Improving QED-Tutrix by Automating the Generation of Proofs

Ludovic Font
École Polytechnique de Montréal
Montréal, Qc, Canada
ludovic.font@polymtl.ca

Philippe R. Richard
Université de Montréal
Montréal, Qc, Canada
philippe.r.richard@umontreal.ca

Michel Gagnon
École Polytechnique de Montréal
Montréal, Qc, Canada
michel.gagnon@polymtl.ca

The idea of assisting teachers with technological tools is not new. Mathematics in general, and geometry in particular, provide interesting challenges when developing educative softwares, both in the education and computer science aspects. QED-Tutrix is an intelligent tutor for geometry offering an interface to help high school students in the resolution of demonstration problems. It focuses on specific goals: 1) to allow the student to freely explore the problem and its figure, 2) to accept proofs elements in any order, 3) to handle a variety of proofs, which can be customized by the teacher, and 4) to be able to help the student at any step of the resolution of the problem, if the need arises. The software is also independent from the intervention of the teacher. QED-Tutrix offers an interesting approach to geometry education, but is currently crippled by the lengthiness of the process of implementing new problems, a task that must still be done manually. Therefore, one of the main focuses of the QED-Tutrix' research team is to ease the implementation of new problems, by automating the tedious step of finding all possible proofs for a given problem. This automation must follow fundamental constraints in order to create problems compatible with QED-Tutrix: 1) readability of the proofs, 2) accessibility at a high school level, and 3) possibility for the teacher to modify the parameters defining the "acceptability" of a proof. We present in this paper the result of our preliminary exploration of possible avenues for this task. Automated theorem proving in geometry is a widely studied subject, and various provers exist. However, our constraints are quite specific and some adaptation would be required to use an existing prover. We have therefore implemented a prototype of automated prover to suit our needs. The future goal is to compare performances and usability in our specific use-case between the existing provers and our implementation.

1 Introduction

Geometry is a delicate subject to teach. One could believe that, since it is a science, a student will learn solely by accepting the concepts and following the processes of the existing theoretical model, with no regards for the reality of the world of space and shapes. By also accepting mathematics as an activity, we can consider the teaching of geometry as the development of an intuition about geometry, by the interaction between the theoretical model and the reality. In other words, "doing geometry" means working inside the theoretical model, by accepting all the codes and axioms, but "learning geometry" means developing this geometrical intuition of the way the theoretical model represents reality. In the light of this consideration, solving a demonstration problem, with or without technological tools, both develops the geometrical sense of the student and allows him to do his mathematician's work, which are the ultimate goal of mathematics education.

The theory of Mathematical Working Spaces (MWS) considers the outset that mathematics is both a science and an activity [37]. In a learning context, when the student does his mathematician's work, this dual perspective allows to interpret the development or the manifestation of geometric sense following **three geneses** at play between the epistemological and cognitive plans (see Fig. 1): **instrumental**, **semiotic** and **discursive**. In a few words, the first one represents the familiarization with the tools, either ruler and compass, or software, and the use of them. The second one represents the association between mathematical concepts and processes, and their formal or systemic representation, such as symbols, semiotic representation systems and the technical mathematical lexicon. The third one represents the articulation of mathematical concepts and processes to form a proof, as the reasoning or calculations that proceed by discursive-graphic expansion [51]. These geneses allow us to classify crucial points, such as the creation of meaning, validation of properties, or usage of technological tools, in terms of genesis coordination in the model of MWS. For instance, a student solving a demonstration problem using software has, at a point, to identify the hypotheses and conclusion of the problem, including 1) understanding how the demonstration is articulated around them (discursive aspect), 2) finding the terms of statements carrying mathematical meaning (semiotic aspect), and 3) understanding how to manipulate, transform and control these mathematical elements through the software (instrumental aspect).

For the intelligent tutor software developer who wants to integrate users (students or teachers) very early in the design process, the question of respect for human learning is a challenge at all times. When planning the possible interactions between the user and the machine, the developer has to know beforehand the hypotheses and conclusion, that are already part of a well-constructed, mechanically explorable proof. As a consequence, when the goal is to maximize the educational value of the software, it is necessary to introduce from the very beginning some cooperation between the experts of different domains (computer science, education science, cognitive science, user interface, etc.) by following collaboration-oriented research methods.

This requirement of integrating users as soon as the software is designed is crucial. Furthermore, in the theory of mathematics education situations [17], and most specifically considering the notion of **didactical contract**, we consider that both student and teacher have reciprocal responsibilities regarding the knowledge devolution. The understanding of these various responsabilities is fundamental for the development of an educational software. There are some explicit rules for knowledge sharing, such as a precise way of writing mathematics, the propriety used to justify the passage from one expression to another, etc., but most are implicit, being more a consequence of a class habit than a necessary constraint. Typically, in a deductive logic, the teacher can accept several inferential shortcuts, by accepting demonstrations that are not completely rigorous, in order to smoothen the flow of the resolution or improve its pedagogic efficiency. However, by doing so, he changes the operational structure based on definitions and properties in a given axiomatic. Besides, the student will build his own shortcuts while he gets used to the mathematics deductive paradigm. Both the question of the readability of proofs and the acceptability of an inferential shortcut are closely related to the didactical contract, i.e. mathematics as an activity.

In this paper, we present our work on the intelligent tutor software, QED-Tutrix or QEDX (abbreviated acronym), that aims at providing a technological tool to improve the learning of geometry in highschool, by offering an interface to solve problems of proof while staying closely attached to the didactical contract. More specifically, we present the problematic of the **readability** and **accessibility** of proofs, and the **adaptability** of proofs to inferential shortcuts. These three constraints are the basis for the development of a system to automatically generate proofs, therefore easing considerably the task of adding new problems to QED-Tutrix.

In Section 2, we present systems similar to QED-Tutrix, and also existing tools for automated theo-

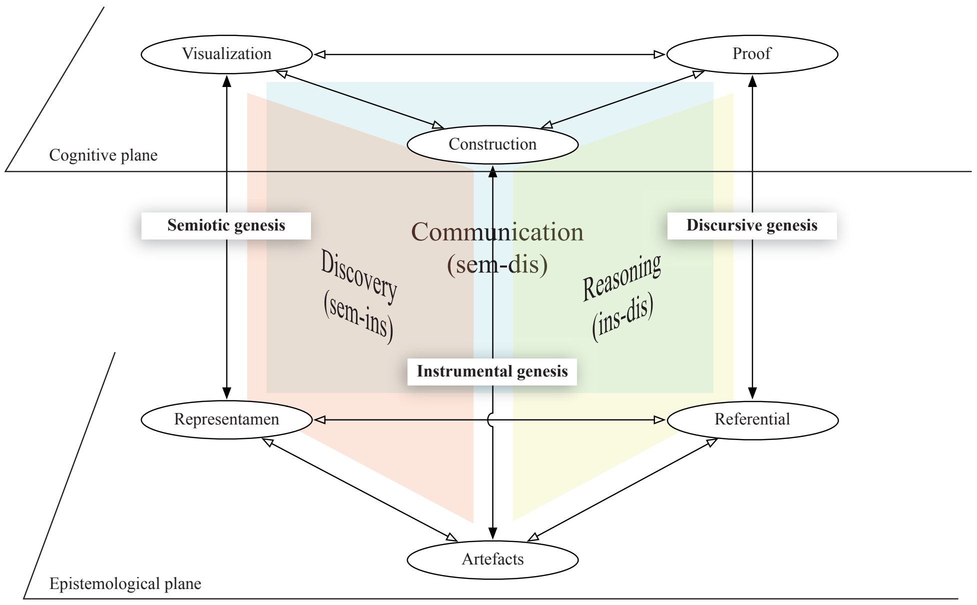


Figure 1: The model of mathematical working space in the theory.

rem proving that could be useful for the task of generating proofs. In Section 3, we provide a brief explanation of the functionalities of QED-Tutrix and its internal handling of proofs. Then, in Section 4, we present our goals for the improvement of QEDX and the needs for such improvements, more specifically our need for an automated proof generator. We give a quick overview of the results of our preliminary work on a custom proof generator in Section 5, and discuss the limits of our work in Section 6. Finally, we present other avenues of research and conclude in Section 7.

2 Related Work

This paper presents a project of improvement of an existing software, QED-Tutrix, in its capacity as an intelligent tutoring system, which allows the student to do his work as a mathematician when solving problems of proof in geometry. As a consequence, most of the related work has already been discussed in great detail in the theses of Nicolas Leduc [38] and Michèle Tessier-Baillargeon [56]. In Section 2.1, we summarize the important elements and, in Section 2.2, discuss the state-of-the-art for the novel problem of automated deduction.

2.1 Tutor softwares

In [38], Nicolas Leduc analyzed the existing solutions for learning mathematics. He first identified non-tutor systems, divided in four groups: tools for autonomous learning; tools modeling the learning path and curriculum; micro-worlds; and tools for automated proving.

Tools for autonomous learning, typically websites providing answers to specific questions, such as Mathway [3] or private tutoring companies. These are a fantastic knowledge bases, but are either passive tools or non-automated.

Learner modeling, the student is guided on a learning path and his knowledge is taken into account when giving him new content. Examples include one of the first such systems, ELM-ART [58] for learning LISP, and ALEKS [26], ActiveMath [44] and Wayang Outpost [11] for mathematics specifically. These require a modelization of the user's knowledge, that can be done using various techniques, such as fuzzy logic [31] or Knowledge Components [6]. These systems are based solely on the problems solved, and not on the way the problem was solved, which is one of the foci of QED-Tutrix.

Micro-worlds, the student, unlike in the paper-pencil environment, can manipulate a dynamic geometrical figure, following certain rules, such as Euclide's axioms. A typical example is GeoGebra [29], a popular dynamic geometry software with an open source code and an active community. These tools are usually dependent on the presence of the teacher, since they offer little to no control over the student's actions, and no help towards the resolution of a problem.

Automated proof, these systems allow to verify statements, or discover new facts. These systems are discussed in detail in Section 2.2.

In a second part, he analyzed in detail 10 tutoring systems for geometry. Each of these systems offers interesting characteristics, but none combines them all. The goal for QEDX is to offer:

- an interface to explore the problem by allowing the student to freely manipulate the figure;
- a liberty in the construction of the proof, allowing the student to construct his proof in any order;
- a handling of all possible proofs, acceptable at a high-school level;
- a tutor system to help the student, based on the identification of the step of the proof on which he is working;
- autonomy from the teacher, allowing an unsupervised use by the students.

In the remaining of this section, we provide a short summary of the systems analyzed and explain their shortcomings.

One of the first tutor system developed for geometry is GeometryTutor. It is based on a solid theoretical model, the ACT-R cognitive theory [9, 10]. However, it does not allow the student to explore the problem outside of the rigorous path identified by the software.

A few years later, the PACT Geometry Tutor has been developed [4, 5], that evolved into Geometry Cognitive Tutor [8, 7, 55]. This system is limited to problems based in cartesian coordinates, since it handles elements numerically. In QEDX, we want to be able to handle any geometry problem.

To include proofs that require an additional construction, the Advanced Geometry tutor was developed by Matsuda [43], based on the GRAMY theorem prover [42]. This system has interesting characteristics, since it is one of the few to handle proofs with intermediate constructions. However, it is quite rigid on the accepted proof and does not allow the student to explore the problem or use a proof that is not the optimal proof calculated by the prover.

The software ANGLE [33, 35], unlike the previous ones, aims at helping the student to construct his proof. It is based on the Diagram Configurations [34] theory, based on interesting configurations of the geometrical figure, used by the experts to produce the proof. It gives the student freedom to explore the solutions. However, the Diagram Configurations are modeled on the work of experts, which can be quite far from the formal geometry taught in highschool. Besides, even though the student can explore the problem, he has no interface to manipulate the figure.

An interesting approach is the Baghera system [12, 57], providing a web platform. This system offers two sides: one for the teacher, where he can create new problems and follow in real-time or replay the progress of the student; and one for the students, which can chose and solve problems. The weakness of this system when compared to our goals is that it offers no automated tutor.

To provide interactive figure manipulation, it is a logical step to use an interactive geometry software. Two systems are based on the Cabri software [13, 36], Cabri-DEFI [41] and Cabri-Euclide [39, 40]. The first one helps the student to plan his proof by asking him questions about the figure that ultimately direct him towards a proof. It is an interesting approach, but it offers no freedom of exploration, since it is the system that asks questions. Besides, there is no tutor system to help the student when he is stuck in his resolution. The second one goes in the opposite direction, by allowing the student to explore freely and enter conjectures, that are later organized in a graph. However, there is no help provided to the student and no mechanism to ensure that the problem is ultimately solved, which can be an issue for unsupervised use.

The system Mentoniezh [47, 48, 49] offers a novel approach by dividing the proof in four steps: understanding the problem; exploration of the figure; planing of the proof; and redaction. The software helps and directs the student during each step. The division of the proof in steps is one of the foundation of QEDX. However, the software provides no help to find the next proof element, and a student can therefore encounter an impasse in his resolution that will force him to ask the teacher for help.

Another system dividing the proof in steps is Geometrix [2], where the student can first construct a figure, and then allows him to create a problem based on that figure and to solve it. It therefore allows the creation of exercises by the teachers, including customized error messages to help the student. However, it remains mainly a demonstration assistant, and offers little in the tutoring aspect.

Finally, the Turing system [24, 52], that largely inspired QEDX, provides an interface for the student that allows him to manipulate a dynamic figure, and to provide statements to construct his proof, in any order. The integrated tutor system analyzes his input and gives feedback depending on the validity of the statement. After a period of inactivity, the tutor gives him hint to restart his resolution process. However, the hint is based only on the last element provided by the student, which may not be the step on which he is currently working.

Overall, all of these systems, except ANGLE, are based on fully formal geometry, which limits the number of acceptable proofs, even though less formal proofs are typically accepted by the teachers. ANGLE is based on a model of the reasoning of experts, which is quite far from the proofs used in class. Besides, only Mentoniezh keeps the previous work of the student in memory, but does not use it to provide hints towards the next step. The systems that provide hints are based on forward or back-tracking, limiting their usefulness. Finally, no system allows the student to explore different proof paths at the same time. This illustrates the need that gave birth to the QEDX project.

2.2 Automated theorem proving

The main issue faced currently by QED-Tutrix is the difficulty of the task of adding new problems. Indeed, to help the student in his resolution, one must know the element(s) of the proof on which he's working, which requires a knowledge of all possible proofs accessible for the student at a given point in the school curriculum. Therefore, the software must internally handle a representation of these. This information is stored in the form of inference graphs (see Section 3.2). These graphs can be quite large even for simple problems, and we currently have to construct them manually, hence the need for a tool to automatically find all possible proofs. However, we have three fundamental constraints:

- the proofs must be **readable**;
- they must use only properties available at a **high-school level**;
- there must be a way to handle the **inferential shortcuts**, i.e. the inference chains that can be deemed too formal by some teachers and are therefore skipped in a demonstration.

These three points are detailed in Section 5.1, along with an example of an inferential shortcut.

The constraints direct our search for a way to automatically find proofs. Indeed, there currently exist two general research avenues for geometry automated theorem provers (GATP): algebraic methods and synthetic, or axiomatic, methods. The first one is based on a translation of the problem into some form of algebraic resolution, and the second one uses an approach closer to the natural, human way of solving problems, by chaining inferences.

One of the main goals of the research community in automated theorem proving is the performance. Since synthetic approaches are typically slower, most solvers are based on an algebraic resolution. Algebraic methods include the application of Gröbner bases [18, 32], Wu's method [20, 59] and the exact check method [60]. Practical applications include the recent integration of a deduction engine in GeoGebra [14], which is based on the internal representation of geometrical elements in complex numbers inside GeoGebra. Other examples include the systems based on the area method [15, 30], the full-angle method [21, 22], and many others. These systems seldom provide readable proofs, and when they do, they are far from what a high-school student would write. Given our readability and accessibility goals, all these systems are not relevant to our interests.

For this reason, we focus on synthetic methods. A popular approach is to use Tarski's axioms, which have interesting computational properties [16, 45]. However, the geometry taught in high-school is based on Euclide's axioms, which are not trivially correlated to Tarski's. Therefore, proofs based on Tarski's axioms are quite inaccessible for high-school students, violating our second constraint.

A prover that has very similar goals is GRAMY [42]. It is based solely on Euclidean geometry, with an emphasis on the readability of proofs. Besides, it has been developed as a tool for the Advanced Geometry Tutor, that has been presented in Section 2.1. It is therefore able to generate all proofs for the given problem. Finally, one of its major strengths is the ability to construct geometrical elements.

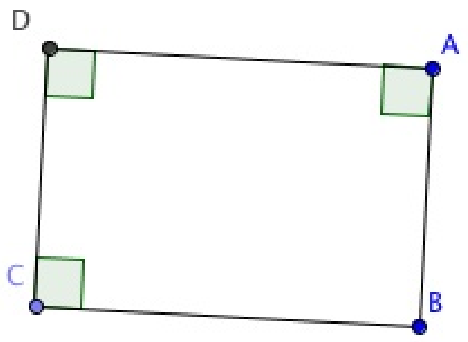
To the best of our knowledge, GRAMY is the only work close to our goals. It is therefore one of the basis for our work on automated generation of proofs. The only limit to the compatibility of the two systems is our third constraint: the axioms used by our system must be flexible and easily changed by the teacher to suit his personal teaching style and to follow the evolution of the axioms taught in class throughout the year.

Overall, given our very specific needs dictated by the focus on educational interest, we identified only one system, GRAMY, that would be suitable. However, since its code is closed and no work has been done on it since 2004, we chose to develop a custom engine inspired by it. This will allow us to integrate it directly to QED-Tutrix.

3 Presentation of QED-Tutrix

The QED-Tutrix software, or QEDX, has benefited from the technological and scientific achievements of its precursors, notably the geogebraTUTOR and Turing systems. It aims at providing an interface in which the students can solve geometrical proof problems, such as the one in Fig. 2.

It is first and foremost a research project, with the central goal of adapting the technology to the student, instead of adapting the student to the technology. This project is therefore conducted by a close cooperation between researchers in mathematics education and in information technology, and every step of its development included a profound reflexion on the educational value of the software.



“Prove that any quadrilateral with three right angles is a rectangle.”

Figure 2: An example of a simple geometrical proof problem.

3.1 Conception guidelines

To ensure the educational interest, we attempted to follow several guidelines. The most general one is to adapt the software to a typical student’s thought process. In other words, to reduce as much as possible the work that has to be done to understand and use QEDX. This way, it becomes a tool that can be used almost intuitively to write proofs. This required several test sessions in class, during which the behavior and reactions of the students were closely monitored. These studies are explained in detail in the work of Michèle Tessier-Baillargeon [56] .

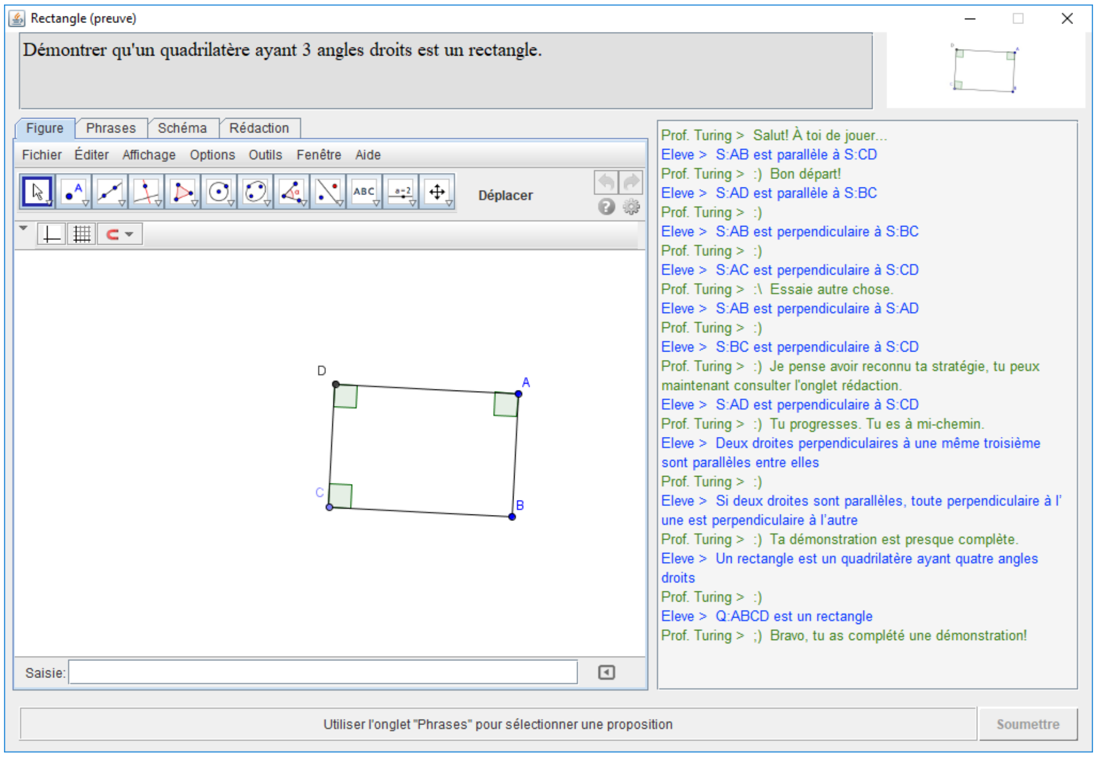
Since QEDX is a software for solving proof problems, we based our work on the way a typical student solves such problems. Usually, the resolution of a proof problem implies three steps:

- **Exploration**, during which the student gets acquainted to the problem and searches for ideas, without a plan on the exact proof he is going to write;
- **Construction** of the proof, during which the student finds out which elements of the class (theorems or lemmas, for example) can be used to materialize his ideas;
- **Redaction** of the proof, during which the student organizes his ideas to write a formal proof.

These steps are not usually followed linearly. A student will often try a proof, to realize during its redaction that it does not work, or that an inference is missing somewhere, sending him back to the exploration.

The interface of QED-Tutrix, represented in Fig. 3 reflects this reality. It is composed of four sections:

- The topmost is simply the statement of the problem, including a figure.
- The left is the core of the communication from the student to QEDX. It is composed of three tabs, plus one currently not implemented.
- The right is the text box containing the artificial conversation going on between the tutor, embodied by Prof. Turing, and the student. We explain the role of this conversation later in this section.
- The bottom is the input interface, where the student can complete the statement he wants to put in his proof.

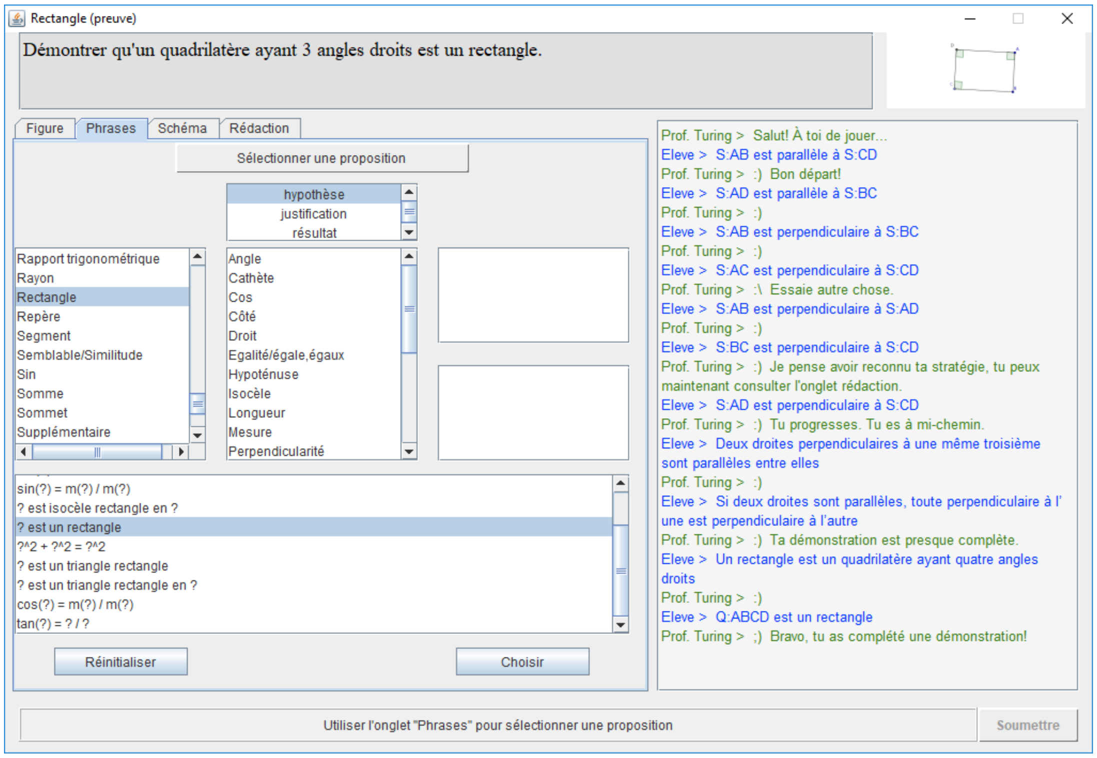


Démontrer qu'un quadrilatère ayant 3 angles droits est un rectangle.

Prof. Turing > Salut! À toi de jouer...
Eleve > S:AB est parallèle à S:CD
Prof. Turing > :) Bon départ!
Eleve > S:AD est parallèle à S:BC
Prof. Turing > :)
Eleve > S:AB est perpendiculaire à S:BC
Prof. Turing > :)
Eleve > S:AC est perpendiculaire à S:CD
Prof. Turing > :\ Essaie autre chose.
Eleve > S:AB est perpendiculaire à S:AD
Prof. Turing > :)
Eleve > S:BC est perpendiculaire à S:CD
Prof. Turing > :) Je pense avoir reconnu ta stratégie, tu peux maintenant consulter l'onglet rédaction.
Eleve > S:AD est perpendiculaire à S:CD
Prof. Turing > :) Tu progresses. Tu es à mi-chemin.
Eleve > Deux droites perpendiculaires à une même troisième sont parallèles entre elles
Prof. Turing > :)
Eleve > Si deux droites sont parallèles, toute perpendiculaire à l' une est perpendiculaire à l'autre
Prof. Turing > :) Ta démonstration est presque complète.
Eleve > Un rectangle est un quadrilatère ayant quatre angles droits
Prof. Turing > :)
Eleve > Q:ABCD est un rectangle
Prof. Turing > ;) Bravo, tu as complété une démonstration!

Utiliser l'onglet "Phrases" pour sélectionner une proposition

Figure 3: The main interface of QED-Tutrix.



Démontrer qu'un quadrilatère ayant 3 angles droits est un rectangle.

Rapport trigonométrique
Rayon
Rectangle
Repère
Segment
Semblable/Similitude
Sin
Somme
Sommet
Supplémentaire

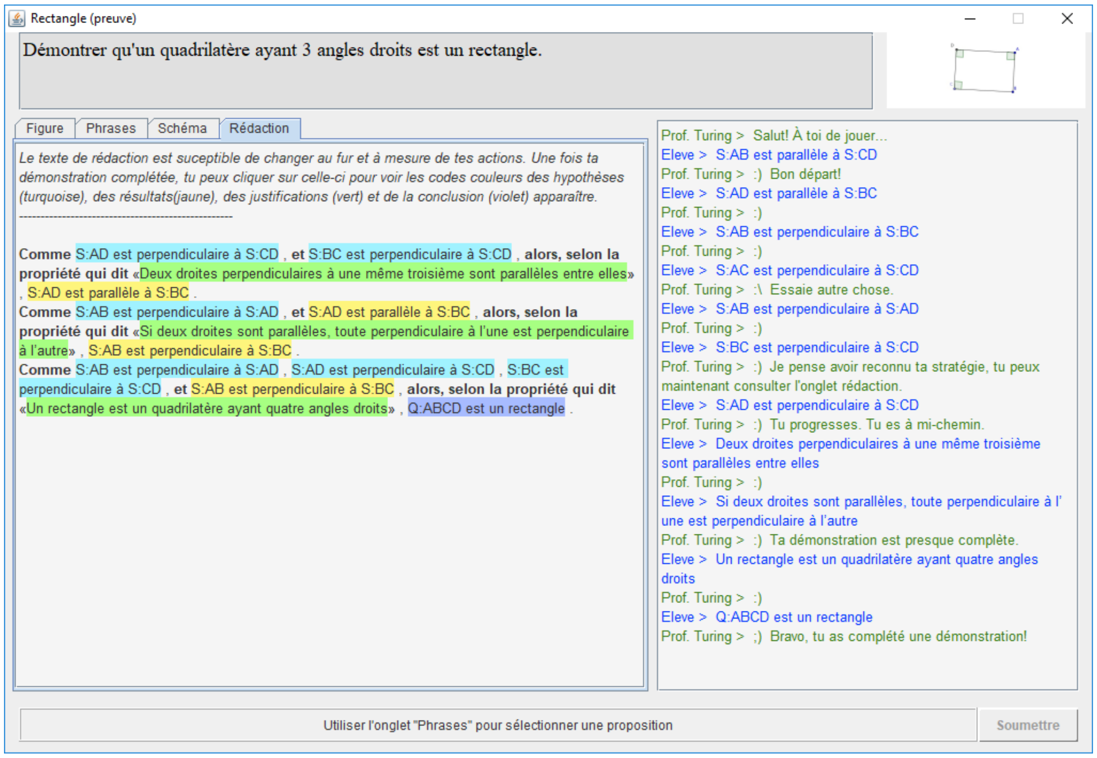
Angle
Cathète
Cos
Côté
Droit
Egalité/égale,égaux
Hypoténuse
Isocèle
Longueur
Mesure
Perpendicularité

sin(?) = m(?) / m(?)
? est isocèle rectangle en ?
? est un rectangle
?^2 + ?^2 = ?^2
? est un triangle rectangle
? est un triangle rectangle en ?
cos(?) = m(?) / m(?)
tan(?) = ? / ?

Prof. Turing > Salut! À toi de jouer...
Eleve > S:AB est parallèle à S:CD
Prof. Turing > :) Bon départ!
Eleve > S:AD est parallèle à S:BC
Prof. Turing > :)
Eleve > S:AB est perpendiculaire à S:BC
Prof. Turing > :)
Eleve > S:AC est perpendiculaire à S:CD
Prof. Turing > :\ Essaie autre chose.
Eleve > S:AB est perpendiculaire à S:AD
Prof. Turing > :)
Eleve > S:BC est perpendiculaire à S:CD
Prof. Turing > :) Je pense avoir reconnu ta stratégie, tu peux maintenant consulter l'onglet rédaction.
Eleve > S:AD est perpendiculaire à S:CD
Prof. Turing > :) Tu progresses. Tu es à mi-chemin.
Eleve > Deux droites perpendiculaires à une même troisième sont parallèles entre elles
Prof. Turing > :)
Eleve > Si deux droites sont parallèles, toute perpendiculaire à l' une est perpendiculaire à l'autre
Prof. Turing > :) Ta démonstration est presque complète.
Eleve > Un rectangle est un quadrilatère ayant quatre angles droits
Prof. Turing > :)
Eleve > Q:ABCD est un rectangle
Prof. Turing > ;) Bravo, tu as complété une démonstration!

Utiliser l'onglet "Phrases" pour sélectionner une proposition

Figure 4: The main interface of QED-Tutrix, on the “Sentence” tab.


Rectangle (preuve)

Démontrer qu'un quadrilatère ayant 3 angles droits est un rectangle.

Le texte de rédaction est suceptible de changer au fur et à mesure de tes actions. Une fois ta démonstration complétée, tu peux cliquer sur celle-ci pour voir les codes couleurs des hypothèses (turquoise), des résultats(jaune), des justifications (vert) et de la conclusion (violet) apparaître.

Comme S:AD est perpendiculaire à S:CD , **et** S:BC est perpendiculaire à S:CD , **alors, selon la propriété qui dit** «Deux droites perpendiculaires à une même troisième sont parallèles entre elles» , S:AD est parallèle à S:BC .
Comme S:AB est perpendiculaire à S:AD , **et** S:AD est parallèle à S:BC , **alors, selon la propriété qui dit** «Si deux droites sont parallèles, toute perpendiculaire à l'une est perpendiculaire à l'autre» , S:AB est perpendiculaire à S:BC .
Comme S:AB est perpendiculaire à S:AD , S:AD est perpendiculaire à S:CD , S:BC est perpendiculaire à S:CD , **et** S:AB est perpendiculaire à S:BC , **alors, selon la propriété qui dit** «Un rectangle est un quadrilatère ayant quatre angles droits» , Q:ABCD est un rectangle .

Prof. Turing > Salut! À toi de jouer...
Eleve > S:AB est parallèle à S:CD
Prof. Turing > :) Bon départ!
Eleve > S:AD est parallèle à S:BC
Prof. Turing > :)
Eleve > S:AB est perpendiculaire à S:BC
Prof. Turing > :)
Eleve > S:AC est perpendiculaire à S:CD
Prof. Turing > :\ Essaie autre chose.
Eleve > S:AB est perpendiculaire à S:AD
Prof. Turing > :)
Eleve > S:BC est perpendiculaire à S:CD
Prof. Turing > :) Je pense avoir reconnu ta stratégie, tu peux maintenant consulter l'onglet rédaction.
Eleve > S:AD est perpendiculaire à S:CD
Prof. Turing > :) Tu progresses. Tu es à mi-chemin.
Eleve > Deux droites perpendiculaires à une même troisième sont parallèles entre elles
Prof. Turing > :)
Eleve > Si deux droites sont parallèles, toute perpendiculaire à l'une est perpendiculaire à l'autre
Prof. Turing > :) Ta démonstration est presque complète.
Eleve > Un rectangle est un quadrilatère ayant quatre angles droits
Prof. Turing > :)
Eleve > Q:ABCD est un rectangle
Prof. Turing > ;) Bravo, tu as complété une démonstration!

Utiliser l'onglet "Phrases" pour sélectionner une proposition
Soumettre

Figure 5: The main interface of QED-Tutrix, on the “Redaction” tab.

The three functional tabs of the left section are associated with the three steps for writing a proof. Indeed, the first tab, “Figure”, is the one present in Fig. 3. It uses a GeoGebra plugin that allows the student to freely manipulate the figure, for example by measuring angles or adding lines, and therefore **exploring** the problem. GeoGebra is a popular tool for dynamic geometry that contains many functionalities, and is therefore an excellent way for the student to experiment. Besides, its popularity ensures that many students will be familiar with it.

The second tab, “Sentences” (“Phrases” in French), shown in Fig. 4, allows the student to **construct** his proof by selecting mathematical statements or properties. This selection is done by four narrowing lists of mathematical topics: the first one is fixed and contains all the topics accessible to the student, then the second, third and fourth adapt to the choice made in the first one by narrowing the options. At every step, the bottom window contains a list of statements compatible with the selected topics. When his choice is made, the bottom of the interface allows him to choose the geometrical element(s) concerned. For instance, in the situation depicted in Fig. 6, if he wants to indicate the result “AH is the height of triangle ABC through A”, he can start by selecting “Triangle” in the first list of topics. Then, in the second list, he selects “Height”. This combination of topics allows the statement “? is the height of ? through ?” to appear in the bottom list. He clicks on it, then fills the fields that appears at the bottom with “AH”, “ABC” and “A”.

Finally, the third tab, “Redaction”, shown in Fig. 5, provides a fully written rigorous proof. This tab is accessible only when the software is able to determine that the student has provided enough elements (arbitrarily fixed at 50%) of any proof. The full formal proof, however, contains only the elements already entered by the student: the remaining ones are blanks. This helps the student in the **redaction** of

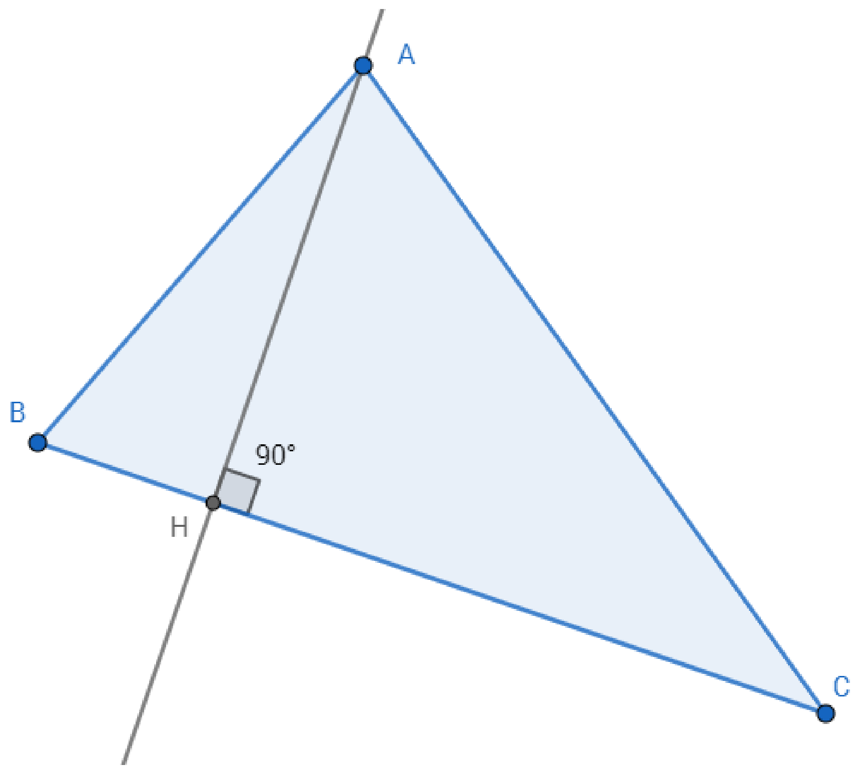


Figure 6: A geometrical construction of the height of a triangle.

his proof, by directing him towards the missing elements.

Overall, the typical usage of our software consists in a constant navigation between the tabs, switching between exploration, construction and redaction. In this aspect, QED-Tutrix embraces the reasoning of its user.

A crucial element of the software, that has not been covered yet, is the tutor interface. Indeed, the goal of QEDX is not simply to provide an interface to solve problems, but also to be an intelligent tutor, helping the student throughout his resolution of the problem.

Our tutoring system works by finding out the proof the student is trying to write. The software has an internal representation of all possible proofs for a given problem, and finding the one is simply a question of calculating the proof for which the student has entered the most elements. More details on the internal representation and exploration of proofs are given in the following section.

This calculation is done dynamically, every time the student enters a new element into the software. This way, when the student reaches an impasse in his resolution, the software identifies an element of his proof that is yet missing and directs him towards it by printing a customized message in the right panel. An example of such a message could be:

“What are the definitions of the height of a triangle ?”

If the student does not find anything after several advices, the software tries to direct him towards another missing element. If this is still not enough to exit the impasse, we consider that the student is either completely stuck or has become distracted, and the tutor asks the student to consult his teacher.

The interest of this system is to allow all students of a class to obtain a personalized help during the entirety of the problem resolution, compared to a typical class format where the teacher cannot physically help all his students at the same time. Even though the help provided by the software is more crude and mechanical than the one provided by a teacher, that has experience and intuition, the combination of both seems like an excellent way to help the students in their learning process.

3.2 Internal engine

To allow this versatility, we introduced some novel elements in QEDX' core. First, for it to run on as many system as possible, QED-Tutrix is coded in Java. Furthermore, it allowed us to use the GeoGebra API [1] to provide our exploration interface very simply.

A central element of our engine is the **HPDIC graph**, standing for Hypothesis, Property, Deduction, Intermediate result, Conclusion. In a few words, this graph contains all possible proofs for a given problem, in a set of axioms. This set of proofs evolves as the students can (and should) use more advanced properties, opening the possibility for more diverse proofs. We therefore consider by default the maximum graph, or **complete graph**, containing all proofs that could be used at any point in the high-school curriculum. If we see a proof as a succession of inferences (hypotheses + property $\rightarrow$ conclusion), then a proof is a directed graph, whose starting nodes are the hypotheses, and finishing node the conclusion. Then, to represent all the proofs of a problem, we fuse these into a larger graph. For example, given the problem in Fig. 2, *"Prove that any quadrilateral with three right angles is a rectangle"*, its graph would be the one in Fig. 7. In such a graph, a proof is a subgraph in which every "intermediate result" node has only one parent, i.e. one justification. This figure is taken from a previous paper [54], where we first discussed the possibility of including connected problems in QEDX.

During the resolution of a problem by a student, every input, either statement or property, must be present on the graph, otherwise we consider that it is not part of the proof. At the beginning of the resolution, all nodes of the graph are "unchecked". When the student adds an element, it is "checked" on the graph. This way, identifying the proof he is working on is a matter of checking which subgraph has the highest proportion of checked nodes.

This check, however, must be done by exploring the space of all possible proofs, i.e. of all subgraphs from the hypotheses to the conclusion. In the rectangle problem, whose graph is represented in Fig. 7, this number remains small. However, this problem has an unusually simple graph. In another problem that is not much more complicated, shown in Fig. 8, the HPDIC graph has several thousands nodes, for over five million possible solutions. This makes the search of subgraphs impossible in real-time, especially on computers typically used in class. Therefore, to avoid this issue, the graph is converted into a series of tree. This operation also simplifies the creation of the space of possible proofs by removing cycles. The exact process is described in the thesis of Nicolas Leduc [38]. This transformation is lengthy, but only has to be done once, and allows the software to explore the set of trees in real-time, rendering it possible to find, at every instant, the proof the student is likely working on, and to provide an instant feedback.

4 Goals for the improvement of QED-Tutrix

Even though QED-Tutrix is currently functional and already provides interesting educative features, we identified many improvement avenues. The most research-oriented one is drawn from the following observation: when a student is in an impasse in a problem resolution, most teachers help the student

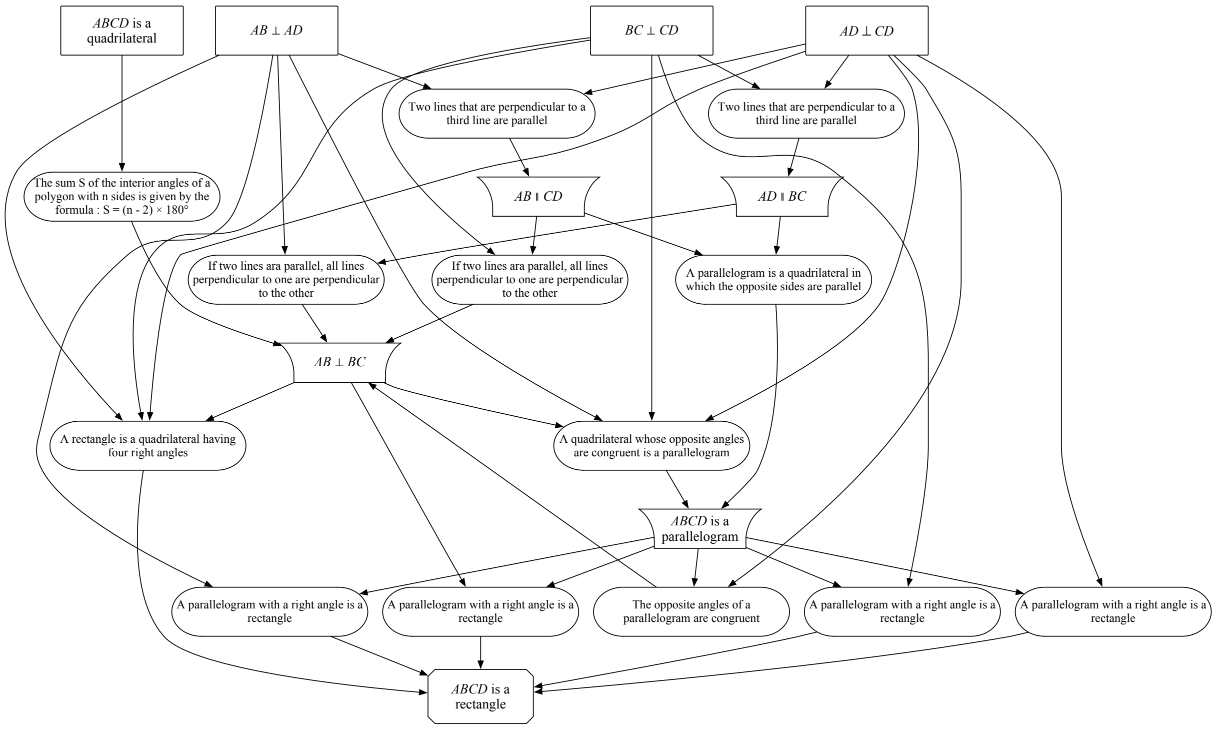
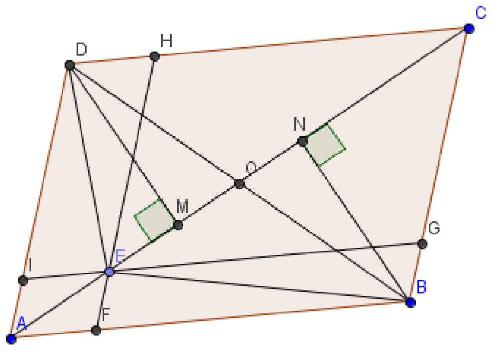

ABCD is a quadrilateral
AB ⊥ AD
BC ⊥ CD
AD ⊥ CD
Two lines that are perpendicular to a third line are parallel
Two lines that are perpendicular to a third line are parallel
The sum S of the interior angles of a polygon with n sides is given by the formula : S = (n - 2) × 180°
AB ∥ CD
AD ∥ BC
If two lines ara parallel, all lines perpendicular to one are perpendicular to the other
If two lines ara parallel, all lines perpendicular to one are perpendicular to the other
A parallelogram is a quadrilateral in which the opposite sides are parallel
AB ⊥ BC
A rectangle is a quadrilateral having four right angles
A quadrilateral whose opposite angles are congruent is a parallelogram
ABCD is a parallelogram
A parallelogram with a right angle is a rectangle
A parallelogram with a right angle is a rectangle
The opposite angles of a parallelogram are congruent
A parallelogram with a right angle is a rectangle
A parallelogram with a right angle is a rectangle
ABCD is a rectangle

Figure 7: The HPDIC graph for the rectangle problem.



"If E is a point of the diagonal AC of parallelogram ABCD, what is the relation between the areas of triangles AEB and AED ?"

Figure 8: The parallelogram area problem.

towards the missing steps of the problem; however, some teachers chose an alternative approach by giving the student another problem, **connected** to the original one. QEDX currently mimics only the first approach, and one of our long-term goals would be to also provide the second approach, by providing connected problems in impasse situations.

More specifically, we consider that two problems are **connected** if their proofs use at least some common mathematical elements. For instance, two problems using the Pythagorean theorem are connected, whereas a third problem about the properties of the hexagon is not connected to the first two. There is no order relation, the problems can be of equal difficulty and length, or very different. However, in this context, they should be approximately close in difficulty, since there is limited educational interest in helping a student stuck on a simple problem by asking him a difficult one. The **connectedness** of two problems is their similarity, a low connectedness means that two problems have few elements in common, whereas a high connectedness means that they are almost similar. The **difficulty** of a problem is not trivial to establish. Providing a rigorous definition to represent our intuition will be one of the very next steps of our work on connected problems.

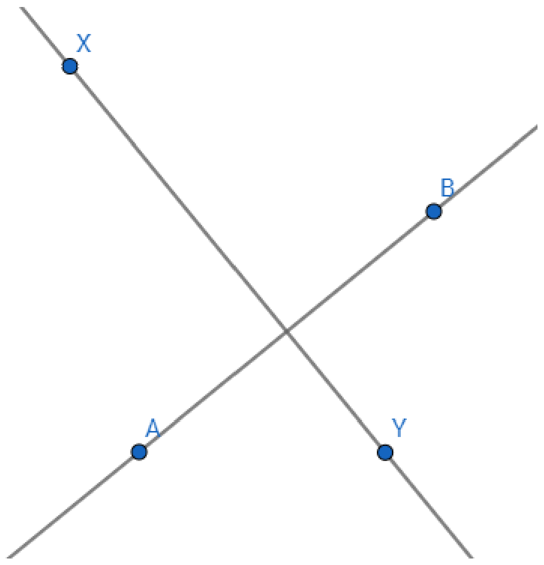
A previous paper by Fortuny, Gagnon and Richard [54] illustrates the educational interest of this approach, both for the student's learning and for the mathematics education research community. In class, the submission of a new, connected problem to the student can restart a halted solving process. Our goal is to mimic this behavior. The main difficulty is that the teacher choses the new problem almost instinctively, helped by his experience. It is therefore a difficult task, even for a teacher, to identify the exact criteria used to determine, first, if giving another problem instead of clues towards the solution would be profitable, and second, **which** problem to give in every particular situation. The first point could be tackled by experimenting with both approaches in class. Concerning the second point, our intuition is that it essentially boils down to identifying the exact step of the proof the student is stuck, which is exactly one of the functionalities of QEDX, and then find another problem using a similar proof. Of course, before working on the connected problems functionality, this intuition will have to be thoroughly verified. This, while interesting, still remains a goal for the future.

Indeed, a fundamental issue arises when trying to tackle the connected problems approach. As we mentioned previously, the HPDIC graph, that must be provided when implementing a problem in QEDX, could be huge. Currently, we have **no method** to automatically generate such a graph, and it therefore has to be constructed manually, which is a tedious and lengthy operation. For this reason, QEDX only contains five problems in its current state, but an approach such as the connected problems one requires a consequent number of available problems covering each topic seen in class. Five is not nearly enough, especially since we chose those to be as different as possible, in order to cover various topics and proof types with few problems.

Therefore, a necessary step, both for the implementation of connected problems and for the usability of QEDX in general, is to pipeline the production of new problems. This includes two steps: identifying interesting problems, which can easily be done by searching the geometry textbooks, and implementing them in QEDX, which includes the difficult step of writing down all possible proofs. Among the research avenues for the improvement of QEDX (see Section 7), this paper specifically presents the automated generation of proofs, in order to supply QED-Tutrix with a healthy amount of problems.

5 Automated generation of proofs

The idea of proving geometry theorem automatically is not new. As soon as 1960, Gelernter [27] proposed a synthetic proof method for geometry theorem proving. Other early approaches include those by



“X and Y are equidistant from A and B. Prove that (XY) is the perpendicular bisector of [AB] ”

Figure 9: A perpendicular bisector problem.

Nevins [46], Elcock [25], Greeno et al. [28], Coelho and Pereira [23], and Chou, Gao and Zhang [19]. Since then, the methods have evolved considerably. However, our use-case is very specific for several reasons.

5.1 Pre-requisites

First, our usage of proofs is to spot the position of a student in a graph of proofs during his resolution process. Therefore, the only proofs that are relevant are proofs that could have been written by a high-school student. We consider a proof to be **readable** if it only uses properties available at a high-school level, and if its length, i.e. the number of inferences, is not unreasonably high. This definition is still informal and subjective, and we have yet to reach a consensus, internally and with external teachers, on what exactly is a readable proof. We discuss part of this issue in Section 7. Still, even without a rigorous definition, this criteria already eliminates all tools based on an algebraic methods, since those provide proofs that are very far from classic Euclidean geometry.

Second, we mentioned that a proof must be accessible to a high-school student. This criteria changes throughout high-school classes, only some properties are available at first, and, as the class advances, more properties are studied, and, therefore, available for the students to use. Hence our second criteria, that the set of properties on which the inference engine works must be fully customizable by non-experts, typically teachers, who will adjust the available properties as the school-year advances.

Finally, teachers are all different in their teaching methods. More specifically, when they ask students to solve a proof problem, they don’t require the same level of rigor in the student’s proof. Hence the notion of **deductive isle**. This notion has been defined in [54]: “*Deductive isles* is our translation from the French *îlot déductif* that considers the network of mathematical properties and definitions accepted or actually used in a given class, which includes the implicit hypothesis and the inferential shortcuts tolerated in the didactic contract.” For example, in the situation described in Fig. 9, most teachers would accept the following proof:

1. *Every point on the perpendicular bisector of a segment [AB] is equidistant from A and B.*
2. *Since X and Y are both equidistant from A and B, the line (XY) is the perpendicular bisector of [AB].*

However, some teachers would not consider it as rigorously valid. Instead, they expect this:

1. *Every point on the perpendicular bisector of a segment [AB] is equidistant from A and B.*
2. *Since X is equidistant from A and B, X is on the perpendicular bisector of [AB].*
3. *Same reasoning for Y.*
4. *Since X and Y are two distinct points {1}, there is one and only one line (XY) passing through X and Y.*
5. *Since X and Y are both on the perpendicular bisector of [AB], (XY) is the perpendicular bisector of [AB].*

In this example, these two inference chains put together constitute a deductive isle, since they use a different granularity of properties to draw the same conclusion (the line (XY) is the perpendicular bisector of [AB]) from the same hypotheses ([AB] is a segment, and X and Y are equidistant from A and B).

The handling of, and the ability to parametrize these deductive isles is essential for the future development of QED-Tutrix. Indeed, we want teachers to be able to use QEDX as a projection of themselves, allowing them to provide a personalized help to twenty or thirty students at the same time.

These three points, readability of the proof, flexibility in the axioms, and handling of the deductive isles, are the fundamental criteria for the automated generation of proofs.

5.2 Our custom engine

We initially had the option to adapt an existing theorem prover or inference engine to suit our needs. Given the fact that optimization and efficiency are not our goal, and that our constraints are quite specific, we decided to work on the implementation and integration of a light custom engine in QEDX, written in Prolog. Prolog is a well-suited language for this task, and the engine itself is quite straightforward to program. The difficulty arises in the encoding of problems. Indeed, if we look back at an extremely simple problem such as the one of the perpendicular bisector in Fig. 9, we already have to add the implicit hypothesis that X and Y are distinct to write the formal proof. This is a typical example of an hypothesis that is obvious both for the students and the teachers, but not for a mechanical tool. Therefore, the hypotheses must be carefully enumerated for the engine to work.

Besides, a recurrent problem in geometry theorem proving is the **construction of intermediate elements**. For instance, many geometry problems require the construction by the student of a geometric element, such as the height of a triangle or the diameter of a circle. Knowing which element to construct is a difficult step for an automated prover. Since the main goal is not to develop a general and well-rounded engine, but instead to reduce the workload to add new problems to QED-Tutrix, we opted for an alternative solution, by imposing that the figure given as an input to the solver include all those intermediate elements. We define that figure as the **super-figure** of the problem, whereas the **figure** represents only the geometrical elements given to the student as part of the problem. This requisite is fundamental, since it defines the nature of the prover, it is not strictly speaking a reasoning tool, but only a mechanical tool. We provide details on the way it works later in this section. Since this research project is not an automated deduction project, we deemed this compromise acceptable for the time being. We keep in mind the possibility of exploring ways to automate this construction step in the future.

```
Require: $problemFile.pl$ $exists$
Require: $properties.pl$ $exists$
 1: $S,C \leftarrow problemFile.pl$ {S for Statements, C for Conclusion}
 2: $P \leftarrow properties.pl$ {P for Properties}
 3: repeat
 4:   $R \leftarrow Reasoner(S,P)$ {R for Results}
 5:   $N \leftarrow R - S$ {N for New results. "-" is the relative complement operation on two sets}
 6:   $S \leftarrow S + N$ {"+" is the Union of two sets}
 7: until $IsEmpty(N)$
 8: if C in S then
 9:   $G \leftarrow ConstructGraph(C,S)$ {G for (HPDIC) Graph}
10:   return $G$
11: else
12:   print $Error$
13:   return 0
14: end if
```

Figure 10: The inference engine algorithm.

The operation of our engine is quite simple. It is composed of a Java handler, calling a Prolog reasoner iteratively, building new results at every step, until nothing remains to be inferred. The initial problem is defined by a set of Prolog statements, one for each hypothesis of the problem. “Hypothesis” in this context is quite broad, since we must define all objects. `point(a)` is a hypothesis, `line(a,b)` as well.

Then, Prolog explores all known properties to extract the ones that can be used with this set of hypotheses to infer new results. In the next iteration, the obtained results are added to the set of hypotheses usable by the properties. We continue iterating until no new inference can be made. The Java handler checks at every step if the result is already known before adding it to the set. The algorithm is detailed in Fig. 10.

The result of this algorithm is, first, a set of all inferred statements, and, second, if the conclusion is among them, as it should be, the HPDIC graph for the problem. Therefore, the pipeline for the addition of a new problem in QEDX would be:

1. Creation of the problem, redaction of the hypotheses and conclusion;

2. Construction of the super-figure, i.e. all geometrical elements that are useful for at least one proof;

3. Translation of the super-figure and hypotheses into Prolog facts;

4. Generation of the HPDIC graph by the inference engine;

5. Integration of the new problem to QEDX.

In this list, steps 1 to 3 must be done by humans, but the bulk of the workload is in step 4. Step 5 is a target for automation later in the project. Overall, the usage of an inference engine reduces drastically the time needed to create new problems for QEDX.

6 Limitations

This project has a number of limitations. We categorized these in two categories: the limitations of the software itself as it is currently, and the limitations of the inference engine we are currently building.

6.1 QED-Tutrix in its current state

QED-Tutrix has globally been well received when tested in class during the experimentations conducted by Michèle Tessier-Baillargeon [56], following the precepts of [53] and *conception by usage* [50]. Teachers and students both appreciated the value provided by its usage. However, we also collected several suggestions for possible improvements.

First, the interface has been designed to be as close to the reasoning of the student as possible. Still, some elements are not yet as easy to use as we want them to be. Typically, the current "sentences" tab is not an ideal way for students to select statements and properties: it requires some search, the statements themselves are not always properly organized, and it is sometimes not easy to know where to search for an element. An ideal solution for this would be to implement a natural language parser, in order to allow the student to write directly his element, and then giving him a list of possible geometrical elements corresponding to his input. Another team is currently working on this possibility.

Second, as mentioned in Section 4, there are only five problems that have been manually implemented in QEDX, with no way to add any significant number in a reasonable time. This issue is the result of the difficulty and time consumption of finding and encoding all possible proofs by hand: these five problems are the result of three years of development, and have been chosen to be as diverse as possible. Still, until it is addressed, QEDX will remain a tool that cannot be used for real, long-term experiments. This important issue of the software is the reason behind the research project described in this paper.

6.2 Our inference engine

One must keep in mind that this project is still in its infancy. The inference engine we started to build is currently only a proof of concept. It still is advanced enough to allow us to identify several limitations to our approach.

We mentioned in Section 5.2 that there can be no implicit hypothesis when using our inference engine. The hypotheses given in input to the inference engine must contain all the relevant information, such as the geometrical elements involved, including the intermediate elements not given to the student their relation, and other hypotheses not contained in the figure. This requires some manual work, including giving names to the objects. For example, the statement "Prove that any quadrilateral with three right angles is a rectangle" is not usable directly. A correct set of statements would be "Given that A,B,C and D are distinct points, that AB,BC,CD and AD lines, that $AB \perp BC$, $BC \perp CD$ and $AB \perp AD$, and that $ABCD$ is a quadrilateral, prove that $ABCD$ is a rectangle". The redaction of completely rigorous hypotheses is a tedious task for the teacher that wants to add a problem to the software.

Furthermore, the necessity of providing the super-figure when using the engine, albeit as a temporary solution, limits the interest of our engine. Indeed, it transfers a great deal of the difficulty to the human. Still, on the five problems currently implemented in QEDX, only two need intermediate constructions, three can be solved completely without construction. We plan to analyze further the proportion of problems that need constructions among the problems accessible to a high-school student. If needed, we plan to explore solutions later in the project to automatically or semi-automatically generate these super-figures. The approach used in the prover GRAMY [42] may be a possible solution.

7 Other avenues and conclusion

In this paper, we presented our plans for the improvement of QED-Tutrix by automatically generating proofs, allowing us to integrate new problems easily, and therefore increasing the range of the software. It is however not the sole research avenue. Indeed, several other aspects of QEDX offer room for improvement. Even when the generation of proofs will be fully automated, the task of encoding the hypotheses, including the implicit and figural ones, remains a tedious task, especially for a teacher who does not want to learn a meta-language. It would therefore be extremely interesting to be able to extract the formal hypotheses, explicit and implicit, and the conclusion, from the statement of the problem in natural language. This task is directly related to the semiotic genesis, i.e. to transform conceptual, abstract knowledge into formal elements, whereas the automated generation of proofs is in the discursive genesis. Another team in the QEDX research group is currently working on this project.

Another interesting aspect is the enhancement of the tutor system. Indeed, the detection of impasses is currently quite simple: the system provides a hint when the student has not given any new element in an arbitrarily fixed time. It would be interesting to improve this impasse detection, for instance by logging and analyzing the actions of the student on the software. These logs could also be used to improve the user interface. A third team in the research group is devoted to this task.

QED-Tutrix is a tool that has an immense potential for educational and research purposes. In its current state, it already provides interesting features that have generally been well received both by teachers and students. It is however crippled by the very limited selection of problems, with only five problems included currently. This limitation is due to the lengthiness of manually writing down all possible proofs for each problem. It is therefore a priority for us to handle that issue by automating the generation of proofs. The generated proofs must follow three specific criteria: readability, accessibility at a high-school level and adaptability to various deductive isles, in accordance to the didactical contract. We experimented briefly with the development of a very simple deduction engine, that offered promising results by successfully generating the inference graph for the two problems we encoded.

The next step is to confirm these results by generating the five inference graphs for the problems currently in QEDX, to ensure the validity of the method. Then, we must address the handling of inferential shortcuts by allowing the teacher to customize the allowed properties and the level of formality required from the students. Finally, we could explore possibilities to include the automated generation of intermediate constructions to the deductive engine.

References

[1] *GeoGebra API*. Available at `http://dev.geogebra.org/trac/wiki/WikiStart`.

[2] *Géométrix*. Available at `http://geometrix.free.fr/site/`.

[3] *Mathway | Math Problem Solver*. Available at `https://www.mathway.com/Algebra`.

[4] Vincent Aleven, Kenneth Koedinger, H Colleen Sinclair & Jaclyn Snyder (1998): *Combatting shallow learning in a tutor for geometry problem solving*. In: Intelligent Tutoring Systems, Springer, pp. 364–373, doi:10.1007/3-540-68716-5_42.

[5] Vincent Aleven & Kenneth R Koedinger (2000): *Limitations of student control: Do students know when they need help?* In: Intelligent tutoring systems, 1839, Springer, pp. 292–303, doi:10.1007/3-540-45108-0_33.

[6] Vincent Aleven & Kenneth R Koedinger (2013): *Knowledge Component (KC) Approaches to Learner Modeling*. Design Recommendations for Intelligent Tutoring Systems 1, pp. 165–182.

[7] Vincent Aleven, Bruce Mclaren, Ido Roll & Kenneth Koedinger (2006): *Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor.* International Journal of Artificial Intelligence in Education 16(2), pp. 101–128.

[8] Vincent AWMM Aleven & Kenneth R Koedinger (2002): *An effective metacognitive strategy: Learning by doing and explaining with a computer-based Cognitive Tutor.* Cognitive science 26(2), pp. 147–179, doi:10.1207/s15516709cog2602_1.

[9] John R Anderson (1996): *ACT: A simple theory of complex cognition.* American Psychologist 51(4), p. 355, doi:10.1037/0003-066X.51.4.355.

[10] John R Anderson & C Schunn (2000): *Implications of the ACT-R learning theory: No magic bullets.* Advances in instructional psychology, Educational design and cognitive science, pp. 1–33.

[11] Ivon Arroyo, Carole Beal, Tom Murray, Rena Walles & Beverly P Woolf (2004): *Web-based intelligent multimedia tutoring for high stakes achievement tests.* In: Intelligent Tutoring Systems, Springer, pp. 142–169, doi:10.1007/978-3-540-30139-4_44.

[12] Nicolas Balacheff, Ricardo Caferra, Michele Cerulli, Nathalie Gaudin, Mirko Maracci, Maria Alessandra Mariotti, Jean-Pierre Muller, Jean-François Nicaud, Michel Occello, Federica Olivero et al. (2003): *Baghera Assessment Project, designing an hybrid and emergent educational society.*

[13] Yves Baulac (1990): *Un micromonde de géométrie, Cabri-géomètre.* Ph.D. thesis, Université Joseph-Fourier-Grenoble I.

[14] Francisco Botana, Markus Hohenwarter, Predrag Janičić, Zoltán Kovács, Ivan Petrović, Tomás Recio & Simon Weitzhofer (2015): *Automated theorem proving in GeoGebra: Current achievements.* Journal of Automated Reasoning 55(1), pp. 39–59, doi:10.1007/s10817-015-9326-4.

[15] Pierre Boutry, Gabriel Braun & Julien Narboux (2016): *From Tarski to Descartes: formalization of the arithmetization of euclidean geometry.* In: SCSS 2016, the 7th International Symposium on Symbolic Computation in Software Science, 39, EasyChair, p. 15, doi:10.29007/k47p.

[16] Gabriel Braun & Julien Narboux (2017): *A synthetic proof of Pappus' theorem in Tarski's geometry.* Journal of Automated Reasoning 58(2), pp. 209–230, doi:10.1007/s10817-016-9374-4.

[17] Guy Brousseau (2006): *Theory of didactical situations in mathematics: Didactique des mathématiques, 1970–1990.* 19, Springer Science & Business Media, doi:10.1007/0-306-47211-2.

[18] Bruno Buchberger (1988): *Applications of Gröbner bases in non-linear computational geometry.* In: Trends in computer algebra, Springer, pp. 52–80, doi:10.1007/3-540-18928-9_5.

[19] S-C Chou, X-S Gao & J-Z Zhang (1993): *Automated production of traditional proofs for constructive geometry theorems.* In: Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on, IEEE, pp. 48–56, doi:10.1109/LICS.1993.287601.

[20] Shang-Ching Chou (1988): *An introduction to Wu's method for mechanical theorem proving in geometry.* Journal of Automated Reasoning 4(3), pp. 237–267, doi:10.1007/BF00244942.

[21] Shang-Ching Chou, Xiao-Shan Gao & Jing-Zhong Zhang (1994): *Machine proofs in geometry: Automated production of readable proofs for geometry theorems.* 6, World Scientific, doi:10.1142/9789812798152_0002.

[22] Shang-Ching Chou, Xiao-Shan Gao & Jing-Zhong Zhang (1996): *Automated generation of readable proofs with geometric invariants. II. Theorem proving with full-angles.* Journal of Automated Reasoning 17(3), pp. 349–370, doi:10.1007/BF00283134.

[23] Helder Coelho & Luis Moniz Pereira (1986): *Automated reasoning in geometry theorem proving with Prolog.* Journal of Automated Reasoning 2(4), pp. 329–390, doi:10.1007/BF00248249.

[24] Simon El-Khoury, Philippe R Richard, Esma Aïmeur & Josep M Fortuny (2005): *Development of an Intelligent Tutorial System to Enhance Students' Mathematical Competence in Problem Solving.* In: E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Association for the Advancement of Computing in Education (AACE), pp. 2042–2049.

[25] EW Elcock (1977): *Representation of knowledge in geometry machine*. Machine Intelligence 8, pp. 11–29.

[26] Jean-Claude Falmagne, Eric Cosyn, Jean-Paul Doignon & Nicolas Thiéry (2006): *The assessment of knowledge, in theory and in practice*. In: Formal concept analysis, Springer, pp. 61–79, doi:10.1007/11671404_4.

[27] Herbert Gelernter, James R Hansen & Donald W Loveland (1960): *Empirical explorations of the geometry theorem machine*. In: Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference, ACM, pp. 143–149.

[28] James G Greeno (1979): *Constructions in Geometry Problem Solving.* doi:10.3758/BF03198261.

[29] Markus Hohenwarter (2013): *GeoGebra 4.4–From desktops to tablets*. Indagatio Didactica 5(1).

[30] Predrag Janicic (2006): *GCLC-A Tool for constructive euclidean geometry and more than that*. In: ICMS, Springer, pp. 58–73, doi:10.1007/11832225_6.

[31] Stéphanie Jean-Daubias (2000): *PÉPITE: un système d'assistance au diagnostic de compétences*. Ph.D. thesis, Université du Maine.

[32] Deepak Kapur (1986): *Using Gröbner bases to reason about geometry problems*. Journal of Symbolic Computation 2(4), pp. 399–408, doi:10.1016/S0747-7171(86)80007-4.

[33] K Koedinger (1991): *On the design of novel notations and actions to facilitate thinking and learning*. In: Proceedings of the International Conference on the Learning Sciences, pp. 266–273.

[34] Kenneth R Koedinger & John R Anderson (1990): *Abstract planning and perceptual chunks: Elements of expertise in geometry*. Cognitive Science 14(4), pp. 511–550, doi:10.1207/s15516709cog1404_2.

[35] Kenneth R Koedinger & John R Anderson (1993): *Effective use of intelligent software in high school math classrooms*.

[36] Maria Kordaki & Alexios Mastrogiannis (2006): *The potential of multiple-solution tasks in e-learning environments: Exploiting the tools of Cabri Geometry II*. In: E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education, Association for the Advancement of Computing in Education (AACE), pp. 97–104.

[37] Alain Kuzniak & Philippe R Richard (2014): *Espacios de trabajo matemático. Puntos de vista y perspectivas*. Revista latinoamericana de investigación en matemática educativa 17(4), doi:10.12802/relime.13.1741a.

[38] N. Leduc (2016): *QED-Tutrix : système tutoriel intelligent pour l'accompagnement d'élèves en situation de résolution de problèmes de démonstration en géométrie plane*. Ph.D. thesis, École polytechnique de Montréal.

[39] Vanda Luengo (1997): *Cabri-Euclide: Un micromonde de Preuve intégrant la réfutation*. These de doctorat, INPG, France.

[40] Vanda Luengo (2005): *Some didactical and epistemological considerations in the design of educational software: the Cabri-Euclide example*. International Journal of Computers for Mathematical Learning 10(1), pp. 1–29, doi:10.1007/s10758-005-4580-x.

[41] Vanda Luengo & Nicolas Balacheff (1998): *Contraintes informatiques et environnements d'apprentissage de la démonstration en mathématiques*. Sciences et Techniques Educatives 5, pp. 15–45.

[42] Noboru Matsuda & Kurt Vanlehn (2004): *Gramy: A geometry theorem prover capable of construction*. Journal of Automated Reasoning 32(1), pp. 3–33, doi:10.1023/B:JARS.0000021960.39761.b7.

[43] Noboru Matsuda & Kurt VanLehn (2005): *Advanced Geometry Tutor: An intelligent tutor that teaches proof-writing with construction.* In: AIED, 125, pp. 443–450.

[44] Erica Melis, Giorgi Goguadze, Paul Libbrecht & Carsten Ullrich (2009): *Activemath–a learning platform with semantic web features*. The Future of Learning, p. 159.

[45] Julien Narboux (2006): *Mechanical theorem proving in Tarski's geometry*. In: International Workshop on Automated Deduction in Geometry, Springer, pp. 139–156, doi:10.1007/978-3-540-77356-6_9.

[46] Arthur J Nevins (1975): *Plane geometry theorem proving using forward chaining*. Artificial Intelligence 6(1), pp. 1–23, doi:10.1016/0004-3702(75)90013-2.

[47] D Py (1994): *Reconnaissance de plan pour la modélisation de l'élève. Le projet Mentoniezh.* Recherches en didactique des mathématiques 14(1/2), pp. 113–138.

[48] Dominique Py (1996): *Aide à la démonstration en géométrie: le projet Mentoniezh.* Sciences et techniques éducatives 3(2), pp. 227–256.

[49] Dominique Py (2001): *Environnements interactifs d'apprentissage et démonstration en géométrie.*

[50] Pierre Rabardel (1995): *Les hommes et les technologies; approche cognitive des instruments contemporains.* Armand Colin.

[51] Philippe R Richard (2004): *L'inférence figurale: un pas de raisonnement discursivo-graphique.* Educational Studies in Mathematics 57(2), pp. 229–263, doi:10.1023/B:EDUC.0000049272.75852.c4.

[52] Philippe R Richard & Josep M Fortuny (2007): *Amélioration des compétences argumentatives à l'aide d'un système tutoriel en classe de mathématique au secondaire.* In: Annales de didactique et de sciences cognitives, 12, pp. 83–116.

[53] Philippe R Richard, Josep Maria Fortuny, Michel Gagnon, Nicolas Leduc, Eloi Puertas & Michèle Tessier-Baillargeon (2011): *Didactic and theoretical-based perspectives in the experimental development of an intelligent tutorial system for the learning of geometry.* ZDM 43(3), pp. 425–439, doi:10.1007/s11858-011-0320-y.

[54] P.R. Richard, Michel Gagnon & J. M. Fortuny (2018): *Connectedness of Problems and Impass Resolution in the Solving Process in Geometry: a Major Educational Challenge.* In P. Herbst, U.H. Cheah, K. Jones & P.R. Richard, editors: International Perspectives on the Teaching and Learning of Geometry in Secondary Schools, Springer, pp. 311–327.

[55] Ido Roll, Ryan SJ d Baker, Vincent Aleven & Kenneth R Koedinger (2014): *On the benefits of seeking (and avoiding) help in online problem-solving environments.* Journal of the Learning Sciences 23(4), pp. 537–560, doi:10.1016/S0360-1315(99)00030-5.

[56] M Tessier-Baillargeon (2015): *GeoGebraTUTOR : Développement d'un système tutoriel autonome pour l'accompagnement d'élèves en situation de résolution de problèmes de démonstration en géométrie plane et genèse d'un espace de travail géométrique idoine.* Ph.D. thesis, Université de Montréal.

[57] Carine Webber, Loris Bergia, Sylvie Pesty & Nicolas Balacheff (2001): *The Baghera project: a multi-agent architecture for human learning.* In: Workshop-Multi-Agent Architectures for Distributed Learning Environments. In Proceedings International Conference on AI and Education. San Antonio, Texas.

[58] Gerhard Weber & Peter Brusilovsky (2001): *ELM-ART: An adaptive versatile system for Web-based instruction.* International Journal of Artificial Intelligence in Education (IJAIED) 12, pp. 351–384.

[59] H Wu (1979): *An elementary method in the study of nonnegative curvature.* Acta Mathematica 142(1), pp. 57–78, doi:10.1007/BF02395057.

[60] Jingzhong Zhang, Lu Yang & Mike Deng (1990): *The parallel numerical method of mechanical theorem proving.* Theoretical Computer Science 74(3), pp. 253–271, doi:10.1016/0304-3975(90)90077-U.