Automated generation of illustrated proofs in geometry and beyond

Predrag Janičić, Julien Narboux

Automated Generation of Illustrated Proofs in Geometry and Beyond

Predrag Janičić[0000-0001-8922-4948]
Department for Computer Science
Faculty of Mathematics
University of Belgrade
Studentski trg 16
11000 Belgrade, Serbia email: janicic@matf.bg.ac.rs

Julien Narboux[0000-0003-3527-7184]
UMR 7357 CNRS
University of Strasbourg
Pôle API, Bd Sébastien Brant
BP 10413
67412 Illkirch, France email: narboux@unistra.fr

May 3, 2023

Abstract

Illustrations are only rarely formal components of mathematical proofs, however they are often very important for understanding proofs. Illustrations are almost unavoidable in geometry, and in many other fields illustrations are helpful for carrying ideas in a more suitable way than via words or formulas. The question is: if we want to automate theorem proving, can we automate creation of corresponding illustrations too? We report on a new, generic, simple, and flexible approach for automated generation of illustrated proofs. The proofs are generated using Larus, an automated prover for coherent logic, and corresponding illustrations are generated in the GCLC language. Animated illustrations are also supported.

Keywords. proofs illustration, synthetic geometry, automated deduction, diagrams, sketches, abstract term rewriting

1 Introduction

Illustrations typically do not form part of mathematical proofs, but are used for communicating ideas more easily. In geometry, proofs and sketches are closely related. Humans need sketches to visualize geometric statements, to reason

about geometrical problems, and for help in understanding geometric proofs. That is why the use of dynamic geometry software is widely spread in mathematical education. However, many famous examples of incorrect or insufficiently justified steps in geometric reasoning can be attributed to improper use of visual intuition based on geometric figures (i.e., in such cases, illustrations and "proofs" were not properly matched). This goes from unjustified proof steps in Euclid's Elements to proofs and statements which rely on implicit assumptions from the figure in the context of education [28], passing by incorrect proofs of Euclid's fifth postulate such as the one by Legendre [24]. Since their early days, automated deduction systems for geometry have also tried to use the knowledge given by figures (i.e., by concrete models) [17]. Some authors also studied *diagrammatic reasoning* – how illustrations, following some rules, can provide proofs. Diagrammatic reasoning systems can be defined rigorously enough such that they behave as formal systems [37, 25].

In this paper, we present our approach for automated generation of proofs accompanied by visual illustrations. While the approach can be used for various mathematical theories, its primary target is geometry. We are not dealing with diagrammatic reasoning, and we focus rather on generation of illustrations of proofs as sketches similar to what a human would create. The proofs are readable synthetic proofs generated by Larus – our automated theorem prover for coherent logic [20], and the module for illustrations is one of its extensions. The use of coherent logic is an essential ingredient of our approach. The proofs in coherent logic are organized in a purely forward reasoning style, hence the goal to be proved is not changed along the proving process. Put in logical terms: coherent logic enjoys a complete deduction system which never modifies the right hand sides of the sequents. The goal is unchanged, reached by some proof steps, so does not need to be additionally depicted. In out approach, the initial sketch depicts (a model of) the assumptions. Then, in each proof step, existence of new objects and also new facts are derived along the proof and depicted dynamically by extending and decorating the initial sketch. The generated illustrations are stored in a domain specific, geometry oriented, language GCLC (developed by the first author) [21, 23]. The illustrations can also be presented as animations, in a step-by-step manner.

2 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 k 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 l variables $y_0, y_1, \ldots, y_{l-1}$; B_j (for $0 \leq j \leq m-1$) denotes a conjunction of atomic formulas (involving zero or more of the variables from $\vec{x}$ and $\vec{y}$). If there are no formulas A_i, then the left-hand side of the implication is assumed to be $\top$. If

there are no formulas 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 formulas do not involve the negation connective. A coherent theory is a set of sentences, axiomatized by coherent formulas, and closed under derivability [14].

A number of theories and theorems can be formulated directly and simply in coherent logic (CL). Several authors independently point to CL (or rules similar to those of CL) as suitable for expressing (sometimes – also automating) significant portions of mathematics [15, 2]. In contrast to resolution-based theorem proving, in 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 readable synthetic proofs [7].

Every first-order theory has a coherent conservative extension [14], i.e., any first-order theory can be translated into CL, possibly with additional predicate symbols. Translation of FOL formulas into CL involves elimination of negations: 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 formulas [31]. 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 order to enable more efficient proving, some advanced translation techniques are used. Elimination of function symbols, sometimes called *anti Skolemization*, is also done by introducing additional predicate symbols [29].

The problem of provability in CL is semi-decidable. CL admits a simple proof system, a sequent-based variant is as follows [33]:

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

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

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

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

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): $AX, A_0(\vec{a}), \ldots, A_{n-1}(\vec{a}) \vdash \exists \vec{y}(B_0(\vec{a},\vec{y}) \vee \ldots \vee\ B_{m-1}(\vec{a},\vec{y}))$. Proofs can be presented in the manner of forward reasoning, as illustrated by the following example.

Example 1 *Consider the following set of axioms:*
ax1: $\forall x\ (p(x) \Rightarrow r(x) \vee q(x))$
ax2: $\forall x\ (q(x) \Rightarrow \bot)$
and the following conjecture that can be proved as a CL theorem:
$\forall x\ (p(x) \Rightarrow r(x))$

Consider arbitrary a such that: $p(a)$. *It should be proved that* $r(a)$.

1. $r(a) \vee q(a)$ *(by MP, from* $p(a)$ *using axiom ax1; instantiation:* $X \mapsto a$*)*
   2. *Case* $r(a)$*:*
      3. *Proved by assumption! (by QEDas)*
   4. *Case* $q(a)$*:*
      5. $\bot$ *(by MP, from* $q(a)$ *using axiom ax2; instantiation:* $X \mapsto a$*)*
      6. *Contradiction! (by QEDefq)*
7. *Proved by case split! (by QEDcs, by* $r(a), q(a)$*)*

Larus is an automated theorem prover for coherent logic [20]. It has several underlying proof engines, based on different approaches. Larus is publicly available and open-source.[1]

3 Methods for Generating Illustrated Proofs

The basic idea for the proposed methods is simple: if we know how to visually interpret all proof steps that introduce new objects or facts, then we know how to produce a complete illustration. Indeed, following the Curry-Howard correspondence which relates computer programs and proofs (a proof is a program, and the formula it proves is the type for the program), proofs of existential statements can be depicted using a function which takes the universally quantified geometric objects of the statement as input and constructs witnesses of the conclusion (along with its visual counterpart). The illustration is based on a sequence of such witnesses in one universe. For geometry, we will focus on the usual choice – Cartesian space (hence, for instance, geometry points will map to Cartesian points). Of course, our illustration will be just one model for just one proof branch. This idea is very well suited to coherent logic as an underlying logical framework (see Section 2). In CL proofs, new facts are derived using *modus ponens*, so we have to handle only these rule applications in the proof.

The above idea, obviously, cannot be applied in conjunction with resolution or saturation based theorem provers for first-order logic, even less in conjunction with algebraic provers (such as those based on Gröbner bases or Wu's method [10]), nor it can be applied on proof traces produced by such provers.

[1] https://github.com/janicicpredrag/Larus

3.1 Construction of witnesses

As an example, let us assume that, within a proof, there is an application of the following axiom (with the obvious intended meaning):

$$\forall x, y\ (point(x) \wedge point(y) \Rightarrow \exists z\ between(x, z, y))$$

It may have the following visual interpretation attached: "for two concrete Cartesian points a and b, the Cartesian point c such that c is between a and b can be created as the Cartesian midpoint of ab". For example, assuming the points a and b, occurring in the proof, are associated Cartesian coordinates $(2, 5)$ and $(4, 11)$, if the above axiom has been applied to them, then the new witness point will have the associated Cartesian coordinates $(3, 8)$.

3.2 Illustrating individual proof steps

Visual interpretation of all proof steps can be supported in the two ways described below.[2]

Method 1 — method based on illustrating axioms: In this variant of our approach, each axiom needs its associated visual interpretation, its visual counterpart. Within this visual counterpart, possible new witnesses are created and illustrated, but also new facts that were established by the axiom are illustrated. In the above example, we would draw the segment ab. As another example, if a proof step establishes that some three points are collinear, we would draw the Cartesian line that contain the three Cartesian points corresponding to them.

Using this method, the only thing that we need for a visual interpretation of the proof of one conjecture are visual counterparts of all axioms and lemmas/theorems used within the proof. Moreover, the visual interpretation of used lemmas/theorems can also be produced automatically, using the same method (as shown in Section 5). Therefore, what we need, ultimately, are only visual interpretations of all axioms, provided by a human.

Method 2 — method based on illustrating predicate symbols: Compared to Method 1, Method 2 is even less demanding for the user. Instead of providing visual counterparts for each axiom (and theorem), the user needs to provide only (*i*) ways to construct new witnesses by the axioms and (*ii*) visual interpretation for each predicate symbol (recall that CL does not use function symbols). The system automatically uses these visual interpretations for illustrating each proof step: all predicates appearing both in premises and in conclusions of the axioms are illustrated.

[2] This paper extends the work presented at the conference ADG 2021, and published in EPTCS 352 [19]: the previous approach has been extended by Method 2, presented here.

Method 2 can be less demanding for the user, but Method 1 leaves more freedom: an axiom may be illustrated in some specific way, not only by illustrating all predicates that occur in it. Actually, even if the user uses Method 2, he/she is still free to modify fragments of automatically generated illustrations and to provide illustrations of some axioms in the spirit of Method 1.

3.3 Premises and Initial Configuration

We explained how the figure is updated by each *modus ponens* rule application. But how do we start the illustration in the first place? Recall that we prove theorems of the form: $A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}) \Rightarrow \exists \vec{y}(B_0(\vec{x},\vec{y}) \vee \ldots \vee B_{m-1}(\vec{x},\vec{y}))$ (and that all axioms used also have that form). In order to build the initial illustration, we need some constants $\vec{a}$ such that: $A_0(\vec{a}) \wedge \ldots \wedge A_{n-1}(\vec{a})$ holds. But how can we find and illustrate such objects? We can do that by using the same approach as described above, applied to the theorem: $\exists \vec{x}(A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}))$.[3] In fact, here we perform a sort of constraint solving using theorem proving. For example, if we provide to the prover axioms modeling only ruler-and-compass geometry, then we can obtain a constructive proof of existence corresponding to a ruler-and-compass construction. But, if we provide more powerful axioms to the prover, such as the ability of trisecting the angle, then we could also, in principle, illustrate proofs such as that of Morley's theorem (this is only an example since Morley's theorem is currently way out of reach of our prover). This approach also requires some care: the witness of the existential should be taken as general as possible (in order to avoid misleading illustrations). This problem can be addressed by adding non-degenerate conditions, for instance, that the points asserted to exist are pairwise distinct and non-collinear. Alternatively, instead of finding an initial illustration by proving the conjecture $\exists \vec{x}(A_0(\vec{x}) \wedge \ldots \wedge A_{n-1}(\vec{x}))$, the user can chose initial objects himself/herself and create an illutration of the premises manually.

3.4 Case Splits

There can be several case splits in the proof and it would make no sense to illustrate all proof leaves. Moreover, some proof branches may be contradictory. Not only that they are less interesting (they typically correspond to degenerative cases) but they do not have models. Hence, if we have several proof branches in one proof node, we could do the following:

- if all of them end up with contradiction, then they all belong to some upper contradictory proof node, and we illustrate neither of them;[4]

[3] If, instead of just a model of the premises, we want to generate a *construction procedure* for a whole class of initial configurations, we can choose to quantify universally some of the variables, and apply the procedure to the statement: $\forall \vec{x'} \exists \vec{x''}(A_0(\vec{x'},\vec{x''}) \wedge \ldots \wedge A_{n-1}(\vec{x'},\vec{x''}))$. Those variables $\vec{x'}$ quantified universally would then correspond to the *free points* of the figure. Depending on the choice of $\vec{x'}$, the statement can be a theorem or not, hence, the choice of the set of these free points should be made by the user.

[4] If the premises themselves are contradictory, then we do not provide an illustration.

- If there are some proof branches that do not end up with contradiction, then one that corresponds to the model being built should be illustrated.

In practice, we combine the above with other policies. For instance, for all case splits of the form $R(\vec{a}) \vee \overline{R}(\vec{a})$, we follow only the negative branch, $\overline{R}(\vec{a})$ (for instance, three points are non-collinear) as generally they correspond to non-degenerated, more interesting cases.

3.5 Randomization

In order to make illustrations partly unpredictable and more interesting, some randomization may be added to the visual interpretation of the axioms/predicate symbols. For example, if there is an axiom stating that for any two distinct points there is a third one between them, then in the visual counterpart, that third point could be chosen based on a pseudo-random number between 0 and 1.

4 Implementation

We implemented the described methods within our automated theorem prover for coherent logic, Larus [20]. Larus' flexible architecture already had proof export to LaTeX and Coq supported. So, we have implemented just two classes more – two classes that export generated CL proofs (in Larus' internal representation) to visual representation. The new code has less than 500 lines of C++.

For the target language, i.e., for the language of visual representation we chose the GCLC language [23], a rich, special purpose language for mathematical, especially geometry illustrations. The gool GCLC can translate illustration files to different image formats, including BMP, EPS, SVG, and LaTeX formats.

For each theorem $\mathcal{T}$ (given in the TPTP format, the standard format for first-order logic theorem provers [34]), once $\mathcal{T}$ has been proved, the prover generates illustration files (in GCLC formats), using one of the two classes that implement the two presented methods (see Section 3.2):

- One class assumes there are available visual interpretations of all axioms/theorems used, and the main illustration file invokes files for all used axioms/theorems. The "main" generated file invokes also the procedure for existence of premises for $\mathcal{T}$, and then the procedure for $\mathcal{T}$ itself, which in turn invokes illustrations for each axiom application.

- One class assumes there are available visual interpretations of all predicate symbols used and also procedures that describe how new witnesses are constructed (by the axioms used). The main illustration file invokes the procedure for existence of premises for $\mathcal{T}$, and then the procedure for $\mathcal{T}$ itself, that consists of illustrations of all facts occurring in the proof steps.

Actually, in some cases we could also somehow illustrate contradictory branches (as it is done in some books [32, page 39]) – for instance, if it is proved that three points are both collinear and non-collinear, we could draw a curved line that connect them.

In both cases, the prover generates a TPTP formulation of the theorem that corresponds to existence of premises for $\mathcal{T}$ (see Section 3.3). If that theorem can be proved automatically, its visual representation can also be obtain automatically. Alternatively, the user may visualize objects used in the premises of the theorem.

In both cases, the prover generates an illustration in small chunks of GCLC code, GCLC procedures, like a procedure for existence of objects given in the premises, or like a procedure that illustrates application of one axiom.

Animations (actually, sequences of images) are obtained simply by showing visualizations of proof steps one by one. For that purpose, each proof step in the illustration is annotated by a layer and, during animations, more and more layers get visible. In addition, visualization for each proof step can go in stages: for instance, first in some emphasized manner (or in another color), and then in a regular way.

Generated illustrations, stored as readable GCLC files, can be further modified and improved by a human.

Generated (and possibly modified) GCLC files can be exported to different formats, including the LaTeX format `TikZ`. Thank to this, illustrations embedded into proofs and proof text itself can share the same fonts and visual spirit.

5 Using Method 1: Euclid's *Elements*, Book I, Proposition 11

As a use-case for Method 1 (based on illustrations of the axioms), we use Proposition 11 from Euclid's *Elements*, Book I. Its original form reads as follows: "To draw a straight line at right angles to a given straight line from a given point on it." The statement represented in first-order logic, following the formalization of first book of Euclid's *Elements* as proposed by Beeson et al. [3], is the following:

$$\forall A, B, C \ \text{BetS}\ ACB \ \Rightarrow \ \exists X \ \ \text{Per}\ ACX$$

(BetS ACB means that C is strictly between A and B, Per ACX means that ACX is a right angle with the vertex C). The TPTP file[5] with the axioms and lemmas[6] (listed as axioms as well), relevant in a wider context, is as follows:

[5] In TPTP files, each `fof` term gives one first-order formula, along with its name and kind (`axiom` or `conjecture`). In the concrete syntax, `!...:` and `?...:` denote universal and existential quantification, `&` and `|` denote conjunction and disjunction, `=>` denotes implication, `!=` denotes non-equality, etc. Atomic formualae can be simply expressed as `betS(A,C,B)`.

[6] The proof checked by Coq can be found here: https://github.com/GeoCoq/GeoCoq/blob/master/Elements/OriginalProofs/proposition_11.v

```
fof(lemma_betweennotequal,axiom, (! [A,B,C] :
    ((betS(A,B,C)) => ((( B != C ) & ( A != B ) & ( A != C )))))).
fof(lemma_extension,axiom, (! [A,B,P,Q] : (? [X] :
    ((( A != B ) & ( P != Q )) => ((betS(A,B,X) & cong(B,X,P,Q))))))).
fof(proposition_01,axiom, (! [A,B] : (? [X] : ((( A != B )) =>
    ((equilateral(A,B,X) & triangle(A,B,X))))))).
fof(defequilateral,axiom, (! [A,B,C] :
    ((equilateral(A,B,C)) => ((cong(A,B,B,C) & cong(B,C,C,A)))))).
fof(defequilateral2,axiom, (! [A,B,C] :
    ((cong(A,B,B,C) & cong(B,C,C,A)) => ((equilateral(A,B,C)))))).
fof(lemma_doublereverse,axiom, (! [A,B,C,D] :
    ((cong(A,B,C,D)) => ((cong(D,C,B,A) & cong(B,A,D,C)))))).
fof(lemma_congruenceflip,axiom, (! [A,B,C,D] :
    ((cong(A,B,C,D)) => ((cong(B,A,D,C) & cong(B,A,C,D) & cong(A,B,D,C)))))).
fof(defcollinear,axiom, (! [A,B,C] : ((col(A,B,C)) =>
    ((( A = B )) | (( A = C )) | (( B = C )) |
    (betS(B,A,C)) | (betS(A,B,C)) | (betS(A,C,B)))))).
fof(defcollinear2a,axiom, (! [A,B,C] : ((( A = B )) => ((col(A,B,C)))))).
fof(defcollinear2b,axiom, (! [A,B,C] : ((( A = C )) => ((col(A,B,C)))))).
fof(defcollinear2c,axiom, (! [A,B,C] : ((( B = C )) => ((col(A,B,C)))))).
fof(defcollinear2d,axiom, (! [A,B,C] : ((betS(B,A,C)) => ((col(A,B,C)))))).
fof(defcollinear2e,axiom, (! [A,B,C] : ((betS(A,B,C)) => ((col(A,B,C)))))).
fof(defcollinear2f,axiom, (! [A,B,C] : ((betS(A,C,B)) => ((col(A,B,C)))))).
fof(lemma_collinearorder,axiom, (! [A,B,C] : ((col(A,B,C)) =>
    ((col(B,A,C) & col(B,C,A) & col(C,A,B) & col(A,C,B) & col(C,B,A)))))).
fof(deftriangle,axiom, (! [A,B,C] : ((triangle(A,B,C)) =>
    ((~ (col(A,B,C))))))).
fof(deftriangle2,axiom, (! [A,B,C] : ((~(col(A,B,C))) =>
    ((triangle(A,B,C)))))).
fof(defrightangle,axiom, (! [A,B,C] : (? [X] : ((per(A,B,C)) =>
    ((betS(A,B,X) & cong(A,B,X,B) & cong(A,C,X,C) & ( B != C ))))))).
fof(defrightangle2,axiom, (! [A,B,C,X] : ((betS(A,B,X) &
    cong(A,B,X,B) & cong(A,C,X,C) & ( B != C )) => ((per(A,B,C)))))).
fof(proposition_11,conjecture,(! [A,B,C] : (? [X] : ((betS(A,C,B)) =>
    ((per(A,C,X))))))).
```

The proposition was proved automatically by the Larus prover, giving the following proof in LaTeX[7]:

[7] This LaTeX proof presentation is still very verbatim and it provides much more information than a traditional proof; for future work we are planning to produce a more natural output.

Theorem 1 *proposition_11 :* $\forall A\ \forall B\ \forall C\ (betS(A,C,B) \Rightarrow \exists X\ (per(A,C,X))\ )$

Proof:

Consider arbitrary a, b, c such that: $betS(a,c,b)$. It should be proved that $\exists X\ \ per(a,c,X)$.

1. Let w be such that $betS(a,c,w) \wedge cong(c,w,a,c)$ (by MP, from $betS(a,c,b)$, $betS(a,c,b)$ using axiom lemma_extension; instantiation: $A \mapsto a$, $B \mapsto c$, $P \mapsto a$, $Q \mapsto c$)
2. Let $w1$ be such that $equilateral(a,w,w1) \wedge triangle(a,w,w1)$ (by MP, from $betS(a,c,w) \wedge cong(c,w,a,c)$ using axiom proposition_01; instantiation: $A \mapsto a$, $B \mapsto w$)
3. $w1 = c \vee w1 \neq c$ (by MP, using axiom eq_excluded_middle; instantiation: $A \mapsto w1$, $B \mapsto c$)
   4. Case $w1 = c$:
      5. $col(a,w,w1)$ (by MP, from $betS(a,c,w) \wedge cong(c,w,a,c)$, $w1 = c$ using axiom colEqSub2; instantiation: $A \mapsto a$, $B \mapsto w$, $C \mapsto c$, $X \mapsto w1$)
      6. $\perp$ (by MP, from $col(a,w,w1)$, $equilateral(a,w,w1) \wedge triangle(a,w,w1)$ using axiom nnncolNegElim; instantiation: $A \mapsto a$, $B \mapsto w$, $C \mapsto w1$)
      7. Contradiction! (by QEDefq)
   8. Case $w1 \neq c$:
      9. $per(a,c,w1)$ (by MP, from $betS(a,c,w) \wedge cong(c,w,a,c)$, $betS(a,c,w) \wedge cong(c,w,a,c)$, $equilateral(a,w,w1) \wedge triangle(a,w,w1)$, $w1 \neq c$ using axiom defrightangle2; instantiation: $A \mapsto a$, $B \mapsto c$, $C \mapsto w1$, $X \mapsto w$)
      10. Proved by assumption! (by QEDas)
11. Proved by case split! (by QEDcs, by $w1 = c, w1 \neq c$)

Note that the above Larus' proofs omits applications of "simple axioms" (although this feature can be turned off) — axioms that are universal implications from one atomic formula to another. The above proof explicitly uses only the following lemmas (not counting those implied by equality axioms and those introducing $\perp$): *lemma_extension*, *proposition_01*, *defrightangle2*. Hence, we need visual interpretation of these and we can write them ourselves. For instance, the visual counterpart for *proposition_01*, the famous Euclid's proposition 1 (on existence of a equilateral triangle on a given segment), can be like the following:

```
procedure proposition_01 { A B X } {
  circle c1 A B
  drawcircle c1
  circle c2 B A
  drawcircle c2
  intersec2 X2 X c1 c2
  cmark X
}
```

In this visual interpretation, a point X is obtained (in the Cartesian model) as an intersection of two circles – one with the center A with B on it,[8] and one the center B with A on it (each circle is drawn by the command `drawcircle`),

[8] The procedure `proposition_01` assumes that A and B are distinct points. That condition has to be ensured by the procedures that invoke this one, or by selection of initial points.

and the point X is annotated by a small circle (using the command `cmark`). Of course, one could have chosen the other intersection of two circles. Alternatively, instead of providing this visual interpretation for this theorem (*proposition_01*), we could run the prover on it and could get another procedure, expressed in terms of axioms (or, possibly, also in terms of some other lemmas). This alternative provides a simple, uniform approach that requires only a little manual labour. However, this alternative is not always possible: maybe some axioms necessary to prove the theorem are not available to the prover or, even if they all are, the automated theorem prover may not be able to prove the statement. Finally, even if the conjecture can be automatically proven and illustrated, in some situations, one may still prefer a figure designed by a human, customized for a certain context.

The prover also generates the conjecture that establishes existence of objects such that the premises of the main conjecture hold:

```
fof(proposition_11, axiom, ( ? [A,B,C] : (betS(A,C,B))))
```

This conjecture can be proved automatically (if there are suitable axioms available) or can be illustrated by hand, as here (within the file `proposition_11_exists.gcl`):

```
procedure proposition_11_exists { a b c  }  {
  point a 8 2
  point b 22 7
  towards c a b 0.7
  cmark_t a
  cmark_t b
  cmark_t c
}
```

In this concrete illustration of initial objects, the constants a and b are associated Cartesian coordinates $(8, 2)$ and $(22, 7)$ (in millimeters, when it comes to exporting images). The point c is chosen to divide the segment (a, b) such that $ac : ab = 0.7 : 1$. All three points are annotated by small circles and by their names (by the commands `cmark_t`).

The main conjecture is described by the following procedure, generated automatically by the prover (within the file `proposition_11.gcl`):

```
procedure proposition_11 { a b c w  }  {
  call lemma_extension { a c a c w }
  mark_t w
  call proposition_01 { a w w1 }
  mark_t w1
  % --- Illustration for branch 2
  call defrightangle2 { a c w1 w }
}
```

One point (w) is introduced by the application of *lemma_extension*, and another (w_1) by the application of *proposition_01*. The illustration follows the second, non-contradictory branch of the proof. The final proof step does not introduce

new objects, but only establishes a property $ac \perp cw_1$ (which is stressed by drawing a small square at the vertex of the angle).

Finally, the main file, that will be processed by GCLC, is generated automatically and looks like the following:

```
% ----- Proof illustration -----
include proposition_11.gcl
include proposition_11_exists.gcl
include lemma_extension.gcl
include proposition_01.gcl
include defrightangle2.gcl
%------------------------------
call proposition_11_exists { a b c }
call proposition_11 { a b c w  }
```

After the `include` section with all needed files/procedures, there is a call to the procedure which illustrates the objects such that the premises hold. After that, with the concrete points `a`, `b`, `c` (provided by the previous procedure), the main procedure is invoked. The illustration can be modified such that it provides an animation (within GCLC graphical environment, but it can be also exported as a sequence of images). This is implemented using layers, as in the following addition to the main GCLC file, so the proof steps are shown one-by-one.

```
animation_frames 7 1
point A0 0 0
point A1 1 0 7 0
distance dA A0 A1
hide_layers_from dA
```

The above code defines an animation with 7 frames (one per second). Each proof step can be associated with one layer, identified by numbers 1, 2, 3, etc. GCLC provides support to generate animation with moving objects, but does not provide an explicit access to the current frame number. Hence, a small trick is used to show layers one by one: during the animation, the point `A1` moves from the coordinates (1,0) to (7,0), hence will have the coordinates (1,0), (2,0), ..., (7,0), and the distance to the point `A0` with coordinates (0,0), will be equal to the frame number: 1, 2, ..., 7. In the ith frame, all layers from i upward will be hidden, so the contents will be shown step by step (while keeping the contents of the old frames).

Processing the above file gives the animation within the GCLC environment, i.e., the illustration that gets extended in several steps, as shown in Figure 1.

6 Using Method 2: Examples in Abstract Term Rewriting

The proposed approach for creating illustrated proofs is not limited to geometry: it can be used for any theory that admits reasonable illustration of proof steps.

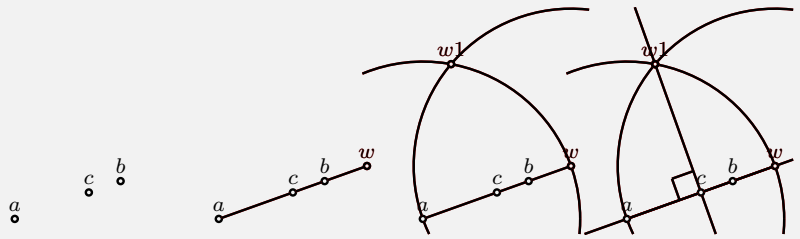


Figure 1: Four steps in illustration of the proposition 11

One such example is theory of abstract rewriting systems. Visualization of proofs in this theory has been studied by the second author [26]. In this section we provide two examples of automatically generated proofs and diagrams, both using Method 2 (based on visual counterparts for each predicate symbol).

6.1 A Simple Property of Abstract Term Rewriting

First we consider the following proposition: if two transitive relations R_1 and R_2 are such that $R_2.R_1 \subseteq R_1.R_2$ then $R_1.R_2$ is transitive[9]. We translate this statement to coherent logic, by specializing the generic definitions of transitivity and composition of two relations to the relations R_1 and R_2. We obtain the following list of axioms, the goal, and finally the illustrated proof[10]:

Axioms:

1. r1transitive : $\forall A\ \forall B\ \forall C\ (r1(A,B) \wedge r1(B,C) \Rightarrow r1(A,C)\ )$
2. r2transitive : $\forall A\ \forall B\ \forall C\ (r2(A,B) \wedge r2(B,C) \Rightarrow r2(A,C)\ )$
3. r2r1inr1r2 : $\forall X\ \forall Y\ (r2r1(X,Y) \Rightarrow r1r2(X,Y)\ )$
4. defr1r2 : $\forall X\ \forall Y\ (r1r2(X,Y) \Rightarrow \exists Z\ (r1(X,Z) \wedge r2(Z,Y))\ )$
5. defr2r1 : $\forall X\ \forall Y\ (r2r1(X,Y) \Rightarrow \exists Z\ (r2(X,Z) \wedge r1(Z,Y))\ )$
6. rdefr1r2 : $\forall X\ \forall Y\ \forall Z\ (r1(X,Y) \wedge r2(Y,Z) \Rightarrow r1r2(X,Z)\ )$
7. rdefr2r1 : $\forall X\ \forall Y\ \forall Z\ (r2(X,Y) \wedge r1(Y,Z) \Rightarrow r2r1(X,Z)\ )$

Theorem 2 *goal :* $\forall X\ \forall Y\ \forall Z\ (r1r2(X,Y) \wedge r1r2(Y,Z) \Rightarrow r1r2(X,Z)\ )$

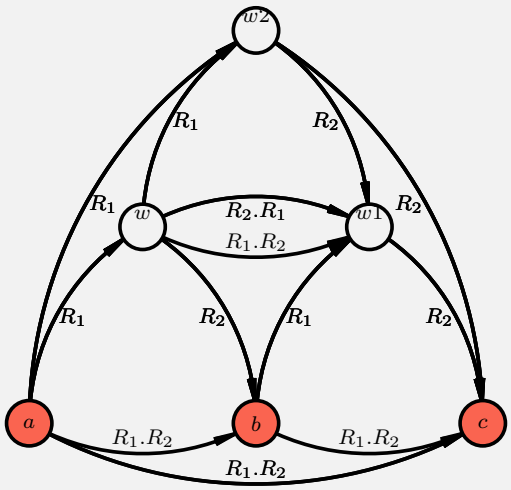
[9] $x\ R_1.R_2\ y$ means that there is a z such that xR_1z and zR_2y

[10] Note that relations are usually depicted using upper-case letters, but the TPTP/FOL language requires predicates to be denoted by a name starting by a lower-case letter.

Proof:

Consider arbitrary a, b, c such that: $r1r2(a,b)$, $r1r2(b,c)$. It should be proved that $r1r2(a,c)$.

1. Let w be such that $r1(a,w) \wedge r2(w,b)$ (by MP, from $r1r2(a,b)$ using axiom defr1r2; instantiation: $X \mapsto a$, $Y \mapsto b$)
2. Let $w1$ be such that $r1(b,w1) \wedge r2(w1,c)$ (by MP, from $r1r2(b,c)$ using axiom defr1r2; instantiation: $X \mapsto b$, $Y \mapsto c$)
3. $r2r1(w,w1)$ (by MP, from $r1(a,w) \wedge r2(w,b)$, $r1(b,w1) \wedge r2(w1,c)$ using axiom rdefr2r1; instantiation: $X \mapsto w$, $Y \mapsto b$, $Z \mapsto w1$)
4. Let $w2$ be such that $r1(w,w2) \wedge r2(w2,w1)$ (by MP, from $r2r1(w,w1)$ using axiom defr1r2; instantiation: $X \mapsto w$, $Y \mapsto w1$)
5. $r1(a,w2)$ (by MP, from $r1(a,w) \wedge r2(w,b)$, $r1(w,w2) \wedge r2(w2,w1)$ using axiom r1transitive; instantiation: $A \mapsto a$, $B \mapsto w$, $C \mapsto w2$)
6. $r2(w2,c)$ (by MP, from $r1(w,w2) \wedge r2(w2,w1)$, $r1(b,w1) \wedge r2(w1,c)$ using axiom r2transitive; instantiation: $A \mapsto w2$, $B \mapsto w1$, $C \mapsto c$)
7. $r1r2(a,c)$ (by MP, from $r1(a,w2)$, $r2(w2,c)$ using axiom rdefr1r2; instantiation: $X \mapsto a$, $Y \mapsto w2$, $Z \mapsto c$)
8. Proved by assumption! (by QEDas)



As already mentioned, the above illustration was created using Method 2, which is based on provided visual counterparts of all predicate symbols. One fragment (provided by the user) is as follows:

```
...
procedure draw_r1 { X Y i }
{
  call draw_edge { X Y L 1 }
  printat L { R_1 }
}
...
```

The above code is invoked for illustrating facts of the form $r1(x, y)$. It uses an auxiliary procedure `draw_edge` also provided by the user. The last parameter in the above procedure can be used to control the drawing style. In the above procedure it was not used, but it can be used to visualy distinguish what Duval calls the *operational status* of a statement [13]: at each step of the proof a statement can either be an already known fact, or a premise, or a conclusion of the current proof step. The main generated GCLC file invokes functions like `draw_r1` with different values of the last argument, depending on the *operational status* (see below).

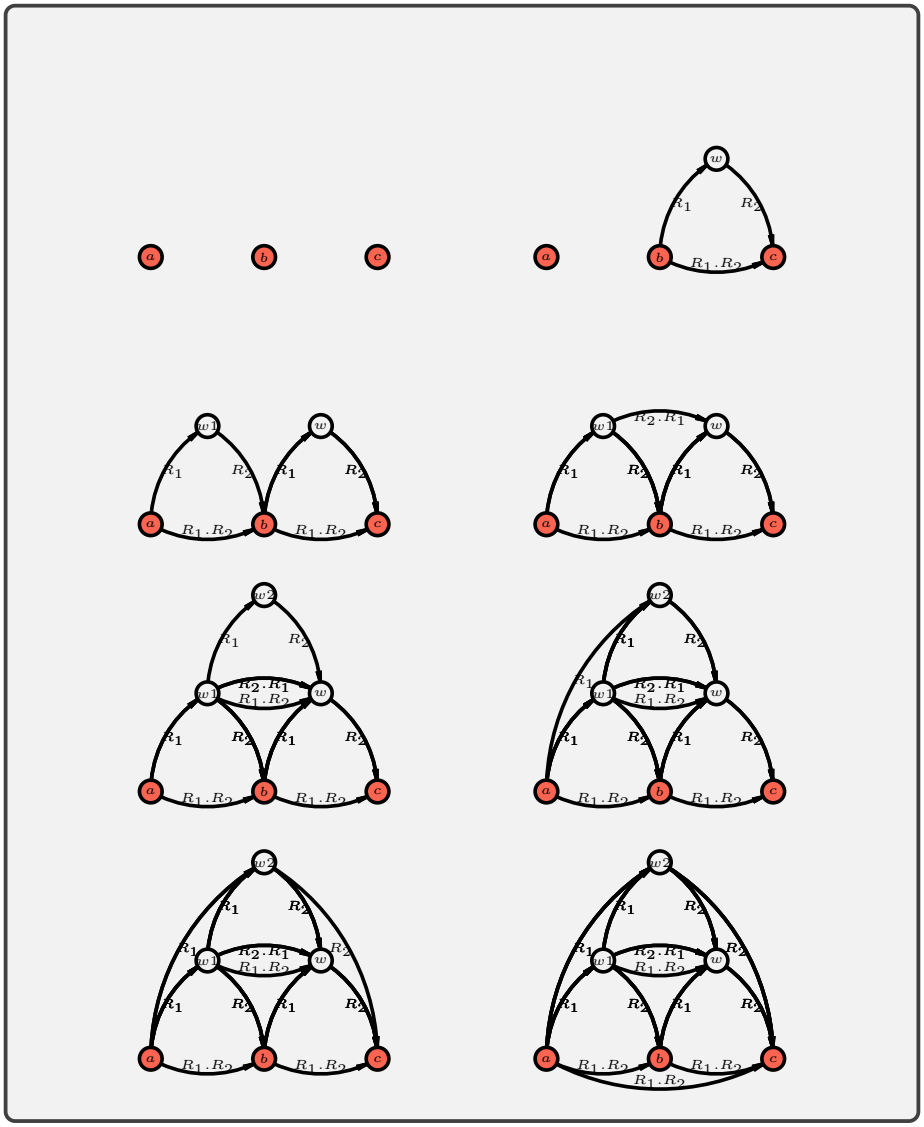
In Method 2, specific procedures for axioms or lemmas used in the proof are needed only if they introduce new objects and they need to specify only how the corresponding objects are created in the model (and not how they can be visualized). For instance, the following procedure for the axiom `defr1r2` creates a point Z such that it forms a regular triangle with the given points X and Y (the command "`rotate A B angle C`" constructs a point `A` such that it is the image of `C` under the rotation by angle of measure `angle` degrees about the point `B`.):

```
procedure defr1r2 { X Y Z }
{
  rotate Z X 60 Y
}
```

Finally, the following, automatically generated main file, invokes the file `predicates.gcl` that should be provided by the user and that should contain procedures for visualization of each predicate symbol. The main file also includes a file `goal_exists.gcl` and invokes a procedure for existence of initial objects. For each proof step, procedures for creating new objects (if there are such in that proof step) and for illustrating appearing facts are called. Here is a fragment of the code covering the first three steps of the illustration given below (the comments have been added manually to ease reading of the code):

```
include predicates.gcl
include goal_exists.gcl
% Call the procedure to create the initial objects called a, b and c
call goal_exists { a b c }

include defr1r2.gcl
% Call the procedure that creates w for given a and b
call defr1r2 { a b w }
% Draw and annotate w
cmark_t w

% Application of axiom (defr1r2):
% (! [X,Y] : (? [Z] : ((r1r2(X,Y)) => ((r1_r2_3(X,Z,Y))))))
layer 0
% Draw an arrow between a and b annotated by R1.R2
call draw_r1r2 { a b  0 }
% Draw an arrow between a and w annotated by  R1
call draw_r1 { a w  1 }
% Draw an arrow between w and b annotated by  R2
call draw_r2 { w b  1 }
layer 1
call draw_r1 { a w  2 }
call draw_r2 { w b  2 }

% Call the procedure that creates w1 for given b and c
call defr1r2 { b c w1 }
% Draw and annotate w1
mark_t w1
...
...
```

The sequence of illustrations can be viewed as animation within GCLC, and the images in the sequence are given below. We can notice that the automatically generated diagrammatic proof is very similar to the diagrammatic proof present in the literature [26, page 13].



6.2 Newmann's Lemma

Our second example is Newmann's lemma, a well known property of abstract term rewriting systems, stating that if a relation is terminating and locally confluent, then it is confluent. Huet has proposed a simple inductive proof [18]. Newmann's lemma has been formalized in Coq by Coquand and Huet as early as 1985 [12]. This automatic proof of this property was studied in the context of coherent logic by Bezem and Coquand [6]. As Bezem and Coquand, we use our coherent logic prover to reconstruct the induction step in Huet's inductive

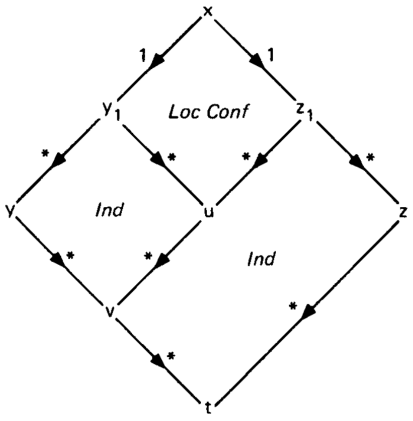


Figure 2: Huet's original diagram for the proof

proof of Newman's Lemma [18] and to generate a corresponding illustration. To our knowledge, it is the first diagram for this proof that is generated automatically. For comparison, in Figure 2 we show the diagram for the proof provided by Huet. Notice that in this, original diagram only steps 1, 2 and 4 of our automatically generated proof are depicted. The fact that R^* is transitive is implicit. To generate the diagrammatic proof, we only had to write GCLC procedures for illustrating the predicates, and procedures that introduce suitable new objects for local confluence and for induction hypothesis – a new node forms a parallelogram with the three given nodes (and display the labels "Local Confl" or "Ind" at the center of the parallelogram). For local confluence, we write the following procedure:

```
procedure localconfluence { A B C W }
{
  translate W A B C
  midpoint L B C
  printat L { Local Confl }
}
```

Using the following axioms:

Axioms:

1. localconfluence : $\forall XYZ\ r(X,Y) \wedge r(X,Z) \Rightarrow \exists T\ rstar(Y,T) \wedge rstart(Z,T)$
2. rstarrefl : $\forall XY\ X = Y \Rightarrow rstar(X,Y)$
3. rrstar : $\forall XY\ r(X,Y) \Rightarrow rstar(X,Y)$
4. rstardef2 : $\forall ABC\ r(A,B) \wedge rstar(B,C) \Rightarrow rstar(A,C)$
5. rstartrans : $\forall ABC\ rstar(A,B) \wedge rstar(B,C) \Rightarrow rstar(A,C)$
6. assump : $r(a,bp) \wedge rstar(bp,b) \wedge r(a,cp) \wedge rstar(cp,c)$
7. inductivehyp : $\forall ABC\ r(a,A) \wedge rstar(A,B) \wedge rstar(A,C) \Rightarrow \exists D\ rstar(B,D) \wedge rstar(C,D)$

the prover automatically generates the following proof:

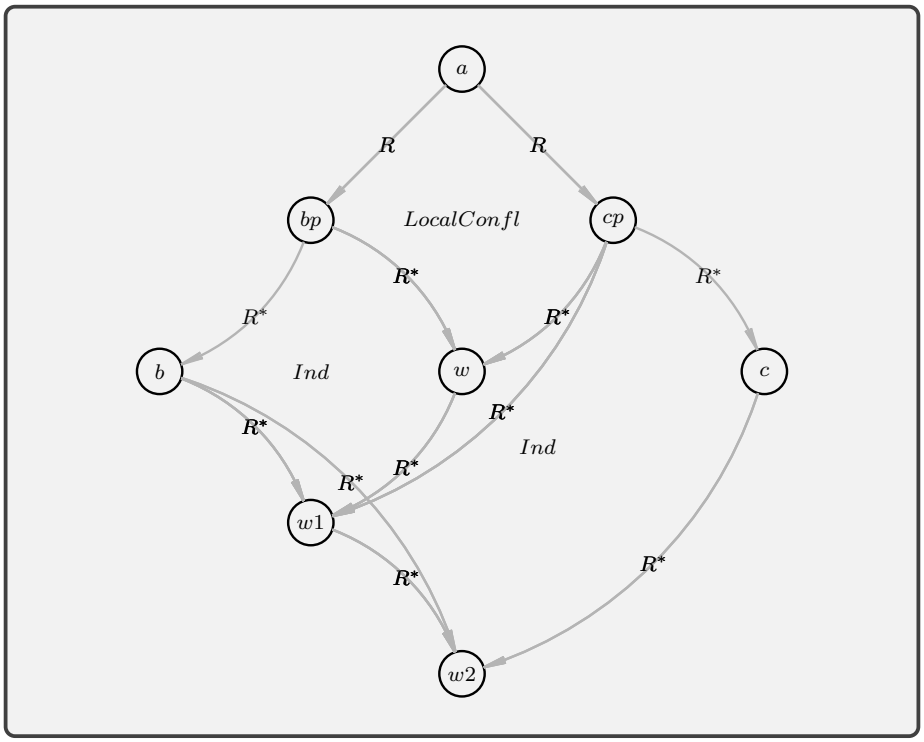
Theorem 3 *goal :* $\exists A\ rstar(b,A) \wedge rstar(c,A)$

Proof:

It should be proved that $\exists A\ \ rstar(b,A) \wedge rstar(c,A)$.

1. Let w be such that $rstar(bp,w) \wedge rstar(cp,w)$ (by MP, from $r(a,bp)$, $r(a,cp)$ using axiom localconfluence; instantiation: $X \mapsto a$, $Y \mapsto bp$, $Z \mapsto cp$)
2. Let $w1$ be such that $rstar(w,w1) \wedge rstar(b,w1)$ (by MP, from $r(a,bp)$, $rstar(bp,w) \wedge rstar(cp,w)$, $rstar(bp,b)$ using axiom inductivehyp; instantiation: $A \mapsto bp$, $B \mapsto w$, $C \mapsto b$)
3. $rstar(cp,w1)$ (by MP, from $rstar(bp,w) \wedge rstar(cp,w)$, $rstar(w,w1) \wedge rstar(b,w1)$ using axiom rstartrans; instantiation: $A \mapsto cp$, $B \mapsto w$, $C \mapsto w1$)
4. Let $w2$ be such that $rstar(c,w2) \wedge rstar(w1,w2)$ (by MP, from $r(a,cp)$, $rstar(cp,c)$, $rstar(cp,w1)$ using axiom inductivehyp; instantiation: $A \mapsto cp$, $B \mapsto c$, $C \mapsto w1$)
5. $rstar(b,w2)$ (by MP, from $rstar(w,w1) \wedge rstar(b,w1)$, $rstar(c,w2) \wedge rstar(w1,w2)$ using axiom rstartrans; instantiation: $A \mapsto b$, $B \mapsto w1$, $C \mapsto w2$)
6. $rstar(b,w2) \wedge rstar(c,w2)$ (by MP, from $rstar(b,w2)$, $rstar(c,w2) \wedge rstar(w1,w2)$ using axiom trivial)
7. Proved by assumption! (by QEDas)

The prover also generates a GCLC file that leads to the following illustration, similar to the one proposed by Huet. The symbol 1 in Huet's diagram corresponds to R in ours, $*$ to R^*, x to a, y to b, z to c, $y1$ to bp, $z1$ to cp, u to w, v to $w1$ and t to $w2$.



7 Comparison of the Two Methods: Varignon's theorem

In this section we compare the two presented methods in terms of code needed to be written by hand, using Varignon's theorem as an example. Varignon's theorem states that the quadrilateral formed by joining the midpoints of the four side of a quadrilateral is a parallelogram. Here we give a proof, found by the theorem prover, using the midpoint theorem, and the fact that a quadrilateral with opposite sides parallel is a parallelogram. This proof requires some non-degeneracy conditions. First, we list the most important axioms and lemmas (listed explicitly in the proof). $tP(A, B, C, D)$ denotes that the lines AB and CD are parallel and distinct, while $pG(A, B, C, D)$ denotes that the points A, B, C, and D form a parallelogram.

Axioms:

1. triangle_mid_par_strict : $\forall A \ \forall B \ \forall C \ \forall P \ \forall Q \ (\neg col(A,B,C) \wedge midpoint(B,P,C) \wedge midpoint(A,Q,C) \Rightarrow par(A,B,Q,P)$)
2. lemma_par_trans : $\forall A \ \forall B \ \forall C \ \forall D \ \forall E \ \forall F \ (par(A,B,C,D) \wedge par(C,D,E,F) \wedge \neg col(A,B,E) \Rightarrow par(A,B,E,F)$)
3. defparallelogram2 : $\forall A \ B \ C \ D \ (par(A,B,C,D) \wedge par(A,D,B,C)) \Rightarrow pG(A,B,C,D)$)

Theorem 4 $\forall A \ \forall B \ \forall C \ \forall D \ \forall I \ \forall J \ \forall K \ \forall L$
$(\neg col(B,D,A) \wedge \neg col(B,D,C) \wedge \neg col(A,C,B) \wedge \neg col(A,C,D) \wedge \neg col(I,J,K) \wedge$
$B \neq D \wedge A \neq C \wedge$
$midpoint(A,I,B) \wedge midpoint(B,J,C) \wedge midpoint(C,K,D) \wedge midpoint(A,L,D)$
$\Rightarrow pG(I,J,K,L)$)

Proof:

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

```
procedure th_varignon_exists { a b c d e f g h  }  {
  point a 5 5
  point b 25 7
  point c 30 28
  point d 2 24
  midpoint e a b
  midpoint f b c
  midpoint g c d
  midpoint h a d
  cmark_t a
  cmark_t b
  cmark_t c
  cmark_t d
  cmark_t e
  cmark_t f
  cmark_t g
  cmark_t h
  drawsegment a b
  drawsegment b c
  drawsegment c d
  drawsegment d a
}
```

We do not list the GCLC code generated by either method. Both methods require the manually created file and the procedure `th_varignon_exists`, like the one above, that illustrates the object given in the premises (for both methods, the alternative is to automatically prove that such object exist and obtain a corresponding illustration; in this case, that could require a number of axioms/lemmas).

In the following, we give the (complete) code needed by Method 1 (left) and by Method 2 (right). For Method 1, the user needs to provide illustrations for each axiom/lemma used (or to ensure that they are proved automatically, yielding the illustration on the way). In our example, there are three such axioms/lemmas. For Method 2, the user needs to provide illustrations for each predicate symbol and (in this concrete case) does not need to provide any specific illustration for the used axioms/lemmas (since they do not introduce new objects). So, it is sufficient for the user to provide the following GCLC code:

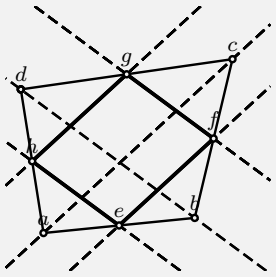
```
procedure triangle_mid_par_strict
    { A B C P Q } {
  drawdashline A B
  drawdashline Q P
}
```

```
procedure lemma_par_trans
    { A B C D E F } {
  drawdashline A B
  drawdashline E F
}
```

```
procedure defparallelogram2
    { A B C D } {
  linethickness -3
  drawsegment A B
  drawsegment C D
  drawsegment B C
  drawsegment A D
  normal
}
```

```
procedure draw_midpoint { a b c l }
{
   cmark c
}

procedure draw_nnncol { a b c l }
{
   drawdashline a b
}

procedure draw_par { a b c d l }
{
   drawdashline a b
   drawdashline c d
}

procedure draw_tP { a b c d l }
{
}

procedure draw_pG { a b c d l }
{
   linethickness -3
   drawsegment a b
   drawsegment c d
   drawsegment b c
   drawsegment a d
   normal
}
```

Both methods generate the same illustration:



As it can be seen above, the code to be written by hand is short using either method. However, the real benefit comes when one needs to prove and illustrate a number of theorems in a single theory. In such a case, the amount of manual labour needed for new theorems becomes very small. The user may have preferences over the available methods: if he/she wants some specific visual message for some axioms/lemmas, then Method 1 can be a better choice. If there is a large number of available and potentially used axioms/lemmas, then Method 2 can be a better choice.

8 Related Work

There is a huge number of sources addressing the role of figures in general, so we focus here only on a closely related work about the visualization of proofs in geometry which have either been generated automatically (by automated theorem proving) or machine checked (by interactive theorem proving).

The most significant contribution related to visualization of geometric proofs is the work of Ye et al. implemented in JGEX [40]. Ye et al. use the algebraic characterization of geometric predicates and triangulation of systems of polynomial equations to generate figures automatically for the statements. JGEX proposes several modes for illustration of proofs: illustration by animated diagrams, or animated diagrams with text, and they apply this method to two kinds of proofs: manually crafted proofs [39], or proofs generated automatically by the full-angle method or the deductive database method [38]. Their system provides interactive visualization using animations that follow proof steps as the user explore the proofs (within one integrated tool), while our illustrations are static images or can be viewed as sequences of images within the tool GCLC (which does not show corresponding proofs). The approach for illustrations generated by the full-angle method or the deductive database are similar to ours, but our approach is generic and not limited to geometry – it can be used for any coherent theory for which the user provides visualization for the axioms or for all predicates used.

Wilson and Fleuriot designed and implemented a tool for the visualization of proofs as direct acyclic graph with nodes depicted using geometric figures [36] for the full-angle method [9].

The above works are focused on the visualization of proofs. There is a larger number of approaches and systems for visualization of geometric statements. Bertot et al. [4] proposed to embed a dynamic geometry software into PCoq [5, 1] (PCoq is a user interface for the Coq proof assistant [11]). The second author implemented a prototype dynamic geometry software GeoProof which can export statements in the syntax of Coq [27]. Gao's MMP/geometer software, an automatic geometry theorem prover based on algebraic methods, offered a way to draw figures corresponding to the statements [16]. Wang proposed generating figures from a set of geometrical constraints by decomposing the system of polynomials into irreducible representative triangular sets, and finding an adequate numerical solution from each triangular set [35]. The first author implemented the GCLC software in which geometry configurations are described using a custom language and then illustrated, while conjectures about configurations can be expressed and then proved automatically by several available automated theorem provers [21, 22]. Pham and Bertot implemented a prototype connecting GeoGebra to PCoq [30]. Botana et al. implemented automated theorem provers based on algebraic methods within GeoGebra [8]. In these last two works, the initial configuration of the geometric statements can be visualized automatically using the dynamic geometry environment.

9 Conclusions and Perspectives

We presented a new approach for automated generation of illustrated proofs, primarily of geometry theorems. The approach, with its two variations, is simple, and it is a small extension to our prover for coherent logic, Larus. The approach is modular, as all illustrations rely only on individual visual interpretations of axioms used or of predicate symbols used. The approach is flexible, as one can provide (created by hand) different visual counterparts of the axioms and predicate symbols, but also of particular lemmas used within other proofs. The illustrations are generated in the GCLC language and they can also be viewed as animations, where proof steps are shown step-by-step. The generated illustrations can be exported to different image formats.

When Larus finds a proof, it can generate the corresponding Coq proof and illustration. In the future, this work could be extended to deal with coherent logic proofs which are not necessarily obtained automatically by Larus, but constructed interactively within Coq.

We also plan to improve the translation of proofs to text form, to generate more natural English text.

Funding No funding was received to assist with the preparation of this manuscript.

Conflicts of interest/Competing interests All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

The first author of this paper is a co-editor of the special issue this paper is submitted to, hence this submission is handled by the editor in chief.

Availability The implementation associated to this work is publicly available and open-source: https://github.com/janicicpredrag/Larus.

References

[1] Ahmed Amerkad, Yves Bertot, Loïc Pottier, and Laurence Rideau. Mathematics and Proof Presentation in Pcoq. In *Workshop Proof Transformation and Presentation and Proof Complexities in connection with IJCAR 2001*, Siena, June 2001.

[2] Jeremy Avigad, Edward Dean, and John Mumma. A Formal System for Euclid's Elements. *The Review of Symbolic Logic*, 2:700–768, 2009.

[3] Michael Beeson, Julien Narboux, and Freek Wiedijk. Proof-checking Euclid. *Annals of Mathematics and Artificial Intelligence*, 85(2-4):213–257, 2019. Publisher: Springer.

[4] Yves Bertot, Frédérique Guilhot, and Loïc Pottier. Visualizing Geometrical Statements with GeoView. *Proceedings of the Workshop User Interfaces for Theorem Provers 2003*, 103:49–65, 2004.

[5] Yves Bertot and Laurent Thery. A Generic Approach to Building User Interfaces for Theorem Provers. *The Journal of Symbolic Computation*, 25:161–194, 1998.

[6] Marc Bezem and Thierry Coquand. Newman's Lemma – a Case Study in Proof Automation and Geometric Logic. *Current trends in Theoretical Computer Science*, 2:267–282, 2004.

[7] Marc Bezem and Thierry Coquand. Automating Coherent Logic. In Geoff Sutcliffe and Andrei Voronkov, editors, *12th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning — LPAR 2005*, volume 3835 of *Lecture Notes in Computer Science*, pages 246–260. Springer-Verlag, 2005.

[8] Francisco Botana, Markus Hohenwarter, Predrag Janičić, Zoltán Kovács, Ivan Petrović, Tomás Recio, and Simon Weitzhofer. Automated Theorem Proving in GeoGebra: Current Achievements. *Journal of Automated Reasoning*, 55(1):39–59, 2015.

[9] Shang-Ching Chou, Xiao-Shan Gao, and Ji Zhang. Automated Generation of Readable Proofs with Geometric Invariants, II. Theorem Proving With Full-Angles. *Journal of Automated Reasoning*, 17(13):349–370, 1996.

[10] Chou Shang-Ching and Gao Xiao-Shan. A Survey of Geometric Reasoning Using Algebraic Methods. pages 97–119. Birkhäuser Boston, Boston, MA, 1996.

[11] Coq development team, The. *The Coq proof assistant reference manual, Version 8.3*. LogiCal Project, 2010.

[12] T. Coquand and Gérard Huet. Concepts mathematiques et informatiques formalises dans le calcul des constructions. Technical Report RR-0463, INRIA, December 1985.

[13] Raymond Duval and Marie-Agnès Egret. Introduction à la démonstration et apprentissage du raisonnement déductif. *Repères-IREM*, 12:114–140, July 1993.

[14] Roy Dyckhoff and Sara Negri. Geometrization of first-order logic. *The Bulletin of Symbolic Logic*, 21:123–163, 2015.

[15] M. Ganesalingam and W. T. Gowers. A fully automatic problem solver with human-style output. *CoRR*, abs/1309.4501, 2013.

[16] Xiao-Shan Gao and Qiang Lin. MMP/Geometer - A Software Package for Automated Geometric Reasoning. In *Proceedings of Automated Deduction in Geometry (ADG02)*, volume 2930 of *Lecture Notes in Computer Science*, pages 44–66. Springer-Verlag, 2004.

[17] Herbert Gelernter, J. R. Hansen, and Donald Loveland. Empirical explorations of the geometry theorem machine. In *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*, IRE-AIEE-ACM '60 (Western), pages 143–149, San Francisco, California, 1960. ACM.

[18] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[19] Predrag Janicic and Julien Narboux. Automated generation of illustrations for synthetic geometry proofs. In Predrag Janicic and Zoltán Kovács, editors, *Proceedings of the 13th International Conference on Automated Deduction in Geometry, ADG 2021, Hagenberg, Austria/virtual, September 15-17, 2021*, volume 352 of *EPTCS*, pages 91–102, 2021.

[20] Predrag Janičić and Julien Narboux. Theorem proving as constraint solving with coherent logic. *Journal of Automated Reasoning*, 2022. To appear.

[21] Predrag Janičić. GCLC — A Tool for Constructive Euclidean Geometry and More Than That. In Andrés Iglesias and Nobuki Takayama, editors, *Mathematical Software - ICMS 2006*, volume 4151 of *Lecture Notes in Computer Science*, pages 58–73. Springer, 2006.

[22] Predrag Janičić. GCLC 9.0/WinGCLC 2009. Manual for the GCLC Dynamic Geometry Software, 2009.

[23] Predrag Janičić. Geometry Constructions Language. *Journal of Automated Reasoning*, 44(1-2):3–24, 2010.

[24] Toshimitsu Karasawa. Historic Investigation of Legendre's Proof about the 5th Postulate of "Elements" for Reeducation of Mathematics Teacher. *Journal of Modern Education Review*, pages 926–931, December 2013.

[25] Nathaniel Miller. *A diagrammatic formal system for Euclidean geometry.* PhD thesis, Cornell University, May 2001.

[26] Julien Narboux. A formalization of diagrammatic proofs in abstract rewriting. working paper or preprint, 2006.

[27] Julien Narboux. A Graphical User Interface for Formal Proofs in Geometry. *Journal of Automated Reasoning*, 39(2):161–180, 2007.

[28] 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: How Intelligence can serve mathematical human learning.* Springer, 2021. In press.

[29] Hans de Nivelle and Jia Meng. Geometric Resolution: A Proof Procedure Based on Finite Model Search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2006.

[30] Tuan Minh Pham and Yves Bertot. A Combination of a Dynamic Geometry Software With a Proof Assistant for Interactive Formal Proofs. *Electron. Notes Theor. Comput. Sci.*, 285:43–55, September 2012.

[31] Andrew Polonsky. *Proofs, Types and Lambda Calculus*. PhD thesis, University of Bergen, 2011.

[32] Wolfram Schwabhäuser, Wanda Szmielew, and Alfred Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, Berlin, 1983.

[33] Sana Stojanović, Julien Narboux, Marc Bezem, and Predrag Janičić. A Vernacular for Coherent Logic. In StephenM. Watt, JamesH. Davenport, AlanP. Sexton, Petr Sojka, and Josef Urban, editors, *Intelligent Computer Mathematics*, volume 8543 of *Lecture Notes in Computer Science*, pages 388–403. Springer International Publishing, 2014.

[34] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[35] Dongming Wang. Automated Generation of Diagrams with Maple and Java. In Michael Joswig and Nobuki Takayama, editors, *Algebra, Geometry and Software Systems*, pages 277–287, Berlin, Heidelberg, 2003. Springer.

[36] Sean Wilson and Jacques D. Fleuriot. Combining Dynamic Geometry, Automated Geometry Theorem Proving and Diagrammatic Proofs. In *ETAPS Satellite Workshop on User Interfaces for Theorem Provers (UITP)*, Edinburgh, 2005. Springer.

[37] Daniel Winterstein. Dr.Doodle: A Diagrammatic Theorem Prover. In *Proceedings of IJCAR 2004*, 2004.

[38] Zheng Ye, Shang-Ching Chou, and Xiao-Shan Gao. Visually Dynamic Presentation of Proofs in Plane Geometry, Part 1 Basic Features and the Manual Input Method. *Journal of Automated Reasoning*, 45(3):213–241, October 2010.

[39] Zheng Ye, Shang-Ching Chou, and Xiao-Shan Gao. Visually Dynamic Presentation of Proofs in Plane Geometry, Part 2 Automated Generation of Visually Dynamic Presentations with the Full-Angle Method and the Deductive Database Method. *Journal of Automated Reasoning*, 45(3):243–266, December 2010.

[40] Zheng Ye, Shang-Ching Chou, and Xiao-Shan Gao. An Introduction to Java Geometry Expert. In *Post-proceedings of Automated Deduction in Geometry (ADG 2008)*, volume 6301 of *Lecture Notes in Computer Science*, pages 189–195. Springer-Verlag, 2011.