Automated Completion of Statements and Proofs in Synthetic Geometry: an Approach based on Constraint Solving

Salwa Tabet Gonzalez, Predrag Janičić, Julien Narboux

Automated Completion of Statements and Proofs in Synthetic Geometry: an Approach based on Constraint Solving

Salwa Tabet Gonzalez, Predrag Janičić, and Julien Narboux

[1] UMR 7357 CNRS, University of Strasbourg
tabetgonzalez@unistra.fr
ORCID: 0009-0008-5874-9440
[2] Department for Computer Science, Faculty of Mathematics, University of Belgrade
janicic@matf.bg.ac.rs
ORCID: 0000-0001-8922-4948
[3] UMR 7357 CNRS, University of Strasbourg
narboux@unistra.fr
ORCID: 0000-0003-3527-7184

Abstract. Conjecturing and theorem proving are activities at the center of mathematical practice and are difficult to separate. In this paper, we propose a framework for completing incomplete conjectures and incomplete proofs. The framework can turn a conjecture with missing assumptions and with an under-specified goal into a proper theorem. Also, the proposed framework can help in completing a proof sketch into a human-readable and machine-checkable proof. Our approach is focused on synthetic geometry, and uses coherent logic and constraint solving. The proposed approach is uniform for all three kinds of tasks, flexible and, to our knowledge, unique such approach.

Keywords: synthetic geometry, automated deduction, proofs, constraint solving, abducts

1 Introduction

Automated theorem provers take as input the formal statement of a conjecture in a theory described by axioms and lemmas, and try to generate a proof or a counter-example for this conjecture. In the field of geometry, several efficient automated theorem proving approaches have been developed, including algebraic ones such as Wu's method, Gröbner bases method, and semi-synthetic methods such as the area method. In these approaches, typically, the conjecture and the axioms being considered are fixed. However, in mathematical practice, in the context of education and also in mathematical research, the conjecturing and proving activities are not separated but interleaved. The practitioner may try to prove a statement which is valid only assuming some implicit or unknown assumptions, while the list of lemmas and theorem which can be used may not be complete. In education, for some kind of exercises, a precise formulation of

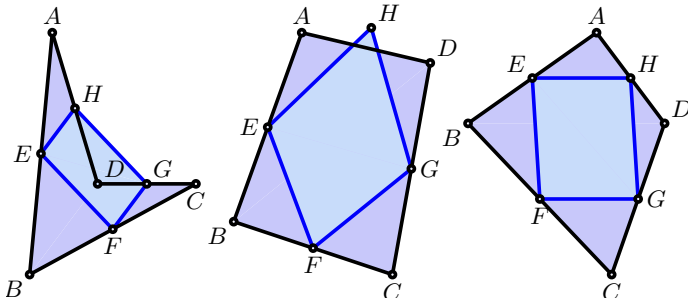

Fig. 1: Illustrations for five problems related to Varignon's theorem, respectively: Problem 1; Problem 2; Problem 3.

the statement to be proved is also left to the student, with questions such as: "What is the nature of the quadrilateral $ABCD$?". Hence, the conjecture can contain unknown assumptions called *abducts*, and the goal may be not completely specified. One may also ask for a proof using a particular theorem or an intermediate fact, i.e., a proof partially specified using constraints specifying some proof steps.

In this paper, we consider the problems of (simultaneously) completing (a) the assumptions of the conjecture; (b) the goal of the conjecture; (c) a proof sketch for the conjecture. The completion process should lead to a proof that is both machine-checkable and human-readable. Because we aim at producing intelligible and readable proofs, with a similar level of granularity as paper-and-pencil proofs, our approach is logic-based, uses a fragment of first-order logic called coherent logic, and is focused on synthetic geometry (in contrast to algebraic methods). Our approach for dealing with partial conjectures and partial proofs is implemented as an extension of the automated theorem prover Larus developed previously [14] . The approach is uniform for all three kinds of completion tasks, flexible and, to our knowledge, unique such approach.

We list five high-school level synthetic geometry problems related to Varignon's theorem (Fig. 1), that we will try to solve using our approach.

Problem 1 (Fully specified statement) Consider a quadrilateral $ABCD$, let E, F, G and H be the midpoints of AB, BC, CD, DA respectively. Prove that the quadrilateral $EFGH$ is a parallelogram (assuming that there are no two sides that are aligned).

Problem 2 (First inverse problem) Consider a quadrilateral $ABCD$, let E, F, and G be the midpoints of AB, BC and CD respectively. Let H be a point. Under which assumption is the quadrilateral $EFGH$ a parallelogram?

Problem 3 (Second inverse problem) Consider a quadrilateral $ABCD$, let E, F, G and H be the midpoints of AB, BC, CD, DA respectively. Under which assumption is the quadrilateral $EFGH$ a rectangle?

Problem 4 (Partially specified goal) Consider a quadrilateral $ABCD$, let E, F, G and H be the midpoints of AB, BC, CD, DA respectively. What is the nature of the quadrilateral $EFGH$?

Problem 5 (Partially specified proof) Consider a quadrilateral $ABCD$, let E, F, G and H be the midpoints of AB, BC, CD, DA respectively. We have that $EG = FH$. Prove that $EFGH$ is a rectangle using the axiom "If the diagonals of a parallelogram are congruent, then it's a rectangle".

The above examples are inspired by exercises given in a teacher training session. A more detailed discussion about how these examples can be used in a didactic context, issues related to the formalization can be found in [11, 19]

2 Background

This section provides some necessary background information on a fragment of first-order logic called coherent logic that our approach uses. There are several automated provers for coherent logic, including Larus, which is based on "theorem proving as constraint solver" paradigm.

2.1 Coherent Logic

A formula of first-order logic is said to be *coherent* if it has the following form:

$$A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x}, \vec{y}) \vee \ldots \vee \ B_{m-1}(\vec{x}, \vec{y}))$$

where universal closure is assumed, and where $\vec{x}$ denotes a sequence of variables $x_0, x_1, \ldots, x_{k-1}$; A_i (for $0 \leq i \leq n-1$) denotes an atomic formula (involving zero or more variables from $\vec{x}$); $\vec{y}$ denotes a sequence of variables $y_0, y_1, \ldots, y_{l-1}$; B_j (for $0 \leq j \leq m-1$) denotes a conjunction of atomic formulae (involving zero or more of the variables from $\vec{x}$ and $\vec{y}$) [14]. If there are no formulae A_i, then the left-hand side of the implication is assumed to be $\top$. If there are no formulae B_j, then the right-hand side of the implication is assumed to be $\bot$. There are no function symbols with arity greater than zero. Coherent formulae do not involve the negation connective. A coherent theory is a set of sentences, axiomatized by coherent formulae, and closed under derivability. A number of theories and theorems can be formulated directly and simply in coherent logic (CL). In addition, any first-order theory can be translated into CL, possibly with additional predicate symbols [12, 21]. Synthetic geometry can be expressed easily using CL. For example, the central part of axioms system of Euclid (as formalized by Beeson et. al. [3]), or Hilbert (as formalized by Braun et. al [6]), or Tarski [26] can be expressed in first-order logic without function symbols, and the axioms are mostly in CL form.

Translation of FOL formulae into CL involves elimination of the negation connectives: negations can be kept in place and new predicates symbols for corresponding sub-formula have to be introduced, or negations can be pushed down to atomic formulae [21]. In the latter case, 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 introduced: $\forall\vec{x}(R(\vec{x})\wedge\overline{R}(\vec{x})\Rightarrow\bot)$, $\forall\vec{x}(R(\vec{x})\vee\overline{R}(\vec{x}))$.

In contrast to resolution-based theorem proving, in forward reasoning for CL, the conjecture being proved is kept unchanged and proved without using refutation, Skolemization and clausal form. Thanks to this, CL is suitable for producing human-readable synthetic proofs and also machine verifiable proofs [4, 12]. The problem of provability in CL is semi-decidable. CL admits a simple proof system, a sequent-based variant is as follows [27]:

$$\frac{\Gamma, ax, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}), \underline{B_0(\vec{a},\vec{b})\vee\ldots\vee B_{m-1}(\vec{a},\vec{b})} \vdash P}{\Gamma, ax, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash P}\ \text{MP}$$

$$\frac{\Gamma, \underline{B_0(\vec{c})} \vdash P \quad \ldots \quad \Gamma, \underline{B_{m-1}(\vec{c})} \vdash P}{\Gamma, B_0(\vec{c})\vee\ldots\vee B_{m-1}(\vec{c}) \vdash P}\ \text{QEDcs (case split)}$$

$$\frac{}{\Gamma, \underline{B_i(\vec{a},\vec{b})} \vdash \exists\vec{y}(B_0(\vec{a},\vec{y})\vee\ldots\vee\ B_{m-1}(\vec{a},\vec{y}))}\ \text{QEDas (assumption)}$$

$$\frac{}{\Gamma, \bot \vdash P}\ \text{QEDefq (ex falso quodlibet)}$$

In the rules given above, it is assumed: ax is a formula $A_0(\vec{x})\wedge\ldots\wedge A_{n-1}(\vec{x})\Rightarrow\exists\vec{y}(B_0(\vec{x},\vec{y})\vee\ldots\vee\ B_{m-1}(\vec{x},\vec{y}))$;[4] $\vec{a}$, $\vec{b}$, $\vec{c}$ denote sequences of constants (possibly of length zero); in the rule MP (*extended modus ponens*), $\vec{b}$ are fresh constants; $\vec{x}$ and $\vec{y}$ denote sequences of variables (possibly of length zero); $A_i(\vec{x})$ (respectively $B_i(\vec{x},\vec{y})$) have no free variables other than from $\vec{x}$ (respectively $\vec{x}$ and $\vec{y}$); $A_i(\vec{a})$ are ground atomic formulae; $B_i(\vec{a},\vec{b})$ and $B_i(\vec{c})$ are conjunctions of ground atomic formulae; $\underline{\Phi}$ denotes the list of conjuncts in Φ if Φ is conjunction, and otherwise Φ itself. In the proving process, the rules are read from bottom to top, i.e., by a rule application one gets the contents (new sub-goals) above the line.

For a set of coherent axioms AX and the statement $A_0(\vec{x})\wedge\ldots\wedge A_{n-1}(\vec{x})\Rightarrow\exists\vec{y}(B_0(\vec{x},\vec{y})\vee\ldots\vee\ B_{m-1}(\vec{x},\vec{y}))$ to be proved, within the above proof system one has to derive the following sequent (where $\vec{a}$ denotes a sequence of new symbols of constants): $AX, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash \exists\vec{y}(B_0(\vec{a},\vec{y})\vee\ldots\vee\ B_{m-1}(\vec{a},\vec{y}))$.

Notice that, in the above proof system, case split may occur only at the end of a (sub)proof. However, it is not a substantial restriction: any proof with unrestricted use of case split can be transformed to such form.

[4] Notice the hidden link between the formulae $B_i(\vec{a},\vec{b})$ from the rule MP and the formula ax: the formulae $B_i(\vec{a},\vec{b})$ from the rule are instances of the formulae $B_i(\vec{x},\vec{y})$ from ax.

2.2 Theorem Proving as Constraint Solving and the Larus System

"Theorem proving as constraint solving" is a paradigm for automated theorem proving recently proposed [14]. In contrast to common automated theorem proving approaches, in which the search space is a set of some formulae and what is sought is again a (goal) formula, this new approach is based on searching for a proof (of a given length) as a whole. Namely, a proof of a formula in a fixed logical setting can be encoded as a sequence of natural numbers obeying some constraints. A suitable solver can find such a sequence and from that sequence a sought proof can be reconstructed. This approach is implemented in C++, within an open-source prover Larus,[5] specialized in proofs in coherent logic and using SAT, SMT, and CSP solvers for solving sets of constraints. Larus can generate readable, human understandable proofs in natural language and also machine-verifiable proofs for the interactive provers Coq, Isabelle, and Mizar.

Each CL proof consists of several proof steps, while each of them has one of the following kinds (with obvious meaning): Assumption, MP, FirstCase, SecondCase, QEDbyCases, QEDbyAssumption, QEDbyEFQ. The information relevant for MP steps include: `AxiomApplied`, `From` (the ordinal numbers of proof steps justifying premises of the axiom applied), `Instantiation` (of the variables in the axiom), `Contents` (the atoms in formula in the proof step), etc. `Nesting` denotes the nesting of the proof step (the nesting of the first step is 1).

The proof can be represented by a sequence of numbers, meeting some constraints (that correspond to definitions of inference steps given in Section 2.1). For instance, if the proof step s is of the kind QEDbyEFQ, then the following conditions must hold (given almost in verbatim as in our C++ code):[6]

1. `StepKind` (s) = QEDbyEFQ;
2. $s > 0$;
3. `Contents` $(s-1)(0) = \bot$;
4. `Goal` (s);
5. `Nesting` (s) =`Nesting` $(s-1)$.

The above conditions can be understood in the following way: if there is a proof of the given conjecture, the proof step s in that proof is of the kind QEDbyEFQ iff the natural number `StepKind` (s) equals the unique code for QEDbyEFQ, $s > 0$ (since there must be a previous step), the contents of the previous proof step is $\bot$, the contents of the step is the goal itself, and the nesting of the steps $s-1$ and s is the same.
Each proof step has one of the listed kinds and meet corresponding conditions. There are also some additional, global constraints, like that the last proof step has `Nesting` equal 1.

Larus works in the following way. If there is a set of axioms, a conjecture, and a proof length, a corresponding proof can be represented as a sequence of

[5] `https://github.com/janicicpredrag/Larus`
[6] The corresponding C++ implementation is an improved version of the implementation presented earlier [14].

natural numbers, still unknown, so they will be represented by variables V. The constraints that have to be met for each proof step and for the proof as a whole can be expressed in terms of these variables V. If a solver can find a model for the constraint, from it the proof in logical terms can be reconstructed. All constraints involved are linear constraints over natural numbers. Since linear arithmetic is decidable, decision procedures for it can decide, for each input constrains, whether or not it has a model. For this purpose, Larus can use SAT, SMT, and CSP solvers. For input, Larus uses axioms and conjectures stored in a file in TPTP/fof format.

3 Abducts and Completing Assumptions

There are three major types of logical inference: induction, deduction, and abduction. The concept of abduction has been introduced by Peirce [20]. In deduction, everything inferred is necessarily true, while it is not the case with the remaining two types of inference. Induction tries to infer general rules based on individual instances. The aim of abduction is to produce additional hypotheses to explain observed facts. Abduction has a wide spectrum of implicit or explicit applications – in everyday life, in education, and in scientific reasoning, including in building mathematical theories, or in software verification. One definition of abduct is given below.

Definition 1. *Given a theory T and a formula G (the goal to be proved), such that $T \not\models G$, an* explanations *or* abduct *is a formula A meeting conditions: $T, A \models G$ and $T, A \not\models \bot$.*

It is clear that some abducts are not *interesting*, so there are often some additional restrictions given. There is no general agreement about such restrictions, but two types are most usual: *syntactical restrictions* (abducts should be of a specific syntactical form) and *minimality restrictions* (for any other abduct A', if $T, A \models A'$ then $A \equiv A'$). It is reasonable to ask that A is not G, as it is trivial. Some authors also add stronger a restriction that $A \not\models G$ (i.e., at least one axiom of T has to be used to prove G).

Approaches for Computing Abducts. Various algorithms to produce different kind of abducts have been developed [1]. In *abductive logic programming*, techniques for abductive reasoning are developed in the context of logic programming. Rules are considered to be Horn clauses [8]. According to Russo et al. [25], some systems assume that predicate symbols appearing in abducts do not appear in the conclusion of any rule and that negation does not appear in the conclusion of any rule. This restriction is not realistic in the context of geometry. In our example, we want to accept geometric predicate symbols both in abducts, and in the assumptions and conclusion of theorems. Some approaches are based on Robinson's resolution algorithm, extended such that when no more clauses can be produced, the atomic clauses are considered as a potential abduct and consistency if checked [17]. There are also approaches developed for the context of SMT solving, dealing with decidable theories like linear arithmetic [10, 23]

In the context of geometry, some algebraic algorithms can generate additional assumptions for the statement to be true. For example, Wu's method [28] can produce non-degeneracy conditions. Algebraic methods can also be used to generate more general abducts [22]. These methods are more efficient than ours, but more specific so cannot be used for arbitrary geometric theories. Also, they cannot generate readable proofs. Moreover, expressing algebraic non-degeneracy conditions in simple geometrical terms is not easy and not always possible [7].

Abduction in Synthetic Euclidean Geometry. In this paper, the theory T from Definition 1 is a synthetic Euclidean geometry. In this context, automated finding of proofs allowing abducts may have several applications. For instance, an automated system may help a student or a researcher who tries to prove (or formalize) a theorem with a missing assumption. Barbosa et al. have proposed such goal in the context of interactive proof assistants where conjectures are sent to an SMT solver [2].

Non-degeneracy conditions are often overlooked and missing in informal geometry statements. Abductive reasoning is also a task which can be asked explicitly to students. The answer expected by the teacher for Problem 2 is that H should be the midpoint of AD.

Finding Abducts using Larus. In this paper, we restrict consideration of abduction only to coherent logic and only to abducts that are conjunctions of ground atomic formulae. Larus was not implemented with abduction in mind, yet implementation of support for abduction turned out to be very simple, almost trivial, and took less than 100 lines of C++ code. In order to find abducts using Larus, we treat them as a special case of proof steps, in the main proof branch, just after assumptions. We have to add constraints on what such an abduct can be:

1. the abduct is treated as an assumption;
2. the nesting of the abduct equals 1;
3. the abduct is an atomic formula (no branching);
4. the predicate symbol is one of the predicate symbols in the signature;
5. the arguments are among existing symbols of constants;
6. the abduct is not the goal itself;
7. the abduct is not $\bot$.

The given conditions may be written in the following way, assuming that the abduct is placed in i-th step of the proof:

1. `StepKind` $(i) =$ASSUMPTION
2. `Nesting` $(i) = 1$
3. `Cases` $(i) = \mathit{false}$
4. `ContentsPredicate` $(i, 0) < \mathit{sizeof}(\mathit{Signature})$
5. for each argument j (up to maximal arity): `ContentsArgument` $(i, 0, j) < \mathit{sizeof}(\mathit{Constants})$
6. `Goal` $(i) = \mathit{false}$
7. `ContentsPredicate` $(i, 0) \neq \bot$

One can also choose a number of abducts, each leading to the constraints given above. With such additional constraints for each abduct (for additional proof steps in specific positions in a proof sought), with a given set of axioms and a conjecture, and with a concrete proof length, we run Larus as usual. The solving/proving process is the same as without abducts: the constraint solver finds a way to specify a full proof, including the abducts, i.e., under-specified assumptions.

In the above list of conditions, the last two do not follow the basic definition of abduct. Like in some other variants of the definition, the abduct may not be equal to the goal atom because such abducts are trivial. Also, the abduct may not be equal to $\bot$, since it is inconsistent. It is important to discard other inconsistent abducts early, so we add one more restriction: the proof of $T, A \models G$ should not end with QEDbyEFQ. Some constructed abducts may still be inconsistent with other assumptions, and we use an external, more efficient automatic theorem prover, Vampire [16], to discard such abducts.

Example 1. For the first inverse problem (Problem 2 from Section 1), Larus produces two consistent, symmetric abducts (the proof obtained with the first abduct is presented in Appendix 7.2):

- "H is the midpoint of AD"
- "H is the midpoint of DA"

Example 2. For the second inverse problem (Problem 3 from Section 1), Larus produces more than 150 consistent abducts, most of which give degenerate cases, hence are less interesting. Apart from such abducts, we obtained the following abducts and their symmetric variants (the proof with the first abduct is presented in Appendix 7.3):

- "the diagonals HF and EG are congruent"
- "$\angle FGH$ is a right angle"
- "$\angle EHG$ is a right angle"
- "$\angle HEF$ is a right angle"
- "$\angle EFG$ is a right angle"

4 Deducts and Completing Goals

Non-trivial first-order logic theories have infinite number of theorems. Approaches based on refutation cannot be used with under-specified goals and, hence, cannot be used for completing them. In principle, a controlled forward-reasoning (for instance, based on some kind of breadth-first search) can enumerate all theorems of a theory. However, such a systematic approach can be hardly useful for some practical applications, like looking for possible conjectures of a specific form. Our framework allows (but does not require) specifying partially the form of the goal: for instance, one may specify the dominant predicate symbol in the goal atom, or some of its arguments.

Finding Deducts in Synthetic Euclidean Geometry. In the field of geometry, deduct candidates can also be guessed based on an illustration, giving a concrete model. However, these deduct candidates still have to be verified i.e., proved. Potential deducts could also be listed as large disjunctions of atomic formulae, but this method does not scale when the list of potential deducts is too long.

Finding Deducts using Larus. In Larus, if the goal is given i.e., fully specified, corresponding constraints are added to the full constraint representing a proof sought. Let us assume that the final step of the proof is (some fixed) n and, for simplicity, let us assume that the goal is just a single atom. The corresponding constraint then includes:

1. `Nesting` $(n) = 1$
2. `Cases` $(n) = \mathit{false}$
3. `ContentsPredicate` $(n, 0) =$ the goal predicate symbol
4. for each argument j (except for existentially quantified variables): `ContentsArgument` $(n, 0, j) =$ the argument from the given goal instantiated.

If the goal is under-specified, for instance, if the predicate symbol is not given (it is given as _ in the TPTP file), the third condition is just ignored. The same holds for the arguments.[7] During the solving process, if there is a model, these slots are filled-in by some concrete values, giving a concrete goal. Overall, support for finding under-specified deducts is very simple. The current implementation finds one possible deduct, but it can be extended to list all possible deducts, similarly as for abducts (as explained in Section 3).

Example 3. For Problem 4 from Section 1, Larus produces the deduct "$EF \parallel GH$". The proof obtained with such deduct is presented in Appendix 7.4.

5 Hints and Completing Proofs

Informal proofs, for instance from textbooks, are often partial and incomplete. They may even provide only a part of a full proof, or some instructions like for filling gaps by analogy. Reconstructing proofs using such hints is very important task, as discussed by Gowers and Hales [13]: "One dream was to develop an automated assistant that would function at the level of a helpful graduate student. The senior mathematician would suggest the main lines of the proof, and the automated grad student would fill in the details."

Completing Proofs in Synthetic Euclidean Geometry. In the context of geometry, completing proofs could be interesting either as a way to render the formalization process simpler (automation would bring in all the details that are overlooked in pen and paper proofs), or as a tool working behind the scene for providing guidance for what could be the next step in the proof. This objective has been studied by Richard et al. [24]. In geometry, hints can also be based on some observations from an associated illustration.

[7] Actually, underspecified arguments can be also handled using existential quantification.

Completing Proofs using Larus. Larus can be instructed to look for a proof of a given conjecture (also possibly only partially specified) meeting some conditions (that we call "hints") [14]. Therefore, Larus can try, for instance, to reconstruct a proof given only in outline (like proofs in textbooks). Larus use hints in a much more general way than just splitting the problem into sub-problems: for instance, some hint may be used in just one proof branch and cannot be proved itself. Hints do not have to be ordered (one can ask for a proof using X and Y in no particular order), they can be vague, imposed only by partial constraints ("find a proof that uses this particular predicate symbol", or "find a proof using some specific axiom", without the way it is instantiated, etc.).

Completing proofs in Larus is supported similarly as for abducts and under-specified goals – by modifying the corresponding constraints. The main difference is that abducts and incomplete goals are under-specified, so some constraints have to be omitted, while partial proof introduce additional constraints, on top of the common constraints that must be met by all proof steps. For expressing hints, we slightly extended the language TPTP/fof to allow a simple but still quite expressible semantics. Some kinds of hints (not all) are illustrated below.

```
fof(hintname0, hint, r(_,_), _ , _).
fof(hintname1, hint, r(_,_), 5 , _).
fof(hintname2, hint, _, 5, ax2(_,_)).
```

The first hint specifies that some proof step will have an atom of the form $r(\ldots,\ldots)$. The second hint specifies that the 5th proof step will have atom of the form $r(\ldots,\ldots)$. The third hint specifies that in the 5th proof step the axiom $ax2$ is applied.

Example 4. For Problem 5 from Section 1, Larus was able to find a proof around 20% faster with a suitable hint presented in Appendix 7.5.

6 Conclusion and Future Work

In this paper we have shown how a prover using the "theorem proving as constraint solving" paradigm can be extended such that it can complete partially specified conjectures and partially specified proofs. This extension is simple, and the implementation update is very small. The completion algorithm is uniform, since all three completion tasks (completing assumptions, completing goals, completing proofs) are handled in the same spirit – in terms of adding or deleting some constraints. To our knowledge, this approach is new, and we are not aware of other systems that can address all three sorts of completion tasks. The presented approach is flexible as different variations of completion tasks can be supported. The strength of this approach is also that it can generate both proofs that are human-readable and machine-checkable. The proposed framework has two main limitations. First, in current stage, it can deal only with coherent logic, hence the theories cannot involve function symbols, which excludes geometry proof that use (non-trivial) arithmetic. Second, the framework cannot deal with conjectures whose proofs are long (say, longer than 50 proof steps).

To our knowledge, there is only one other approach in which some kind of proof is encoded, and reconstructed from a model for the corresponding set of constraints – the approach in which rigid connection tableaux are encoded as SAT and SMT instances [9, 5, 18]. However, in this line of research, neither machine verifiable or readable proofs, nor any of completion tasks are considered.

The presented work can be extended in several directions. One of our goals is to use our framework to help transfer geometry knowledge from informal sources to proof assistants and between proof assistants, while keeping its high-level structure. In informal sources, statements of theorems may be incomplete, while proofs may be given just in outline. Still, using our approach such contents can be, at least in some cases, completed and turned into a verifiable form. For transferring knowledge from a proof assistant, one would need to go into its specifics, but only to grab (some) proof steps and make hints out of them. We are still to explore these ideas on a larger scale, like one geometry textbook. In the same spirit as the work proposed by Jiang et al. [15], our approach could be combined with large language models to perform automatic formalization by extracting data from natural language proofs. More specific to abduction, we are planning to make an in-depth comparison (both qualitative and quantitative) of our tool to other tools for generating abducts.

Acknowledgement. The work related to this paper has been partially supported by the European Cost project CA20111 EUROProofNet.

References

1. Aliseda Atocha. *Abductive Reasoning*, volume 330 of *Synthese Library*. Kluwer, 2006.
2. Haniel Barbosa, Chantal Keller, Andrew Reynolds, Arjun Viswanathan, Cesare Tinelli, and Clark Barrett. An Interactive SMT Tactic in Coq using Abductive Reasoning. In *EPiC Series in Computing*, volume 94. EasyChair, 2023.
3. Michael Beeson, Julien Narboux, and Freek Wiedijk. Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, 85(2-4). Springer, 2019.
4. Marc Bezem and Thierry Coquand. Automating Coherent Logic. *12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2005*, volume 3835 of *LNCS*. Springer, 2005.
5. Jeremy Bongio, Cyrus Katrak, Hai Lin, Christopher Lynch, and Ralph Eric McGregor. Encoding First Order Proofs in SMT. *Electronic Notes in Theoretical Computer Science*, 198(2), 2008.
6. Gabriel Braun and Julien Narboux. From Tarski to Hilbert. *Post-proceedings of Automated Deduction in Geometry 2012*, volume 7993 of *LNCS*, Springer, 2012.
7. XueFeng Chen and DingKang Wang. The Projection of Quasi Variety and Its Application on Geometric Theorem Proving and Formula Deduction. *Post-proceedings of Automated Deduction in Geometry 2002*, volume 2930 of *LNCS*, Springer, 2004.
8. Marc Denecker and Antonis C. Kakas. Abduction in Logic Programming. *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part I*, volume 2407 of *LNCS*. Springer, 2002.

9. Todd Deshane, Wenjin Hu, Patty Jablonski, Hai Lin, Christopher Lynch, and Ralph Eric McGregor. Encoding First Order Proofs in SAT. CADE-21, volume 4603 of *LNCS*. Springer, 2007.
10. Isil Dillig and Thomas Dillig. Explain: A Tool for Performing Abductive Inference. CAV-13 - *Computer Aided Verification*, volume 8044 of *LNCS*, Springer, 2013
11. Viviane Durand-Guerrier, Paolo Boero, Nadia Douek, Susanna S. Epp, and Denis Tanguay. Examining the Role of Logic in Teaching Proof. *Proof and Proving in Mathematics Education*, number 15 in New ICMI Study Series. Springer, 2012.
12. Roy Dyckhoff and Sara Negri. Geometrization of first-order logic. *The Bulletin of Symbolic Logic*, 21, 2015.
13. Thomas Hales. An argument for controlled natural languages in mathematics. `https://jiggerwit.files.wordpress.com/2019/06/header.pdf` 2019.
14. Predrag Janičić and Julien Narboux. Theorem Proving as Constraint Solving with Coherent Logic. *Journal of Automated Reasoning*, 66(4), 2022.
15. Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs, February 2023. arXiv:2210.12283 [cs].
16. Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. *Computer Aided Verification - 25th International Conference*, CAV 2013, volume 8044 of *LNCS*. Springer, 2013.
17. P. Marquis. Extending abduction from propositional to first-order logic. *Fundamentals of Artificial Intelligence Research*, volume 535 of *LNCS*. Springer, 1991.
18. Ralph Eric McGregor. *Automated Theorem Proving Using SAT*. PhD Thesis, Clarkson University, 2011.
19. Julien Narboux and Viviane Durand-Guerrier. Combining pencil/paper proofs and formal proofs, a challenge for Artificial Intelligence and mathematics education. In *Mathematics Education in the Age of Artificial Intelligence*. volume 17 of Mathematics Education in the Digital Era, Springer, 2022.
20. Charles Peirce. *Collected papers of Charles Sanders Peirce*. Belknap Press, 1932.
21. Andrew Polonsky. *Proofs, Types and Lambda Calculus*. PhD thesis, University of Bergen, 2011.
22. T. Recio and M. P. Vélez. Automatic Discovery of Theorems in Elementary Geometry. *Journal of Automated Reasoning*, 23(1), 1999.
23. Andrew Reynolds, Haniel Barbosa, Daniel Larraz, and Cesare Tinelli. Scalable Algorithms for Abduction via Enumerative Syntax-Guided Synthesis. *Automated Reasoning - IJCAR 2020*, volume 12166 of *LNCS*. Springer, 2020.
24. Philippe R. Richard, Josep Maria Fortuny, Michel Gagnon, Nicolas Leduc, Eloi Puertas, and Michèle Tessier-Baillargeon. Didactic and theoretical-based perspectives in the experimental development of an intelligent tutorial system for the learning of geometry. *ZDM*, 43(3), 2011.
25. Alessandra Russo and Bashar Nuseibeh. On The Use Of Logical Abduction In Software Engineering. In Handbook of Software Engineering and Knowledge Engineering. World Scientific, 2001.
26. Wolfram Schwabhäuser, Wanda Szmielew, and Alfred Tarski. *Metamathematische Methoden in der Geometrie*. Springer, Berlin, 1983.
27. Sana Stojanović, Julien Narboux, Marc Bezem, and Predrag Janičić. A Vernacular for Coherent Logic. *Intelligent Computer Mathematics*, volume 8543 of *LNCS*. Springer, 2014.
28. Wen-Tsün Wu. On the Decision Problem and the Mechanization of Theorem-Proving in Elementary Geometry. Scientia Sinica 21(2), 1978.

7 Appendix

In this appendix, we provide a complete list of lemmas and axioms (in coherent logic form) used in our examples, and the results obtained using Larus. The results were obtained on a PC computer with Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz processor running under Linux (the time spent should give just a general picture of the efficiency of the system).

7.1 Problem 1: Varignon's Theorem

The TPTP file used for Problem 1 is the following:

```
fof(triangle_mid_par_strict, axiom, (! [A, B, C, P, Q] : ( ((~ col
    (A,B,C)) & midpoint(B,P,C) & midpoint(A,Q,C)) => par(A,B,Q,P))
    )).
fof(lemma_par_trans, axiom, (! [A, B, C, D, E, F] : ((par(A,B,C,D)
     & par(C,D,E,F) & (~col(A,B,E))) => par(A,B,E,F)))).
fof(defparallelogram2,axiom, (! [A,B,C,D] : ((par(A,B,C,D) & par(A
    ,D,B,C)) => ((pG(A,B,C,D)))))).
fof(lemma_parallelNC,axiom, (! [A,B,C,D] : ((par(A,B,C,D)) => ((~
    (col(A,B,C)) & ~ (col(A,C,D)) & ~ (col(B,C,D)) & ~ (col(A,B,D)
    ))))).
fof(lemma_parallelflip,axiom, (! [A,B,C,D] : ((par(A,B,C,D)) => ((
    par(B,A,C,D) & par(A,B,D,C) & par(B,A,D,C)))))).
fof(lemma_parallelsymmetric,axiom, (! [A,B,C,D] : ((par(A,B,C,D))
    => ((par(C,D,A,B)))))).
fof(midpoint_sym, axiom, (! [A, B, I] : (midpoint(A,I,B) =>
    midpoint(B,I,A)))).
fof(lemma_tP_trans, axiom,  (! [A, B, C, D, E, F] : ((tP(A,B,C,D)
    & tP(C,D,E,F)) => tP(A,B,E,F)))).

fof(th_varignon,conjecture,(! [A,B,C,D,E,F,G,H] : (( (~(col(B,D,A)
    )) & (~(col(B,D,C))) & (~(col(A,C,B))) & (~(col(A,C,D))) & (~
    (col(E,F,G))) & midpoint(A,E,B) & midpoint(B,F,C) & midpoint(C
    ,G,D) & midpoint(A,H,D)) => pG(E,F,G,H) ))).
```

If Larus is invoked as: `./larus -l100 -m8` (`-l100` means the time limit is 100s, `-m8` means that we look for a proof with 8 or fewer steps), it produces the following human-readable proof in 2.00 seconds:

Consider arbitrary a, b, c, d, e, f, g, h such that:

- $\neg col(b, d, a)$,
- $\neg col(b, d, c)$,
- $\neg col(a, c, b)$,
- $\neg col(a, c, d)$,
- $\neg col(e, f, g)$,

- $b \neq d$,
- $a \neq c$,
- $midpoint(a, e, b)$,
- $midpoint(b, f, c)$,
- $midpoint(c, g, d)$,
- $midpoint(a, h, d)$.

It should be proved that $pG(e, f, g, h)$.

1. $par(a, c, h, g)$ (by MP, from $\neg col(a, c, d)$, $midpoint(c, g, d)$, $midpoint(a, h, d)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto d$, $P \mapsto g$, $Q \mapsto h$)
2. $par(b, d, f, g)$ (by MP, from $\neg col(b, d, c)$, $midpoint(c, g, d)$, $midpoint(b, f, c)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto b$, $B \mapsto d$, $C \mapsto c$, $P \mapsto g$, $Q \mapsto f$)
3. $par(a, c, e, f)$ (by MP, from $\neg col(a, c, b)$, $midpoint(b, f, c)$, $midpoint(a, e, b)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto b$, $P \mapsto f$, $Q \mapsto e$)
4. $par(b, d, e, h)$ (by MP, from $\neg col(b, d, a)$, $midpoint(a, h, d)$, $midpoint(a, e, b)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto b$, $B \mapsto d$, $C \mapsto a$, $P \mapsto h$, $Q \mapsto e$)
5. $par(e, f, g, h)$ (by MP, from $par(a, c, e, f)$, $par(a, c, h, g)$, $\neg col(e, f, g)$ using axiom lemma_par_trans; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto a$, $D \mapsto c$, $E \mapsto g$, $F \mapsto h$)
6. $par(f, g, h, e)$ (by MP, from $par(b, d, f, g)$, $par(b, d, e, h)$, $par(e, f, g, h)$ using axiom lemma_par_trans; instantiation: $A \mapsto f$, $B \mapsto g$, $C \mapsto d$, $D \mapsto b$, $E \mapsto h$, $F \mapsto e$)
7. $pG(e, f, g, h)$ (by MP, from $par(e, f, g, h)$, $par(f, g, h, e)$ using axiom defparallelogram2; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto g$, $D \mapsto h$)
8. Proved by assumption! (by QEDas)

7.2 Problem 2: First Inverse Problem

The list of axioms used for the first inverse problem (Problem 2) is the same as in Section 7.1. Only the conjecture is different – the assumption `midpoint(A,H,D)` is ommitted:

```
fof(th_varignon,conjecture,(! [A,B,C,D,E,F,G,H] : (( (~(col(B,D,A)
    )) & (~(col(B,D,C))) & (~(col(A,C,B))) & (~(col(A,C,D))) & (~
    (col(E,F,G))) & (B != D) & (A != C) & midpoint(A,E,B) &
    midpoint(B,F,C) & midpoint(C,G,D)) => pG(E,F,G,H) ))).
```

If Larus is invoked as: `./larus -l100 -m8 -b1` (`-l100` means the time limit is 100s, `-m8` means that we look for a proof with 8 or fewer steps, `-b1` means that we look for one atomic formula as an abduct), it finds a first consistent abduct (after two inconsistent ones) and produces the following human-readable proof in 3.26 seconds (the abduct found is highlighted):

Consider arbitrary a, b, c, d, e, f, g, h such that:

- $\neg col(b, d, a)$,
- $\neg col(b, d, c)$,
- $\neg col(a, c, b)$,
- $\neg col(a, c, d)$,
- $\neg col(e, f, g)$,
- $b \neq d$,
- $a \neq c$,
- $midpoint(a, e, b)$,
- $midpoint(b, f, c)$,
- $midpoint(c, g, d)$.

It should be proved that $pG(e, f, g, h)$.

Abducts found:

- $midpoint(d, h, a)$

1. $par(a, c, e, f)$ (by MP, from $\neg col(a, c, b)$, $midpoint(b, f, c)$, $midpoint(a, e, b)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto b$, $P \mapsto f$, $Q \mapsto e$)
2. $par(b, d, f, g)$ (by MP, from $\neg col(b, d, c)$, $midpoint(c, g, d)$, $midpoint(b, f, c)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto b$, $B \mapsto d$, $C \mapsto c$, $P \mapsto g$, $Q \mapsto f$)
3. $par(b, d, e, h)$ (by MP, from $\neg col(b, d, a)$, $midpoint(d, h, a)$, $midpoint(a, e, b)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto b$, $B \mapsto d$, $C \mapsto a$, $P \mapsto h$, $Q \mapsto e$)
4. $par(a, c, h, g)$ (by MP, from $\neg col(a, c, d)$, $midpoint(c, g, d)$, $midpoint(d, h, a)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto d$, $P \mapsto g$, $Q \mapsto h$)
5. $par(e, f, g, h)$ (by MP, from $par(a, c, e, f)$, $par(a, c, h, g)$, $\neg col(e, f, g)$ using axiom lemma_par_trans; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto a$, $D \mapsto c$, $E \mapsto g$, $F \mapsto h$)
6. $par(e, h, g, f)$ (by MP, from $par(b, d, e, h)$, $par(b, d, f, g)$, $par(e, f, g, h)$ using axiom lemma_par_trans; instantiation: $A \mapsto e$, $B \mapsto h$, $C \mapsto b$, $D \mapsto d$, $E \mapsto g$, $F \mapsto f$)
7. $pG(e, f, g, h)$ (by MP, from $par(e, f, g, h)$, $par(e, h, g, f)$ using axiom defparallelogram2; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto g$, $D \mapsto h$)
8. Proved by assumption! (by QEDas)

7.3 Problem 3: Second Inverse Problem

The list of axioms used for the second inverse problem (Problem 3) is the same as in section 7.1, extended with the following axioms.

```
fof(defmidpoint,axiom, (! [A,B,C] : ((midpoint(A,B,C)) => ((betS(A
    ,B,C) & cong(A,B,B,C))))).
fof(defmidpoint2,axiom, (! [A,B,C] : ((betS(A,B,C) & cong(A,B,B,C)
    ) => ((midpoint(A,B,C)))))).
```

```
fof(midpoint_NC, axiom, (! [A, B, I] : ((midpoint(A,I,B) & (A != B
    )) => ( (A != I) & ( B != I))))).
fof(defrectangle,axiom, (! [A,B,C,D] : ((rectangle(A,B,C,D)) => ((
    pG(A,B,C,D) & per(A,B,C) & per(B,C,D) & per(C,D,A) & per(D,A,B
    )))))).
fof(defrectangle2a,axiom, (! [A,B,C,D] : ((pG(A,B,C,D) & per(A,B,C
    )) => rectangle(A,B,C,D)))).
fof(defrectangle2b,axiom, (! [A,B,C,D] : ((pG(A,B,C,D) & per(B,C,D
    )) => rectangle(A,B,C,D)))).
fof(defrectangle2c,axiom, (! [A,B,C,D] : ((pG(A,B,C,D) & per(C,D,A
    )) => rectangle(A,B,C,D)))).
fof(defrectangle2d,axiom, (! [A,B,C,D] : ((pG(A,B,C,D) & per(D,A,B
    )) => rectangle(A,B,C,D)))).
fof(defrectangle2e,axiom, (! [A,B,C,D] : ((per(A,B,C) & per(B,C,D)
     & per(C,D,A) & per(D,A,B)) => rectangle(A,B,C,D)))).
%fof(defrectangle3a,axiom, (! [A,B,C,D] : (? [X] : ((rectangle(A,B
    ,C,D)) => cong(A,C,B,D) & midpoint(A,X,C) & midpoint(B,X,D))))
    ).
fof(defrectangle3b,axiom, (! [A,B,C,D,X] : ((cong(A,C,B,D) &
    midpoint(A,X,C) & midpoint(B,X,D)) => rectangle(A,B,C,D)))).
fof(defrectangle4a,axiom, (! [A,B,C,D] : ((rectangle(A,B,C,D)) =>
    (pG(A,B,C,D) & cong(A,C,B,D))))).
fof(defrectangle4b,axiom, (! [A,B,C,D] : ((pG(A,B,C,D) & cong(A,C,
    B,D)) => rectangle(A,B,C,D)))).
fof(lemma_8_2,axiom, (! [A,B,C] : ((per(A,B,C)) => ((per(C,B,A))))
    )).
fof(varignon_th,axiom,(! [A,B,C,D,E,F,G,H] : (( (~(col(B,D,A))) &
    (~(col(B,D,C))) & (~(col(A,C,B))) & (~(col(A,C,D))) & (~ (col(
    G,F,E))) & (B != D) & (A != C) & midpoint(A,E,B) & midpoint(B,
    F,C) & midpoint(C,G,D) & midpoint(A,H,D)) => pG(E,F,G,H) ))).
```

The conjecture is also different – the goal is to find under which assumption the quadrilateral $EFGH$ is a rectangle.

```
fof(th_varignon_rect,conjecture,(! [A,B,C,D,E,F,G,H] : (( (~(col(B
    ,D,A))) & (~(col(B,D,C))) & (~(col(A,C,B))) & (~(col(A,C,D)))
    & (~ (col(G,F,E))) & (B != D) & (A != C) & midpoint(A,E,B) &
    midpoint(B,F,C) & midpoint(C,G,D) & midpoint(A,H,D)) =>
    rectangle(E,F,G,H) ))).
```

If Larus is invoked as: `./larus -l100 -m8 -b1`, it produces the following human-readable proof in 14.19 seconds (the abduct found is highlighted):

Consider arbitrary a, b, c, d, e, f, g, h such that:

- $\neg col(b, d, a)$,
- $\neg col(b, d, c)$,
- $\neg col(a, c, b)$,
- $\neg col(a, c, d)$,
- $\neg col(f, g, e)$,
- $b \neq d$,
- $a \neq c$,
- $midpoint(a, e, b)$,
- $midpoint(b, f, c)$,
- $midpoint(c, g, d)$,
- $midpoint(a, h, d)$.

It should be proved that $rectangle(e, f, g, h)$.

Abducts found:

- $cong(e, g, h, f)$

1. $midpoint(b, e, a)$ (by MP, from $midpoint(a, e, b)$, $midpoint(a, e, b)$ using axiom defmidpoint2; instantiation: $A \mapsto b$, $B \mapsto e$, $C \mapsto a$)
2. $pG(e, f, g, h)$ (by MP, from $\neg col(b, d, a)$, $\neg col(b, d, c)$, $\neg col(a, c, b)$, $\neg col(a, c, d)$, $\neg col(f, g, e)$, $b \neq d$, $a \neq c$, $midpoint(b, e, a)$, $midpoint(b, f, c)$, $midpoint(c, g, d)$, $midpoint(a, h, d)$ using axiom varignon_th; instantiation: $A \mapsto a$, $B \mapsto b$, $C \mapsto c$, $D \mapsto d$, $I \mapsto e$, $J \mapsto f$, $K \mapsto g$, $L \mapsto h$)
3. $rectangle(e, f, g, h)$ (by MP, from $pG(e, f, g, h)$, $cong(e, g, h, f)$ using axiom defrectangle4b; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto g$, $D \mapsto h$)
4. $rectangle(e, f, g, h)$ (by MP, from $rectangle(e, f, g, h)$, $rectangle(e, f, g, h)$, $rectangle(e, f, g, h)$, $rectangle(e, f, g, h)$ using axiom defrectangle2e; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto g$, $D \mapsto h$)
5. Proved by assumption! (by QEDas)

7.4 Problem 4: Partially Specified Goal

The list of axioms used for Problem 4 is the same as presented in Section 7.1. Only the conjecture is different: the goal does not have the predicate symbol specified:

```
fof(th_varignon,conjecture,(! [A,B,C,D,E,F,G,H] : (( (~(col(B,D,A)
    )) & (~(col(B,D,C))) & (~(col(A,C,B))) & (~(col(A,C,D))) & (~
    (col(E,F,G))) & (B != D) & (A != C) & midpoint(A,E,B) &
    midpoint(B,F,C) & midpoint(C,G,D) & midpoint(A,H,D)) => _(E,F,
    G,H) ))).
```

If Larus is invoked as `./larus -l100 -m8`, it produces the following human-readable proof (for the goal $par(e, f, g, h)$, highlighted in the proof) in 2.15 seconds:

Consider arbitrary a, b, c, d, e, f, g, h such that:

- $\neg col(b, d, a)$,
- $\neg col(b, d, c)$,
- $\neg col(a, c, b)$,
- $\neg col(a, c, d)$,
- $\neg col(e, f, g)$,
- $b \neq d$,
- $a \neq c$,
- $midpoint(a, e, b)$,
- $midpoint(b, f, c)$,
- $midpoint(c, g, d)$,
- $midpoint(a, h, d)$.

It should be proved that $\_(e, f, g, h)$.

1. $par(a, c, e, f)$ (by MP, from $\neg col(a, c, b)$, $midpoint(b, f, c)$, $midpoint(a, e, b)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto b$, $P \mapsto f$, $Q \mapsto e$)
2. $par(a, c, h, g)$ (by MP, from $\neg col(a, c, d)$, $midpoint(c, g, d)$, $midpoint(a, h, d)$ using axiom triangle_mid_par_strict; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto d$, $P \mapsto g$, $Q \mapsto h$)
3. $par(e, f, g, h)$ (by MP, from $par(a, c, e, f)$, $par(a, c, h, g)$, $\neg col(e, f, g)$ using axiom lemma_par_trans; instantiation: $A \mapsto e$, $B \mapsto f$, $C \mapsto a$, $D \mapsto c$, $E \mapsto g$, $F \mapsto h$)
4. Proved by assumption! (by QEDas)

7.5 Problem 5: Partially Specified Proof

The list of axioms used for Problem 5 is the same as presented in Section 7.3 (with the axiom `defrectangle3a` deleted). The conjecture is the same plus the abduct as an assumption, but we add the following hint:

```
fof(hint1,hint,_,_,defrectangle4b(4,5,6,7)).
```

If Larus is invoked as `./larus -l100 -m8`, it produces the same proof as in Section 7.3 in 4 seconds, but if the hint is omitted, it takes 5 seconds.