Automated Generation of Machine Verifiable and Readable Proofs: A Case Study of Tarski's Geometry

Sana Stojanovic Durdevic, Julien Narboux, Predrag Janicic

Automated Generation of Machine Verifiable and Readable Proofs: A Case Study of Tarski's Geometry *

Sana Stojanović Đurđević, Julien Narboux, and Predrag Janičić

Faculty of Mathematics, University of Belgrade
Studentski trg 16, 11000 Belgrade, Serbia
ICube, UMR 7357 CNRS, University of Strasbourg
Pôle API, Bd Sébastien Brant, BP 10413, 67412 Illkirch, France
sana@matf.bg.ac.rs,narboux@unistra.fr,janicic@matf.bg.ac.rs

Abstract. The power of state-of-the-art automated and interactive theorem provers has reached the level at which a significant portion of non-trivial mathematical contents can be formalized almost fully automatically. In this paper we present our framework for the formalization of mathematical knowledge that can produce machine verifiable proofs (for different proof assistants) but also human-readable (nearly textbook-like) proofs. As a case study, we focus on one of the twentieth century classics – a book on Tarski's geometry. We tried to automatically generate such proofs for the theorems from this book using resolution theorem provers and a coherent logic theorem prover. In the first experiment, we used only theorems from the book, in the second we used additional lemmas from the existing Coq formalization of the book, and in the third we used specific dependency lists from the Coq formalization for each theorem. The results show that 37% of the theorems from the book can be automatically proven (with readable and machine verifiable proofs generated) without any guidance, and with additional lemmas this percentage rises to 42%. These results give hope that the described framework and other forms of automation can significantly aid mathematicians in developing formal and informal mathematical knowledge.

Keywords: Automated theorem proving, Interactive theorem proving, Tarski's geometry

1 Introduction

For decades, or even centuries, mathematicians have dreamed of an automated system, usable in everyday practice, that can be used as help in proving math-

* The research presented in this paper was partly funded by the Serbian-French Technology Co-Operation grant EGIDE/"Pavle Savić" 680-00-132/2012-09/12. The first and the third author are partly supported by the grant ON174021 of the Ministry of Science of Serbia.
The final publication is available at Springer via:
http://dx.doi.org/10.1007/s10472-014-9443-5.

ematical theorems. It would be beneficial to have computer programs not only capable of proving open or very complex mathematical conjectures, but also the technical, yet non-trivial, lemmas and theorems that appear in the development of large mathematical theories. Over the past few decades, significant progress has been made in both interactive and automated theorem proving and they are nowadays used in many areas of mathematics and computer science. In this paper we explore and demonstrate the benefits of integrating interactive and automated theorem proving in the formalization of mathematical knowledge (e.g., mathematical heritage, textbooks, etc.). For that task, we use our framework that proves given theorems and generates both machine-verifiable and human-readable, nearly textbook-like proofs. The framework uses a fragment of first order logic, known as coherent logic, as an underlying logic. It combines several approaches and tools, including resolution theorem provers, a coherent logic theorem prover, interactive theorem provers, and a set of XML tools for coherent logic (used for translation of proofs to machine verifiable proofs and to natural language).

Within our framework we use coherent logic because it is suitable for expressing many mathematical theories while allowing for the construction of natural, intuitive, human readable proofs (in the style of forward reasoning and a variant of natural deduction) [5]. Coherent logic is essentially the modified Horn fragment which allows having a whole disjunction of existentially quantified conjunctions of atoms within the conclusion. Coherent logic can be considered as an extension of resolution logic, but in contrast to the resolution-based proving, the conjecture being proved is left unchanged and is proved directly (refutation, Skolemization, and transformation to clausal form are not used). Proofs in coherent logic are suitable for formalization projects as they can easily be translated into the input language of different proof assistants and in a natural language form. Several authors independently point to coherent logic or similar fragments of the first order logic as suitable for expressing significant portions of standard mathematics (specifically geometry), for instance, Avigad et.al. in the context of a new axiomatic foundations of Euclidean geometry [1], Ganesalingam and Gowers in the context of importance of automated generation of readable proofs [13], Tarski in the context of geometry [34], etc.

In this paper, we focus on one case-study — a computer supported development of the first part of a book on foundations of geometry: *Metamathematische Methoden in der Geometrie*, by Wolfram Schwabhäuser, Wanda Szmielew, and Alfred Tarski [28]. In the rest of this paper we will refer to this text as SST. This book is a culmination of a series of Tarski's axiomatizations along with a decision procedure for that theory [33]. According to Beeson [2], the first part of the book has been adapted from the Szmielew's lecture notes. It also includes results from the PhD thesis of Gupta [15]. The theory is described in terms of first-order logic, it uses only one primitive object — the point, has only two primitive predicates, and only eleven axioms. This choice for our case study is motivated by the following reasons:

– The book is one of the twentieth century mathematical classics.

- The book is self-contained as all the theorems in it are provable from the set of starting axioms.
- The used axioms are simple and all of them are expressed in terms of primitive notions only.
- The proofs in the book are very rigorous.
- The theory developed belongs to first-order logic.
- All sentences can be expressed in coherent logic (directly or using trivial transformations) [34].
- The set of theorems in the book is well-rounded, starting from easy ones to very complex ones, forming a non-trivial, important mathematical knowledge base.
- Computer proofs of SST, using both automated theorem proving [24, 2, 3] and interactive theorem proving [22, 11], have already been developed. These developments can be used as a reference point but also for guiding the search within our framework.

Our study, presented in the rest of this paper, shows that our framework can produce proofs for 37% of the theorems from chapter 1 to 12 of SST completely automatically (using all preceding theorems from the book as premises). One of the outputs of our framework is a digital version of SST, with all axioms, definitions, theorems, and generated proofs filled-in, all in a natural language form.

Overview of the paper: We first provide some background information (Section 2); then we describe our framework that links automated theorem provers and machine verifiable proofs (Section 3); then we present its performance on a case study — Tarski's geometry (Section 4). In Section 5 we discuss related work, and in Section 6 we draw final conclusions and discuss future work.

2 Context

Interactive theorem proving. In all areas of mathematics and computer science, with a history of a huge number of flawed published proofs, formal, machine-verifiable proofs, given in an object-level form, have been gaining more and more importance. *Interactive theorem provers* (or *proof assistants*) are trusted tools used for proving mathematical theorems in a rigorous and machine verifiable way. Many extremely complex conjectures about software correctness or mathematical theorems were proved using interactive theorem provers [16, 14]. There are growing efforts to develop repositories of proved theorems (within large corpora of formalized mathematics that can be used as building-blocks in new developments), and many software tools for producing and checking formal proofs. Among the most popular theorem proving assistants nowadays are Isabelle, Coq, Mizar, and HOL-light [37]. Despite many successful results, proof assistants are still not widely used by a scientific community. This may change with recent efforts aimed at bridging the gap between internal notions of a proof and the abstractions necessary to make them user friendly at a higher level.

Automated theorem proving. Automated theorem provers, both general purpose (like provers based on the resolution method [26]) and solvers for specific first-order theories, have reached high levels of maturity and can be used to prove (or disprove) extremely difficult or huge conjectures (involving hundreds of thousands of variables) coming from various areas of mathematics and industrial applications.

Integration of interactive and automated theorem proving. Proof arguments of automated theorem provers are not fully trusted because there can be a bug in the implementation of the prover. This problem is avoided if the prover works in conjunction with a proof assistant (which verifies the prover's outputs). Because of this, following the progress in both worlds, over the past few years much effort has been invested in combining the power of automated and interactive theorem proving: interactive theorem provers are now equipped with trusted support for SAT solving, SMT solving, resolution method, etc. These combinations open new frontiers for the application of theorem proving in software and hardware verification, but also in the formalization of mathematics and for helping mathematicians in everyday practice. Automated theorem provers can handle technical conjectures and can also reveal what axioms and lemmas are sufficient for proving some theorem. At the same time a human user, having deeper insights in the subject matter, can aid the automated theorem prover by supplying it with additional lemmas that could prove critical in the proof generation process.

One of very successful examples of cooperation between a proof assistant and an automated theorem prover is the Sledgehammer system [7, 8]. This system works within the proof assistant Isabelle and can invoke an external resolution-based theorem prover such as Vampire [25], E [27], or SPASS [35]. If a conjecture is successfully proved by an external prover, the list of axioms used in the proof can be passed to the resolution-based internal prover Metis (otherwise helpless, because it is not as efficient as external provers). Metis can then construct a verified proof for the given conjecture.

Readable proofs. In formalizing mathematical knowledge, apart from the issue of trusted proofs (guaranteed by proof assistants) and the issue of automation (provided by automated theorem provers), there is also an issue of *readable proofs*. Readable proofs (e.g., textbook-like proofs), are often not critical in fields such as software verification (although can also be useful in such areas — e.g. for proving an understandable argument why some command leads to a bug), but are very important in mathematical practice. For mathematicians, the main goal is often, instead of only a *yes* or *no* answer, a clear and intuitive proof, that serves not only as a justification, but more importantly as an explanation as well. The Intelligible semiautomated reasoning (Isar) approach to readable formal proof documents [36] is one of the systems following this direction — Isar allows the user to express proofs in a human-friendly way, but they are still automatically verifiable by the underlying proof system.

Coherent Logic and ArgoCLP. Certain fragments of first-order logic allow automated theorem proving with a simple production of readable proofs — one of them is coherent logic. Coherent logic (CL), initially defined by Skolem, has gained new attention in recent years [5, 12, 6]. It consists of formulae of the following form:

$$A_1(\boldsymbol{x}) \wedge \ldots \wedge A_n(\boldsymbol{x}) \Rightarrow \exists \boldsymbol{y}(B_1(\boldsymbol{x}, \boldsymbol{y}) \vee \ldots \vee \ B_m(\boldsymbol{x}, \boldsymbol{y})) \tag{1}$$

which are implicitly universally quantified, and where $0 \leq n$, $0 \leq m$, $\boldsymbol{x}$ denotes a sequence of variables $x_1, x_2, \ldots, x_k$ $(0 \leq k)$, A_i (for $1 \leq i \leq n$) denotes an atomic formula (involving zero or more of the variables from $\boldsymbol{x}$), $\boldsymbol{y}$ denotes a sequence of variables $y_1, y_2, \ldots, y_l$ $(0 \leq l)$, and B_j (for $1 \leq j \leq m$) denotes conjunction of atomic formulae (involving zero or more of the variables from $\boldsymbol{x}$ and $\boldsymbol{y}$). For simplicity, we assume that there are no function symbols with arity greater than zero (so, we only consider symbols of constants as ground terms).

The definition of CL does not involve negation. For a single atom A, its negation $\neg A$ can be represented in the form $A \Rightarrow \bot$, where $\bot$ stands for the empty disjunction, but more general negation must be expressed carefully in coherent logic. In order to deal with negation in general, new predicate symbols are used to abbreviate subformulae. Furthermore, for every predicate symbol R (that appears in negated form), a new symbol $\overline{R}$ is introduced that stands for $\neg R$, and the following axioms are postulated (cf. [23]): $\forall \boldsymbol{x}(R(\boldsymbol{x}) \wedge \overline{R}(\boldsymbol{x}) \Rightarrow \bot)$, $\forall \boldsymbol{x}(R(\boldsymbol{x}) \vee \overline{R}(\boldsymbol{x}))$.

A number of theories and theorems can be formulated directly and simply in CL. It can be proved that any first-order formula can be translated into a set of CL formulae (in a different signature) preserving satisfiability [23] (however, this translation itself is not always constructive). Coherent logic is semi-decidable and there are several implemented semi-decision procedures for it [5]. The weakness of theorem provers for coherent logic is that they are not very efficient and cannot prove complex mathematical theorems.

ArgoCLP [30] is a generic theorem prover for coherent logic, which is based on a simple proof procedure with forward chaining and iterative deepening. ArgoCLP can read problems given in TPTP form[1] and can export proofs to a custom XML representation. This proof representation can further be translated, by a simple XSLT stylesheet, to Isabelle/Isar proofs, Coq proofs, and proofs in natural language (English) formatted in LaTeX or in HTML [29]. Here we list one theorem (4.19) from SST and its proof in natural language form (formatted in LaTeX) generated automatically by ArgoCLP and our XML tools (for better readability, $\cong$ was used as the layout of the predicate *cong*):

[1] `http://www.cs.miami.edu/~tptp/`

Theorem 1 (th_4_19). *Assuming that* $bet(A, B, C)$ *and* $AB \cong AD$ *and* $CB \cong CD$ *it holds that* $B = D$.

Proof:

1. It holds that $bet(B, A, A)$ (using th_3_1).
2. From the fact(s) $bet(A, B, C)$ it holds that $col(C, A, B)$ (using $ax\_4\_10\_3$).
3. From the fact(s) $AB \cong AD$ it holds that $AD \cong AB$ (using th_2_2).
4. It holds that $A = B$ or $A \neq B$.
   5. Assume that: $A = B$.
      6. From the fact(s) $AD \cong AB$ and $A = B$ it holds that $AD \cong AA$.
      7. From the fact(s) $AD \cong AA$ it holds that $A = D$ (using ax_3).
      8. From the fact(s) $A = B$ and $A = D$ it holds that $B = D$.
      9. The conclusion follows from the fact(s) $B = D$.
   10. Assume that: $A \neq B$.
      11. It holds that $A = C$ or $A \neq C$.
         12. Assume that: $A = C$.
            13. From the fact(s) $bet(A, B, C)$ and $A = C$ it holds that $bet(A, B, A)$.
            14. From the fact(s) $bet(A, B, A)$ and $bet(B, A, A)$ it holds that $A = B$ (using th_3_4).
            15. From the fact(s) $A \neq B$ and $A = B$ we get contradiction.
         16. Assume that: $A \neq C$.
            17. From the fact(s) $A \neq C$ it holds that $C \neq A$.
            18. From the fact(s) $C \neq A$ and $col(C, A, B)$ and $CB \cong CD$ and $AB \cong AD$ it holds that $B = D$ (using th_4_18).
            19. The conclusion follows from the fact(s) $B = D$.
      20. The conclusion follows in all cases.
21. The conclusion follows in all cases.

QED

---

3 Framework Description

All of the proving systems described in Section 2 have their strengths and weaknesses: proof assistants are trusted, but the level of automation within them is often low; resolution provers are automated and efficient, but they do not produce machine-verifiable or human-readable proofs; coherent logic provers are automated and can export both machine-verifiable and readable proofs, but are not efficient enough for proving complex mathematical theorems. Combining the power of these systems, we aim to produce a framework for automation of formalization of mathematical theories with all axioms and conjectures expressed in coherent logic.

The framework combines the power and features of resolution theorem provers (Vampire, E, and Spass), an automated theorem prover for coherent logic (ArgoCLP), and a set of XML tools for dealing with proofs in coherent logic.

Given a list of axioms and theorems, proving conjectures in the presented framework work as follows:

1. The available axioms and theorems are passed (in TPTP form) to resolution based automated theorem provers.
2. The resolution provers try to prove the conjecture using the given list of axioms and theorems both in normal and in reversed order (the order of premises can have a significant impact on the proving process). If one or more resolution provers proves the conjecture, the smallest list of used axioms/theorems (returned by the resolution prover) is used for proving the conjecture again, in the same manner. This process is repeated until the list of used axioms/theorems remains unchanged between two consecutive iterations.[2]
3. With the obtained list of axioms, ArgoCLP prover is invoked, and (if successful) the proofs are exported in the XML form, which can further be translated to the Isar, Coq, and a natural-language form.

In the first step, the input premises can be either all axioms and theorems that precede the conjecture within a development of some mathematical theory, or some smaller set or relevant formulae. In the last step the ArgoCLP prover is used to generate machine verifiable and natural-language proofs. One could potentially construct such proofs directly from traces of the resolution provers [4], but it is not a trivial task and instead we used our prover ArgoCLP. One negative of ArgoCLP is that it may fail to prove some theorems even if the list of all axioms/theorems, that was successfully used by the resolution provers, is given.

The described framework is implemented in C++ and can be used in a wider context of building a mathematical theory — for proving a sequence of conjectures and also in completing a mathematical textbook in LaTeX form, with axioms and theorems in TPTP form, a new (completed) version can be generated with automatically generated proofs filled-in.

4 Case Study: Tarski's Geometry

In this section we focus on using the framework described in Section 3 on the first part of SST (the second part is concerned with metamathematical issues). Our goal is to produce readable and machine verifiable proofs of theorems from SST automatically. Just like in the Coq formalization [11], we will focus only on the theorems found in the first 12 chapters of SST that belong to plane geometry. Those chapters, covering 120 pages of the book, contain 203 theorems, while 179 of them belong to plane geometry.

[2] Of course, if a resolution prover has an option of returning minimal inconsistent set of clauses, this iterative process does not need to be applied.

4.1 Tarski's Axioms

Tarski's axioms are expressed in terms of first-order logic with equality, without sorts (only primitive objects are *points*, denoted by small Latin letters), and with two primitive predicate symbols: D (for *congruent*) and B (for *between*). For the use of the dominant naming scheme and better readability, we will denote points by capital Latin letters, we will denote sets of points by capital Greek letters, and we will denote the predicate symbols D and B by *cong* (in prefix form) and *bet*. We will use this notation in the following text, even when citing the original Tarski's material. Tarski's axioms are as follows (it is assumed that all axioms are universally closed):

Axiom A1 (symmetry): $cong(A, B, B, A)$
Axiom A2 (pseudo-transitivity): $cong(A, B, P, Q) \ \wedge \ cong(A, B, R, S) \ \Rightarrow \ cong(P, Q, R, S)$
Axiom A3 (cong identity): $cong(A, B, C, C) \ \Rightarrow \ A = B$
Axiom A4 (construction): $\exists X \ (bet(Q, A, X) \ \wedge \ cong(A, X, B, C))$
Axiom A5 (five segments): $A \neq B \ \wedge \ bet(A, B, C) \ \wedge \ bet(A', B', C') \ \wedge \ cong(A, B, A', B') \ \wedge \ cong(B, C, B', C') \ \wedge \ cong(A, D, A', D') \ \wedge \ cong(B, D, B', D') \ \Rightarrow \ cong(C, D, C', D')$
Axiom A6 (bet identity): $bet(A, B, A) \ \Rightarrow \ A = B$
Axiom A7 (Pasch):
$bet(A, P, C) \ \wedge \ bet(B, Q, C) \ \Rightarrow \exists X \ (bet(P, X, B) \ \wedge \ bet(Q, X, A))$
Axiom A8 (lower dimension):
$\exists A \ \exists B \ \exists C \ (\neg bet(A, B, C) \ \wedge \ \neg bet(B, C, A) \ \wedge \ \neg bet(C, A, B))$
Axiom A9 (upper dimension):
$P \neq Q \ \wedge \ cong(A, P, A, Q) \ \wedge \ cong(B, P, B, Q) \ \wedge \ cong(C, P, C, Q) \ \Rightarrow \ (bet(A, B, C) \vee bet(B, C, A) \vee bet(C, A, B))$
Axiom A10 (euclid): $bet(A, D, T) \ \wedge \ bet(B, D, C) \ \wedge \ A \neq D \ \Rightarrow \ \exists X \ \exists Y \ (bet(A, B, X) \ \wedge \ bet(A, C, Y) \ \wedge \ bet(X, T, Y))$
Axiom A11 (continuity): $\forall \Phi \ \forall \Psi \ \exists A \ \forall X \ \forall Y \ ((X \in \Phi \wedge Y \in \Psi \Rightarrow bet(A, X, Y)) \ \Rightarrow \exists B \ \forall X \ \forall Y \ (X \in \Phi \wedge Y \in \Psi \Rightarrow bet(X, B, Y))$

The axiom A8 states that there are three non collinear points. This implies that the dimension of the space is at least 2.

The axiom A9 states that if three points are at the same distance of two points then they are collinear. This implies that the dimension of the space is at most 2. This is actually only a special instance of the axiom A9 which has the following general form:

Axiom A9':

$$\left[\bigwedge_{1 \leq i < j \leq n} P_i \neq P_j \wedge \bigwedge_{i=2}^{n} cong(A, P_1, A, P_i) \right.$$
$$\left. \wedge \bigwedge_{i=2}^{n} cong(B, P_1, B, P_i) \wedge \bigwedge_{i=2}^{n} cong(C, P_1, C, P_i) \right]$$
$$\Rightarrow [(bet(A, B, C) \vee bet(B, C, A) \vee bet(C, A, B)]$$

One of the beautiful features of Tarski's geometry is that the dimension of the space is controlled by only one parameter (n) in only one axiom (the one above). We will use only the instance where $n = 2$ (as given by axiom A9), i.e., the plane geometry.

The axiom A10 is equivalent (in the context of the other axioms) to Euclid's parallel postulate. It is used onwards from Chapter 12 of SST.

The axiom A11 (the axiom of continuity) that involves sets of points Φ and Ψ can be reformulated into a schema of first-order axioms:

Axiom A11': $\exists A\, \forall X\, \forall Y\, ((\phi(X) \wedge \psi(Y) \Rightarrow bet(A, X, Y)) \Rightarrow \exists B\, \forall X\, \forall Y\, (\phi(X) \wedge \psi(Y) \Rightarrow bet(X, B, Y))$

where ϕ and ψ are any first-order formulae with no free occurrences of A or B.

Notice that all the axioms except A11 belong to coherent logic. The axiom 11 is not used in the first 12 chapters of SST.

Gupta has shown several independence results about this axiom system [15]. Recently Timothy Makarios proposed a further simplification of this system [19]. Swapping two points in the axiom A5 allows omitting the axiom A2. All axioms are shown to be independent except the axioms A3 and A7, which remain open problems.

4.2 Expressing Tarski's Geometry in Coherent Logic

Most of the contents of SST can be simply expressed in coherent logic. In this part of the text we discuss how we formulated all the axioms, definitions, and theorems, as faithfully to the original formulations as possible. This process resulted in 238 coherent logic theorems, a slight increase from the starting 179. This process was not always straightforward and we had to make some adaptations and decisions which are described next (some of these adaptations were also used within other formalizations of SST).

Treatment of Negations. As mentioned before, in coherent logic there is no negation, so typically for every predicate symbol R there is a new predicate symbol $\overline{R}$ that corresponds to $\neg R$, and the following two axioms: $R(\boldsymbol{x}) \wedge \overline{R}(\boldsymbol{x}) \Rightarrow \bot$, $R(\boldsymbol{x}) \vee \overline{R}(\boldsymbol{x})$. In the concrete case of Tarski's geometry, it can be proved that this general approach can be simplified. Indeed, Boutry and Narboux have recently shown that, in the context of Tarski's axiom system, both $cong(\boldsymbol{x}) \vee \overline{cong}(\boldsymbol{x})$ and $bet(\boldsymbol{x}) \vee \overline{bet}(\boldsymbol{x})$ are equivalent to the excluded middle axiom for equality. The Coq formalization has been rebuilt since the first version [11] and it does not use excluded middle in its general form anymore. The first 12 chapters can be formalized assuming only decidability of equality and intersection of lines [9].

Treatment of Function Symbols. Apart from function symbols related to lines and planes (for instance, $L(P, Q)$ corresponds to the line determined by two

distinct points P and Q), the use of function symbols is very limited in SST and the first time that the authors use a function symbol is in Chapter 7, Definition 7.5, page 49, for the definition of the mirror image of a point:

$$S_A(P) = P' := is\_midpoint(P, A, P'),$$

where $is\_midpoint(P, A, P')$ is defined as $bet(P, A, P') \wedge cong(A, P, A, P')$. Function symbols are not used in coherent logic, so we use an additional predicate symbol $is\_symmetric$ defined as follows:

$$is\_symmetric(P, P', A) := is\_midpoint(P, A, P').$$

With this definition at hand, some changes in formulations of theorems (theorems 7.13, 7.15, 7.16, 7.18, 7.19) had to be made. Each time when $S_A(P)$ is used implicitly (without stating the symmetric point), a new point P' with the property $is\_symmetric(P, P', A)$ has to be introduced.

The next function symbol is introduced in Chapter 10, Definition 10.1:

$$M(A, B) = X := is\_midpoint(A, X, B).$$

Since there is already a predicate symbol that can mimic this function, we do not need to introduce a new predicate symbol. The use of this function symbol in the following formulae is analogous to the previous one.

There is only one more function symbol – is_image, defined in Chapter 10, Definition 10.3.

Treatment of Sets. Tarski aimed at building elementary geometry within first-order logic and using only one primitive sort — intuitively *points*. In some definitions, however, sets are used and it is not explicit what set theory or which fragment of a set theory is required. For instance, in Chapter 6, *half-lines* (rays) are defined in the following way (Definition 6.8, p44):

$$H(PA) := \{X | X \simeq_P A\},$$

where $A \simeq_P B$ (intuitively: the points A and B are on the same side of the point P) is defined as $A \neq P \wedge B \neq P \wedge (bet(P, A, B) \vee bet(P, B, A))$. Also, given P and Q are two distinct points, the *line* $L(PQ)$ determined by P and Q is defined (Definition 6.14, p45) as follows ($col(A, B, C)$ is defined as $bet(A, B, C) \vee bet(B, C, A) \vee bet(C, A, B)$):

$$L(PQ) := \{X | col(P, Q, X)\}.$$

In addition, there is a predicate Ln (p45) that corresponds to lines, used in many of the subsequent theorems:

$$Ln(a) := \exists P \exists Q (P \neq Q \wedge Ln(a) = L(PQ)).$$

Within our formalization, sets are not used and, instead, additional predicates (that mimic dealing with sets of points) are introduced. For instance, in our development $point\_on\_line(A, X, Y)$ replaces $A \in p$ and is defined as follows:[3]

$$point\_on\_line(A, X, Y) \ := \ (X \neq Y \ \wedge \ col(A, X, Y))).$$

Similarly, $same\_lines(P1, P2, Q1, Q2)$, replaces $p = q$ (for the line p determined by points $P1$ and $P2$, and the line q determined by points $Q1$ and $Q2$) and is defined as follows:

$$same\_lines(P1, P2, Q1, Q2) \ :=$$

$$P1 \neq P2 \wedge Q1 \neq Q2 \wedge point\_on\_line(P1, Q1, Q2) \wedge point\_on\_line(P2, Q1, Q2).$$

There is a number of predicate symbols introduced this way, including (r/k denotes a predicate symbol r of arity k): $point\_on\_plane3p/4$, $point\_on\_plane2l/5$, $line\_on\_plane3p/5$, $same\_planes2l/8$.

It can be proved (using a large number of theorems from the book) that all our new predicates are equivalent to the original predicates. Proving these conjectures formally (within Isabelle) is the subject of our current work and is beyond the scope of this paper.

Coherent Form. The first 10 axioms and all theorems (179) from the first 12 chapters of SST (after the modifications described above) can be divided into the following four groups:

1. *Formulae that are in the coherent logic form.* As expected, most of the theorems from SST (94) are already in coherent form (which is one of the reasons why coherent logic is used).
2. *Formulae that can be* easily *translated into the coherent logic form.* Theorems from this group (31) can easily be transformed into one or more coherent logic formulae, using the following transformations (formulae are implicitly universally quantified; $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, $\mathcal{D}$ denote conjunctions of atomic formulae; x is not free in $\mathcal{A}$ in $\mathcal{A} \Rightarrow \forall x\mathcal{B}$, etc.):

$$\begin{aligned}
\mathcal{A} \Leftrightarrow \mathcal{B} &\equiv \mathcal{A} \Rightarrow \mathcal{B} \wedge \mathcal{B} \Rightarrow \mathcal{A} \\
\mathcal{A} \Leftrightarrow (\mathcal{B} \vee \mathcal{C}) &\equiv (\mathcal{A} \Rightarrow \mathcal{B} \vee \mathcal{C}) \wedge (\mathcal{B} \Rightarrow A) \wedge (\mathcal{C} \Rightarrow \mathcal{A}) \\
\mathcal{A} \Rightarrow (\mathcal{B} \Leftrightarrow \mathcal{C}) &\equiv (\mathcal{A} \wedge \mathcal{B} \Rightarrow \mathcal{C}) \wedge (\mathcal{A} \wedge \mathcal{C} \Rightarrow \mathcal{B}) \\
\mathcal{A} \Leftrightarrow \mathcal{B} \wedge (\mathcal{C} \vee \mathcal{D}) &\equiv A \Leftrightarrow (\mathcal{B} \wedge \mathcal{C}) \vee (\mathcal{B} \wedge \mathcal{D}) \\
\mathcal{A} \Rightarrow (\mathcal{B} \wedge (\mathcal{C} \Rightarrow \mathcal{D})) &\equiv (\mathcal{A} \Rightarrow \mathcal{B}) \wedge (\mathcal{A} \wedge \mathcal{C} \Rightarrow \mathcal{D}) \\
\mathcal{A} \Rightarrow \forall x\mathcal{B} &\equiv \forall x(\mathcal{A} \Rightarrow \mathcal{B}) \\
\mathcal{A} \Leftrightarrow \exists x\mathcal{B} &\equiv ((\mathcal{A} \Rightarrow \exists x\mathcal{B}) \wedge (\forall x(\mathcal{B} \Rightarrow \mathcal{A}))) \\
\mathcal{A} \Rightarrow \mathcal{B} \wedge \exists x\mathcal{C} &\equiv \mathcal{A} \Rightarrow \exists x(\mathcal{B} \wedge \mathcal{C}) \\
\mathcal{A} \Rightarrow \exists^{\leq 1} x\mathcal{B}(x) &\equiv (\forall x_1 \forall x_2(\mathcal{A} \wedge \mathcal{B}(x_1) \wedge \mathcal{B}(x_2) \Rightarrow x_1 = x_2))
\end{aligned}$$

[3] This definition can trivially be broken into two implications that are in coherent form and therefore can be used within our framework.

$$\mathcal{A} \Rightarrow \exists^{=1} x \mathcal{B}(x) \equiv (A \Rightarrow \exists x \mathcal{B}(x)) \wedge \\ (\forall x_1 \forall x_2 (\mathcal{A} \wedge \mathcal{B}(x_1) \wedge \mathcal{B}(x_2) \Rightarrow x_1 = x_2))$$

3. *Formulae that can be* reformulated *to fit into the coherent logic form.* Typically, most of the theorems from this group (48) involve predicates that correspond to the sort line and deal with lines as sets of points and hence need (simple) reformulation. There is only one definition (Definition 8.11) that needs other type of reformulation (a and a' are lines, $per(U, X, V)$ holds if the angle UXV is the right angle):

$$a \perp_X a' \ := \ X \in a \wedge X \in a' \wedge \forall U \forall V (U \in a \wedge V \in a' \Rightarrow per(U, X, V)).$$

The first step in the transformation is formulation in terms of pairs of points (instead of lines) and elimination of inner universal quantifiers ($\forall U \forall V$), based on a geometric property:

$$perp\_in(X, A, B, C, D) := (A \neq B \wedge C \neq D \wedge X \in lineAB \wedge X \in lineCD \wedge \\ ((A \neq X \wedge C \neq X \wedge per(A, X, C)) \vee (A \neq X \wedge D \neq X \wedge per(A, X, D)) \vee \\ (B \neq X \wedge C \neq X \wedge per(B, X, C)) \vee (B \neq X \wedge D \neq X \wedge per(B, X, D)))).$$

The above definition is then transformed (as described above) into eight coherent logic formulae, one of which is (written in TPTP form):

```
fof(ax_8_11_1_1, axiom, (! [X,A,B,C,D] :
   ((perp_in(X,A,B,C,D) & A!=X & C!=X) =>
   (A!=B & C!=D & point_on_line(X,A,B) &
   point_on_line(X,C,D) & per(A,X,C))))).
```

Transformations like the above may introduce complex definitions, and obviously there is some trade-off between readability on one hand and using coherent form and avoiding using sets on the other. Still, the above ad-hoc transformation allows us to have more readable proofs than had we used a general encoding algorithm of first order logic into coherent logic as the one presented in [5] which would have created a number of auxiliary predicates.

4. *Formulae that involve n-tuples (for arbitrary n).* We treated the theorems from this group (5) only as special cases for $n \leq 2$.

TPTP Form. The TPTP format[4] [31] supports first order logic and since coherent logic is a fragment of first order logic, the transformed Tarski's formulae (axioms and theorems) can be directly written in the TPTP form.

The list of all axioms, definitions, and theorems, used in our development (with information on kinds of formula and on matching with theorems from SST and from the Coq development) is available online.[5]

[4] `http://www.cs.miami.edu/~tptp/`

[5] `http://argo.matf.bg.ac.rs/?content=downloads`

4.3 Consistency of the Development

In developing a formalization of a mathematical theory, proof assistants ensure (to a very high degree) that proof derivations are correct, but the sole responsibility for the formulations of the axioms is on the human side. A human can unfaithfully formulate axioms from a mathematical theory, or can simply make a typo and make a system inconsistent. Furthermore, a human can introduce errors in definitions and theorems that could propagate and influence many theorems. For these reasons, availability of automated theorem provers can be very helpful in detecting possible inconsistencies.

While building a computer version of SST we used the resolution provers to check for inconsistencies (as a consequence of our errors) — within a given time-limit, we tried to derive $\perp$ from the set of axioms and theorems. This is just a simple heuristic, but still, we discovered several errors in our formalization (most often typos or missing predicates). With that type of errors, some conjectures were not valid anymore, and, even worse, introduced inconsistencies with the axioms and theorems which led to the trivial proofs of the following theorems (because of the inconsistent premises).

4.4 Overview of the Existing Computer Developments

There are three existing computer developments of SST: one by Quaife based on automated theorem proving [24], one by Beeson and Wos based on automated theorem proving [2, 3], and one by Narboux and Braun based on interactive theorem proving [22, 11].

In his development, Quaife used the resolution-based prover Otter for proving theorems from the first five chapters of the book (his proving process took around two weeks). Beeson and Wos again used Otter (a newer version) for replaying and extending Quaife's work. By instructing Otter with "hints" and "resonators" they managed to prove all the theorems up to and including the theorem 9.6 and many key theorems up to Chapter 12 [2, 3]. The formalization by Beeson and Wos can be considered semi-automatic as the authors provide many guidelines to Otter for the difficult theorems: they provide the points to be constructed and also long lists of proof steps. Beeson and Wos also provide intermediate lemmas not appearing in the original book, some of which seem to be inspired by the Coq formalization (see the lemma in Chapter 5[6]). Within this development, neither formal proofs (verifiable by proof assistants) nor readable and natural-language proofs were considered.

Braun and Narboux proved most theorems (theorems about three-dimensional geometry are omitted) from the first twelve chapters within the proof assistant Coq (with minimal automation) [22, 11]. Recently, Braun, Narboux, and Boutry proved some high-level geometry theorems based on Tarski's axiom system: existence of the center of gravity, orthocenter, circumscribed circle as well as stan-

[6] `http://www.michaelbeeson.com/research/FormalTarski/index.php?include=archive5`

dard properties of quadrilaterals [10]. Like the development presented in this paper, both the Coq and Otter formalizations do not use sets and all formulations from SST are changed accordingly. For instance, none of the above developments use the concept of lines – instead, lines are implicit and represented by pairs of points. However, unlike our approach, these approaches do not use additional predicates that mimic dealing with sets of points. These developments also do not use function symbols: n-ary functions are represented by predicates of arity $n + 1$. In both formalizations this requires proving some intermediate lemmas such as pseudo transitivity of collinearity:

$$A \neq B \wedge col(A, B, C) \wedge col(A, B, D) \Rightarrow col(A, C, D).$$

The chapters of SST are uneven in depth and difficulty of their contents and this is also reflected in the computer developments. One (of course, informal) illustration of depth and complexity of the chapters are the numbers of theorems in the computer developments. The total number of theorems is: 565 in the Coq development, [7] 119 in the Otter development (up to Section 9.6), and 238 in our TPTP development in coherent logic. Figure 1 shows the number of theorems in SST and in the computer developments. As an informal measure of the difficulty of theorems, we use the number of lines in the proofs within the Coq development (of course, some theorems can be proved in some other way, so this measure is not perfect, but still can be used as an illustration). Figure 2 shows the distribution of theorems with respect to the proof lengths in the Coq development.

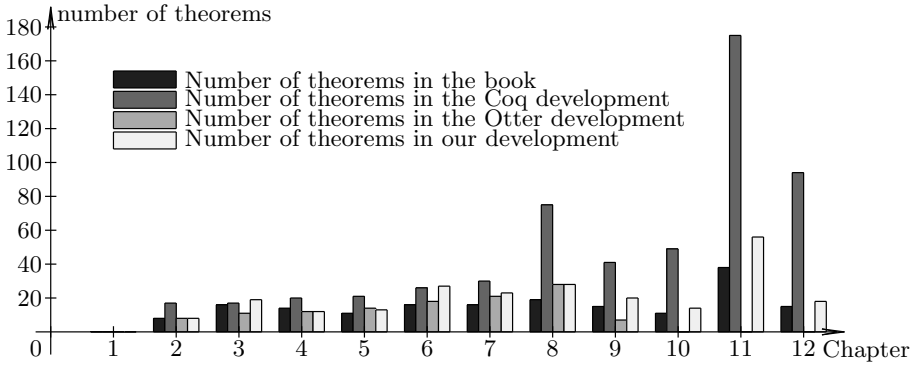

Fig. 1. Numbers of theorems in the book and in the computer developments.

[7] Note that the whole of the available Coq formalization involves also some lemmas not used for proving theorems from SST, but for other tasks. Out of 565 lemmas and theorems, 456 are used for proving theorems from SST.

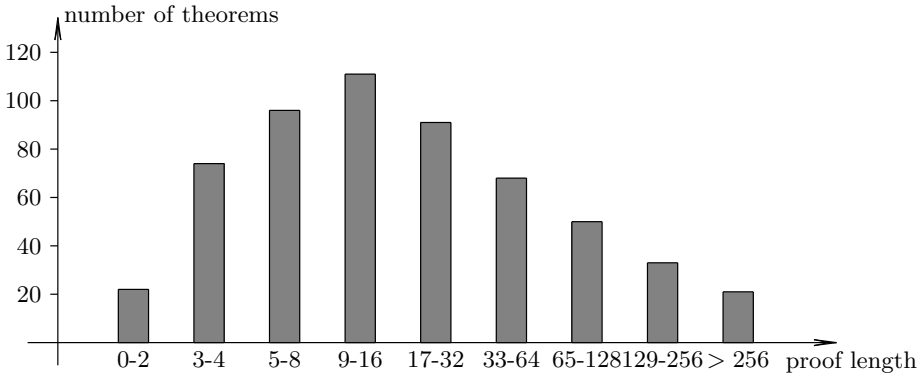

Fig. 2. Histogram of the lengths of the Coq proofs

4.5 Merging Together Computer Developments

In order to use the existing Coq development for guiding our automated proving framework, we first needed to match it with the SST lists of theorems and with our TPTP list of theorems (we performed these tasks partly using our *ad-hoc* tools and partly manually). Most often, it was easy, but there were also some challenges. For instance, in computer developments, many SST theorems were broken into several ones often belonging to coherent logic (although the authors of the Coq development did not intend to use any tool for coherent logic), so the matching with SST was not one-to-one.

For one of our key experiments, we needed to integrate all the lemmas from the Coq formalization into our TPTP development. First, all the Coq lemmas were (if needed) translated to coherent logic and the TPTP format. Second, all the Coq lemmas (in CL form) had to be matched with corresponding SST theorems (SST theorems were not always uniformly named in the Coq formalization) and then inserted into our list of theorems. Using dependency files, the two formalizations were merged such that every Coq lemma (not existing in our original TPTP list) was positioned at the latest possible place (before the first theorem whose proof uses that lemma). Merging two computer developments (in this case – the Coq and the TPTP version) was rather challenging, not only because of different syntax, naming conventions, and available features, but also because of little differences in formulation of the original contents. SST is very rigorous, so it seems a bit unexpected that there are ambiguities and, consequently, differences in computer developments.[8] Looking closer reveals the main reason for these problems: as said above, in contrast to SST, neither of the exist-

[8] In formalizing mathematical knowledge, because of different ambiguities there is a general problem of finding the intended meaning of the author. This problem is notorious for some mathematical classics, e.g., Hilbert's *Grundlagen der Geometrie* [17], as it was demonstrated within several formalization projects [20, 21].

ing computer developments uses sets. It is even claimed that SST substantially does not use sets or that sets can be trivially avoided. Still, by avoiding sets, translations from SST are not always straightforward and some errors can be introduced. As an illustration, we will briefly present (using a common mathematical notation) an error that we discovered, in the existing Coq formalization of SST, while taking over the definitions and lemmas from the Coq development and integrating them in our TPTP list. In the Coq development, the predicate $is\_image(P', P, A, B)$ (P and P' are symmetric w.r.t. the line AB) was defined (following the first part of SST definition 10.3) as:

$\exists X(is\_midpoint(X, P, P') \wedge col(A, B, X) \wedge (perp(A, B, P, P') \vee P = P')$,

where $is\_midpoint(X, P, P')$ holds if X is the midpoint of PP', $col(A, B, C)$ holds if A,B,C are collinear, and $perp(A, B, P, P')$ holds if AB is perpendicular to PP'. The Coq development contains the following lemma:

$$col(A, B, X) \Rightarrow is\_image(X, X, A, B).$$

But there is a special case when $A = B$ in Definition 10.3 ($is\_symmetric(P, P', A)$ holds if P and P' are symmetric w.r.t. A):

$$is\_image(P, P', A, B) := \begin{cases} \exists X \begin{pmatrix} is\_midpoint(X, P, P') \wedge col(A, B, X) \\ \wedge (perp(A, B, P, P') \vee P = P') \end{pmatrix} & \text{if} A \neq B \\ is\_symmetric(P, P', A) & \text{if } A = B \end{cases}$$

So, from the fact that $col(A, A, P)$ holds it follows that $is\_image(P, P, A, A)$ and then $is\_symmetric(P, P, A)$ will hold, which is not the case in general (i.e., not in the case when $P \neq A$). The problem occurred because SST uses lines (as sets of points), while the Coq development uses only points and the translation from the original formulations to Coq was not completely straightforward. This small unfaithfulness to the SST definition actually led to the inconsistency which was discovered during integration with our formalization (use of resolution provers discovered the inconsistency). Changing the definition in the Coq development to be faithful to the SST definition gave us lemmas with fewer non-degeneracy conditions. Many conditions $A \neq B$ could be removed thanks to the more general definition of the predicate is_image.

The use of the automated provers in the merging process also helped in detecting some duplicated or redundant lemmas (for instance, if a lemma can be proved using only one other lemma, it can be eliminated as redundant).

4.6 Automation and Results

With all Tarski's axioms, definitions, and theorems (reformulated if needed, as described in Section 4.2) from the first twelve chapters of SST stored in TPTP form, we performed several experiments trying to explore how many theorems (out of the total of 238) can be proved automatically, with some sort of guidance. The existing developments can be used not only as a reference (for a comparison with an automated approach), but also as a simulation of real human guidance. We used information that we got from interactively developed proofs within Coq [22, 11].

The experiments were performed on a set of four 12-core servers with AMD Opteron 6168 CPUs. For each conjecture, all resolution provers were invoked twice (for premises given in normal and reversed order), with the time limit of 60s for each call,[9] while ArgoCLP prover had the time limit of 1000s. [10]

Full Automation, No Human Guidance. In our first experiment, we applied our framework (described in Section 3) to the list of 238 theorems corresponding to the contents of the first twelve chapters of SST. The axioms, definitions, and theorems are listed in exactly the same way as in the book. When attempting to prove one theorem, all axioms, definitions, and theorems (proved automatically or not) that precede that theorem are used (i.e., passed to the provers). In addition, we used the following specifics:

- for proving theorems from the first eleven chapters, we pass axioms A1-A9 to the provers; for proving theorems from Chapter 12, we also pass the axiom A10. We did not use the axiom A11.
- we pass the axioms of the form: $\forall \boldsymbol{x}(R(\boldsymbol{x}) \wedge \overline{R}(\boldsymbol{x}) \Rightarrow \bot)$, $\forall \boldsymbol{x}(R(\boldsymbol{x}) \vee \overline{R}(\boldsymbol{x}))$, for all primitive and defined predicate symbols R.

By this fully automated approach, out of 238 theorems, 114 (48%) were proved by (at least one of) the resolution provers, 87 (37%) of them were then proved by ArgoCLP, and for 41 (17%) of them all used theorems and lemmas were also proved by ArgoCLP. Detailed information (by chapters) is given in Table 1 and in Figure 3.

Automation with Implicit Guidance (automation with extended list of lemmas). In our second experiment, we expanded the list of theorems with the list of auxiliary lemmas used within the Coq formalization. The idea is that auxiliary lemmas could be easier to prove than original theorems, and with those lemmas at hand, the proving process of the following theorems should be easier. The experiment simulates a benefit that could be gained in an interaction between a human mathematician and a computer: a mathematician builds a theory, formulates additional lemmas and tries to prove lemmas/theorems first automatically, and if that does not succeeded, interactively. We call this and the next experimental setting "automation with guidance", because Coq lemmas

[9] The relatively low time limit is chosen to reflect possible real-world applications (Sledgehammer uses a time limit of 30s). Higher time-limits do not change the overall picture much: in the experiments without guidance, the resolution provers proved 48% theorems for the 1 minute time limit, 49% theorems for the 2 minutes time limit, 51% theorems for the 5 minutes time limit, 54% theorems for the 30 minutes time limit.

[10] The time limit for ArgoCLP is relatively high, but it is not much higher than the time limit for the resolution provers (since they take $6 \times 60s$). The time limit of 1000s is acceptable for an offline formalization work. Actually, the framework is usable also in an interactive regime: 84% conjectures proved by ArgoCLP within 1000s were also proved within the 60s time limit.

Chapter	number of	basic			extended			dep.lists		
	theorems	res	argo	full	res	argo	full	res	argo	full
1	0	0	0	0	0	0	0	0	0	0
2	8	8	6	6	7	6	6	8	7	7
3	19	18	14	7	18	14	7	18	15	13
4	12	5	5	2	5	5	2	7	5	2
5	13	6	5	2	6	6	2	9	6	2
6	27	23	16	8	24	16	8	23	14	9
7	23	18	17	8	18	17	8	19	17	11
8	28	11	9	2	15	10	2	19	8	2
9	20	11	7	4	10	6	4	9	5	4
10	14	5	4	1	9	7	1	9	7	1
11	56	7	3	0	13	8	0	24	8	0
12	18	2	1	1	5	4	1	5	3	1
total	238	114	87	41	130	99	41	150	95	52
percentage	100	48%	37%	17%	55%	42%	17%	63%	40%	22%
additional lemmas	218				137	88	27	147	76	38
total	456				267	187	68	297	171	90
percentage	100				59%	41%	15%	65%	38%	20%

Table 1. Numbers of theorems proved by our (basic) framework, numbers of theorems proved with auxiliary lemmas imported from Coq (extended), numbers of theorems proved using dependency lists imported from Coq proofs (dep.lists). For each category, 'res' stands for theorems proved by one of resolution provers, 'argo' stands for theorems proved then by ArgoCLP, and 'full' stands for theorems that were proved when all the theorems used as premises were also proved.

generated by a human are used as a guidance to the automated provers. We call this setting "with *implicit* guidance" because the resolution provers were not given a short list of axioms/lemmas/theorems actually used in the proof of the theorem being considered, unlike in the following experiment.

The setting of this experiment was the same as for the first one. By this approach, out of 238 theorems, 130 (55%) were proved by (at least one of) the resolution provers, 99 (42%) of them were then proved by ArgoCLP, and for 41 (17%) of them all used theorems and lemmas were also proved by ArgoCLP. Concerning the lemmas taken from the Coq development, the performance is similar for all of 218 lemmas: 137/88/27, giving the following totals for 456 theorems: 267/187/68. Detailed information (by chapters) is given in Table 1 and in Figure 4.

Automation with Explicit Guidance (automation with direct dependency lists). In our third experiment, for a concrete theorem being proved, we used only its dependency list – the list of lemmas/theorems used for proving that theorem within the Coq formalization. This experiment simulates a scenario of formalizing a proof of a mathematical theorem when it is known what lem-

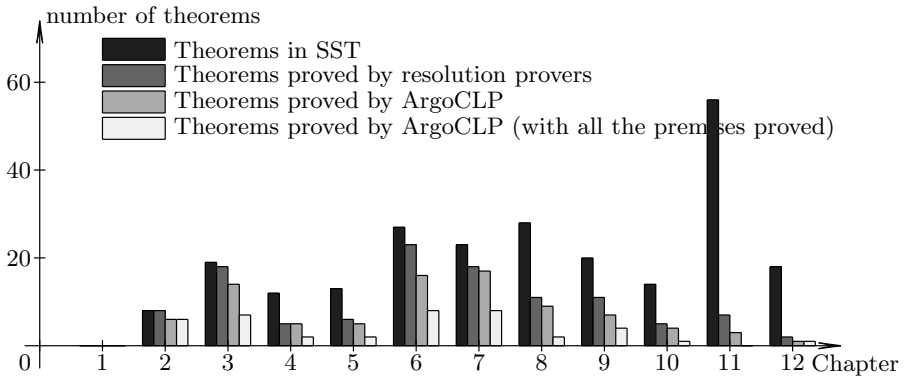

Fig. 3. The number of theorems proved by our framework

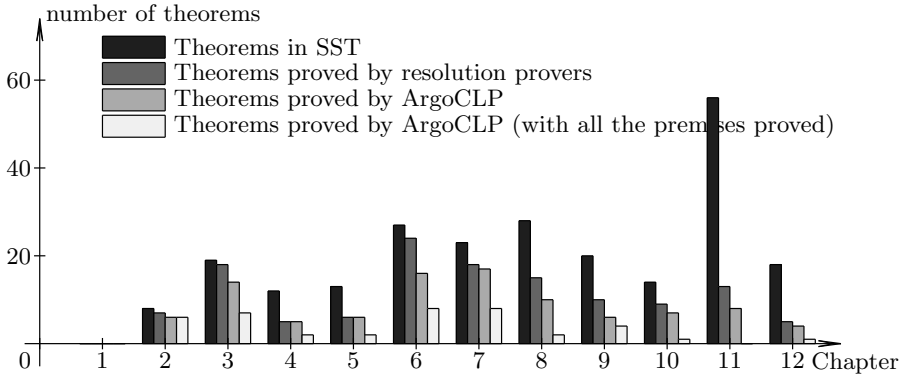

Fig. 4. The number of theorems proved using the list of auxiliary lemmas from the Coq development

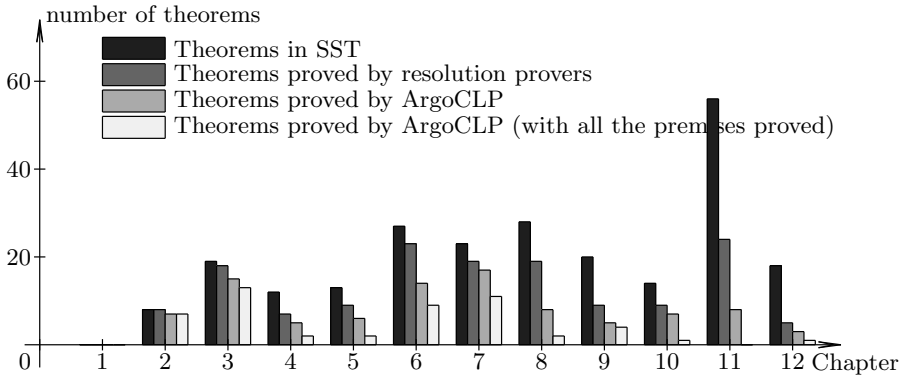

Fig. 5. The number of theorems proved using the list of auxiliary lemmas and the dependency lists from the Coq development

mas are (directly) relevant, but still the task of constructing a formal, machine verifiable proof is non-trivial (for instance, in a textbook there may be only a proof-outline available, or a mathematician may have only a vague picture of a proof and can guess relevant lemmas). In this context, we initially used only definitions and lemmas/theorems given in the dependency lists, and all the axioms including the axioms of the form $\forall \boldsymbol{x}(R(\boldsymbol{x}) \wedge \overline{R}(\boldsymbol{x}) \Rightarrow \bot)$, for all primitive or defined predicate symbols R (if the proving process failed, then we also used all the definitions that precede the conjecture). By this approach, out of 238 theorems, 150 (63%) theorems were proved by (at least one of) the resolution provers, 95 (40%) were then proved by ArgoCLP, and for 52 (22%) of them all used theorems and lemmas were also proved by ArgoCLP. Concerning lemmas taken from the Coq development, the performance is similar for all 218 lemmas: 147/76/38, giving the following totals for 456 lemmas/theorems: 297/171/90. Detailed information (by chapters) is given in Table 1 and in Figure 5.

Discussion. As we expected, the presented results show that automated theorem provers can prove a significant portion of goals within a formalization task (out of the three used resolution provers, Vampire had the best performance). Also, inserting intermediate lemmas (from formalization developed by a human) helps the automated provers which is not surprising. It is also not surprising that giving the dependency lists (from the Coq formalization) to the resolution provers significantly increases their performance. However, we were surprised that ArgoCLP can prove less theorems (95) when provided the dependencies from the Coq proofs (via the resolution provers) than the dependencies discovered by a resolution prover (99). Looking deeper into the data reveals that for some theorems it happens that the dependencies from the Coq proofs contain more lemmas than proofs discovered by the resolution provers. This shows that resolution provers can be used to find shorter, alternative proofs, that can simplify some of the interactive proofs.

Concerning the time spent, the experiment that spent the least overall time is the third one. This was expected, as the provers were given short lists of premises. This performance shows potential for usability of this experiment, when it is applicable.

Concerning the difficulty of the proved theorems, again as expected, our framework performed better on simple theorems (i.e., on theorems with shorted Coq proofs). Figure 6 shows the percentage of theorems proved by ArgoCLP, using the list of auxiliary lemmas against the lengths of corresponding Coq proofs.

Above we presented numbers of proved theorems with and without all premises proved. Although it might seem that only the latter ones are relevant, we believe that the former ones are also (or even more so) practically important and better estimate the needed amount of human's work in real-world tasks. Namely, these numbers reflect a realistic scenario in which a human, in a formalization project, uses synergy with interactive and automated theorem proving. The human could try, within a proof assistant, to prove conjectures sequentually, one by one, and

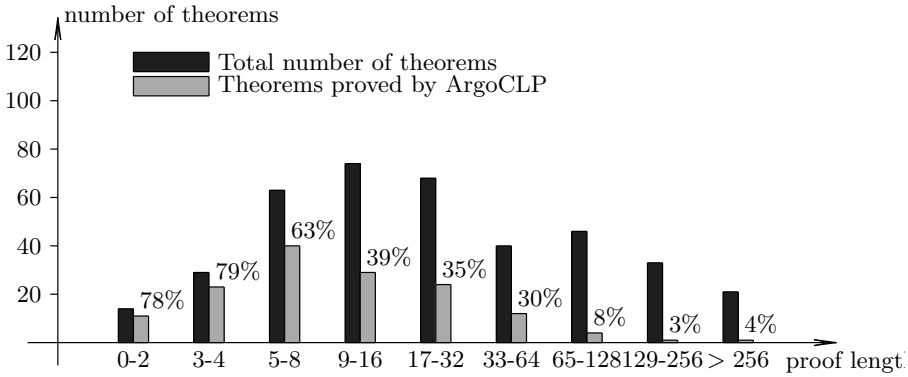

Fig. 6. The number of theorems proved by ArgoCLP and using the list of auxiliary lemmas against the lengths of corresponding Coq proofs.

he would first try to obtain a formal proof automatically, using all previous axioms and theorems. If that failed, he would have to prove the theorem himself (and for the next conjectures he would have all premises already proven).

5 Related Work

Our proving framework is closely related to the Sledgehammer approach [8, 7], but Sledgehammer does not aim at producing readable proofs, it only suggests the axioms and theorems to be used. Sledgehammer can use additional tools that can redirect refutation proofs constructed by resolution provers [4], while our framework can be used for producing proofs for a range of proof assistants and also for producing proofs in readable, natural language form.

Closely related to the spirit of our study is a study of Kaliszyk and Urban [18]. They explored how many theorems an automated theorem provers (for first-order logic, for higher-order logic, and for SMT) can handle (in conjunction with the HOL Light theorem prover) out of 14185 theorems appearing within the Flyspeck project. They used a number of automated theorem provers within their study, with Vampire, E, and Z3 having the most important role. In the first approach, the dependencies from the formalization (lists of theorem/lemmas used within the existing formal proofs) were used and only the relevant theorems/lemmas were passed to the automated theorem provers. The success rate in this approach was 43.2% (for the time limit of 900s) and it serves as an upper limit for what one might expect from fully automated approach. In another approach, all theorems that precede (in chronological order) the current conjecture are considered, but only some of them would be selected by a set of special heuristics (for *premise selection*) and passed to automated theorem provers. With this approach, the success rate was 39.5% (with 192-fold parallelization and time-limit equal 30s, on a 48-core server with AMD Opteron 6174 2.2 GHz CPUs, 320 GB RAM, and 0.5M B L2 cache per CPU). The same authors also work on informal readable

proofs [32] and linking them with formal proofs. Their case study are the proofs from the Flyspeck project [16]. They also provide a tool for visualizing the proof steps dynamically from a tactic based formal proof (either from Isabelle or Coq).

Ganesalingam and Gowers work on automated generation of readable proofs [13], and they propose inference rules which are very similar to our coherent logic based proof system.

6 Conclusions and Future Work

In this paper we promote the use of state-of-the-art automated theorem provers in formalization tasks and within a wider framework that combines different sorts of provers. For one classical twentieth century mathematical book, it turned out that for 37% theorems, readable and machine verifiable proofs can be generated completely automatically. This shows that automated theorem proving can provide significant help in formalization tasks within proof assistants (of course, typically in proving less difficult theorems).

Apart from success rates presented above and a generally optimistic message, during our case study we learned some additional lessons:

- automated theorem proving can help in detecting inconsistency (involving axioms, definitions, and unproved conjectures) within a computer development of a mathematical theory.
- automated theorem proving can help in merging available pieces of mathematical knowledge formalized within the same or different settings.

One of the products (available also from `http://argo.matf.bg.ac.rs`) of our work is a TPTP representation of SST, and a document with all axioms, definitions, theorems, and generated proofs filled-in, given in a natural language form. This material can also serve for training of machine learning techniques for choosing relevant axioms/definitions/lemmas for premises, in the spirit of techniques used within the Flyspeck project [18].

The existing developments of Tarski's geometry can be considered interactive or semi-interactive. For future work we are planning to further explore potentials for full automation, for example to use information from resolution provers more deeply and guide our coherent prover more precisely. This way we should cover at least some of the theorems proved by the resolution provers and not by ArgoCLP. In particular, we would focus on the axioms $\forall \boldsymbol{x}(R(\boldsymbol{x}) \vee \overline{R}(\boldsymbol{x}))$ that can heavily extend the search space. To further improve readability, we will consider using function symbols and work on extending ArgoCPL to deal with function symbols. We will explore ways for using our framework for simplifying existing proofs within proof assistants (by using dependencies from automated theorem provers). In particular, we will use the presented framework for formalizing other classical mathematical sources and we will explore if our framework sometimes gives simpler proofs than those available (so, some fragments of the formalized knowledge can be simplified). We will also try to use our framework to tackle theorems from SST that require using the continuity axiom (i.e., the schema of axioms).

Acknowledgements. We are grateful to Stephan Schulz for his help and comments about using the prover E, and to Filip Marić for his advice about Isabelle proofs and his feedback on earlier versions of this paper. We are also grateful to the anonymous reviewers for a very helpful feedback on earlier versions of this paper.

References

1. Avigad, J., Dean, E., Mumma, J.: A formal system for Euclid's Elements. Rev. Symb. Log. 2(4), 700–768 (2009)
2. Beeson, M.: Proof and computation in geometry. In: Automated Deduction in Geometry – ADG 2012, volume 7993 of Lecture Notes in Computer Science, pp 1–30. Springer (2013)
3. Beeson, M., Wos, L.: OTTER proofs in Tarskian geometry. In: Automated Reasoning - 7th International Joint Conference, IJCAR 2014, volume 8562 of Lecture Notes in Computer Science, pp. 495–510. Springer (2014)
4. Blanchette, J.C.: Redirecting proofs by contradiction. In: Third International Workshop on Proof Exchange for Theorem Proving, PxTP 2013, Lake Placid, NY, USA, volume 14 of EPiC Series, pp. 11–26. EasyChair (2013)
5. Bezem, M., Coquand, T.: Automating coherent logic. In: 12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2005, volume 3835 of Lecture Notes in Computer Science. Springer-Verlag (2005)
6. Bezem, M., Hendriks, D.: On the mechanization of the proof of Hessenberg's theorem in coherent logic. J. Autom. Reason., 40(1) (2008)
7. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. J. Autom. Reason. 51(1), 109–128 (2013)
8. Blanchette, J.C., Bulwahn, L., Nipkow, T.: Automatic proof and disproof in Isabelle/HOL. In: Frontiers of Combining Systems, 8th International Symposium, Proceedings, volume 6989 of Lecture Notes in Computer Science, pp. 12–27. Springer (2011)
9. Boutry, P., Narboux, J., Schreck, P., Braun, G.: A short note about case distinctions in Tarski's geometry. 10th International Workshop on Automated Deduction in Geometry (ADG 2014), pp. 51–66. TR 2014/01, University of Coimbra (2014)
10. Boutry, P., Narboux, J., Schreck, P., Braun, G.: Using small scale automation to improve both accessibility and readability of formal proofs in geometry. 10th International Workshop on Automated Deduction in Geometry (ADG 2014), pp. 31–50. TR 2014/01, University of Coimbra (2014)
11. Braun, G., Narboux, J.: From Tarski to Hilbert. In: Automated Deduction in Geometry – ADG 2012, volume 7993 of Lecture Notes in Computer Science, pp. 89–109. Springer (2013)
12. Fisher, J., Bezem, M.: Skolem machines and geometric logic. In: 4th International Colloquium on Theoretical Aspects of Computing — ICTAC 2007, volume 4711 of Lecture Notes in Computer Science. Springer-Verlag (2007)
13. Ganesalingam, M., Gowers, W.T.: A fully automatic problem solver with human-style output. CoRR, abs/1309.4501 (2013)
14. Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the Odd Order Theorem. In: 4th Conference on Interactive Theorem Proving – ITP 2013, volume 7998 of Lecture Notes in Computer Science, pp. 163–179. Springer (2013)

15. Gupta, H.N.: Contributions to the axiomatic foundations of geometry. PhD thesis, University of California, Berkley (1965)
16. Hales, T.C.: Introduction to the Flyspeck project. In: Mathematics, Algorithms, Proofs, volume 05021 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
17. Hilbert, D.: Grundlagen der Geometrie. Leipzig (1899)
18. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. CoRR, abs/1211.7012 (2012)
19. Makarios, T.: A further simplification of Tarski's axioms of geometry. CoRR, abs/1306.0066 (2013)
20. Meikle, L., Fleuriot, J.: Formalizing Hilbert's Grundlagen in Isabelle/Isar. In: Theorem Proving in Higher Order Logics, pp. 319–334 (2003)
21. Meikle, L., Fleuriot, J.: Mechanical theorem proving in computation geometry. In: Automated Deduction in Geometry – ADG 04, volume 3763 of Lecture Notes in Computer Science, pp. 1–18. Springer-Verlag (2005)
22. Narboux, J.: Mechanical theorem proving in Tarski's geometry. In: Proceedings of Automatic Deduction in Geometry 06, volume 4869 of Lecture Notes in Artificial Intelligence, pp. 139–156. Springer-Verlag (2007)
23. Polonsky, A.: Proofs, Types and Lambda Calculus. PhD thesis, University of Bergen (2011)
24. Quaife, A.: Automated development of Tarski's geometry. J. Autom. Reason. 5(1),97–118 (1989)
25. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Communications, 15(2-3), 91–110 (2002)
26. Robinson, J.A.: A machine oriented logic based on the resolution principle. J. ACM, 12, 23–41 (1965)
27. Schulz, S.: E - a brainiac theorem prover. AI Commun., 15(2-3):111–126 (2002)
28. Schwabhuser, W., Szmielew, W., Tarski, A.: Metamathematische Methoden in der Geometrie. Springer-Verlag, Berlin (1983)
29. Stojanović, S., Narboux, J., Bezem, M., Janičić, P.: A vernacular for coherent logic. In: Conferences on Intelligent Computer Mathematics, volume 8543 of Lecture Notes in Computer Science, pp. 388–403. Springer (2014)
30. Stojanović, S., Pavlović, V., Janičić, P.: A coherent logic based geometry theorem prover capable of producing formal and readable proofs. In: Automated Deduction in Geometry, volume 6877 of Lecture Notes in Computer Science. Springer (2011)
31. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. J. Autom. Reason. 43(4), 337–362 (2009)
32. Tankink, C., Kaliszyk, C., Urban, J., Geuvers, H.: Communicating formal proofs: The case of Flyspeck. In: Interactive Theorem Proving - 4th International Conference, Proceedings, volume 7998 of Lecture Notes in Computer Science, pp. 451–456. Springer (2013)
33. Tarski, A.: What is elementary geometry? In: P. Suppes L. Henkin and A. Tarski (eds.) The axiomatic Method, with special reference to Geometry and Physics, pp. 16–29, Amsterdam, North-Holland (1959)
34. Tarski, A., Givant, S.: Tarski's system of geometry. Bull. Symb. Log. 5(2) (1999)
35. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: Spass version 3.5. In: Automated Deduction - CADE-22 Proceedings, volume 5663 of Lecture Notes in Computer Science, pp. 140–145. Springer (2009)

36. Wenzel, M.: Isar - a generic interpretative approach to readable formal proof documents. In: Theorem Proving in Higher Order Logics (TPHOLs'99), volume 1690 of Lecture Notes in Computer Science, pp. 167–184. Springer (1999)
37. Wiedijk, F. (ed.): The seventeen provers of the World volume 3600 of Lecture Notes in Computer Science. Springer (2006)