, 2005.

[10] L. Wos and G.W. Pieper. *Automated Reasoning and the Discovery of Missing and Elegant Proofs*. Rinton Press, 2003.