

**Titre:** Génération automatique de preuves pour un logiciel tuteur en géométrie  
Title: [Génération automatique de preuves pour un logiciel tuteur en géométrie](#)

**Auteur:** Ludovic Font  
Author: [Ludovic Font](#)

**Date:** 2021

**Type:** Mémoire ou thèse / Dissertation or Thesis

**Référence:** Font, L. (2021). Génération automatique de preuves pour un logiciel tuteur en géométrie [Ph.D. thesis, Polytechnique Montréal]. PolyPublie.  
Citation: <https://publications.polymtl.ca/9090/>

## Document en libre accès dans PolyPublie

Open Access document in PolyPublie

**URL de PolyPublie:** <https://publications.polymtl.ca/9090/>  
PolyPublie URL: <https://publications.polymtl.ca/9090/>

**Directeurs de recherche:** Michel Gagnon, & Philippe R. Richard  
Advisors: [Michel Gagnon](#), [Philippe R. Richard](#)

**Programme:** Génie informatique  
Program: [Génie informatique](#)

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

**Génération automatique de preuves pour un logiciel tuteur en géométrie**

**LUDOVIC FONT**

Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*  
Génie informatique

Juillet 2021

**POLYTECHNIQUE MONTRÉAL**

affiliée à l'Université de Montréal

Cette thèse intitulée :

**Génération automatique de preuves pour un logiciel tuteur en géométrie**

présentée par **Ludovic FONT**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*  
a été dûment acceptée par le jury d'examen constitué de :

**Daniel ALOISE**, président

**Michel GAGNON**, membre et directeur de recherche

**Philippe R. RICHARD**, membre et codirecteur de recherche

**Michel DESMARAIS**, membre

**Pedro Henrique e Figueiredo QUARESMA DE ALMEIDA**, membre externe

## REMERCIEMENTS

En tout premier lieu, je remercie Michel et Philippe, mes directeurs, qui, par leurs conseils, leur encadrement, et leur accompagnement en général, ont réussi l'exploit de me maintenir focalisé et productif sur ce projet, et ce jusqu'à sa complétion. Je remercie également Sébastien, collègue et ami, dont le travail acharné en didactique des mathématiques, allant bien au-delà d'un travail de maîtrise, a grandement participé à donner à ce projet toute sa dimension multidisciplinaire. Enfin, je remercie ma famille, qui, malgré la distance, m'ont offert de précieux moments de répit et n'ont jamais failli dans leurs encouragements, ainsi que la communauté Eclipse, sans qui cette thèse aurait été finie six mois plus tôt.

## RÉSUMÉ

Le logiciel QED-Tutrix est un tuteur intelligent qui offre un cadre aidant les élèves de secondaire à résoudre des problèmes de géométrie. Une de ses caractéristiques fondamentales est l'aspect tuteur, qui accompagne l'élève dans sa résolution du problème, et l'aide à avancer en cas de blocage par le biais de messages, personnalisés selon son avancement dans la preuve. Une autre de ses caractéristiques est de rester proche de la façon dont un élève résoudrait le problème avec un papier et un crayon, et, par conséquent, lui permet d'explorer la construction de sa preuve dans l'ordre de son choix. En d'autres termes, si le premier instinct de l'élève est de remarquer une caractéristique de la situation géométrique n'étant ni un conséquent direct des hypothèses, ni un élément directement nécessaire à la conclusion, QED-Tutrix lui permet de commencer à résoudre le problème à partir de ce résultat intermédiaire.

Ces deux caractéristiques créent une difficulté importante. En effet, le logiciel doit fournir une aide personnalisée selon l'avancement de l'élève, qui peut contenir des éléments de preuve très variés et sans nécessairement de rapport direct entre eux, pouvant par exemple appartenir à plusieurs chemins de preuve distincts. Par conséquent, le logiciel doit connaître à l'avance les différents chemins de résolution possibles pour chaque problème, et être capable d'analyser ces chemins pour déterminer dans quelle direction l'élève se dirige. Pour résoudre cet enjeu, le logiciel associe à chaque problème une structure de données, nommée le graphe HPDIC et présentée plus en détail dans les sections centrales de ce document, qui permet de représenter sous forme de graphe toutes les preuves possibles pour un problème donné. Ainsi, il devient possible d'analyser finement l'avancement de l'élève dans sa résolution du problème, et ce quel que soit l'ordre dans lequel il a fourni les éléments de la preuve.

La difficulté devient alors de générer ce graphe pour chacun des problèmes que l'on souhaite ajouter au logiciel. À la première itération de développement, ces graphes étaient générés à la main par un des experts en didactique du projet. Cette procédure, convenable pour une petite quantité de problèmes, devient fastidieuse et peu appropriée lorsque l'on souhaite ajouter un grand nombre de problèmes. Il est donc rapidement devenu nécessaire d'automatiser ce processus. Bien que la démonstration automatique de théorème, ou Automated Theorem Proving (ATP), soit un domaine de recherche dynamique, les enjeux très spécifiques de ce projet rendent difficile l'utilisation de techniques ou outils existants. Cette difficulté est explorée en détail dans la revue de littérature, au chapitre 2.

Par conséquent, nous avons opté pour créer notre propre générateur de preuves, afin de pouvoir ajuster son fonctionnement aux besoins, à la fois didactiques et techniques, du projet.

Nous avons également choisi de créer cet outil en utilisant la programmation logique, plus précisément le langage Prolog, car le paradigme de la programmation logique est tout à fait approprié pour représenter des démonstrations mathématiques basées sur des inférences. En effet, les parallèles entre une preuve mathématique et la résolution d'une requête en programmation logique sont flagrants. Une preuve mathématique peut être vue comme une succession d'inférences, chacune constituée d'antécédents, d'une justification, qui peut être une définition, une propriété ou un théorème, et d'un conséquent. Le conséquent d'une inférence peut ensuite servir d'antécédent à une autre inférence, créant ainsi un chemin inférentiel partant des hypothèses du problème et arrivant à la conclusion attendue. En programmation logique, la base de connaissance contient des faits ("facts"). Ces faits permettent d'activer des règles ("rules") afin de générer de nouveaux faits, qui sont ensuite ajoutés à la base de connaissances, et permettront éventuellement d'activer de nouvelles règles. Le parallèle est aisé, en faisant correspondre les résultats mathématiques aux faits, les inférences aux applications de règles. La démonstration complète consiste alors à générer la conclusion du problème à partir de l'ensemble des faits représentant les hypothèses initiales.

Fort de cette conviction, nous avons donc développé notre générateur de preuves en Prolog. Cet outil, à partir d'un ensemble de règles représentant les justifications mathématiques présentes au programme de géométrie du secondaire au Québec, et d'un ensemble d'hypothèses décrivant une situation géométrique, est capable de générer l'ensemble des informations qu'un élève de secondaire pourrait inférer. Ainsi, lorsque la situation géométrique correspond à un problème, le générateur de preuves fournit l'intégralité des preuves pour ce problème, sous la forme d'un graphe HPDIC, tel que mentionné précédemment.

Cependant, bien que les parallèles soient forts entre la programmation logique et une preuve mathématique rigoureuse, des difficultés apparaissent dès que l'on confronte un système automatisé avec la réalité de l'enseignement en classe. En effet, la description d'une preuve mathématique donnée plus tôt dans ce résumé représente un idéal théorique. En pratique, en particulier dans le cadre de l'enseignement où la démonstration n'est qu'un outil afin de faire comprendre des concepts aux élèves, les preuves sont bien moins rigoureuses, avec par exemple certaines inférences non justifiées, qui seront acceptées parce que les justifications ne sont pas le sujet du cours du moment, ou des hypothèses non fournies, car elles représentent des cas limites dont la considération n'est pas nécessaire au niveau secondaire. Cette constatation rend la création d'un système automatisé difficile. En effet, l'ordinateur, bien que beaucoup plus rapide qu'un humain pour effectuer un travail répétitif, n'est pas capable de raisonner à un niveau aussi abstrait. Il est donc nécessaire, lors de l'élaboration du système, de prendre en compte toutes les nuances découlant, premièrement, de la réalité de l'enseignement en classe, et, deuxièmement, de la grande variabilité de cette réalité d'une

classe, et d'un professeur, à l'autre.

En d'autres termes, un des points centraux de ce travail est qu'il est le fruit d'une collaboration rapprochée avec des experts en didactique des mathématiques. La partie informatique seule n'est en effet définitivement pas suffisante pour apporter une contribution réaliste et pertinente au logiciel tuteur. Par conséquent, une partie importante de cette thèse est consacrée à détailler les enjeux didactiques du projet, et les solutions informatiques qui y ont été apportées.

## ABSTRACT

The QED-Tutrix project has the ultimate goal of creating, improving, and expanding the eponym tutor software, to accompany high school students in the resolution of geometry problems. Based on proven mathematics education theories, it has several distinctive characteristics. First, as a tutor software, the tutoring aspect is paramount. The virtual tutor, aptly named Professor Turing, is tasked with helping the student through the entire resolution process, from the exploration of the problem to the redaction of the proof. Second, the software as a whole stays close to the way students would typically solve a problem with paper and pencil, while adding digital tools to improve this process. This includes a dynamic geometry interface to explore the problem, a listing of available justifications, such as definitions or theorems, and a semi-automated redaction engine. Further, the student is allowed, like in real life, to construct the proof by adding elements in any order, for example by noticing a particular geometrical result on the figure, that is neither a hypothesis of the problem or its conclusion. It is possible in QED-Tutrix to begin the proof “in the middle”, by stating this particular result and working from there.

These educational goals create interesting issues from a computer science point of view. Indeed, the tutoring is tailored to the exact point of the resolution reached by the student, and this point can be quite discontinuous since the elements of the resolution may have been entered in any order, can be unrelated, or pertain to different possible proofs of the problem. As a logical consequence, in order for the software to “know” what the student is doing and, mostly, what he or she is missing to complete the resolution, it is necessary to know, beforehand, all the possible proofs for the problem. To solve that issue, each problem is provided with a structure, the HPDIC graph, containing this set of all possible proofs. Of course, “all possible proofs” is restricted to those that a high school student may be able to do, in order to, first, remain in the realm of high school mathematics education, and, second, avoid having to deal with an infinitely large set of proofs. This graph being provided, it becomes possible for the software, by its exploration, to identify the proof the student is most likely working on, and, therefore, to provide help to complete it, and this regardless of the order the student entered its elements in.

Then, the issue becomes to create such a HPDIC graph for each problem we wish to add to QED-Tutrix. In its first version, these graphs were created by hand, which is acceptable for a small number of problems, but quickly becomes extremely time-consuming and error-prone when the amount of problems increases. Therefore, a logical answer is to automate

this process. The research domain of Automated Theorem Proving (ATP) provides many solutions to generate proofs, but, given our very specific educational goals, was not directly useable for our task.

We therefore opted for the creation of our automated proof generator, working closely with mathematics education experts at every step. Because of the similarities between the paradigm of logic programming and mathematical inferences, we chose to implement it in Prolog. Indeed, a mathematical inference can be seen as a justification, such as a definition, property or theorem, that is combined to a set of mathematical statements, the antecedents, to create a new statement, the consequent. This consequent can then be used as the antecedent of another inference. In Prolog, the knowledge base contains facts, that are combined with rules to create new facts. By encoding the mathematical statements, including the problem hypotheses, as facts, and the justifications available in high school as rules, it becomes quite straightforward to create an inference engine that finds all the possible inferences in a problem. Since a proof is a chain of inferences from the problem hypotheses to its conclusion, this set of all possible inferences contains the set of all possible proofs. As a consequence, the generation of all proofs for a geometry problems boils down to encoding the problem hypotheses as Prolog facts, and providing the inference engine with a set of Prolog rules representing all the justifications of high school geometry, and then extracting and translating the HPDIC graph in a format useable by QED-Tutrix.

However, even though there are strong parallels between logic programming and a mathematical proof, issues appear as soon as we attempt to represent actual classroom situations. Indeed, the previous definition about a mathematical inference is an ideal one. In the context of education, since geometry can be extremely complex, shortcuts, approximation and implicit hypotheses are often made. Indeed, theorems typically require a set of non-degeneracy conditions, that are very often ignored in high school. Further, since a problem is mostly used as an illustration of a concept seen in class, some hypotheses are also ignored since they are out of scope for this particular lesson. This makes the task of automating this process quite difficult, since a computer, albeit extremely fast in repetitive tasks, is perfectly unable to have even such a basic abstract reasoning. It is therefore necessary to explicitly encode these subtelties and possible simplifications.

In other words, a core part of this project is to ensure the harmonious marriage of powerful but unbendingly rigorous logic programming tools, and the subtle reality of actual high school geometry education. To do so, this project has been done in close cooperation with mathematics education experts, since computer science alone is definitely not enough to ensue a relevant and realistic addition to the QED-Tutrix tutor software. An important part

of this thesis is therefore dedicated to presenting such educational constraints and how they were solved in the rigid world of software engineering.

## TABLE DES MATIÈRES

REMERCIEMENTS . . . . .	iii
RÉSUMÉ . . . . .	iv
ABSTRACT . . . . .	vii
TABLE DES MATIÈRES . . . . .	x
LISTE DES FIGURES . . . . .	xiv
LISTE DES SIGLES ET ABRÉVIATIONS . . . . .	xvi
LISTE DES ANNEXES . . . . .	xvii
 CHAPITRE 1 INTRODUCTION . . . . .	1
1.1 Le logiciel QED-Tutrix . . . . .	1
1.2 Le besoin d'un générateur de preuves automatisé . . . . .	4
1.3 Enjeux didactiques . . . . .	6
1.4 Objectifs de recherche . . . . .	8
1.5 Plan de la thèse . . . . .	9
 CHAPITRE 2 REVUE DE LITTÉRATURE . . . . .	10
2.1 Logiciels tuteurs . . . . .	10
2.2 Démonstration automatique . . . . .	11
2.3 Fondations didactiques . . . . .	11
 CHAPITRE 3 MÉTHODOLOGIE . . . . .	13
3.1 Genèse du projet . . . . .	13
3.2 Contraintes didactiques . . . . .	13
3.3 Évaluation . . . . .	14
3.3.1 Validation du processus . . . . .	15
3.3.2 Validation du référentiel . . . . .	15
 CHAPITRE 4 ARTICLE 1 : IMPROVING QED-TUTRIX BY AUTOMATING THE GENERATION OF PROOFS . . . . .	16
4.1 Mise en contexte . . . . .	16

4.2	Abstract . . . . .	16
4.3	Introduction . . . . .	17
4.4	Related Work . . . . .	19
4.4.1	Tutor softwares . . . . .	19
4.4.2	Automated theorem proving . . . . .	23
4.5	Presentation of QED-Tutrix . . . . .	24
4.5.1	Conception guidelines . . . . .	24
4.5.2	Internal engine . . . . .	31
4.6	Goals for the improvement of QED-Tutrix . . . . .	33
4.7	Automated generation of proofs . . . . .	34
4.7.1	Pre-requisites . . . . .	34
4.7.2	Our custom engine . . . . .	36
4.8	Limitations . . . . .	38
4.8.1	QED-Tutrix in its current state . . . . .	38
4.8.2	Our inference engine . . . . .	39
4.9	Other avenues and conclusion . . . . .	39
 CHAPITRE 5 ARTICLE 2 : AUTOMATING THE GENERATION OF HIGH SCHOOL GEOMETRY PROOFS USING PROLOG IN AN EDUCATIONAL CONTEXT . . . . .		41
5.1	Mise en contexte . . . . .	41
5.2	Abstract . . . . .	41
5.3	Context . . . . .	42
5.3.1	The QED-Tutrix software . . . . .	42
5.3.2	The need for an automated problem solver . . . . .	43
5.3.3	Addition of a problem to QED-Tutrix . . . . .	44
5.4	Related Work . . . . .	45
5.5	The problem solver . . . . .	46
5.5.1	Available data . . . . .	46
5.5.2	Encoding of a problem . . . . .	47
5.5.3	Generation of the complete set of proofs . . . . .	50
5.5.4	Generation of mathematical results using logic programming . . . . .	52
5.6	Educational challenges . . . . .	55
5.6.1	Translation of high school properties into a computable format . . . . .	55
5.6.2	Differences between a human's and a computer's reasoning . . . . .	58
5.7	Limitations . . . . .	59
5.8	Other avenues and conclusion . . . . .	61

CHAPITRE 6 ARTICLE 3 : INTELLIGENCE IN QED-TUTRIX : BALANCING THE INTERACTIONS BETWEEN THE NATURAL INTELLIGENCE OF THE USER AND THE ARTIFICIAL INTELLIGENCE OF THE TUTOR SOFTWARE . . . . .	62
6.1 Mise en contexte . . . . .	62
6.2 Abstract . . . . .	63
6.3 Context . . . . .	63
6.3.1 Symbiosis between the Mathematical Work in Schools and Computer Science . . . . .	65
6.3.2 Existing Tutor Softwares . . . . .	67
6.4 Genesis of the QED-Tutrix Project . . . . .	69
6.4.1 Task description . . . . .	70
6.4.2 Software Overview . . . . .	71
6.4.3 The Core Layers of QED-Tutrix . . . . .	76
6.5 The Need for Automated Proof Generation . . . . .	80
6.5.1 Existing Theorem Provers . . . . .	81
6.5.2 The Choice of Logic Programming to Generate Inferences . . . . .	82
6.5.3 Available Data . . . . .	83
6.5.4 Encoding of a problem . . . . .	84
6.5.5 Generation of the Complete Set of Proofs . . . . .	86
6.5.6 Validation . . . . .	87
6.5.7 Limitations . . . . .	90
6.6 Conclusion . . . . .	92
CHAPITRE 7 ARTICLE 4 : CREATION OF A MATHEMATICAL MODEL FOR QED-TUTRIX' AUTOMATED PROOF GENERATOR . . . . .	94
7.1 Mise en contexte . . . . .	94
7.2 Abstract . . . . .	95
7.3 Introduction . . . . .	95
7.4 From a problem statement to a solutions graph . . . . .	96
7.5 Overview of the proof generation process . . . . .	97
7.6 The many names of geometrical objects . . . . .	101
7.7 The degrees of approximation of a problem . . . . .	102
7.8 Level of granularity . . . . .	103
7.9 Future work . . . . .	105
7.10 Conclusion . . . . .	108
CHAPITRE 8 DISCUSSION GÉNÉRALE . . . . .	109

CHAPITRE 9 CONCLUSION . . . . .	110
9.1 Synthèse des travaux . . . . .	110
9.2 Limitations et améliorations futures . . . . .	112
RÉFÉRENCES . . . . .	114
ANNEXES . . . . .	121

## LISTE DES FIGURES

1.1	Le processus de construction du graphe HPDIC . . . . .	4
1.2	Un triangle équilatéral et le triangle formé par les milieux de ses côtés	6
1.3	Une simple situation géométrique . . . . .	8
4.1	The model of mathematical working space in the theory. . . . .	20
4.2	An example of a simple geometrical proof problem. . . . .	25
4.3	The main interface of QED-Tutrix. . . . .	26
4.4	The main interface of QED-Tutrix, on the “Sentence” tab. . . . .	27
4.5	The main interface of QED-Tutrix, on the “Redaction” tab. . . . .	28
4.6	A geometrical construction of the height of a triangle. . . . .	30
4.7	The HPDIC graph for the rectangle problem. . . . .	32
4.8	The parallelogram area problem. . . . .	32
4.9	A perpendicular bisector problem. . . . .	36
4.10	The inference engine algorithm. . . . .	38
5.1	The process of constructing the HPDIC graph . . . . .	43
5.2	The translation of the rectangle problem from its statement. . . . .	48
5.3	A geometrical situation with many angles. . . . .	49
5.4	The graph construction algorithm. . . . .	51
5.5	Translation in Prolog of a property on right triangles . . . . .	53
5.6	The part of a HPDIC graph encoded by one newInference fact . . . . .	54
5.7	Link between a statement and an inference . . . . .	54
6.1	The main components of the mathematical working space. The vertical planes highlight the coordination of the dominant geneses in a specific mathematical work, for example, to signify certain mathematical competencies at stake or types of instrumental proofs [1]. . . . .	66
6.2	QED-Tutrix’s main interface . . . . .	71
6.3	The Parallelogram problem statement . . . . .	72
6.4	The Parallelogram problem super-figure . . . . .	73
6.5	QED-Tutrix’s sentence selection menu . . . . .	74
6.6	QED-Tutrix’s sentence confirmation prompt . . . . .	75
6.7	The informal construction process of an HPDIC graph . . . . .	77
6.8	The translation of the rectangle problem from its statement. . . . .	84
6.9	A geometrical situation with many angles. . . . .	85
6.10	The statements of two of the four intial problems. . . . .	88

7.1	A simple inference. . . . .	98
7.2	The rectangle problem. . . . .	98
7.3	The HPDIC graph for the rectangle problem. . . . .	99
7.4	The parallelogram problem . . . . .	100
7.5	Two mathematical situations . . . . .	102
7.6	Two geometrical situations using circles . . . . .	104
7.7	A perpendicular bissector problem. . . . .	104
7.8	The HPDIC graph for a high granularity proof of the perpendicular bissector problem . . . . .	106
7.9	The HPDIC graph for a low granularity proof of the perpendicular bissector problem . . . . .	106
C.1	Un triangle équilatéral et le triangle formé par les milieux de ses côtés	128

## LISTE DES SIGLES ET ABRÉVIATIONS

(Graphe) HPDIC	Graphe contenant les Hypothèses, Propriétés, Définitions, résultats Intermédiaires, et Conclusion(s)
ETM	Espaces de Travail Mathématique, théorie didactique sur laquelle QED-Tutrix est basé
ATP	Démonstration automatisée de théorèmes, en anglais <i>Automated Theorem Proving</i>

**LISTE DES ANNEXES**

Annexe A	Propriétés implémentées . . . . .	121
Annexe B	Une inférence encodée en Prolog . . . . .	126
Annexe C	Encodage d'un problème en Prolog . . . . .	128

## CHAPITRE 1 INTRODUCTION

Bien que centrales dans notre société moderne et les technologies qui en sont le pilier, les mathématiques sont délicates à enseigner, et divisives auprès des élèves. Alors qu'il est rare d'entendre “je suis absolument incapable de comprendre la géographie” ou “la littérature me rend physiquement malade”, nous avons tous entendu un ou une camarade utiliser ces phrases pour décrire son expérience avec les cours de mathématiques. Ces difficultés s'accentuent au secondaire, avec l'apparition de la notion de démonstration, pourtant mise en avant par les organismes chargés de l'enseignement, qu'il s'agisse du gouvernement du Québec<sup>1</sup>, du conseil des enseignants en mathématiques<sup>2</sup>, ou dans le cadre d'études sur l'enseignement<sup>3</sup>. Ce constat est le point d'origine ayant mené à la création du projet QED-Tutrix, visant à développer un logiciel tuteur pour assister à l'enseignement de la démonstration en géométrie au secondaire. Son développement est le fruit d'une collaboration de bientôt une décennie entre des experts et des étudiants, à la fois en informatique et en didactique des mathématiques. Le projet de recherche décrit dans cette thèse constitue la principale partie informatique de la deuxième phase du développement de QED-Tutrix. Par conséquent, l'objectif premier de cette introduction est de fournir une vue d'ensemble du projet dans son intégralité, et d'illustrer de quelle façon ce développement d'un générateur automatique de preuves, le cœur de cette thèse, s'y inscrit.

### 1.1 Le logiciel QED-Tutrix

Le tuteur intelligent QED-Tutrix est un logiciel ayant pour but de faciliter l'enseignement de la géométrie au secondaire. Une description plus précise du logiciel est fournie dans le premier article, au chapitre 4. Pour simplifier la lecture, cette section contient un résumé des points importants.

Il est tout d'abord important de noter que QED-Tutrix se place dans un paradigme de preuve formelle. Par conséquent, il ne gère que les problèmes pouvant être résolus par un mécanisme de preuve. Plus précisément, nous considérons une “preuve” comme une succession d'inférences. Une inférence est un processus logique permettant d'obtenir un résultat mathématique, le conséquent, à partir d'un ensemble d'autres résultats mathématiques démontrés au préalable, les antécédents, et d'une justification faisant le lien entre les deux, tel un théo-

---

1. <http://www.education.gouv.qc.ca/enseignants/pfeq/>  
 2. <https://www.nctm.org/Standards-and-Positions/Principles-and-Standards/>  
 3. <https://www.oecd-ilibrary.org/docserver/b25efab8-en.pdf>

rème, une propriété, ou une définition. Par exemple, une inférence utilisant le théorème de Pythagore aurait la forme suivante :

- Étant donné que ABC est un triangle rectangle en A, que AB = 3cm et que AC = 4cm (*antécédents*),
- d'après le théorème de Pythagore (*justification*),
- BC vaut  $\sqrt{3^2 + 4^2} = 5\text{cm}$  (*conséquent*).

Les antécédents et les conséquents d'une inférence sont tous des résultats mathématiques. Ainsi, le conséquent d'une inférence peut servir d'antécédent à une seconde inférence. De cette façon, une preuve se construit en enchaînant les inférences. Les hypothèses du problème servent d'antécédents aux premières inférences, qui produisent de nouveaux résultats, permettant d'inférer de nouveaux résultats, et ce jusqu'à arriver à inférer la conclusion du problème. Cette structure rigide de preuve limite donc en partie l'étendue des problèmes que QED-Tutrix est capable de gérer, mais offre en contrepartie un ensemble de règles immuables servant de base à tout le travail informatique.

Pour en revenir au logiciel, le mandat de QED-Tutrix est de fournir un espace dans lequel l'élève peut résoudre des problèmes de preuve formelle en géométrie, d'une façon ressemblant le plus possible à ce qu'il ferait avec un papier et crayon, et recevoir, au besoin, une aide personnalisée de la part du logiciel. Cela permet au professeur de la classe de concentrer ses efforts sur les élèves plus en difficulté, car une grande partie des difficultés courantes peut être gérée par le logiciel. L'on peut donc résumer les enjeux principaux de QED-Tutrix en deux points. Premièrement, la nécessité de permettre à l'élève de résoudre le problème de façon naturelle. Deuxièmement, la capacité d'évaluer la preuve que l'élève est en train de construire afin de pouvoir déterminer les éléments de preuve cruciaux qui lui manquent pour compléter le problème.

Pour le premier point, QED-Tutrix dispose d'une interface Geogebra, un logiciel couramment utilisé en géométrie, dans lequel il peut manipuler la figure librement, la modifier, mesurer des angles ou des longueurs, ou ajouter de nouveaux éléments géométriques. En parallèle, le logiciel offre également un espace permettant à l'élève de rentrer des éléments de preuve, qu'ils soient des justifications telles que "Dans un triangle rectangle, le carré de l'hypoténuse est égal à la somme des carrés des deux autres côtés", ou des résultats mathématiques, tels que "Le segment AB mesure 5cm" ou "Les droites (AB) et (CD) sont parallèles". Afin de rendre le processus de résolution le plus naturel possible, ces éléments peuvent être rentrés dans n'importe quel ordre. L'élève peut donc partir des hypothèses et procéder itérativement, ou au contraire partir de la conclusion, ou même fournir un élément se trouvant au milieu de la preuve, et ce jusqu'à la résolution du problème. Enfin, lorsque le logiciel juge que l'élève est suffisamment avancé dans une preuve, un troisième volet est rendu disponible

par l'interface, où ladite preuve est rédigée au complet par le logiciel, mais avec des espaces vides à compléter pour tous les éléments mathématiques n'ayant pas été fournis. Ces éléments d'interface, à l'exception du volet “rédaction” qui n'est rendu disponible qu'à un certain point, sont accessibles tout au long du processus de résolution. Ainsi, quel que soit son avancement dans la résolution, il est toujours possible pour l'élève de manipuler la figure dans l'interface Geogebra.

Concernant le second point, le logiciel offre également un système tuteur poussé pour assister l'élève dans sa résolution. Les interactions avec le tuteur s'effectuent dans la dernière partie de l'interface, qui prend la forme d'une fenêtre de discussion. Les “messages” de l'élève correspondent en fait aux éléments mathématiques qu'elle a fournis au fur et à mesure de sa résolution. En réponse à chacun de ses éléments, le tuteur, nommé Prof. Turing, lui fournit une rétroaction immédiate pour indiquer si son entrée est valide (i.e. utile pour une preuve du problème), invalide, ou quelques autres variantes : invalide mais dont la réciproque est valide, valide mais a déjà été entrée, ou enfin invalide et a déjà été entrée. De plus, lorsque le logiciel détecte que l'élève est bloqué, c'est-à-dire qu'il n'a pas fourni de résultat valide depuis un certain temps, des messages d'aide sont fournis par le Prof. Turing. Le choix des messages envoyés pour l'aider est expliqué en détail dans les thèses de Nicolas Leduc [2] et Michèle Tessier-Baillargeon [3]. Pour résumer les points importants, ces messages sont déterminés en fonction de plusieurs choses. Premièrement, comme un problème mathématique typique a la plupart du temps plusieurs preuves possibles, le logiciel doit évaluer, parmi ces preuves possibles, celle dans laquelle l'élève est la plus avancée. Deuxièmement, parmi les inférences constituant cette preuve n'ayant pas encore été complétées, le logiciel en choisit une qui implique l'élément mathématique rentré le plus récemment. Par exemple, si l'élève a rentré le Théorème de Pythagore en dernier, alors le logiciel va choisir une inférence utilisant le Théorème de Pythagore. Enfin, le logiciel va fournir des messages pour aider l'élève à compléter cette inférence, par exemple “Quels antécédants sont nécessaires pour démontrer le Théorème de Pythagore ?”, ou “Quelle conclusion peut-on tirer du fait que ABC est un triangle rectangle à l'aide du Théorème de Pythagore ?” Note : ces messages ne sont pas les messages exacts envoyés par QED-Tutrix dans cette situation, mais ont été édités pour alléger la lecture, car les messages originaux, étant générés automatiquement, sont assez verbeux.

Ces deux points, mis en commun, créent des contraintes très spécifiques. En effet, il devient nécessaire que le logiciel connaisse, à l'avance, toutes les preuves possibles pour le problème. Pire, comme certains éléments de résolution peuvent servir dans plusieurs preuves, il est nécessaire d'avoir une structure contenant toutes les preuves simultanément, et de pouvoir parcourir cette structure pour noter les éléments que l'élève a rentrés. Pour cela, Nicolas Leduc a créé plusieurs structures de données, détaillées dans sa thèse [2]. La plus intéres-

sante pour nous est le graphe HPDIC, ces initiales décrivant les divers éléments constituant le graphe, les Hypothèses, Propriétés, Définitions, résultats Intermédiaires, et Conclusion. Ce graphe a pour objectif de regrouper, pour un problème donné, l'ensemble des preuves possibles. Pour cela, comme nous l'avons mentionné précédemment, nous considérons qu'une preuve est une succession d'inférences, le conséquent d'une inférence et les hypothèses du problème pouvant servir d'antécédent à une ou plusieurs inférences. Ainsi, il est assez aisé de combiner les preuves d'un problème dans une seule structure, comme on peut le voir dans la Figure 1.1. Notons au passage que le terme "résultat intermédiaire" englobe tous les résultats mathématiques pouvant servir d'antécédent ou de conséquent à une inférence dans au moins une preuve, qui ne sont ni une hypothèse du problème, ni sa conclusion. Pour résumer, le graphe HPDIC est au coeur du fonctionnement de QED-Tutrix, et l'une des étapes cruciales pour l'ajout de nouveaux problèmes au logiciel est la génération de leur graphe.

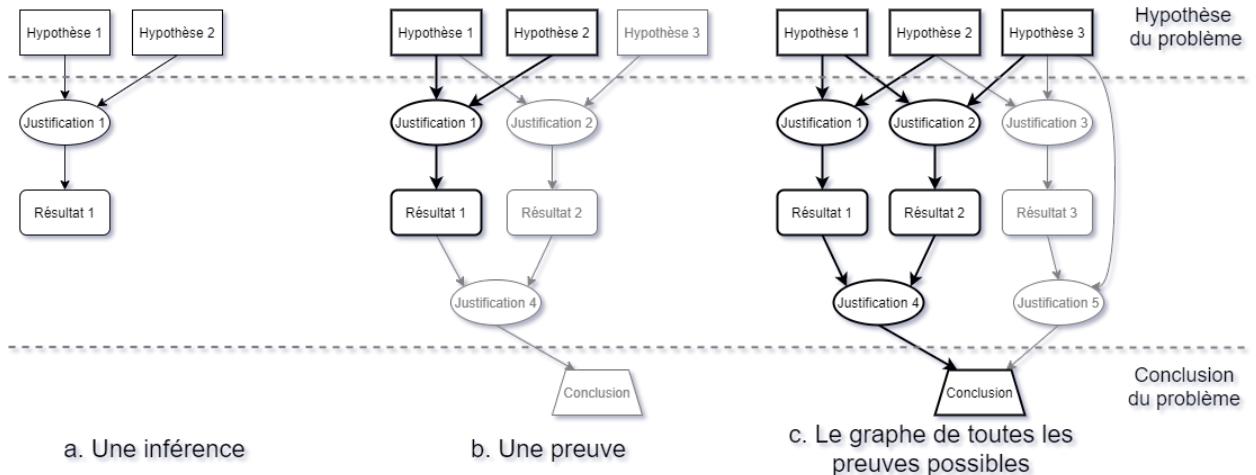


Figure 1.1 Le processus de construction du graphe HPDIC

## 1.2 Le besoin d'un générateur de preuves automatisé

La première itération de QED-Tutrix ne disposait pas d'outil pour générer le graphe HPDIC d'un problème automatiquement. Par conséquent, l'ajout d'un nouveau problème au logiciel requérait une création manuelle de son graphe, ce qui est une tâche fastidieuse, répétitive, et sensible aux erreurs. Pour cette raison, cette première itération ne proposait que 5 problèmes types, ayant été judicieusement choisis pour démontrer les capacités de QED-Tutrix. Chacun de ces problèmes étaient adjoints d'un graphe HPDIC, construit à la main. Bien que les problèmes soient de manière générale d'une complexité semblable dans leur énoncé et leur

possible résolution, la taille des graphes varie considérablement. En effet, il faut prendre en compte que le graphe HPDIC contient *toutes* les preuves possibles.

Prenons comme exemple le problème suivant : “Démontrer que si un quadrilatère ABCD a trois angles droits, alors c'est un rectangle.” Ce problème dispose de plusieurs chemins de résolution. Il est en effet possible de le résoudre par la somme des angles d'un quadrilatère, ou en passant par des propriétés sur le parallélisme et la perpendicularité des droites pour montrer que le quatrième angle est droit, ou encore utiliser ces résultats pour prouver que ABCD est un parallélogramme. Cependant, chacune de ces preuves n'existe qu'en peu de variantes : il n'y a qu'une seule façon de faire le calcul de la somme des angles d'un quadrilatère. Considérons maintenant un autre problème : “Prouver que le triangle formé par les milieux des trois côtés d'un triangle équilatéral est lui aussi équilatéral.” La situation géométrique de ce problème est donnée à la Figure 1.2. Ce problème n'est pas tellement plus complexe que le précédent et peut se résoudre aisément avec quelques calculs de sommes de mesures d'angles. Cependant, le nombre important d'angles et de triangles dans cette figure fait que le nombre de preuves possibles, et donc la taille du graphe HPDIC, est colossal. Pour illustrer ce point, dans un premier temps, considérons uniquement les angles mesurant 60 degrés, et les propriétés réciproques “Si deux angles sont de même mesure, alors ils sont isométriques” et “Si deux angles sont isométriques, alors ils sont de même mesure.” Ces propriétés, bien que triviales (c'est la définition de l'isométrie), sont nécessaires au bon fonctionnement du générateur de preuves. En quelques mots, elles sont nécessaires pour passer de la mesure numérique d'un angle ( $\widehat{ABC} = 60$  et  $\widehat{BCA} = 60$ ) à la notion d'égalité ( $\widehat{ABC} = \widehat{BCA}$ ), représentés en Prolog par des faits distincts. Pour en revenir à l'explosion du graphe, il existe dans notre problème, au total, 12 angles mesurant 60 degrés. Chaque couple d'angles permet d'inférer un résultat sur leur similitude, soit un total de 66 résultats de similitude ( $12 * 11 / 2$ ). À leur tour, chacun de ces résultats de similitude permet, par la réciproque, d'inférer la mesure de deux angles, soit à nouveau 132 inférences. Il suffit maintenant de considérer les propriétés “Deux triangles dont les angles sont isométriques sont eux aussi semblables” et sa réciproque, “La mesure de l'angle formé de deux angles adjacents est égale à la somme des mesures de ces deux angles” et sa réciproque, ou encore “Deux triangles semblables dont deux côtés homologues sont de même longueur sont congrus” et sa réciproque, pour se convaincre que la taille du graphe HPDIC peut aisément exploser, même sur un problème relativement simple.

Ainsi, la création manuelle des graphes n'est pas une solution viable à long terme, et il est donc nécessaire d'automatiser ce processus au maximum. Il s'agit là de l'enjeu central de ce projet de recherche.

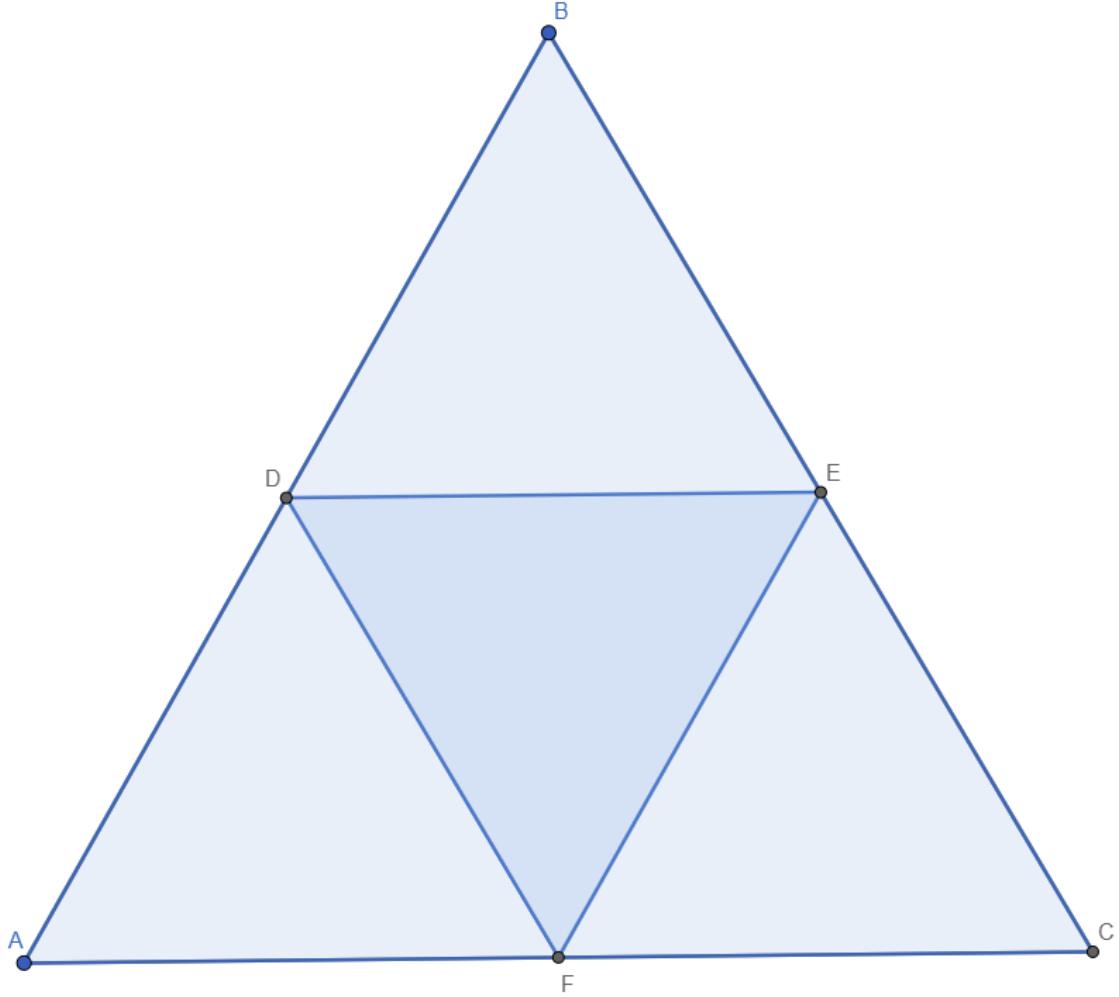


Figure 1.2 Un triangle équilatéral et le triangle formé par les milieux de ses côtés

### 1.3 Enjeux didactiques

D'un point de vue purement informatique, l'utilisation de la programmation logique rend la tâche de génération de l'ensemble des preuves relativement aisée. En effet, la construction du graphe HPDIC d'un problème consiste à partir d'un ensemble de faits mathématiques, les hypothèses du problème, et d'un ensemble de justifications, le référentiel, puis d'inférer tout ce qu'il est possible d'inférer à partir de ces deux ensembles. Par la suite, il suffit d'identifier, parmi les résultats inférés, lequel est la conclusion du problème, et de ne garder que les chemins menant à ce résultat. Cependant, cette vue optimiste se heurte rapidement à la réalité.

En effet, l'objectif ultime de ce projet est d'alimenter un logiciel tuteur, conçu pour s'aligner

sur les situations rencontrées dans une classe réelle, avec tous les raccourcis, approximations et abstractions utilisés. Dans le but d'aligner le processus informatique avec cette réalité, nous nous sommes heurté à deux difficultés majeures. Premièrement, le référentiel utilisé est calqué sur les propriétés et définitions utilisées en classe, telles qu'identifiées par les experts en didactique. Ce référentiel “réel” est composé de propriétés n'étant pas rigoureusement définies, car les cours de mathématique au secondaire n'exigent pas une rigueur absolue dans l'utilisation de propriétés. Dépendamment du professeur, du niveau de la classe et de l'exercice abordé, certains antécédents ou hypothèses peuvent être omis, voire certaines étapes de la démonstration sautées.

Par exemple, considérons la situation présentée en Figure 1.3.  $X$  et  $Y$  sont tous deux équidistants de  $A$  et  $B$ . Dans la plupart des cas, la preuve suivante serait acceptée par le professeur :

1. *Tout point de la médiatrice d'un segment est équidistant des deux extrémités du segment.*
2. *Comme  $X$  et  $Y$  sont tous deux équidistants de  $A$  et  $B$ , alors la droite ( $XY$ ) est la médiatrice du segment  $[AB]$ .*

Cependant, dans le cadre d'un exercice ayant pour but de pratiquer la notion de médiatrice, ou dans le cas d'un professeur particulièrement exigeant, cette démonstration pourrait ne pas être acceptée, car elle passe par plusieurs raccourcis. Il pourrait alors plutôt s'attendre à une démonstration telle que :

1. *Tout point équidistant des deux extrémités d'un segment est sur la médiatrice de ce segment.*
2. *Comme  $X$  est équidistant de  $A$  et  $B$ ,  $X$  est sur la médiatrice de  $[AB]$ .*
3. *Par le même raisonnement,  $Y$  est également sur la médiatrice de  $[AB]$ .*
4. *Comme  $X$  et  $Y$  sont deux points distincts, il existe une et une seule droite passant par  $X$  et  $Y$ .*
5. *Comme  $X$  et  $Y$  sont tous deux sur la médiatrice de  $[AB]$ , alors ( $XY$ ) est la médiatrice de  $[AB]$ .*

Cette grande variabilité crée une contrainte supplémentaire. Il est en effet non seulement nécessaire d'identifier toutes les preuves possibles au sens large, mais également toutes les variations dans chacune de ces preuves.

Deuxièmement, l'encodage informatique de propriétés mathématiques exige une définition précise et non ambiguë. Dans certaines situations, cela est particulièrement délicat, et demande d'introduire des concepts plus complexes sortant du cadre de la géométrie enseignée

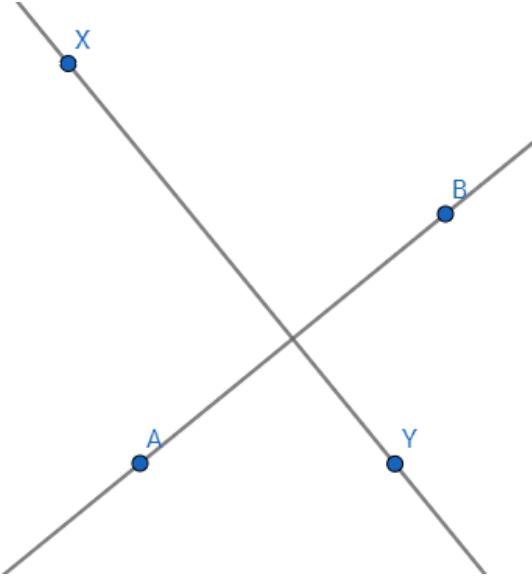


Figure 1.3 Une simple situation géométrique

au secondaire. Par exemple, la simple formule  $\widehat{AOB} + \widehat{BOC} = \widehat{AOC}$  serait acceptée comme justification dans la plupart des démonstrations... mais ne s'applique que si le point  $B$  se trouve dans la section du plan délimitée par les demi-droites  $[OA)$  et  $[OC)$  en ordre trigonométrique. Alternativement, un critère plus simple pour que cette formule s'applique est que les angles  $\widehat{AOB}$ ,  $\widehat{BOC}$  et  $\widehat{AOC}$  soient orientés. Ainsi, pour une opération aussi simple que l'addition d'angles, il devient nécessaire d'introduire des concepts dépassant le cadre d'un cours de secondaire, dans ce cas la notion d'angle orienté.

Ces enjeux sont présentés plus en détail dans le chapitre 7, contenant le second article de ce mémoire.

#### 1.4 Objectifs de recherche

Le but central de ce projet est ainsi de développer un système capable de générer toutes les preuves d'un problème de géométrie, en restant dans un référentiel de résolution accessible à un élève du secondaire. Ce fil conducteur a été divisé en plusieurs questions de recherche.

La première est d'évaluer quelle méthode, dans le domaine de la démonstration automatique, serait la plus pertinente pour générer cet ensemble. Le premier article, chapitre 4, répond à cette question et justifie le choix de la programmation logique. C'est à ce stade que nous avons posé notre première hypothèse de recherche. En effet, nous avons constaté des similarités fortes entre le processus d'inférence de la programmation logique et celui d'inférence dans une démonstration mathématique (voir chapitre 6.5.2). Notre hypothèse est donc que

la programmation logique est suffisamment flexible et puissante pour, premièrement, générer automatiquement le graphe HPDIC d'un problème, et, deuxièmement, prendre en considération les contraintes didactiques lors de cette génération.

La seconde question est de déterminer comment implémenter l'intégralité du référentiel didactique de géométrie au secondaire dans un format exploitable par Prolog, et ce en évitant l'explosion combinatoire typique de tels systèmes de démonstration automatique. Cette question est explorée dans les deuxième et troisième articles, chapitres 5 et 6.

La troisième question, intimement liée à la seconde, est de deviser une méthode de validation pour quantifier la portion du référentiel de géométrie au secondaire théorique couverte par le référentiel concrètement implémenté. La méthode créée est présentée dans le troisième article, chapitre 6.

Enfin, la quatrième question est de valider le deuxième point de notre hypothèse de recherche : la programmation logique est-elle suffisante pour prendre en compte nos contraintes didactiques ? Nous tentons d'y répondre dans le quatrième article, chapitre 7.

## 1.5 Plan de la thèse

Dans le chapitre 2, nous analysons les différents travaux ayant été effectués par la communauté scientifique, qui sont pertinents pour notre travail. Ensuite, le chapitre 3 contient une vue d'ensemble du projet et présente l'articulation des quatre articles scientifiques constituant le cœur de cette thèse. Ces quatre articles forment les chapitres 4, 5, 6 et 7. Le premier et le second sont issus de présentations à la conférence ThEdu (Theorem Proving Components for Educational Software). Le premier, présenté à ThEdu'17<sup>4</sup>, explique comment ce projet de génération automatisée de preuve s'inscrit dans le développement de QED-Tutrix [4]. Le second, présenté deux ans après à ThEdu'19<sup>5</sup>, contient une explication plus détaillée sur le fonctionnement de cet outil et sur le processus de génération de preuves, allant de l'encodage du problème à la création du graphe HPDIC [5]. Le troisième est une publication dans le livre Mathematics Education in the Digital Era, en cours de publication par Springer<sup>6</sup>, et offre une vue d'ensemble *a posteriori* du travail effectué dans ce projet [6]. Enfin, le quatrième a été présenté au symposium ETM6<sup>7</sup> (Espaces de Travail Mathématique), et est axé sur les problématiques issues de la rencontre entre les raisonnements informatique et humain [7].

---

4. <https://www.uc.pt/en/congressos/thedu/thedu17>

5. <https://www.uc.pt/en/congressos/thedu/thedu19>

6. <https://www.springer.com/series/10170>

7. [http://www.pucv.cl/uuaa/site/edic/base/port/didactica\\_de\\_la\\_matematica.html](http://www.pucv.cl/uuaa/site/edic/base/port/didactica_de_la_matematica.html)

## CHAPITRE 2 REVUE DE LITTÉRATURE

Cette thèse comprenant quatre articles, chacun disposant de sa section revue de littérature propre, ce chapitre a pour but de résumer les points centraux des différents volets de la littérature pertinente, plutôt que de répéter ce qui est déjà présenté en détail dans ces articles. Comme ce projet est, premièrement, un outil alimentant le logiciel tuteur QED-Tutrix, et, deuxièmement, à l'intersection du domaine informatique de la démonstration automatique et du domaine de la didactique des mathématiques, la littérature pertinente se divise en trois parties.

### 2.1 Logiciels tuteurs

L'objectif premier de ce projet est d'étendre la portée de QED-Tutrix en simplifiant le processus d'ajout de nouveaux problèmes. Par conséquent, il est pertinent pour ce projet de disposer d'une vue d'ensemble des logiciels tuteurs en mathématiques, et plus particulièrement en géométrie. L'enseignement des mathématiques dispose d'un écosystème d'outils informatiques vaste et varié, pouvant aller de simples recueils d'information passifs, tels que Mathway [8], à des logiciels plus poussés modélisant le chemin d'apprentissage de l'élève tels qu'Active-Math [9], en passant par les outils de manipulation d'objets, précieux pour l'enseignement de la géométrie, tels que GeoGebra [10].

L'originalité de QED-Tutrix parmi ces outils divers provient d'une combinaison de caractéristiques qui n'apparaît dans aucun autre système. En effet, QED-Tutrix a été construit basé sur des fondements didactiques divisant le processus de démonstration en trois étapes : exploration du problème, identification des éléments mathématiques pertinents pour sa résolution, et rédaction de la preuve. Ces trois étapes n'étant pas nécessairement séquentielles ni monolithiques, il est important de permettre à l'élève de passer librement de l'une à l'autre. La première s'incarne par l'utilisation d'une interface GeoGebra, la seconde par la possibilité de construire sa résolution en allant sélectionner des éléments mathématiques (résultats ou justifications) dans n'importe quel ordre, et la troisième par l'utilisation d'une interface de rédaction intuitive. Il est également crucial que le logiciel tuteur soit capable de fournir une aide personnalisée à n'importe laquelle de ces trois étapes, et ce, de façon autonome du professeur dans les cas d'utilisation typiques.

Ces caractéristiques rendent QED-Tutrix unique par rapport aux nombreux autres systèmes existant. De plus amples détails sur les logiciels tuteurs existants sont présentés dans les

revues de littératures du premier et troisième article (sections 4.4 et 6.3.2), ainsi que de façon très détaillée dans la thèse de Leduc [2].

## 2.2 Démonstration automatique

Parmi ces logiciels tuteurs, certains tentent également de modéliser le processus de résolution par une représentation *a priori* de l'ensemble des preuves possibles dans un référentiel donné. L'exemple le plus proéminent est le logiciel Mentoniezh [11–13], un des principaux précurseurs spirituels de QED-Tutrix. Les problèmes implémentés dans ce logiciel ont d'ailleurs alimenté notre base de validation, tel que présentée en section 6.5.6. Ceci étant dit, le domaine de la démonstration automatique dépasse largement le cadre des logiciels tuteurs. En effet, l'usabilité pour l'enseignement des outils développés n'est que rarement considérée comme une priorité, et les efforts sont généralement mis sur la performance. Les outils de démonstration automatique se divisent en deux grandes familles : les approches algébriques, traduisant le problème sous une autre forme plus aisée à résoudre par des méthodes informatiques, comme des polynômes, et les approches synthétiques ou axiomatiques, basées sur une succession d'inférences. La seconde étant typiquement moins performante, la plupart des systèmes développés utilisent une approche algébrique ou semi-algébrique [14–20]. Ce constat réduit drastiquement les possibilités d'utiliser un système existant pour la génération de preuves. Combiné avec nos exigences didactiques précises en termes de malléabilité du référentiel et d'accessibilité des preuves au secondaire, ceci nous a orienté vers la décision d'implémenter notre propre système. Le choix de la programmation logique a été fait pour les raisons énoncées dans l'introduction, et est une option qui a été explorée par le passé [21, 22]. Ces travaux datant de trente ans visant plus à démontrer les capacités de Prolog qu'à concrètement générer des preuves pour un problème mathématique, leur apport est essentiellement de confirmer qu'il s'agit d'une approche valide.

## 2.3 Fondations didactiques

Un autre point clé par lequel ce projet se distingue des autres approches de démonstration automatique est que notre objectif est avant tout d'alimenter le logiciel QED-Tutrix, avec par conséquent un accent important sur les contraintes didactiques que cela impose. En effet, le projet QED-Tutrix est fondé principalement sur deux théories didactiques : les Espaces de Travail Mathématiques [23–25] et la Théorie des Situations Didactiques [26].

En quelques mots, la première tente d'identifier les différents objets, actions et interactions ayant lieu dans une situation d'enseignement des mathématiques. Plus précisément,

elle modélise l'apprentissage par des échanges entre le plan épistémologique, représentant la connaissance mathématique au sens large, et le plan cognitif, représentant les connaissances de l'élève. Ces échanges se font dans un espace délimité par trois axes. Le premier est la genèse instrumentale, représentant l'apprentissage de l'utilisation d'outils, tels que la règle et le compas, ou, dans notre cas, un logiciel tuteur. Le second est la genèse sémiotique, représentant l'apprentissage des concepts mathématiques, tels que la notion d'angle ou le théorème de Pythagore. Enfin, le troisième est la genèse discursive, représentant l'apprentissage des règles régissant le discours mathématique, typiquement l'écriture d'une démonstration cohérente. L'espace délimité par ces deux plans et trois axes représente l'apprentissage mathématique. Ce modèle permet ainsi de décomposer et quantifier de façon précise les échanges entre le professeur, la matière, et l'élève.

La seconde, la Théorie des Situations Didactiques, considère que l'apprentissage des mathématiques se fait en grande partie par la pratique, et plus précisément par la résolution de problèmes. Ainsi, la tâche de l'enseignant n'est pas seulement d'expliquer la matière à la classe, mais également de bien choisir ou construire les problèmes qu'il donne à résoudre aux élèves. De plus, pour un apprentissage optimal, l'enseignant doit s'assurer de la *dévolution* du problème aux élèves. Ce terme signifie que l'élève s'approprie le problème, en s'intéressant personnellement à sa résolution, plutôt que de le voir comme une épreuve ou une corvée, ce qui est bien entendu une tâche particulièrement ardue. Cette théorie considère également que la transmission des connaissances lors de la résolution d'un problème passe également en grande partie par l'exploration, à la fois du problème, par exemple en traçant de nouvelles droites ou en manipulant la figure dans un outil de géométrie dynamique, et à la fois des outils dont il dispose, par exemple de l'ensemble des théorèmes vus en classe. C'est pour cette raison que QED-Tutrix met l'accent sur la possibilité d'explorer le problème en manipulant la figure et permet à l'élève de parcourir l'ensemble des justifications du référentiel lors de sa résolution.

Pour une lecture plus poussée sur les liens entre celles-ci et le projet QED-Tutrix, le lecteur peut s'orienter vers la thèse de Tessier-Baillargeon [3].

## CHAPITRE 3 MÉTHODOLOGIE

### 3.1 Genèse du projet

Comme expliqué dans l'introduction, l'objectif central de ce projet est le développement d'un outil prenant en entrée un “problème mathématique”, encodé selon un processus détaillé plus loin, ainsi qu'un ensemble de justifications (propriétés et définitions) autorisées, le référentiel, et renvoyant en sortie un graphe HPDIC contenant l'ensemble des solutions à ce problème dans le référentiel fourni. Le premier article de cette thèse, dans le chapitre 4, présente l'état de nos réflexions sur la façon de résoudre ce problème après la phase exploratoire du projet. Il présente également brièvement le logiciel tuteur QED-Tutrix, dont nous souhaitons étendre la portée en facilitant le processus d'implémentation de nouveaux problèmes. Le point essentiel à retenir de cet article est : bien que notre objectif de résoudre automatiquement des problèmes entre dans le vaste domaine de recherche de la démonstration automatisée de théorème (ATP ou Automated Theorem Proving), l'utilisation prévue de cet ensemble de preuves par un logiciel tuteur crée des contraintes, à la fois informatiques et didactiques, nous empêchant d'utiliser les systèmes de démonstration déjà existants, ce qui nous a amené à développer notre propre générateur de preuves. En revanche, ces travaux de développement utilisant la programmation logique, en particulier les obstacles didactiques rencontrés, ont suscité un intérêt fort de la part de la communauté.

Le choix de la programmation logique comme réponse à notre première question de recherche n'est pas anodin, comme présenté dans les premier et deuxième articles, chapitres 4 et 5. En effet, le processus d'inférence en programmation logique, où un ou plusieurs faits (*facts*) sont combinés à une règle (*rule*) pour créer de nouveaux faits, est très proche du processus d'inférence dans une démonstration mathématique, où un ou plusieurs faits mathématiques (les antécédents) sont combinés à une justification, comme un théorème, pour obtenir un nouveau fait (le conséquent). Ce simple constat rend relativement simple la création d'un moteur inférentiel, l'élément central du générateur de preuves, en encodant les justifications mathématiques comme autant de règles, et en formalisant l'ensemble des résultats mathématiques pouvant apparaître dans une preuve comme des faits.

### 3.2 Contraintes didactiques

Dans l'introduction, section 1.3, nous avons mentionné deux enjeux didactiques majeurs : la représentation de variations fines dans une preuve, dépendantes du niveau de la classe,

des exigences du professeur et des besoins épisodiques de la leçon en cours, et la différence entre les concepts géométriques rigoureux et ceux utilisés dans une démonstration au secondaire. Ces enjeux sont cruciaux pour QED-Tutrix et ont ainsi guidé le développement de ce générateur de preuves, comme indiqué par notre quatrième question de recherche : la programmation logique est-elle suffisante pour prendre en compte nos contraintes didactiques ? Répondre à cette question de façon absolue serait un projet de recherche en didactique en soi, car cela nécessiterait d'effectuer plusieurs études en classe, en comparant les preuves écrites par les élèves à la main (et acceptées par le professeur) avec celles générées informatiquement, et ce à plusieurs points dans le parcours scolaire, et avec plusieurs professeurs suivant plusieurs styles d'enseignement. Cette validation est donc bien au-delà de la portée de cette thèse. Par conséquent, afin de valider cet aspect précis, ce projet a été étroitement supervisé par des experts en didactique des mathématiques, qui se sont assurés du respect des enjeux didactiques à chaque étape. De plus, il s'est effectué en jumelage avec un projet de référencement des propriétés et définitions utilisées au secondaire [27]. Les deux projets se sont alimentés mutuellement, l'implémentation informatique profitant de la formalisation exhaustive du référentiel du secondaire, et ce référencement tirant des leçons de la modélisation des propriétés comme règles informatiques, à la fois pour la représentation explicite des raccourcis, la formulation canonique des énoncés, et l'identification des propriétés minimales pour contenir tout le référentiel.

Par ailleurs, nos solutions pour respecter ces enjeux et nos résultats ont été présentées à la communauté scientifique de la didactique, par le biais du quatrième article, présenté chapitre 7. Bien qu'imparfaite, cette méthode fournit une bonne garantie que le générateur de preuves créé est, sur le plan didactique, un ajout pertinent à QED-Tutrix.

Ainsi, pour assurer le bon déroulement à la fois de ce projet et de sa contrepartie didactique, un partage de connaissances important s'est effectué. Au delà de la validation par des experts en didactique, ce projet a nécessité une appropriation profonde des concepts didactiques régissant QED-Tutrix. En particulier, il a été nécessaire de nous familiariser avec les notions de référentiel mathématique, de raccourci inférentiel, et plus généralement sur le déroulement du processus d'apprentissage mathématique au sens large.

### 3.3 Évaluation

Une évaluation idéale du générateur de preuves créé lors de ce projet consisterait à valider sa capacité à générer toutes les preuves possibles (dans un référentiel donné), sur l'ensemble des problèmes existant au niveau secondaire. Cet objectif est bien entendu irréalisable dans la réalité, premièrement parce que le nombre de preuves possibles d'un problème peut être

dramatiquement élevé, et deuxièmement parce que l'ensemble des problèmes existant au niveau secondaire est également bien trop vaste, et sa collection consisterait un projet de recherche en soi. Par conséquent, afin de tout de même valider cet outil, nous avons procédé en deux étapes. Ces étapes sont brièvement expliquées dans cette section, et les résultats de l'évaluation sont présentés dans le troisième article, section 6.5.6.

### 3.3.1 Validation du processus

Premièrement, nous disposons des cinq problèmes déjà encodés dans QED-Tutrix, dont l'ensemble des preuves possibles a été encodé dans autant de graphes HPDIC, à la main, par des experts en didactique [28]. Ces problèmes ont été choisis pour recouvrir une grande partie des concepts vus au secondaire. Par conséquent, le premier objectif de la validation consiste à donner ces problèmes en entrée au générateur de preuves, et de comparer le graphe HPDIC fourni en sortie avec celui existant. Ceci assure que le processus de génération de preuves permet en effet de construire des graphes HPDIC complets et valides. Il est important de noter que parmi ces cinq problèmes, seuls quatre sont utilisables dans ce processus. En effet, le cinquième présente une situation impossible, et la conclusion attendue est “on arrive à une contradiction”, ce qui n'est pas un résultat que nous avons implémenté.

### 3.3.2 Validation du référentiel

Deuxièmement, nous disposons également d'un ensemble conséquent de problèmes de géométrie, compilés par un expert en didactique de l'équipe. Parmi cet ensemble, notre expert a sélectionné un sous-ensemble recouvrant l'intégralité des concepts vus au secondaire, mais où chaque problème est individuellement simple, dans le sens où il n'y a pas ou peu de variation dans la façon de le résoudre. La prochaine étape consiste à encoder chacun de ces problèmes en Prolog, et de générer le graphe HPDIC de chacun. Ensuite, nous considérons que le générateur a réussi à résoudre le problème avec succès si au moins une preuve a été trouvée, plutôt que d'entamer la tâche fastidieuse de vérifier que toutes les preuves possibles sont présentes dans le graphe HPDIC généré. Les problèmes étant simples avec peu de preuves possibles, il s'agit d'une bonne approximation. De plus, le but de cet étape n'est pas de valider la capacité du générateur à construire des graphes complets, mais sa couverture du référentiel de justifications utilisé au secondaire. Dans cette optique, la mesure du nombre de problèmes résolus en constitue une bonne estimation.

Ces deux étapes mises en commun assurent à la fois le bon fonctionnement du générateur de preuves et sa capacité à alimenter QED-Tutrix en problèmes recouvrant tout le programme de géométrie du secondaire.

## CHAPITRE 4 ARTICLE 1 : 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

**Reference :** L. Font, P. R. Richard et M. Gagnon, “Improving qed-tutrix by automating the generation of proofs,” *arXiv preprint arXiv :1803.01468*, 2018

### 4.1 Mise en contexte

Cet article, publié à Electronic Proceedings in Theoretical Computer Science [4], est une version étendue de l’article présenté à la conférence Theorem Proving Components for Educational Software (ThEdu’17). Il a été relu et accepté par le comité de la conférence ThEdu’17. Il s’agit de la première publication liée à ce projet. Ainsi, cet article présente le contexte dans le développement du logiciel QED-Tutrix ayant amené au besoin d’un générateur de preuves automatique. L’état de l’art (Section 5.4) y est assez détaillé, en particulier sur les logiciels tuteurs existants et les méthodes de démonstration automatique de théorèmes. Cette seconde partie est particulièrement d’intérêt, car elle explique la décision prise en début de projet de développer notre propre système en utilisant la programmation logique. La Section 4.5 présente le logiciel QED-Tutrix tel qu’il était au début de ce projet. Cependant, le logiciel ayant beaucoup évolué depuis, en particulier dans sa forme, cette section peut être mise de côté au profit de sa version plus récente, présentée au chapitre 6. La section suivante, en revanche, présente les enjeux initiaux, identifiés dès le début du projet, qui ont guidé le développement du générateur de preuves. Sa lecture est donc pertinente. Enfin, la dernière section illustre les limitations de l’approche choisie. Cette réflexion ayant beaucoup progressé, le lecteur est également invité à prioriser la discussion présentée en fin de ce mémoire, au chapitre 9.

### 4.2 Abstract

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.

### 4.3 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 [25]. In a learning context, when the student does his mathematician’s work, this dual perspective allows to interpret the development or the ma-

nifestation of geometric sense following **three geneses** at play between the epistemological and cognitive plans (see Fig. 4.1) : **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 [29]. 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 involve 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 involving users as soon as the software is designed is crucial. Furthermore, in the theory of mathematics education situations [30], 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 responsibilities 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 property 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, 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 5.4, we present systems similar to QED-Tutrix, and also existing tools for automated theorem proving that could be useful for the task of generating proofs. In Section 4.5, we provide a brief explanation of the functionalities of QED-Tutrix and its internal handling of proofs. Then, in Section 4.6, we present our goals for the improvement of QED-Tutrix 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 4.7, and discuss the limits of our work in Section 4.8. Finally, we present other avenues of research and conclude in Section 4.9.

## 4.4 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 [2] and Michèle Tessier-Baillargeon [3]. In Section 4.4.1, we summarize the important elements and, in Section 4.4.2, discuss the state-of-the-art for the novel problem of automated deduction.

### 4.4.1 Tutor softwares

In [2], 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 [8] or private tutoring companies. These are a fantastic knowledge bases, but are either passive tools or non-automated.

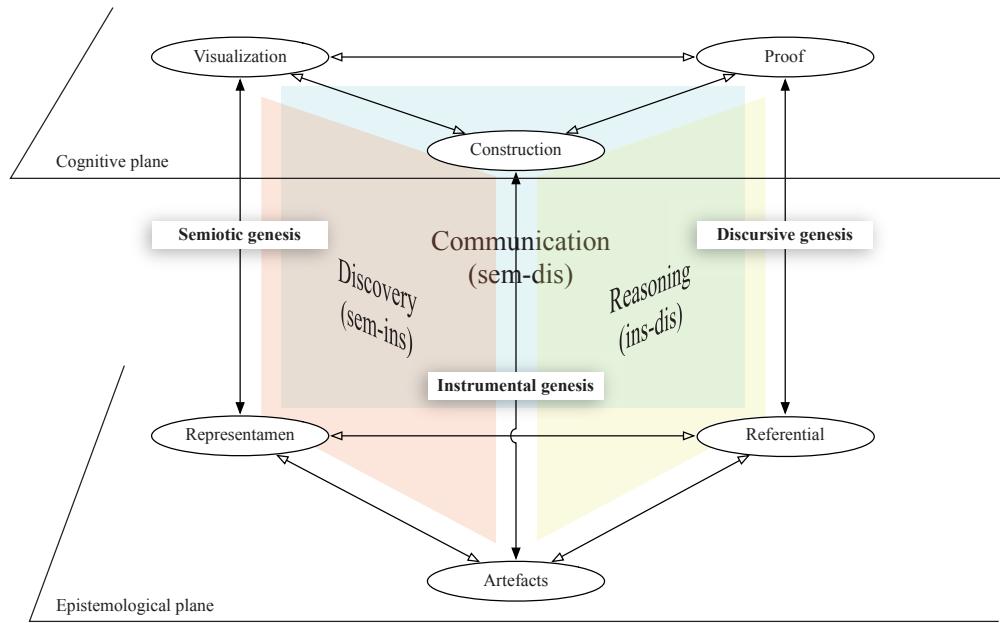


Figure 4.1 The model of mathematical working space in the theory.

**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 [31] for learning LISP, and ALEKS [32], ActiveMath [9] and Wayang Outpost [33] for mathematics specifically. These require a modelization of the user's knowledge, that can be done using various techniques, such as fuzzy logic [34] or Knowledge Components [35]. 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 [10], 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 4.4.2.

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 QED-Tutrix 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 [36,37]. 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 [38,39], that evolved into Geometry Cognitive Tutor [40–42]. This system is limited to problems based in cartesian coordinates, since it handles elements numerically. In QED-Tutrix, 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 [44]. 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 [45,46], unlike the previous ones, aims at helping the student to construct his proof. It is based on the Diagram Configurations [47] 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 [48,49], 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 [50, 51], Cabri-DEFI [52] and Cabri-Euclide [53, 54]. 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 [11–13] 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 QED-Tutrix. 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 [55], 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 [56, 57], that largely inspired QED-Tutrix, 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 QED-Tutrix project.

#### 4.4.2 Automated theorem proving

The main issue faced currently by QED-Tutrix is the difficulty 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 4.5.2). 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 4.7.1, along with an example of an inferential shortcut.

These 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 [14, 15], Wu's method [16, 17] and the exact check method [18]. Practical applications include the recent integration of a deduction engine in GeoGebra [19], 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 [20, 58], the full-angle method [59, 60], 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 [61, 62]. However, the geometry taught in high-school is based on Euclidean'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 [44]. 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 4.4.1. 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.

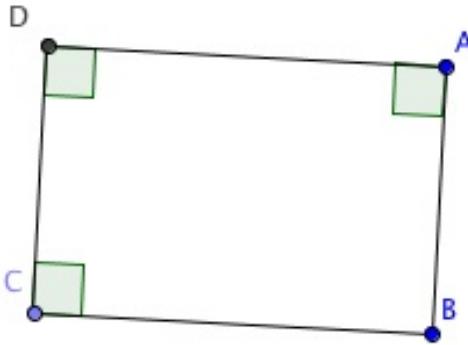
## 4.5 Presentation of QED-Tutrix

The QED-Tutrix software, or QED-Tutrix, 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. 4.2.

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.

### 4.5.1 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 QED-Tutrix. 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



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

Figure 4.2 An example of a simple geometrical proof problem.

students were closely monitored. These studies are explained in detail in the work of Michèle Tessier-Baillargeon [3] .

Since QED-Tutrix 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. 4.3 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 QED-Tutrix. 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.

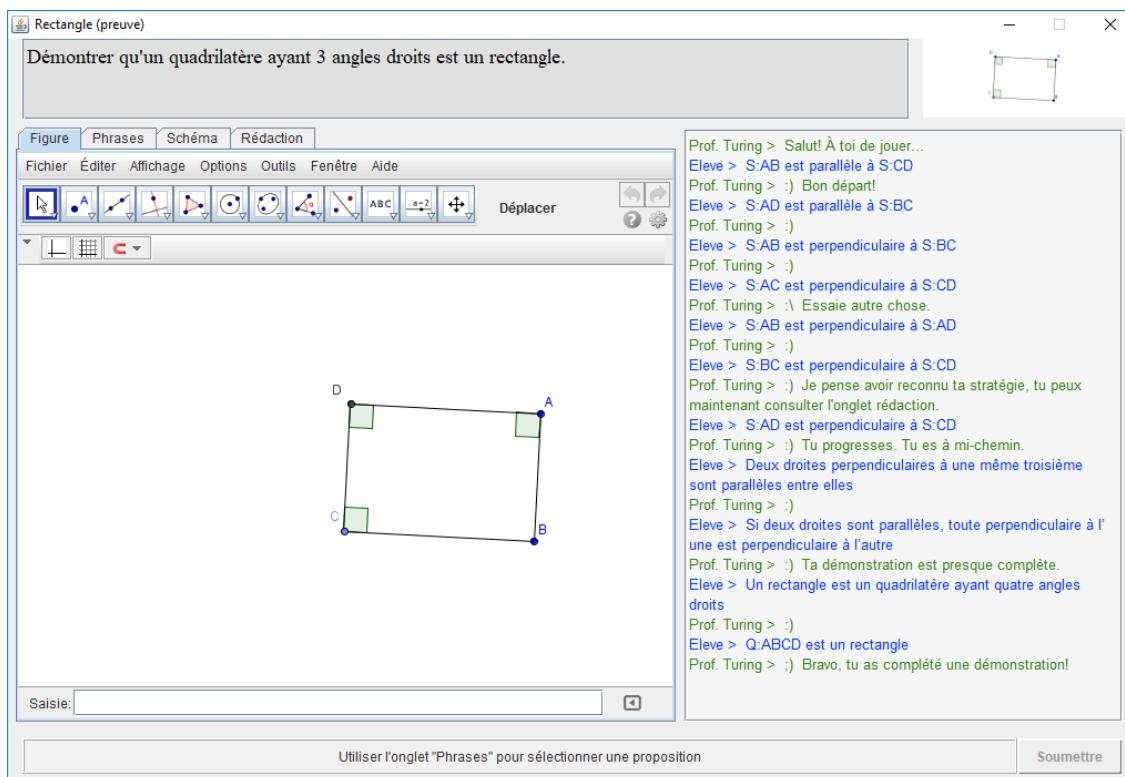


Figure 4.3 The main interface of QED-Tutrix.

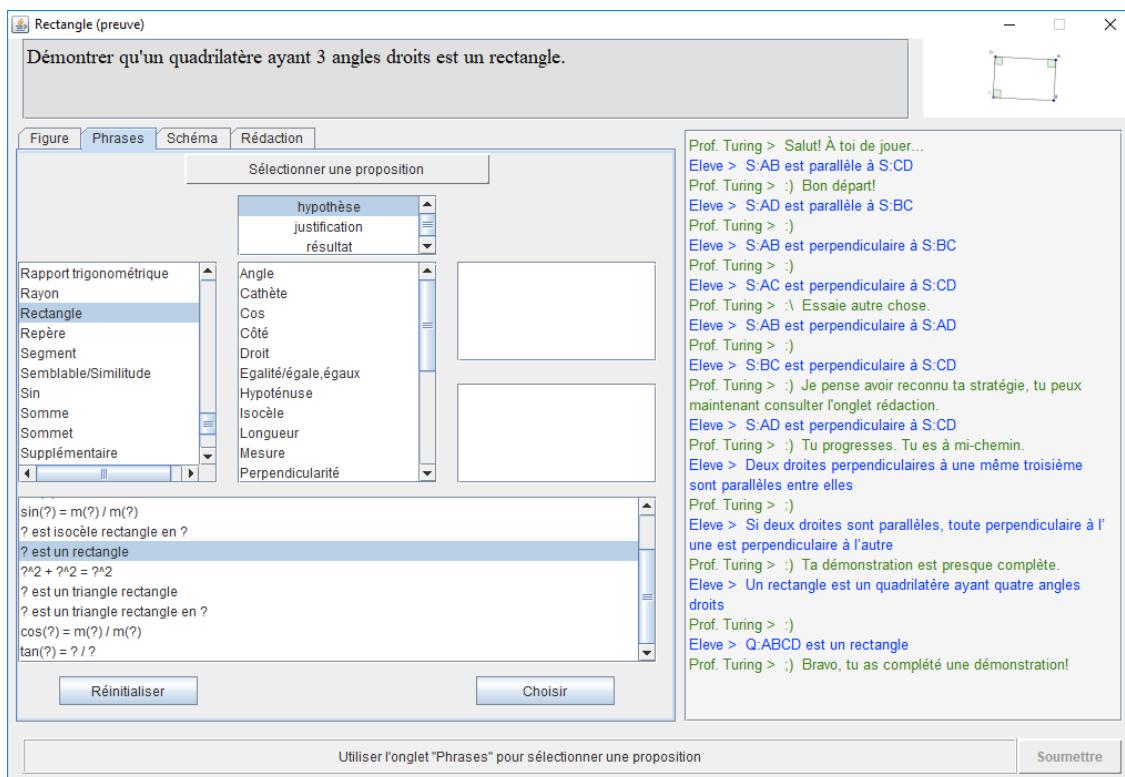


Figure 4.4 The main interface of QED-Tutrix, on the “Sentence” tab.

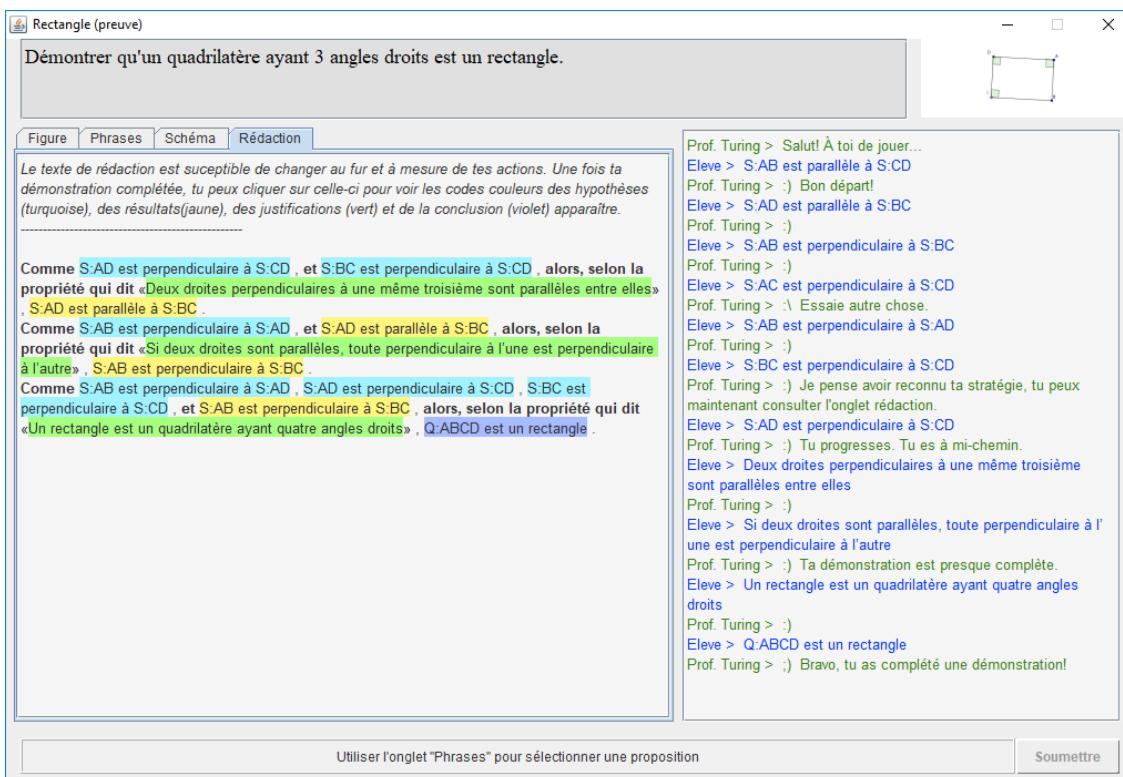


Figure 4.5 The main interface of QED-Tutrix, on the “Redaction” tab.

- The bottom is the input interface, where the student can complete the statement he wants to put in his proof.

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. 4.3. 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. 4.4, 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. 4.6, 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. 4.5, 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 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 QED-Tutrix 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

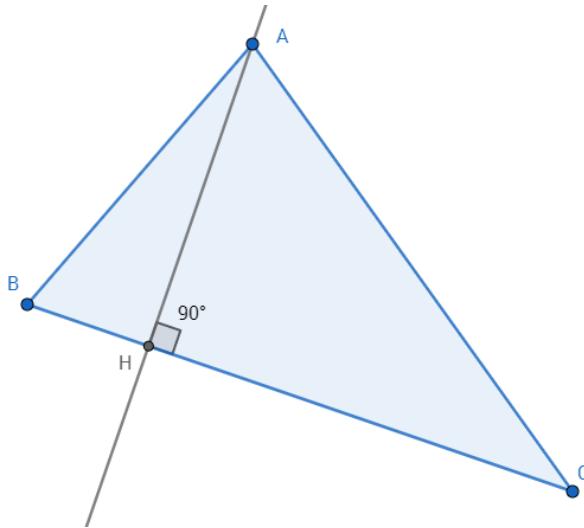


Figure 4.6 A geometrical construction of the height of a triangle.

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, who has experience and intuition, the combination of both seems like an excellent way to help the students in their learning process.

### 4.5.2 Internal engine

To allow this versatility, we introduced some novel elements in QED-Tutrix's' 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 [63] 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, using a set of axioms. The student explores a subset of the proofs represented in the graph. This subset evolves as the student 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 → conclusion), then a proof is a directed graph, whose starting nodes are the hypotheses, and the finishing node is 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. 4.2, “*Prove that any quadrilateral with three right angles is a rectangle*”, its graph would be the one in Fig. 4.7. 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 [64], where we first discussed the possibility of including connected problems in QED-Tutrix.

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. 4.7, this number remains small. However, this problem has an unusually simple graph. In another problem that is not much more complicated, shown in Fig. 4.8, 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 [2]. 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.

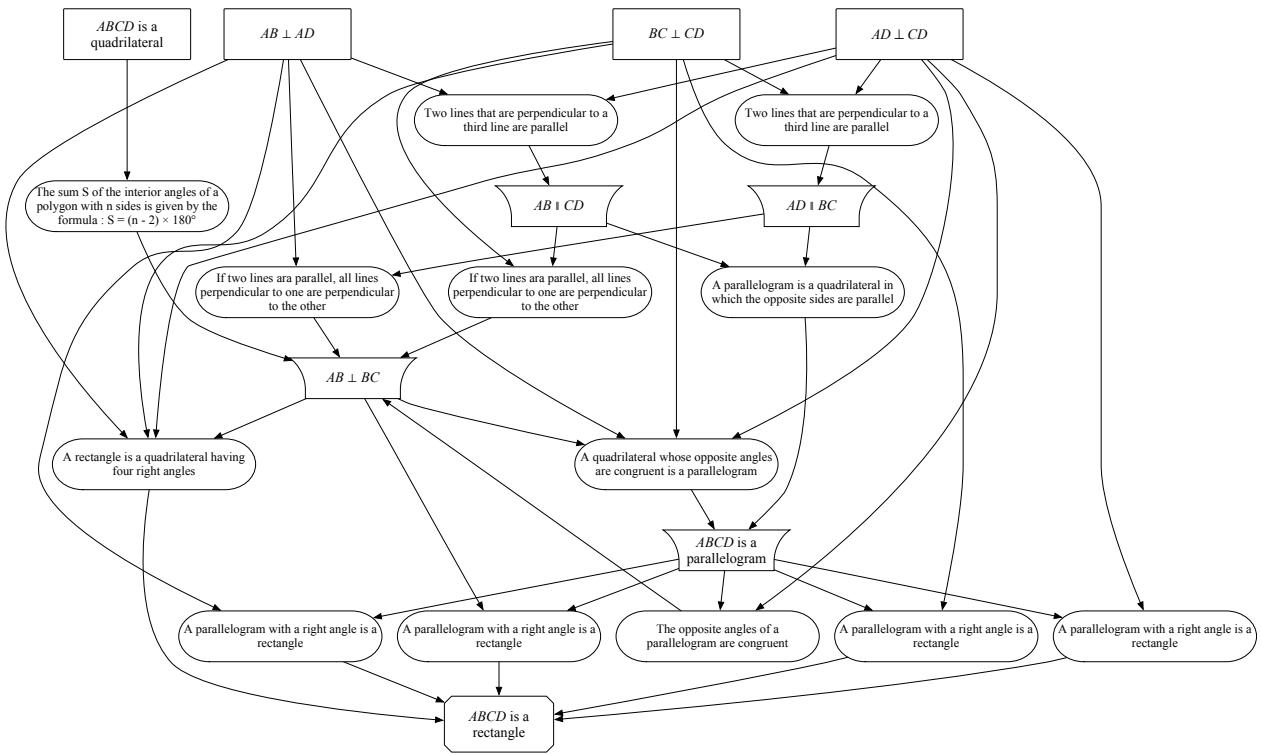
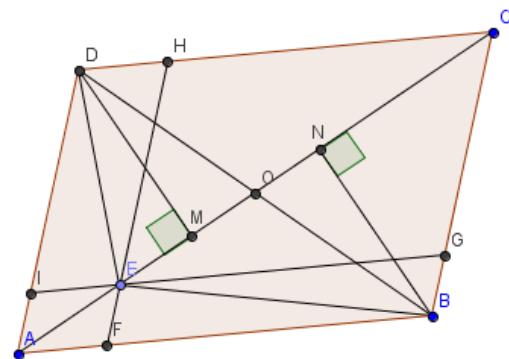


Figure 4.7 The HPDIC graph for the rectangle problem.



*"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 4.8 The parallelogram area problem.

## 4.6 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 towards the missing steps of the problem ; however, some teachers choose an alternative approach by giving the student another problem, **connected** to the original one. QED-Tutrix 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 [64] 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 chooses 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 QED-Tutrix, 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 QED-Tutrix, 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, QED-Tutrix 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 QED-Tutrix 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 QED-Tutrix, which includes the difficult step of writing down all possible proofs. Among the research avenues for the improvement of QED-Tutrix (see Section 4.9), this paper specifically presents the automated generation of proofs, in order to supply QED-Tutrix with a healthy amount of problems.

## 4.7 Automated generation of proofs

The idea of proving geometry theorem automatically is not new. As soon as 1960, Gelernter [65] proposed a synthetic proof method for geometry theorem proving. Other early approaches include those by Nevins [66], Elcock [67], Greeno et al. [68], Coelho and Pereira [69], and Chou, Gao and Zhang [60]. Since then, the methods have evolved considerably. However, our use-case is very specific for several reasons.

### 4.7.1 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 4.9. 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 [64] : “*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. 4.9, most teachers would accept the following proof :

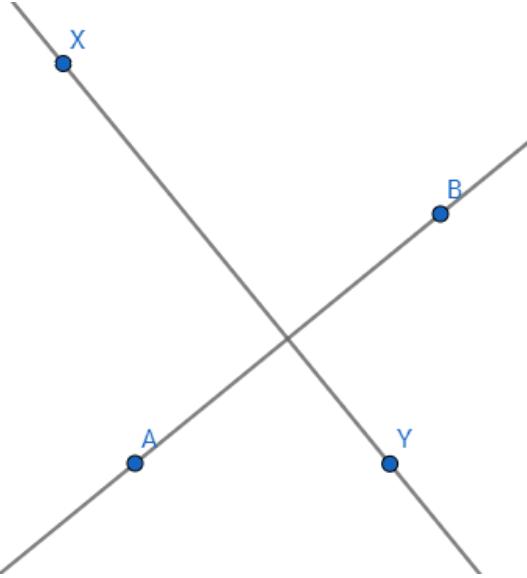
1. *Every point on the perpendicular bisector of a segment  $[AB]$  is equidistant from A and B.*
2. *Since X and Y are both equidistant from A and B, the line  $(XY)$  is the perpendicular bisector of  $[AB]$ .*

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 QED-Tutrix as a



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

Figure 4.9 A perpendicular bisector problem.

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.

#### 4.7.2 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 QED-Tutrix, 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. 4.9, 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.

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. 4.10.

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 QED-Tutrix 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 QED-Tutrix.

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 QED-Tutrix.

```

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

```

Figure 4.10 The inference engine algorithm.

## 4.8 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.

### 4.8.1 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 [3], following the precepts of [70] and *conception by usage* [71]. 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 4.6, there are only five problems that have been manually implemented in QED-Tutrix, 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, QED-Tutrix 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.

#### 4.8.2 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 4.7.2 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 QED-Tutrix, 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 [44] may be a possible solution.

### 4.9 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 QED-Tutrix 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 QED-Tutrix 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 QED-Tutrix, 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.

## CHAPITRE 5 ARTICLE 2 : AUTOMATING THE GENERATION OF HIGH SCHOOL GEOMETRY PROOFS USING PROLOG IN AN EDUCATIONAL CONTEXT

**Ludovic Font**

École Polytechnique de Montréal  
Montréal, QC, Canada  
ludovic.font@polymtl.ca

**Sébastien Cyr**

Université de Montréal  
Montréal, QC, Canada  
sebastien.cyr.1@umontreal.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

**Reference :** L. Font *et al.*, “Automating the Generation of High School Geometry Proofs using Prolog in an Educational Context,” *arXiv preprint arXiv :2002.12551*, 2020

### 5.1 Mise en contexte

Suivant le même format que l’article précédent, celui-ci a été publié dans Electronics Proceedings of Theoretical Computer Science comme version étendue de l’article présenté à la conférence ThEdu’19 [5]. Comme le précédent, il a été relu et accepté par le comité de la conférence ThEdu. Son objectif essentiel est ainsi de présenter l’avancement des travaux par rapport à l’ébauche faite dans la version précédente. La première section présente à nouveau rapidement le logiciel QED-Tutrix, et peut donc être mise de côté lors d’une première lecture. De même, la revue de littérature est fort similaire à celle de l’article précédent. L’intérêt principal de cet article se trouve dans la Section 5.5, où le processus complet de génération de preuve est présenté, de l’encodage initial du problème à la génération finale du graphe HPDIC résultant. Les limitations présentées dans les Sections 5.6 et 5.7 sont en revanche présentées plus en détail dans les Chapitres 7 et 9.2, respectivement.

### 5.2 Abstract

When working on intelligent tutor systems designed for mathematics education and its specificities, an interesting objective is to provide relevant help to the students by anticipating their next steps. This can only be done by knowing, beforehand, the possible ways to solve a

problem. Hence the need for an automated theorem prover that provide proofs as they would be written by a student. To achieve this objective, logic programming is a natural tool due to the similarity of its reasoning with a mathematical proof by inference. In this paper, we present the core ideas we used to implement such a prover, from its encoding in Prolog to the generation of the complete set of proofs. However, when dealing with educational aspects, there are many challenges to overcome. We also present the main issues we encountered, as well as the chosen solutions.

### 5.3 Context

#### 5.3.1 The QED-Tutrix software

The QED-Tutrix software [2,3] provides an environment where a high-school student can solve geometry proof problems. One of its key features is that it allows the student to provide proof elements in any order, not limiting them to forward- or backward-chaining. For instance, when solving the simple problem “prove that a quadrilateral with three right angles is a rectangle”, the student can provide any element of any possible proof, such as a direct consequence of the hypotheses (“if two lines are perpendicular to a third, they are parallel”), a necessary premise for the conclusion (“a rectangle is a quadrilateral that has four right angles”), or anything in between (“the quadrilateral  $ABCD$  is a parallelogram”). A second key feature is the tutoring aspect. When the student is stuck in the resolution, the software is able to provide them with relevant messages. In the previous example, if the student entered “the quadrilateral  $ABCD$  is a parallelogram” and is stuck afterwards, the software identifies that they are working on a proof using parallelogram properties, and will provide them messages such as “what is the definition of a parallelogram ?” or “is there a relation between parallelogram and rectangle ?”

These features, the flexibility in exploration and the tutoring, are very interesting from a mathematics education perspective, but come with a cost. Indeed, to allow this behavior, the software must know, first, all the various mathematical elements that can be used at any step of any proof for the problem at hand, and second, how these elements are used in the various possible proofs of the problem. This requires, first, a structure that contains all the possible proofs, called the HPDIC graph<sup>1</sup> [2], and, second, the generation of such a structure for every problem that is offered in QED-Tutrix [4]. However, even quite simple problems can have a huge amount of possible proofs when considering all the slight variations. This makes the manual generation of HPDIC graphs an extremely tedious and time-consuming task. Furthermore, the software is restricted to proof problems, forbidding the implementation of

---

1. A HPDIC graph contains the Hypotheses, Properties, Definitions, Intermediate results, and Conclusion used in any possible proof for the problem.

an interesting aspect of geometry education, the construction problems. In our software, the geometry construction aspect, encapsulated in a dynamic geometry system, is only a tool for the problem resolution.

### 5.3.2 The need for an automated problem solver

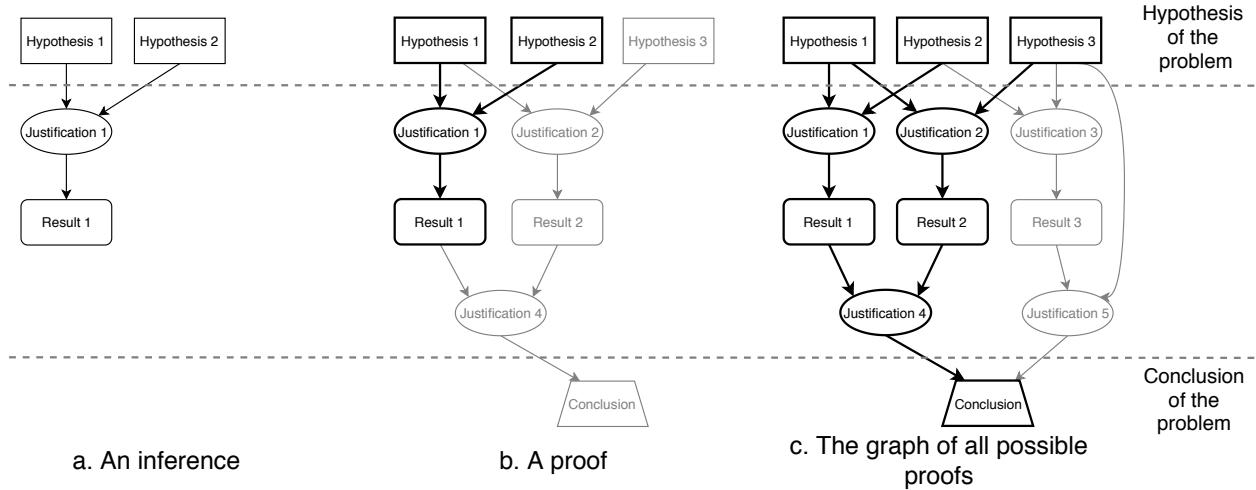


Figure 5.1 The process of constructing the HPDIC graph

To avoid having to create the HPDIC graphs by hand, we implemented a tool that, given a problem and a set of mathematical properties, generates the set of all possible proofs. In theory, a mathematical inference is quite easy to model in a computer, as it is essentially the combination of premises (“ $ABCD$  is a parallelogram” and “ $\widehat{ABC}$  is a right angle”), a property (“a parallelogram with a right angle is a rectangle”), and a result (“ $ABCD$  is a rectangle”), as seen in Figure 5.1a. This structure is extremely similar to the inference mechanism in logic programming, where a program is a set of facts and rules, and where we infer new facts (the result) based on a rule (the property) and existing facts (the premises). Then, because the result of an inference can be used as the premise of another, a proof is simply a chaining of inferences, starting at the hypotheses of the problem, and reaching the conclusion, as seen in Figure 5.1b [72]. Finally, since the mathematical results can be used in several proofs, we can merge the proofs to create a unique structure containing all the possible proofs for a problem, as in Figure 5.1c.

Thus, the core of our problem solver seems to be very simple : we create a Prolog fact for each hypothesis and result, and a Prolog rule for each property. However, as we will see in Section 5.5, this simple approach will not work in practice for most problems, due to technical complications, like combinatorial explosion. Moreover, as shown in Section 5.6,

when confronted with the complexity of human reasoning and behavior in general, and what happens in a classroom in particular, the reality is far more nuanced and difficult to reproduce in a computer.

### 5.3.3 Addition of a problem to QED-Tutrix

Overall, the addition of a new problem to the QED-Tutrix's database is done in a three-step process.

**Creation of the problem :** Quite obviously, the first step is to create the geometry problem, by providing the statement, in the form of a geometrical situation (usually given in a figure, but sometimes in the text as well), the hypotheses, and the desired conclusion. To ensure the functioning of QED-Tutrix, however, one must also ensure that this problem is indeed a **proof** problem that can be solved by using a chain of inferences. Furthermore, since the automated theorem prover is not able to create new geometrical elements such as lines or circles, it is also necessary to provide the **super-figure** of the problem. The super-figure is based on the figure that is provided with the problem's statement, to which we add all the geometrical elements that the student could create during the resolution. Therefore, the process of creating the problems must also include predictions on potential construction of new objects. In the current version of QED-Tutrix, the super-figure is provided to the student when he/she begins the resolution process.

**Translation in Prolog :** Since we chose Prolog for our automated theorem prover, the representation of the problem must be translated into this language. This is achieved by : 1) identifying the geometrical objects that are present in the super-figure ( $AB$  and  $BC$  are lines) ; 2) writing down the hypotheses on these elements ( $AB$  is perpendicular to  $CD$ ) and the conclusion of the problem ( $ABCD$  is a rectangle) ; 3) providing some additional information, such as the angles that the student is allowed to use in his/her demonstration, or alternative names for objects (the line  $AB$  can also be called  $l$ ). More detail on the encoding is provided in Section 5.5.2.

**Generation of the HPDIC graph :** Once the Prolog representation of the problem is provided to the software, along with another Prolog file containing all the available mathematical properties, the HPDIC graph, containing all the possible proofs for the problem, can be generated, using the algorithm detailed in Section 5.5.3. It is first stored locally in a third Prolog file for testing, analysis and debugging purposes, and, second, if authorized, uploaded

directly to QED-Tutrix's problem database. This second part allows for a quick enrichment of the set of problems available in QED-Tutrix.

## 5.4 Related Work

This paper presents the advancement of our work on QED-Tutrix. In one of our previous papers [4], we presented our preliminary results, including a review of research work on tutoring systems and automated theorem proving relevant to this project. Since the basis for our work has not changed, this review is quite similar to the previous one with, however, the addition of some recent work that was not published at the time.

In our work to automatically generate the set of possible proofs to a problem, we have three fundamental constraints that created the need to build our own prover instead of reusing an existing one :

- 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 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 [14, 15], Wu's method [16, 17] and the exact check method [18]. Practical applications include the recent integration of a deduction engine in GeoGebra [19], 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 [73, 74], the full-angle method [59], 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 [61, 62]. However, the geometry taught in

high-school is based on Euclidean'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 [44]. 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. 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. However, to the best of our knowledge, the source is not accessible, and no work has been done on it since 2004. Furthermore, it does not provide the complete set of proofs.

Another very interesting work is the one of Wang and Su [75], as it aims at providing proof for the iGeoTutor, and therefore has the same readability and accessibility at a high school level objectives. In particular, its template-matching algorithm for finding auxiliary constructions is quite promising. However, it requires the use of an external arithmetic engine, and, more importantly, also focuses on finding one proof.

Overall, the very specific needs dictated by the focus on educational interest considerably limit our options. The two only systems with similar goals are GRAMY and iGeoTutor, and they are not suited to our needs. Therefore, we chose to implement our own system.

## 5.5 The problem solver

In this section, we present our method to automatically generate the HPDIC graph that represents all possible solutions to a problem.

### 5.5.1 Available data

**Available properties** One of our development goals is to use technology to assist mathematics education and, more specifically, adapt our software for the different key players within the educational environment (among them, the students, the teachers and the problems' authors). The development of QED-Tutrix is thus an endeavour not only in the field of computer science, but also within the field of mathematics education ; therefore, our team is composed of experts in both domains. Although not detailed in this paper, an important part of this work includes collecting, compiling, and organizing the mathematical elements that are used by the software. To date, we have analyzed 18 Quebec high school textbooks ranging from 7th to 11th grade and extracted 1382 mathematical properties and definitions. This large databank of definitions and properties is necessary to properly adapt the software. We call *referential* [24, 25] the set of properties and definitions that are allowed or expected in

the resolution of a problem. These are known to greatly vary among grades, textbooks, and teachers. Different phrasing might be used from one referential to another even when describing foundational mathematical properties such as *the sum of interior angles in a triangle is 180°*. This variability in phrasing must be accounted for to ensure consistency among the properties and definitions used by QED-Tutrix and to allow teachers to choose their innate preferences to be used by their classes. The final cumulative referential extracted from the various school textbooks includes Euclidean geometry, area and volume formulas, metric relations, transformational geometry, analytic geometry and basic vector geometry. We do not attempt to isolate a minimal set of axioms that are used in high school geometry, but instead aim at implementing enough properties and definitions to cover all the material present in high school geometry textbooks.

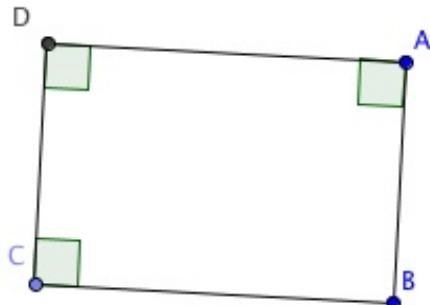
**Available problems** To learn mathematics, one must solve problems [26]. The tasks of finding, adapting and creating new problems plays an important role in a tutoring system such as QED-Tutrix. Currently, there are approximately fifty problems covering a vast array of mathematical topics from high school courses. These problems have been intentionally selected to allow many possible solutions, each one involving several inferences. In our work, problems are also used as a validation method to ensure a proper encoding and the completeness of the implemented referential. Although those first fifty problems can be used to test the referential, we also need simpler problems to test specific elements of the referential. We thus created smaller problems with one or two inferences per property and definition implemented in our software. Since those simpler problems use a small number of properties or definitions that are related to specific small sub-parts of the referential, we can easily detect errors arising in our proof generator. These errors can be due to some coding mistake or to a missing property or definition within the referential.

### 5.5.2 Encoding of a problem

The second step is to translate the mathematical problem into a Prolog file. An example of such a translation is provided in Figure 5.2. The resulting file has several parts :

**Implicit hypotheses** The first eight lines provide names to the geometrical objects present in the problem figure. We refer to these as **implicit hypotheses**, since they are needed for the prover, but are not specified in the problem statement. Furthermore, even though the names of the points are explicit in this case, the Prolog engine needs names for each point, line, etc., even if they are not at all given, neither in the statement nor in the figure. In that

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



(a) Problem statement.

```

hypothesize(point(a)).
hypothesize(point(b)).
hypothesize(point(c)).
hypothesize(point(d)).

hypothesize(line([a,b])).
hypothesize(line([b,c])).
hypothesize(line([c,d])).
hypothesize(line([d,a])).

hypothesize(isAQuad(quad(a,b,c,d))).

hypothesize(angleValue(angle([d],a,[b]),value(90))).
hypothesize(angleValue(angle([b],c,[d]),value(90))).
hypothesize(angleValue(angle([c],d,[a]),value(90))).
```

conclusion(rectangle(quad(a,b,c,d))).  
usefulAngle([a],b,[c]).

(b) Problem encoding.

Figure 5.2 The translation of the rectangle problem from its statement.

case, arbitrary names must be provided (such as P1, P2, etc.). Lines 1 to 4 provide the set of points present in the figure. Then, lines 5 to 8 indicate which point are linked by a line. Here, we have the four lines ( $AB$ ), ( $BC$ ), ( $CD$ ), and ( $AD$ ) (we do not differentiate segments and lines). If we wanted to add the line through A and C, we would add *hypothesize(line([a, c]))*. to the file. The example here is simple enough, but in more complex problems, the encoding of lines can become a delicate issue. Indeed, we must provide, **as soon as the problem is encoded**, all the points that are on the line. This set of points is, as far as Prolog is concerned, the unique identifier for the line. A direct consequence of this implementation is that it becomes impossible to add, during a proof, new points to a line.

**Explicit hypotheses and conclusion** The next four lines provide the hypotheses in a more general sense, meaning the ones that are usually given explicitly in the problem statement, hence the name **explicit hypotheses**. Here, the statement of the problem, "Prove that any quadrilateral with three right angles is a rectangle", with the addition of the figure, provides four hypotheses : there is a quadrilateral named ABCD, and three of its angles,  $\widehat{DAB}$ ,  $\widehat{BCD}$  and  $\widehat{CDA}$ , are right angles. The problem statement also provides the expected conclusion :  $ABCD$  is a rectangle, encoded in the second-to-last line.

**Auxiliary hypotheses** In many problem resolutions, there is, at some step, the need for the construction of additional elements, such as a new line or point. However, the creation of such elements is a difficult issue. Given that our aim in this project is not actually to solve problems in and of itself, but to ease the process of adding new problems to QED-Tutrix, we chose to require the addition of **auxiliary hypotheses**, i.e. elements that are not present on the problem statement directly, but are useful in one or several of its resolutions. For instance, in the rectangle problem, one could write a proof using the diagonals of the rectangle. The fact that  $(AC)$  is a line would therefore enter in this category. Structurally, auxiliary hypotheses are identical to implicit and explicit hypotheses, it is their origin that differs. When adding these auxilliary elements to the figure of the problem, we create the **super-figure**.

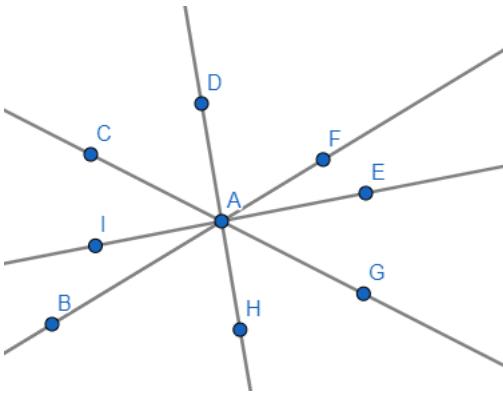


Figure 5.3 A geometrical situation with many angles.

**Additional elements** The last line in our example is neither a hypothesis nor a conclusion, but an additional information provided to the prover. It can be of two types : useful angles and dictionary items. The useful angles are required because in some geometrical situations, such as the one in Figure 5.3, the number of possible angles can become quite huge. Here, there are 8 points around the center A, meaning that there are  $8 \times 7$  angles. When considering inference such as “two angles that share a side are adjacent” and “two angles whose sum measures 180 degrees are supplementary angles”, the number of possible inferences becomes very impractical. For this reason, to limit the combinatorial explosion that plagues many synthetic automated theorem provers, we chose to impose the following requirement : the problem file must contain the list of all angles on which Prolog is allowed to infer results. In other words, if an angle is not explicitly written in the problem file, no result that depends on this angle will be present in the HPDIC graph. Finally, the dictionary allows the user to provide alternative names for geometrical objects. This is useful in problems where lines,

angles, circles, or, more rarely, triangles and quadrilaterals, have alternative names, such as “line  $l$ ” or “rectangle  $R$ ”. This is not needed for the solver, but is important for the tutor software, since the students must be able to enter their sentences with any valid name.

One should note that, as required by the Prolog language, each constant must have its first letter in lowercase, otherwise Prolog would consider it as a variable. This could lead to some collisions if there is a point named  $A$  and a point named  $a$ , but, in our experience, the conventions used in the statement of high school geometry problems prevent that situation. It is however possible, and usual, that two different mathematical objects are named with the same letter, one uppercase and one lowercase, for instance a circle  $c$  and a point  $C$ , but we avoided collisions by giving an explicit type to each object except points. This means that the point  $C$  is named “ $c$ ”, and the circle  $c$  is named “circle( $c$ )” in the Prolog code.

### 5.5.3 Generation of the complete set of proofs

Once the problem is encoded in Prolog, we proceed to the proof generation process. This process follows the algorithm displayed in Figure 5.4. To summarize, we proceed to a construction of the graph by forward chaining. We maintain a set  $ToExplore$ , that contains the mathematical results already known, but from which new results have not been inferred. At the beginning, this set contains only the hypotheses of the problem. We also maintain a knowledge base, that contains statements (hypotheses and inferred results) and the complete inferences. Iteratively, we take one item of the set  $ToExplore$  and we ask Prolog to infer everything that can be inferred by using only inferences that have this item in their premises. Each inference found this way is added to the knowledge base (remember that an inference is a property with its premises and its result), and its result added to the set  $ToExplore$  and to the knowledge base. We loop until there is nothing new to explore, i.e. Prolog inferred everything using the given hypotheses and properties. Since the set of hypothesis, the set of geometrical elements, and the set of available properties are all finite, the set of possible inferences is also finite. Since the algorithm constructs new inferences without any fallback, we can guarantee that this algorithm terminates.

Then, we verify if we were able to infer the conclusion. If not, there is an error. Otherwise, we explore the generated graph in backward chaining, starting from the conclusion, and marking an element on the graph only if it can be used to infer the conclusion. This last step is necessary, because the inference engine can infer results that are valid, but useless for the resolution of the problem, i.e. not used in any proof that reaches the conclusion. These results are marked, but not removed from the graph. This will be useful when the student works on the problem and enters such a result. In this situation, the software should not

**Require:** *problemFile.pl*

```

1: Hypotheses, Conclusion  $\leftarrow$  problemFile.pl
2: ToExplore  $\leftarrow$  Hypotheses
3: KnowledgeBase.statements  $\leftarrow$  Hypotheses
4: KnowledgeBase.inferences  $\leftarrow$   $\emptyset$ 
5: repeat
6:   NewPremise  $\leftarrow$  ToExplore.pop()
7:   NewInferences  $\leftarrow$  INFERUSING(NewPremise, KnowledgeBase)
8:   NewResults = {Inference.result | Inference  $\in$  NewInferences}
9:   ToExplore  $\leftarrow$  ToExplore  $\cup$  NewResults
10:  KnowledgeBase.inferences.ADD(NewInferences)
11:  KnowledgeBase.statements.ADD(NewResults)
12: until IsEmpty(ToExplore)
13: if Conclusion  $\in$  KnowledgeBase.statements then
14:   Graph  $\leftarrow$  CONSTRUCTGRAPH(Conclusion, KnowledgeBase.inferences)
15:   return Graph
16: else
17:   return Error
18: end if

```

```

1: function INFERUSING(Statement, KB)
2:   Inferences  $\leftarrow$   $\emptyset$ 
3:   CandidateInferences  $\leftarrow$  {I | I  $\in$  KB.inferences & Statement  $\in$  I.premises}
4:   for all Inf  $\in$  CandidateInferences do
5:     if KB.PROVE(Inf) then
6:       Inferences = Inferences  $\cup$  {Inf}
7:     end if
8:   end for
9:   return Inferences
10: end function

```

Figure 5.4 The graph construction algorithm.

say “This result is false”, but instead “This result is valid, but not useful for the problem. Try something else.” For this behavior to be possible, the “valid but useless” elements must remain in the graph and be identified as such.

Since the generated graph is based on mathematical properties, which typically have a reciprocal, there are usually many cycles in the resulting graph. For example, if  $ABC$  is a right triangle in  $A$ , then we can infer that  $\widehat{BAC}$  is a right angle. And since  $\widehat{BAC}$  is a right angle, we can infer that  $ABC$  is a right triangle in  $A$ . This may be a problem for the calculation of the students’ progress in the proof resolution, but it is solved by a clever exploration of the graph. This algorithm, however, is not part of the proof generation process, and is not detailed in this paper.

In this algorithm, only the *inferUsing* function uses Prolog. The rest of the algorithm is handled by a Python program combined with the QED-Tutrix central database. This database is the core of all our operations, since it contains all the data provided by the input Prolog files (the one that provides the representation of the problem and the one that contains all the mathematical properties), and all the inferences that have been added and which constitute the generated HPDIC graph.

#### 5.5.4 Generation of mathematical results using logic programming

In the previous algorithm, we abstracted the use of Prolog by a *inferUsing* function, that takes as arguments the knowledge base, representing the set of all results and inferences known by the Prolog engine, and a premise, i.e. a mathematical result that must be used as the premise of the returned set of new inferences. In other words, if the given premise parameter is “ $ABC$  is a right triangle in  $A$ ”, then all the inferences returned will have this statement as a premise. For instance, the inference “ $ABC$  is a right triangle in  $A$ , therefore, since a right triangle has a right angle, the angle  $\widehat{BAC}$  is right” is a valid candidate, but “The angle  $\widehat{BAC}$  is right and  $ABC$  is a triangle, therefore, since a triangle that has a right angle is right,  $ABC$  is a right triangle” is not, since the given statement is not a premise of the inference, but its result. This requirement is translated into Prolog in three parts.

**Translation of mathematical properties in Prolog rules** Each property identified during the work presented in Section 5.5.1 is encoded into Prolog rules. In Figure 5.5, we translated the property “a triangle that has a right angle is right”. Let’s suppose, for example, that at some point in the solving process, the Prolog knowledge base contains a relation of perpendicularity between two lines  $(AB)$  and  $(AC)$ , and that there exists a triangle  $ABC$  with two sides on these lines, encoded by :

```

newInference(rightTriangle(triangle(A,B,C)),
    'droitTrect',
    [perp(line(L1),line(L2))]) :-  

triangle(A,B,C),
line(L1),
line(L2),
in(A,L1),
in(B,L1),
in(A,L2),
in(C,L2),
perp(line(L1),line(L2)),
B < C.

```

Figure 5.5 Translation in Prolog of a property on right triangles

```

perp(line([a,b]),line([a,c])).  

triangle(a,b,c).

```

In this situation, this rule can be invoked to create the new fact :

```

newInference(rightTriangle(triangle(a,b,c),'rightTrPerp',
    [perp(line([a,b]),line([a,c]))]).

```

This contains the whole inference : the premise “( $AB$ ) perpendicular to ( $AC$ )” ; the property “rightTrPerp”, which is an encoding for the sentence “A triangle that has a right angle is right” ; and the result “the triangle  $ABC$  is right”. In other words, the knowledge base of Prolog is actually not constituted of simple statements such as “ $\text{perp}(\text{line}([a,b]),\text{line}([a,c]))$ ”, but of inferences that also contain the **origin** of the statement, i.e. the property and premises that were used to infer it. In Figure 5.6, we highlighted the part of an HPDIC graph that is encoded by one of these Prolog facts.

**Relation between inference and statement** Since statements such as “( $AB$ ) perpendicular to ( $AC$ )” are not encoded as such in the knowledge base, Prolog must have a way to list the available results. Indeed, in Figure 5.5, the second-to-last line checks if the fact  $\text{perp}(\text{line}([a,b]),\text{line}([a,c]))$  is true, but this fact is not known directly, only that there exists a  $\text{newInference}(\text{perp}(\text{line}([a,b]),\text{line}([a,c])), \_, \_)$ . This linking is done by another rule, displayed in Figure 5.7. The translation of a  $\text{newInference}$  fact into an inference fact is done by the Python wrapper (line 10 in the algorithm). The distinction between the two is crucial to avoid infinite loops in the Prolog inference engine. Furthermore, as seen in the figure, the

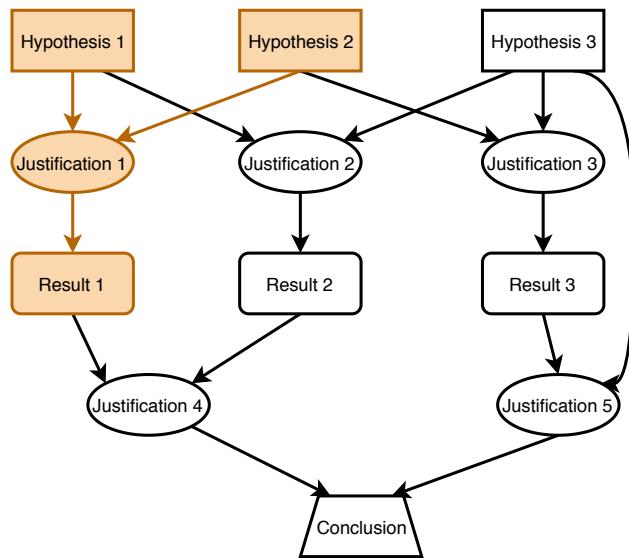


Figure 5.6 The part of a HPDIC graph encoded by one newInference fact

```

perp(L1,L2) :-
    inference(perp(L1,L2),_,_);
    inference(perp(L2,L1),_,_).
  
```

Figure 5.7 Link between a statement and an inference

usage of another rule to obtain the result from the inference allows more flexibility in the order of the elements. Basically, the fact “ $(AB)$  and  $(AC)$  are perpendicular” can stem from an inference whose result is “ $(AC)$  and  $(AB)$  are perpendicular”. Both results are obviously equivalent for a human, but not for Prolog.

**Finding an inference using a given premise** Lastly, in the algorithm displayed in Section 5.5.3, we restrict the inferences to those that use a given premise. This is done for performance reasons. In some moderately difficult problems, the HPDIC graph can become quite big, making its exploration time consuming. In a previous version of the prover [4], we used all the available inferences, stored their results, then started again, until no new result is found. This algorithm had the major drawback of duplicating inferences : at step N, the inferences found at step N-1 are still valid, and are therefore found again, increasing the complexity of the algorithm to quadratic on the number of nodes in the final graph. Using the new algorithm makes the complexity linear.

To achieve this result, we create a Prolog fact for each premise of each mathematical property, linking the two. For instance :

```
usedIn(perp(_,_),rightTrPerp).
```

indicates that the statement “perp”, regardless of its argument, is a premise of the rule shown in Figure 5.5. Then, the function inferUsing(NewPremise, KnowledgeBase) used in the algorithm of Figure 5.4 consists in calling Prolog with the query :

```
usedIn(Premise,J),
newInference(R,J,P).
```

that returns in R the result of the inference, J the code of the justification, and P the array of premises, which contains the premise given in argument.

## 5.6 Educational challenges

### 5.6.1 Translation of high school properties into a computable format

In mathematics education, we define different geometric mindsets as the different paradigms GI, GII and GIII [76]. Our work is a constant back and forth between the latter two (GI concerns tangible problems of geometry, such as measurements, folding or the use of blocks, which are not the focus of our tutoring system). The natural axiomatic geometry (GII) is

best described by the iconic example of the Euclidean geometry. It is a geometry where the reasoning depends on a figure. The component of the inferences are, however, part of the linguistic register, such as the example in Section 5.5.1. A referential in the linguistic register means the statement has semantic and syntactic aspects which can easily be understood by a person. The challenge comes with the translation of these statements into the formal axiomatic geometry (GIII) of Prolog where the semantics disappears.

In one of our previous papers [4], we documented several challenges that arose during the development of the solver. One of the most difficult stems from the fact that the computer must have complete and exact information to infer new results. For instance, when dealing with properties on adjacent or complementary angles, it is crucial to identify exactly each angle. Where most teachers would accept both  $\widehat{ABC}$  and  $\widehat{CBA}$ , interchangeably, in a demonstration, this ambiguity is unacceptable in an automated tool and, if not handled properly, leads to an exploding number of nonsensical results. Therefore, a distinction appears between the process of the solver and what happens in the tutor software. In the solver, the angles are all oriented, and the order of the points is crucial. In QED-Tutrix, however, it depends on the teacher's preferences, but the possibility exists for the teacher to allow both notations. If it is allowed and two results exist on the two opposite angles, for instance  $\widehat{ABC} = 120$  and  $\widehat{CBA} = 240$ , then all four propositions, including  $\widehat{ABC} = 240$  and  $\widehat{CBA} = 120$ , would be accepted. In this kind of problem, though, it is advisable to disallow this behavior, since it uses angles larger than 180 degrees and can lead to some confusion.

This example also touches on another delicate issue, the naming of objects. Here, we mentioned that a teacher might want to allow the students to refer to the angle as  $\widehat{ABC}$  or  $\widehat{CBA}$ . However, if there is a point  $D$  on the line  $(AB)$ , then the angle can also be named  $\widehat{DBC}$  (and  $\widehat{CBD}$  if allowed). If there is another point  $E$  on the line  $(BC)$ , the number of possible names doubles, with the addition of  $\widehat{DBE}$  and  $\widehat{ABE}$ , and their opposites. All these must be accepted in the tutor software when the student enters a result on the angle. The same is true for lines, since a line containing the three points  $A$ ,  $B$  and  $D$  can be referred to as  $(AB)$ ,  $(BD)$ ,  $(AD)$ , etc.

This creates two issues in the implementation. First, Prolog facts representing geometrical objects, such as lines and angles, must contain all their constituting points. For instance, the angle of the previous example is represented in Prolog by :

```
angle([a,d],b,[c,e]).
```

where the two lists between brackets contain the points on the two half-lines constituting the angle, and the second argument,  $b$ , is the center point. This way, this simple fact contains all

the information to find the valid names for this angle. In the same way, the line containing  $A, B$  and  $D$  is represented by :

```
line([a,b,d]).
```

listing all the points on the line.

Second, since objects can have many names, we must ensure that their representation is unique. This is especially apparent when inferring new results. For instance, if we infer that  $(AB)$  is perpendicular to  $(BC)$ , we only want to infer one Prolog fact representing  $(AB) \perp (BC)$ , and not  $(BC) \perp (AB)$ . This required the creation of some rules ensuring the uniqueness of the encoding of a result. In this example, we want the two lines to be alphabetically sorted.

In Section 5.5.4, we explained that each mathematical property is encoded into several Prolog rules. One of them, such as the one in Figure 5.5, is the main one, and contains the restrictions on when that property can be used. Ideally, since a mathematical inference is very close to a Prolog inference, there should be only one such rule per property. However, in reality, it can become necessary to implement several rules for one property. For instance, there is only one property for Pythagoras' theorem : “In a right-angled triangle, the square of the hypotenuse side is equal to the sum of squares of the other two sides”. There are basically two use cases here. Either we know the value of the two sides and want to calculate the value of the hypotenuse, or we know the value of the hypotenuse and one side and want to calculate the value of the other. This distinction requires the implementation of two Prolog rules. Another example is in the property “if two of the remarkable lines (bisector, perpendicular bisector, median or altitude) of a triangle are identical, then the triangle is isosceles”. Here, any combination of two remarkable lines is valid, for a total of six possible cases, and, therefore, six Prolog rules.

In these two examples, the need for several implementations is more of a technical need, but in other cases, it happens because the mathematical property is complex and provides many ways to use it. For instance, let us consider the property “the perpendicular bisectors of the three sides of a triangle intersect in a point that is the center of the circumscribed circle”. There are many ways to use this property : to infer that the three lines intersect at one point ; that this point is equidistant to each vertex of the triangle ; that the circle whose center is the intersection of the perpendicular bisector and passes through a vertex also passes through the other two. Even though it can be argued that this difficulty lies in the mathematical statement itself, it is a statement that is used in class and, therefore, must be present in QED-Tutrix. Then, when encoding this property in Prolog, we need to consider

all the possible use cases and create (at least) one rule for each. In the final result, each inference that uses this property will have the same justification.

### 5.6.2 Differences between a human's and a computer's reasoning

As mentioned in 5.6.1, the computer is very good with formal proofs, in contrast to students who are better at making mathematical arguments. When solving a geometric problem, students will typically first explore the figure and then reason on it. For example, they will see a triangle, whether or not it is explicated in the problem statement, and will immediately think about what they know about triangles. With the use of informal properties, that we can somehow formalize by a statement such as *I can see it on the figure*, they might determine that the triangle is isosceles. From there the students may start writing their proof. After writing down the aspect related to the isosceles triangle, they might just come back to observe the figure to keep building their reasoning. At some point, they might also use tools (ruler, compass, dynamic geometry software, etc.) to help with this process [1]. Overall, this whole process is extremely different from what a computer would do. The algorithm presented in Section 5.5 attempts to generate proofs similar to what a student would write, but the process to obtain them is very different. This distinction creates its share of issues.

The main one, and the only one we present in this paper, is the use of different levels of granularity. Indeed, in the mathematics education process, the properties available in the students' referential changes depending on many factors, such as the year of the class, the chapter currently studied, or even the teachers' personal preferences. For instance, when proving a property or theorem, such as "the sum of the angles in a triangle equals 180 degrees", students are obviously not authorized to use that property directly, otherwise the proof is trivial. However, after seeing this proof in the classroom, the property can be used to solve exercises. In other words, when considering the HPDIC graph for this problem ("prove that the sum of the angles in a triangle equals 180 degrees"), the resulting property can be seen as a shortcut from the problem hypotheses to the conclusion. And because both the low-level properties used in the complete proof and the high-level, direct property can be useful in class, QED-Tutrix must be able to handle both. Therefore, we identified three ways to implement this duality. The first one is to implement only a basic set of axioms, forbidding the use of such shortcuts, and to scan the generated graph afterwards to add the shortcuts. The second one is to implement only "advanced" properties, and, when they appear in the graph, to add the subgraph of their proof to the final graph. Finally, the third way is to implement both, and to generate both paths in the resolution. Even though the handling of shortcuts is still a work in progress at the moment, we chose, for the time being, the third

option, to implement both high-level and low-level properties. Further work is needed on both the mathematics education and computer science aspects to determine which solution is the best for the final version.

## 5.7 Limitations

Despite the encountered educational challenges, logic programming is indeed quite adapted to generate the proofs of high school geometry problems. However, it is a lengthy task, and, in its current state, our solver still has several limitations whose correction would require work of varying magnitudes : possible in the current implementation ; possible but requiring fundamental changes in the implementation ; intrinsically difficult or impossible.

**Possible in the current implementation** Currently, the solver does not handle algebra. For instance, it is impossible to represent the result “the length of segment AB is three times the length of the segment CD” without knowing either values. This would require the implementation in Prolog of basic arithmetic operations, but it would be possible, since the use of such results remain in the domain of inferences. Furthermore, we do not handle proofs by contradiction, but, similarly, these kinds of proofs still follow an inferential format. The difference is that the hypotheses represent an impossible situation, and the conclusion is “we reach a contradiction”. This would require to implement all the numerous possible inferences that result in a contradiction, such as having a triangle with two parallel sides, but it would fit in our solving process. Those two limitations cause another problem concerning an axiomatic referential. Indeed, Euclid’s axioms generate most of high school’s properties, but most of his work is done by proofs by contradiction which is not currently feasible by our system. On the other hand, Clairaut’s adaptation of Euclid’s work also generates this geometry and without using proof by contradiction. However, Clairaut uses an algebra system to bypass this difficulty, something that, again, our system cannot do. For this reason, our system requires a much more extensive referential that includes those properties that we cannot generate to be able to solve the problems. This limitation adds another layer of complexity concerning the challenge of the level of granularity discussed in [4].

**Possible with some fundamental change** One of the biggest limitations of our solver is the necessity to provide, beforehand, the whole geometrical situation (the super-figure), including the set of elements that could be useful in a proof. This is very difficult to solve, as the construction of new elements in automated theorem proving in geometry is a whole research domain in itself. However, we envisioned a possible solution. Since QED-Tutrix is

an online tool, and that the generation of the possible proofs can usually be done in a matter of seconds, it would be possible to wait for the student to construct a new element in the interface. Then, when constructed, QED-Tutrix sends it to the prover, that attempts to infer new results from this new hypothesis. If we can reach the conclusion, then the hypothesis is indeed useful, and is added, in real time, to the HPDIC graph of the problem. This solution, however, would require a complete change of the interactions between QED-Tutrix and the solver, and comes with its share of issues. A direct, less profound consequence of this limitation on the possible proofs is that it is impossible to discover, in the middle of a proof, that a line actually passes through a point. This is because it is necessary to provide the complete set of points the line passes through from the beginning when encoding the problem. This limitation could be solved, but it would require a drastic change to the way we represent elements in Prolog.

**Intrinsically difficult** A crucial issue comes from the floating-point precision of calculations in the machine. For example, let us assume that there is a right triangle  $ABC$ , whose dimensions we know. From these dimensions, we can calculate the cosine, sinus and tangent of the angles. This calculation is done with a certain precision. This creates a first problem : what level of imprecision from the student is acceptable ? If the cosine of the angle has a value of 0.7654321, is 0.8 acceptable ? Is 0.77 ? Is 0.76 ? Still, while delicate, this question concerns the tutor aspect, and not the automated proving, and we will not discuss it further. However, there is another issue concerning approximations. In the previous example, now that we know the cosine of the angle, by using the reciprocal of the property, we can calculate the value of the sides of the triangle. This calculation is also done with a certain precision, and if the resulting value is even imperceptibly different from the already known one, then it is considered a new result, that will lead to the calculation of a slightly different cosine, and so on. Hopefully, we were able to solve this issue by only considering the first result. When we already know the value of something, every inference that results in another measurement of that value does not do any calculation, but uses the already known one. In this example, if we already know that the hypotenuse of the triangle is of length 5, then any inference who results in the measurement of that line segment will not even calculate it, but reuse the 5 that has been calculated previously. Lastly, the precision can also create problems when using an inference such as “two angles are equal if they have the same measure”. Just checking the equality of their measures is not enough, since they could be very slightly different due to rounding. In that case, we allow for a difference of 1% between the two values. We chose that precision by trial-and-error, with the argument that, in high school geometry problems, two different values that are supposed to be measured will differ by more than 1%. This is not a

mathematically ideal solution, but for our purposes, it is enough.

## 5.8 Other avenues and conclusion

The future work will consist of several steps. In the short term, we will be completing the implementation of all the identified high school properties in Prolog, and at the same time, add problems to QED-Tutrix as soon as all the useful properties for its resolution are available. Then, one of our major goals is to implement the teacher interface, where they will be able to create new problems, indicate referentials, both for the problem and in general, generate proofs, and obtain statistics of the use of QED-Tutrix by their students (which properties they use, in what order, where they are stuck, etc.). In parallel, we are compiling inferential shortcuts used in class to enrich the possible referentials. Lastly, we are also working on a tool to automate as much of the problem encoding as possible, with, ideally, a software that translates a problem statement in natural language (with, possibly, a figure) into a Prolog file.

Overall, while much work remains to be done, and some limitations of varying degrees exist to our theorem prover, we demonstrated that logic programming is indeed an adequate tool for automating the generation of proofs at a high school level. Furthermore, logic programming offers a high flexibility that allows us to handle the educational challenges stemming from the fundamental differences between the reasoning of a human and the automated theorem proving process. Although challenges remain, the current version is already sufficient to handle an interesting number of high school problems, allowing us to use that prover to create problems for the current version of QED-Tutrix.

**CHAPITRE 6 ARTICLE 3 : INTELLIGENCE IN QED-TUTRIX :  
BALANCING THE INTERACTIONS BETWEEN THE NATURAL  
INTELLIGENCE OF THE USER AND THE ARTIFICIAL INTELLIGENCE  
OF THE TUTOR SOFTWARE**

**Ludovic Font**

École Polytechnique de Montréal  
Montréal, QC, Canada  
ludovic.font@polymtl.ca

**Nicolas Leduc**

École Polytechnique de Montréal  
Montréal, QC, Canada  
nicolas.leduc@polymtl.ca

**Michel Gagnon**

École Polytechnique de Montréal  
Montréal, QC, Canada  
michel.gagnon@polymtl.ca

**Philippe R. Richard**

Université de Montréal  
Montréal, QC, Canada  
philippe.r.richard@umontreal.ca

**Reference :** L. Font *et al.*, “Intelligence in QED-Tutrix : Balancing the Interactions between the Natural Intelligence of the User and the Artificial Intelligence of the Tutor Software,” *Mathematics Education in the Digital Era*, 2021

## 6.1 Mise en contexte

Cet article est, à l’heure de l’écriture de cette thèse, en cours de publication dans l’édition spéciale *Mathematics Education in the Digital Era*, publiée par Springer. La publication est, à l’heure d’écriture de cette thèse, imminente, et l’article a été relu et accepté par le comité en charge. Il s’agit donc de la publication la plus récente et aboutie de ce projet, et a pour but de présenter de façon auto-suffisante les travaux effectués dans cette thèse. Sa lecture au complet est donc pertinente. Cependant, une attention particulière peut être portée sur la Section 6.4, présentant en détail le fonctionnement de QED-Tutrix. Bien que cet aspect du logiciel n’ait pas été développé dans le cadre des travaux présentés dans ce mémoire, les structures complexes utilisées à l’interne dans QED-Tutrix ont guidé le développement de notre générateur de preuves, et cette section offre donc une perspective intéressante. La Section 6.5 présente à nouveau le fonctionnement du générateur de preuves, et, plus loin, ses limitations. Bien que plus aboutie que dans les articles précédents, une lecture rapide est recommandée. En revanche, la partie 6.5.6 est d’un intérêt tout particulier, car elle présente notre processus de validation, crucial dans notre méthodologie de recherche.

## 6.2 Abstract

In order to explore the complexity stemming from the interactions between technology, and in particular computer science, and geometry education, we developed the QED-Tutrix intelligent tutor software, as well as an automated proof generator. QED-Tutrix combines an interface allowing the students to solve high school geometry problems, developed with finely honed mathematics education theories in mind, and an automated tutor able to guide the students in any possible situation in their resolutions. These strengths required the development of specific, highly specialized layers, for representing a set the solutions for a problem, approximating the student's cognitive state, or providing custom-tailored tutoring messages. In a more ancillary role, the proof generator aims at automatically anticipating all the possible ways a high school student could solve a geometry problem, a task combining both computer science issues, that we attempted to answer by exploiting the strength of logic programming, and education issues, since the reasoning simulated by an automated system could not hope to match the finesse and adaptability of a human mind.

## 6.3 Context

When considering the mathematics competences referential [77], whether from Québec Education Program<sup>1</sup>, the Standards of the National Council of Teachers of Mathematics<sup>2</sup> or some large international studies such as the Programme for International Student Assessment<sup>3</sup>, reasoning and mathematical proofs are always at the forefront. Although the notion of proof is limited, in elementary school, to mathematical reasoning and conviction, and in post-secondary education, to demonstration and written communication, the evolution of the treatment reserved for deductive reasoning in secondary school has the appearance, depending on the region, of an eternally shilly-shallying between valorisation and marginalisation. This status quo creates a major source of epistemic injustice [78], if only for mathematical conceptualisation or the development of high-level competences. Therefore, it is a major educational issue at the heart of everything from teaching to learning, including the training of mathematics teachers.

It is already recognised that the field of reasoning and proof is an authentic vector of training during compulsory education [79]. In our digital age, the understanding of the field intertwines as never before didactic, mathematical and computer issues. However, this domain is often the most resilient to technological change, as if the mathematical sciences could only prove

---

1. <http://www.education.gouv.qc.ca/enseignants/pfeq/>  
 2. <https://www.nctm.org/Standards-and-Positions/Principles-and-Standards/>  
 3. <https://www.oecd-ilibrary.org/docserver/b25efab8-en.pdf>

with writing. Without denying the crucial role of discourse in mathematical work, we retain, first, an unavoidable strain between traditional and technological mathematics, the latter still fuzzily defined but flourishing under our eyes, and, second, the lack of an epistemological reference, precise and verifiable in the long run, apt to inspire education. If it is possible to link technological tools to discourse with the notion of instrumental proof – notion developed in mathematics education on epistemological bases [1]–, our argument is based on the idea of interaction between a human and a machine.

Since Rabardel's work [71] on cognitive ergonomics and the cross-look offered in mathematics didactics [80], the notion of instrument has become detached from its usual meaning of a man-made object, used to perform certain operations. The modes of use constructed by a user were then attached to the technical object, thus underlining the idea of a mixed entity consisting of the interaction between an artefact and a subject. What changes is that beyond the direct interaction between a subject and an object, i.e. anything perceived by the senses, new interactions are integrated, such as the interactions between the subject and the instrument, the interactions between the instrument and the object on which it allows acting, as well as the subject-object interactions mediated by the instrument. The instrument then appears as an emergent system which is viewed from different angles, associating with the instrumental genesis two interdependent processes which remain the fact of the subject, i.e. instrumentation and instrumentalisation [71]. We come to distinguish between those aspects of the process of instrumental genesis which are directed towards the subject itself (instrumentation) and those which are directed towards the artefact (instrumentalization). The intelligence of the system is therefore a shared intelligence where one can see above all the machine intelligence, the human intelligence or the intelligence of an evolving system, always looking at the same interactions of mathematical activity under three complementary glances.

In order to explore in depth these complex issues, we created the QED-Tutrix project. At its core, it is a software that, first, provides a platform to experiment on the benefits of using technology to teach geometry, and, second, allows the precise analysis of the interactions between student and subject, since those interactions are all done in a digital space where every action can be logged, stored and automatically included in broad statistics. The aim of this chapter is to provide a standalone report on the goals, stakes (both in computer science and mathematics education aspect), and internal workings of the QED-Tutrix software. This first section explains the context surrounding our project. The second section provides detailed information about the software itself, in particular the various original structures we developed to represent complex situations in a computable form. Finally, the third section contains a report on the ancillary system we created to address the crucial issue of automated proof generation. Most of the content of the latter is similar to one of our previous paper [5],

with the notable addition of the validation process.

### **6.3.1 Symbiosis between the Mathematical Work in Schools and Computer Science**

One of the prominent theories of mathematics education is the one of mathematical working space [24]. The theory of mathematical working spaces facilitates the specific study of mathematical work in schooling, both in terms of learning and implementation. Mathematical work is, in a way, the visible part of mathematical thought and it is constructed progressively as a process of bringing together epistemological and cognitive aspects, in accordance with three genetic developments that are jointly engaged, identified in the theory as the semiotic, instrumental and discursive geneses. With respect to our system, this representation separates the epistemological plane, containing the “absolute” theoretical mathematical knowledge, and the cognitive plane, containing the current knowledge of a student at a given time. The interaction of knowledge between these planes is done on three geneses : the semiotic genesis, representing the mathematical concepts and symbols and their meaning ; the discursive genesis, representing mathematical organisation in a structured form, whether written or oral ; and the instrumental genesis, representing the use of material or symbolic artefacts to manipulate or transform concepts, for example, with the help of a ruler, compass and protocol, or, more relevant to our case, tutorial software and all the tools it engages. The knowledge constructed by a subject (student, teacher, trainer) emerges from this interaction and can, under certain conditions, constitute autonomous instrumented knowledge. This model is summarized in Figure 6.1.

In the light of this representation, we can specify the role of the QED-Tutrix software in supporting mathematical work. Thus, for the learner, it favours the creation of knowledge and the development of mathematical competencies ; for the teacher, it allows him/her to organise the reference knowledge according to the usual mathematical work in his/her lessons ; for the teacher, it makes possible the simulation of mathematical work in a concrete space of necessity, exploring the valence of mathematical work for absent subjects (the learners). This objective has been one of the guiding principles during the whole development process. More generally, QED-Tutrix is the result of a close collaboration between experts in computer science and mathematics education, following the principle of design in use. As a result, all its core functionalities have been validated, implicitly or explicitly, as being relevant for improving the experience of its users, both student, teacher and trainer. We present these functionalities in Section 6.4.

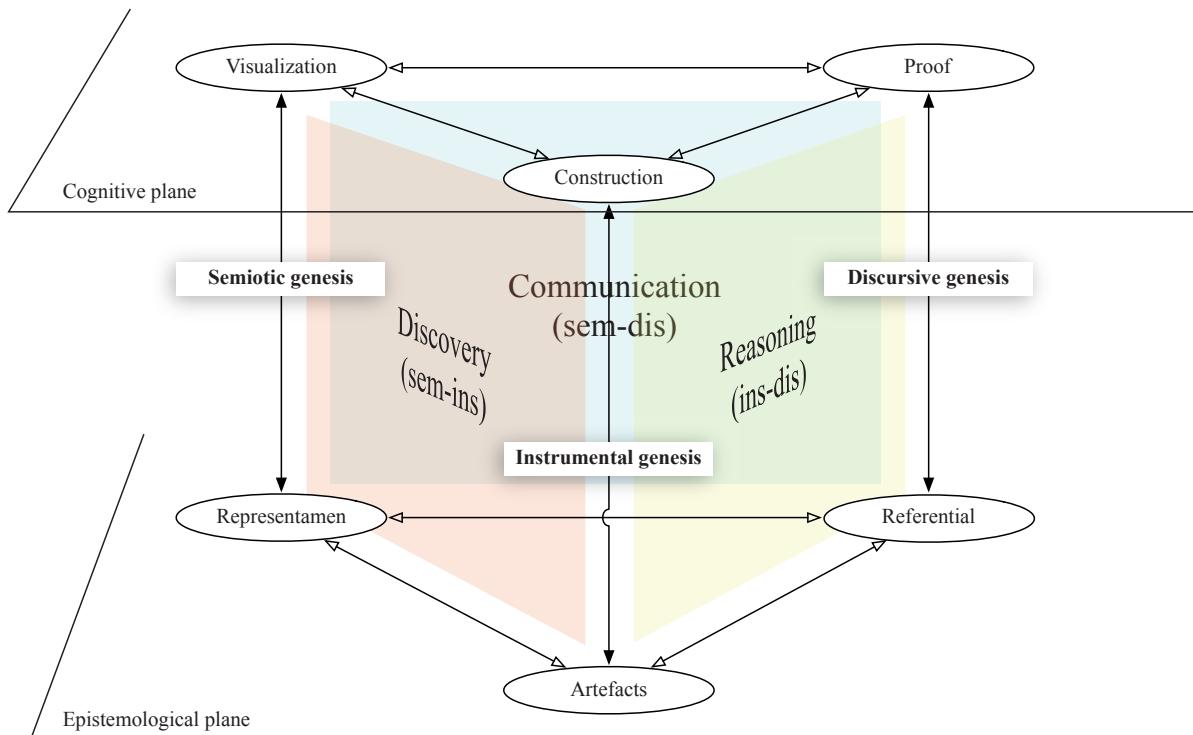


Figure 6.1 The main components of the mathematical working space. The vertical planes highlight the coordination of the dominant geneses in a specific mathematical work, for example, to signify certain mathematical competencies at stake or types of instrumental proofs [1].

### 6.3.2 Existing Tutor Softwares

In [2], 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 [8] or private tutoring companies. These are 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 [31] for learning LISP, and ALEKS [32], ActiveMath [9] and Wayang Outpost [33] for mathematics specifically. These require a modelization of the user's knowledge, that can be done using various techniques, such as fuzzy logic [34] or Knowledge Components [35]. 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 Euclid's axioms. A pioneer in this domain is the Logo framework, developed in 1980 [81], but a typical modern example is GeoGebra [10], 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 verifying statements, or discover new facts. These systems are discussed in detail in Section 6.5.1.

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 of QED-Tutrix is to offer :

- an interface to explore the problem by allowing the student to freely manipulate the figure ;
- freedom in the construction of the proof, allowing the student to construct his proof in any order ;
- handling of all possible synthetic proofs, acceptable at a high-school level, i.e., excluding coordinates or complex numbers-based ones ;
- 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 systems developed for geometry is GeometryTutor. It is based on a solid theoretical model, the ACT-R cognitive theory [36, 37]. 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 [38, 39], that evolved into Geometry Cognitive Tutor [40–42]. This system is limited to problems based in Cartesian coordinates, since it handles elements numerically. In QED-Tutrix, 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 [44]. 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 [45, 46], unlike the previous ones, aims at helping the student to construct his proof. It is based on the Diagram Configurations [47] 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 high school. Besides, even though the student can explore the problem, there is no interface to manipulate the figure.

An interesting approach is the Baghera system [48, 49], 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, who can choose 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 [50, 51], Cabri-DEFI [52] and Cabri-Euclide [53, 54]. 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 [11–13] 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 foundations of QED-Tutrix. 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 [55], 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 [56, 57], that largely inspired QED-Tutrix, 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 a 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 QED-Tutrix project.

## 6.4 Genesis of the QED-Tutrix Project

This section focuses on providing an overview of the QED-Tutrix software itself, both in its goals in terms of mathematics education, its user interface and its internal working.

### 6.4.1 Task description

As we mentioned previously, QED-Tutrix has several well-established goals :

- providing an interface to solve geometry problems by constructing the proof in a way similar to what the student would do in class ;
- accounting for the three geneses of mathematical work (semiotic, discursive and instrumental) ;
- offering a tutoring system to help the student in the problem resolution.

These high-level, conceptual goals directly create low-level constraints in the very conception of the software itself. Indeed, the first goal means that the software must be able to assess, on the fly, the validity of the mathematical proposition the student is entering, both in its form (“The line  $AB$  is parallel to point  $B$ ” is formally invalid) and in its relevance for solving the problem at hand. For instance, if the student begins the proof by providing a theorem that is needed to obtain the conclusion, and is, therefore, at the very end of the proof, the software must be able to assess that this theorem is a relevant element, despite being the first element provided. In turn, this means that the software must know, in advance, the whole set of proofs that a student could provide. Finally, this creates the need for, first, a structure to represent the set of all possible proofs to a problem, and, second, a way to generate such a structure. The definition of such a formal structure formalizes the representation of a proof, and, therefore, ensures that there is not an infinite number of possible proofs. We provide more detail on this structure in Section 6.4.3.

The second goal means that the interface of QED-Tutrix must allow the student to explore the problem, provide mathematical elements (such as results or properties) that are relevant to the proof, and finally organize these elements in a valid, formal proof. These three aspects follow the processes defined by Coutat & Richard [82] of discovery, validation and modelization.

The third goal means that the software must be able to determine the proof the student is working on in real time. Indeed, even simple problems can have up to several millions of possible proofs because of the combinatorial explosion when there are several possible variations at every step. This required the creation of several structures to store and manipulate the progression of the student on his or her proof. Furthermore, the messages sent by the tutor to help the student had to be carefully crafted, as well as a way to choose which one to send in every possible situation.

### 6.4.2 Software Overview

We now present the interface of the current version of QED-Tutrix. The main window is composed of four core elements : the GeoGebra interface (top-left), the sentence selection menu (center), the redaction section (top-right), and the tutor chatbox (bottom). A view of the interface is provided in Figure 6.2.

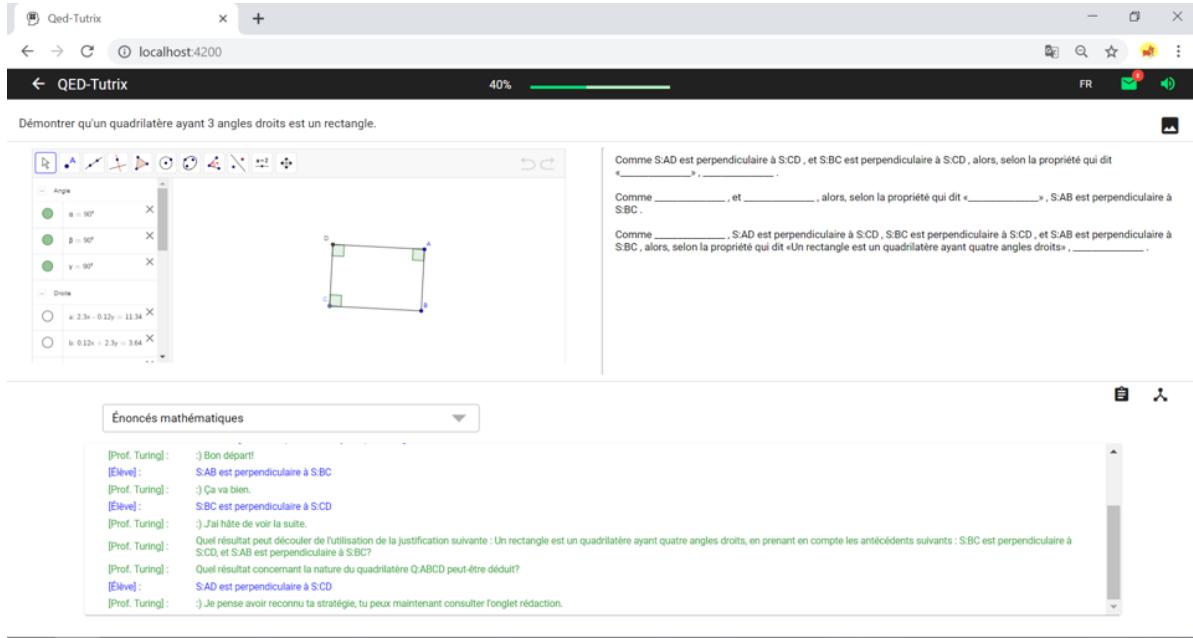


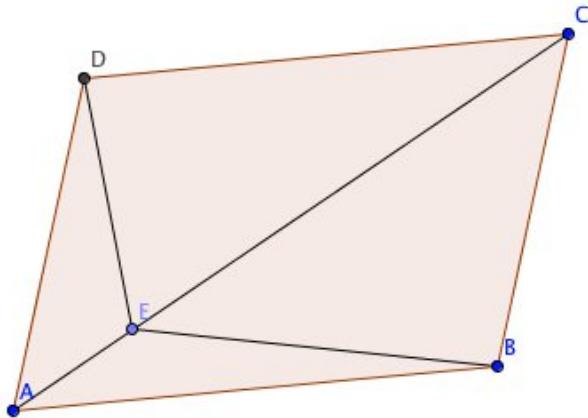
Figure 6.2 QED-Tutrix's main interface

### GeoGebra Interface

The figure section is present on the interface at all times. It displays GeoGebra's [10] main window which was incorporated into QED-Tutrix to allow the student to explore the dynamic figure linked to the problem. An important point is that there is a limited communication between the GeoGebra interface and the software internal structures. If the student creates a new element (such as a new line or circle), then this element will not be useable to solve the problem, since it has not been integrated in the set of possible proofs. As a consequence, we must ensure that the GeoGebra figure contains each and every geometrical element that can appear in any proof of the problem that we want the software to be able to handle. We call such a figure the super-figure.

To provide a more detailed example, let us consider the parallelogram problem, as stated in Figure 6.3. A possible proof uses the heights of triangles  $ACD$  and  $ABC$  through  $D$  and  $B$

respectively, another uses the lines parallel to  $AB$  through  $E$  and parallel to  $AD$  through  $E$ , and another uses the diagonal of  $ABCD$  through  $B$  and  $D$ . Therefore, all these new elements, both lines and points, must have a name, so that the student is able to refer to them in his proof, and the names must correspond between the displayed GeoGebra figure and the internal representation of the software. As a result, these elements are created beforehand and provided to the student in the GeoGebra plugin, forming the super-figure of the problem, as shown in Figure 6.4. If, at some point, a student thinks of another way of solving the problem, such as adding a point  $D'$  on  $AB$  such as  $AD' = AD$  and using the coordinates in the  $DAD'$  system, then the software would not be able to handle this, since it does not know about any  $D'$  point.



In the parallelogram  $ABCD$ , prove that the areas of triangles  $ABE$  and  $ADE$  are equal.

Figure 6.3 The Parallelogram problem statement

### Sentence Selection Menu

This menu is accessed by clicking on the “Énoncés mathématiques” button. This opens an interface, displayed in Figure 6.5, allowing the student to search a mathematical result, property or definition following one or more keywords. In this example, the student selected the keywords “Area” and “Base”. This combination leads to the displaying of four choices in the right menu.

Selecting one of those leads to the next step, shown by the sentence in the middle of Figure 6.6. Indeed, in the case of a mathematical result, the student must indicate the object(s) in which he or she is providing a result. In this example, the sentence is “The angles \_\_\_ and \_\_\_ are supplementary angles”. To enter this result, the two angles must be specified by entering their names.

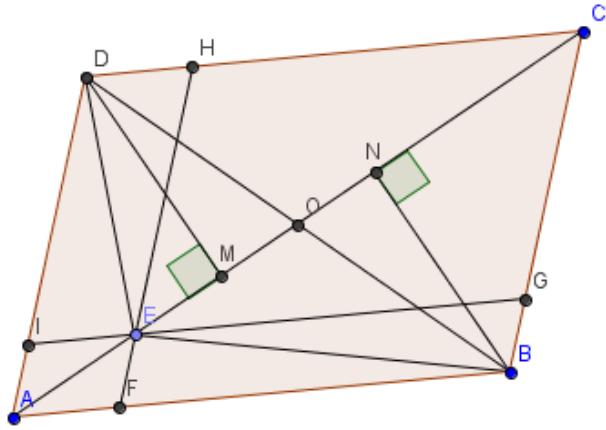


Figure 6.4 The Parallelogram problem super-figure

Lastly, if the entered names are lexically valid (three letters for an angle, one letter for a point, etc.), the student can click the green validation button to send the proposition to the software. He or she immediately receives a feedback from the tutor.

### Tutor Chatbox

Present at the bottom of the interface, this section takes the form of a chat between the student and the virtual tutor, Prof. Turing. It is not, however, a true chatbox, since the only way for the student to “speak” is to enter sentences using the interface presented in the previous section. Regarding the tutor, there are two possible interactions.

The first one is an immediate feedback on the entered sentence : a positive smiley face and message (“Good start”, “This is going well” or “I can’t wait to see what’s next”) in case of a valid sentence, or a sad face and a feedback message in the other case. The feedback message depends on the error. The most common case is simply entering a sentence that is not useful for the problem, but the student could also have entered the same sentence twice. Lastly, the tutor also handles the case where the student entered the reciprocal of an expected property (“If a quadrilateral is a rectangle, then it has four right angles” instead of “If a quadrilateral has four right angles, then it is a rectangle”).

The second type of interaction is a series of help messages in case the student is stuck. The software considers that there is an impasse if the student has not entered any correct sentence recently. Then, until it happens, a help message is sent every minute. To summarize

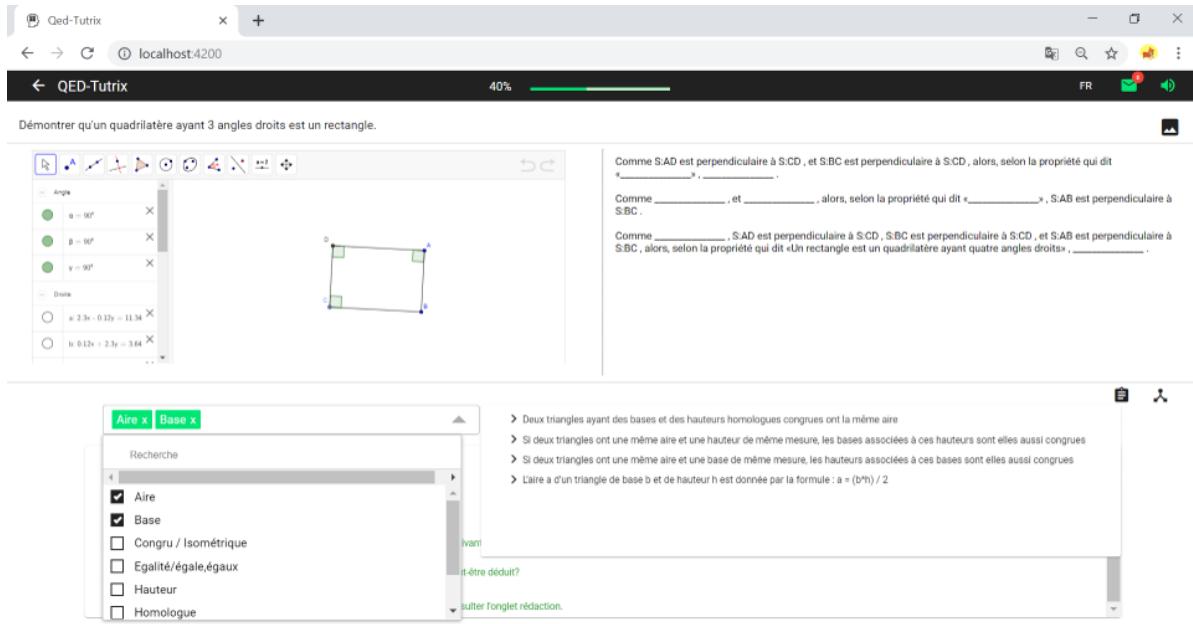


Figure 6.5 QED-Tutrix's sentence selection menu

the process, the software finds out which proof the student is likely to work on. Then, it identifies the element of this proof closest to the last valid sentence he or she entered. Then it attempts to help the student to find this element. The sequence of messages is determined by a finite state machine whose details are provided in the work of Leduc [2]. A summary is available in Section 6.4.3.

For example, let us go back to the parallelogram problem in Figure 6.3. The student has entered the results “ $(DM)$  is the height of the triangle  $AED$  through  $D$ ”, “ $(BN)$  is the height of the triangle  $AEB$  through  $B$ ” and “The area of triangle  $AED$  is equal to the area of triangle  $AEB$ ”. Let us consider that the student entered that last sentence most recently. To complete this inference, “The line segments  $[DM]$  and  $[BN]$  have the same length” and “Two triangles whose height and base are equal have the same area” are missing. The second one is the closest to the most recent valid sentence entered. The tutor will therefore attempt to help the student towards “Two triangles whose height and base are equal have the same area”. To do so, it provides messages that depend on the structure of the inference (“What is the property that allows inferring  $Y$  from  $X_1$ ,  $X_2$  and  $X_3$ ?”), then on the property itself (“How can you obtain the result that two triangles have the same area?”). After the set of possible messages has been exhausted, in general after a dozen or so messages, the tutor recommends the student to consult his or her professor.

This whole series of help messages is reset, and a new series is generated, every time a correct

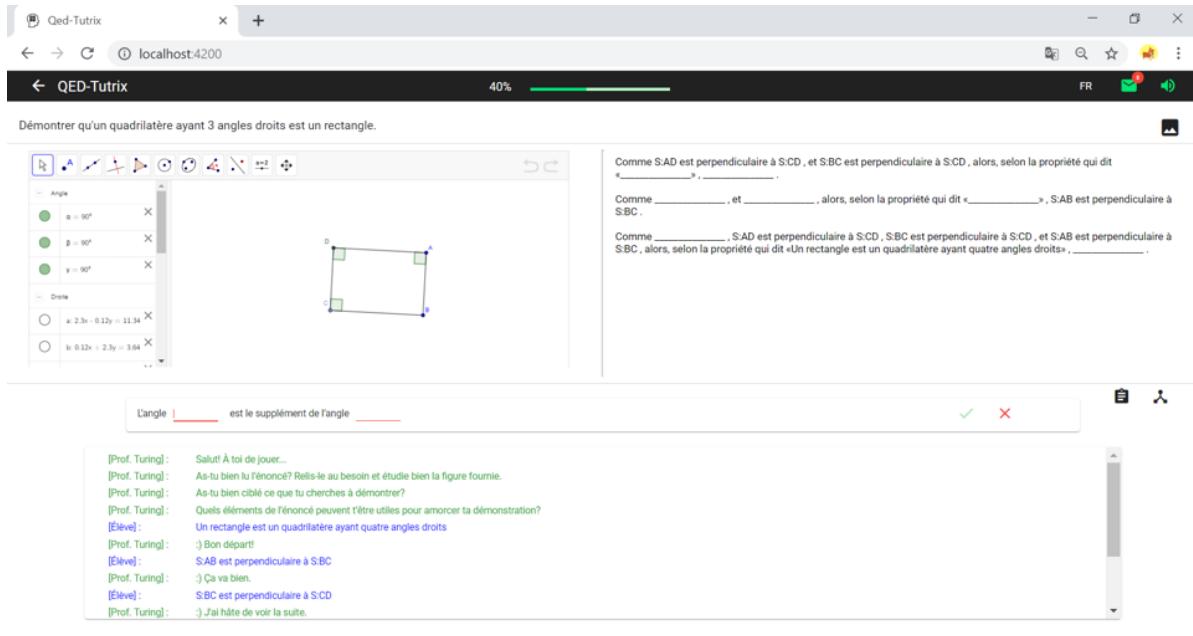


Figure 6.6 QED-Tutrix's sentence confirmation prompt

sentence is entered.

## Redaction Section

This last section, on the top-right of the interface, allows the student to visualise his or her progression by displaying the written proof the student is working on. When any valid proof of the problem is 40% completed, it is identified as the one he or she is working on and is displayed for the student to see. Prof Turing also indicates in the discussion zone that a strategy has been identified and is shown in the writing tab. Thereafter, if the software determines that the most advanced proof has changed, the writing tab displays this new one. The statements are organized in a forward chaining fashion and contains only the elements that have been entered previously. The remaining elements are replaced by a blank. The student can't interact directly with this section and must go through the statement menu submit the missing pieces of the proof.

It is not possible to interact with this section. Its only purpose is to help the student towards the completion of the proof, by highlighting the missing elements and handling most of the redaction process.

### 6.4.3 The Core Layers of QED-Tutrix

The features presented previously require a precise representation of several aspects of problem resolution. First, we need a structure to represent the set of all possible proofs for a given problem. Second, we need to keep in memory the various resolution steps taken by the student, and to be able to explore these steps to identify which proof he or she is working on. Third, we need to be able to generate, automatically, tutor messages in case the student is stuck in the resolution. In this section, we present the core layers of the software that we developed to tackle these needs.

#### The HPDIC Graph

The first structure we had to create is a way to store the set of possible proofs and the inferences composing them. Indeed, it is necessary to be able to navigate this set to find out, first, if the element entered by the student is relevant to this problem, and, second, the proof he or she is likely working on. This representation of the set of possible proofs is possible thanks to the structure of HPDIC graph. This graph, that we present in detail in this section, is the first and most central of the four layers that, by interacting together, form the core of the QED-Tutrix software.

This graph includes **Hypotheses**, **Properties**, **Definitions**, **Intermediary results** and a **Conclusion**, hence the HPDIC appellation. This graph is unique for each problem and is built from the inferences deemed acceptable to solve the problem. Therefore, generating this graph is a prerequisite to add a new problem to QED-Tutrix. It is a static structure which is loaded once when a problem is chosen by the user, and is referred to during the running of the software.

To build the HPDIC graph, we must consider each inference as a (directed) tree structure, at the center of which is the justification that is linked to parent premises, and to an only child, the consequent. This structure allows inferences to be chained, since the consequent of an inference can be used as a premise to another. Therefore, in this representation, a mathematical proof is a chain of inferences that begin with the problem's hypotheses and ends with the problem's conclusion.

Furthermore, since the problem's hypotheses and conclusion do not change from one proof to another, and since several proofs can use common intermediary steps, it is natural to fuse the inference chains of the various proofs together in a single structure. This process is illustrated in Figure 6.7. Subfigure a. represents an inference, subfigure b. represents the creation of a proof by adding two inferences centered on “Justification 2” and “Justification 4”, and subfigure c., with the addition of the two inferences centred on “Justification 3” and

“Justification 5”, represents a set of two proofs for the problem. The grey part in subfigures b. and c. represent the addition of the step compared to the previous one.

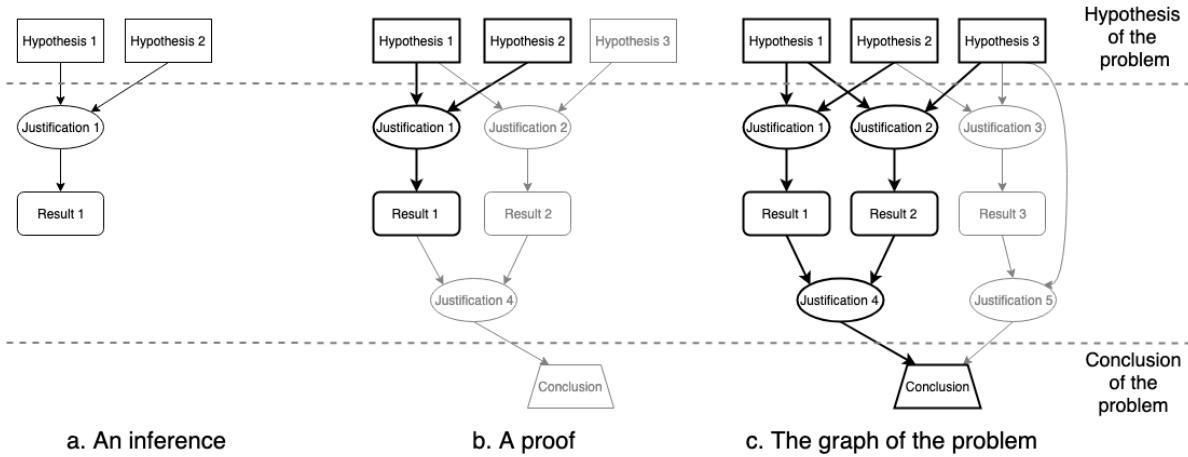


Figure 6.7 The informal construction process of an HPDIC graph

Although both an inference and a proof have tree structures, the complete HPDIC graph does not, since it can include cycles in the case of symmetrical inferences. Symmetrical inferences occur when for a same problem to different admissible proofs use two inferences that differ only by the fact that their premisses and results are reversed. In other words, this happens when two reciprocal justifications appear in two different proofs (“If a quadrilateral is a rectangle, then it has four right angles” and “If a quadrilateral has four right angles, then it is a rectangle”). This is the simplest case where a cycle can occur, but lengthier cycles also exist.

Overall, the HPDIC graph offers a straightforward means of storing the complete set of possible proofs for a problem. Furthermore, the fact that inferences are shared between proofs, allows for an efficient assessment of the progress of the student. Lastly, because of its structure, it is theoretically quite simple to generate a HPDIC graph by forward-chaining, i.e., starting from the hypotheses and generating all possible inferences from there, as we will explain in Section 6.5. Some steps must, however, be taken in practice to avoid combinatorial explosion, such as ordering the premises of each inference.

## The MIA

The second layer, the MIA or Modèle Interactif de l’Apprenant (Interactive Model of the Learner), is used to modelize the student’s progress while she or he drafts a proof. Classic geometry tutorial systems usually use forward or backwards chaining motion to predict the

next suitable action for the student and limit help to that particular step. In order to achieve this, only the list of activated nodes, without regards to the activation order, is necessary. Sometimes, they use the last activated node to offer more personalized help. With QED-Tutrix, our objective was to provide, when a student is stuck, different solution paths like a teacher would do, as we explained previously. We therefore introduced the MIA which accounts for the student's progress, but also for the chronological order of his or her actions, in order to provide a tailored help that respects the student's cognitive state.

The HPDIC graph being static, we had to define a structure to record the student's progression in the drafting of his or her proof. Therefore, the MIA adds information about the student's actions to every node of the HPDIC graph. For every statement form, the HPDIC (valid action) submitted by the student, we take note, in the corresponding node, a value according to the activation time. This information is constantly updated while the student drafts a proof and is used in the superior layers to generate tailored messages and to identify the proof he or she is most likely working on, according to his or her past actions.

## The EDOI

The third layer is the EDOI, Évaluation des Démonstrations et Ordonnancement des Inférences (demonstration evaluation and inference ordering). In order to help the student, QED-Tutrix generates instant messages as well as a series of hints. These aim at reviving a stalled solving process without handing the answers to the problem directly. To do this, we must know which of the admissible proofs is most advanced and arrange the inferences according to the student's cognitive state for them to be used by the next layer, the GMD (see Section 6.4.3). The EDOI layer is used, in the first place, to estimate the progress in each of the potential strategies explored by the student, but only the most advanced solution is considered to evaluate the general progress and to generate the instant messages and the proof which will be displayed in the redaction section. Then, the ordering algorithm will favour the inferences belonging to this same proof, which is considered as the student's current strategy. It will combine this information with the data from the MIA to generate a list of inferences which will be updated after each of the student's valid actions and injected into the GMD.

In QED-Tutrix, we define the most advanced proof as the proof for which the submitted statements to statements left to submit ratio is the highest. In the HPDIC graph, we therefore calculate the percentage of activated nodes for each solution. However, this procedure means the ratio for each proof must be recalculated each time the student submits a valid statement, which can be time consuming for problems for which there are millions of possible

proofs. To avoid these costing calculations, we designed an algorithm that shares information about the percentage of nodes activated in the HPDIC graph, providing that the graph is a tree. If it is not the case, we create beforehand a set of trees that combined include all the different solutions from the original HPDIC graph. We then can, by backwards tracing, almost instantaneously, find the most advanced proof. Next, an instant message is generated to indicate to the student if the submitted statement is accepted as part of a solution, part of the HPDIC graph, or of no use. In the case of an admissible statement, the displayed message depends on the percentage of completion of the proof.

Then, the inference arranging algorithm is solicited to provide the GMD with hints to submit to the student if he or she gets stalled. With the MIA's data, it is possible to know which inference the student has recently worked on in order to help him or her on this inference first and foremost. In fact, we arrange all the HPDIC's inferences according to a reverse chronological order, from the most recent to the first inference the student worked on. In the case of inferences that have not yet been evoked by the student, they are organized in a random order at the end of the list, since they constitute new paths not yet explored by the student. When the most advanced proof reaches a certain percentage of completion, the inferences that belong to it will be treated first and foremost, even if other inferences were worked on more recently, since we want to spur the student to complete this proof. To do so, first, we organise the inferences of the most advanced proof. We then add to the list, in reverse chronological order, the rest of the graph's inferences. Once the organisation is complete, the organised list is provided to the GMD to generate the necessary messages to restart a blocked solving process.

## The GMD

The final layer is the GMD, Génératuer de Messages Discursif (discursive messages generator). One of QEDX's goals is to help the student the way a teacher would, so, in order to reach this goal, we added the GMD layer which generates Prof Turing's messages. The type and order of the messages to display were determined by analyzing the student-teacher interactions. We uncovered a certain structure in the order in which teachers provide hints as well as in the content of these messages, and we used it as an inspiration in programming the messages for QEDX. The GMD layer, which uses to list of organized inferences, therefore completes Prof Turing's intelligence, and constitutes one of the main contributions of our project since it is able to display help messages in a similar fashion to a teacher, while following a predetermined schedule. It is therefore impossible for the student to directly address the tutoring agent with a question or to force it to solve the problem for him or her.

The finite state machine therefore includes the necessary states to take into account most of the behaviours witnessed in class and in our analysis. However, since all states are not used in every situation, it is possible to bypass these states by providing a list of empty messages. Since QED-Tutrix' architecture is guided by data, the messages associated with the different states are defined by didacticians or teachers in a text file. Besides the static messages, it is possible to program dynamic messages which depend on the statements submitted by the student. Also, different levels of messages can be implemented, from the more generic to the most precise. The content designers therefore have much control over Prof Turing's reactions.

To generate these messages, the machine, after the student submits a valid statement, retrieves the new list of organized inferences in the EDOI and positions itself in an initial state. Then, for each state, it displays the messages in the planned order by letting a prescribed delay between each message. At first, it submits generic messages that can be applied to the whole problem. Then, the different states are specific to given forms of inference, the EDOI list. Our finite state machine sequentially treats each help message for a given inference before dealing with the next one. The machine treats the inferences from the EDOI list until it is empty or until a maximum time without valid actions is expired. From there, the machine adopts a terminal help state to tell the student whose statement to submit or by advising him or her to consult a teacher. This last layer, by using the data from the lower layers and messages created by the didacticians, replicates actions of a teacher whose helping a student solve a geometry problem.

In this section, we summarized the four core layers of QED-Tutrix. The base layer includes the HPDIC graph which contains all the different proofs for a given problem and is used by QED-Tutrix's different components. On top of that is the MIA layer that reports the student's progress while storing the action chronology. From this chronology, the EDOI layer determines the most advanced proof, generates an instant message stating the student's progress and arranges the inferences in order to identify the help strategy QED-Tutrix will adopt. Finally, with the arranged list of inferences, the GMD layer generates, in the same fashion a teacher would, Prof Turing's messages. Although these layers are the main components of the software, the QED-Tutrix project encompasses more aspects. See also : [83] about messages ; [84] on blockages ; [85] on extraction of knowledge ; [27] on the theoretical referential system ; and [28] about tutorial systems.

## 6.5 The Need for Automated Proof Generation

As we explained in the previous section, the HPDIC graph is at the core of QED-Tutrix. Therefore, expanding the available set of geometry problems require providing an HPDIC

graph per new problem, meaning that generating those automatically provides a huge boon to the expansion of QED-Tutrix's reach. In this section, we provide a report on the automated proof generator we developed to address this issue.

### 6.5.1 Existing Theorem Provers

The main objective of this part of the project can be summarized in a couple of words : to automate the process of generating the HPDIC graph of a given problem. This broad objective can be separated in two tasks : first, to automatically generate the set of possible proofs for a problem, and, second, to translate this set into a HPDIC graph. The second task is quite straightforward and consists entirely into writing down translation rules. Therefore, this chapter focuses on the first one.

Finding a proof to a problem is identical to the task of proving a theorem, since a theorem is, in essence, simply a way to provide a shortcut between some hypotheses and a conclusion. Indeed, let us go back to the example of the parallelogram problem. Finding a solution to this problem is similar to finding a proof to the theorem : “In a parallelogram  $ABCD$ , with a point  $E$  on the diagonal  $AC$ , the areas of triangles  $AEB$  and  $AED$  are equal”. Therefore, a natural first step is to explore the avenue of (Geometrical) Automated Theorem Proving, (G)ATP. However, unlike “classical” theorem proving, we have several unique 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 constraints direct our search for a way to automatically find proofs. Indeed, there currently exist two general research avenues for geometry automated theorem provers : 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 [14, 15], Wu’s method [16, 17] and the exact check method [18]. Practical applications include the recent integration of a deduction engine in GeoGebra [19], which is based on the internal representation of points in the plane as coordinate tuples inside GeoGebra. Other examples include the systems based on the area method [73, 74], the full-angle method [59], 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 [61, 62]. However, the geometry taught in high school is not based on Tarski’s axioms. Therefore, proofs based on them are quite inaccessible for high-school students, violating our second constraint.

A prover that has very similar goals is GRAMY [44]. 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. 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. However, to the best of our knowledge, the source is not accessible, and no work has been done on it since 2004. Furthermore, it does not provide the complete set of proofs.

Another very interesting work is the one of Wang and Su [75], as it aims at providing proof for the iGeoTutor, and therefore has the same readability and accessibility at a high-school-level objectives. In particular, its template-matching algorithm for finding auxiliary constructions is quite promising. However, it requires the use of an external arithmetic engine, and, more importantly, also focuses on finding one proof.

Overall, the very specific needs dictated by the focus on educational interest considerably limit our options. The two only systems with similar goals are GRAMY and iGeoTutor, and they are not suited to our needs. Therefore, we chose to implement our own system.

### 6.5.2 The Choice of Logic Programming to Generate Inferences

In theory, a mathematical inference is quite easy to model in a computer, as it is essentially a combination of premises (“ $ABCD$  is a parallelogram” and “ $\widehat{ABC}$  is a right angle”), a property (“a parallelogram with a right angle is a rectangle”), and a result (“ $ABCD$  is a rectangle”). It can be quite different in practice, however, as we explain in Section 6.5.7, but, for the moment, let us consider only this ideal case. This structure is extremely similar to the inference mechanism in logic programming, where a program is a set of facts and rules, and where we infer new facts (the result) based on a rule (the property) and existing facts (the premises). Then, because the result of an inference can be used as the premise of another, a proof is simply a chaining of inferences, starting at the hypotheses of the problem, and reaching the conclusion. Finally, since the mathematical results can be used in several proofs, we can merge the proofs to create a unique structure containing all the possible proofs

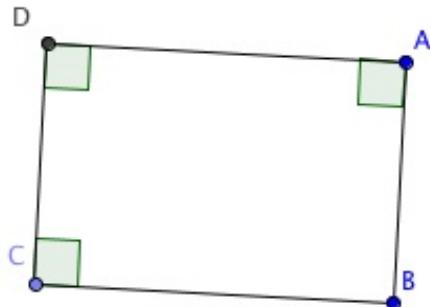
for a problem, as we illustrated in the previous section, in Figure 6.7. Thus, the core of our problem solver is, in theory, very simple : we create a Prolog fact for each hypothesis of the problem, a Prolog rule for each property, and we let the Prolog inference engine infer every possible new fact (mathematical results) from those. Every time a new fact is inferred, we store it in the form of a mathematical inference (premises + justification = consequent).

### 6.5.3 Available Data

**Available properties** Although not detailed in this chapter, an important part of this work includes collecting, compiling, and organizing the mathematical elements that are used by the software. We have analyzed 19 Quebec high school textbooks ranging from 7th to 11th grade and extracted 2855 mathematical statements, representing 707 properties and definitions that can be translated into an inference [27]. This large databank of definitions and properties is necessary to properly adapt the software. We call *referential* [24, 25] the set of properties and definitions that are allowed or expected in the resolution of a problem. These are known to greatly vary among grades, textbooks, and teachers. Different phrasing might be used from one referential to another even when describing foundational mathematical properties such as *the sum of interior angles in a triangle is 180°*. This variability in phrasing must be accounted for to ensure consistency among the properties and definitions used by QED-Tutrix and to allow teachers to choose their innate preferences to be used by their classes. The final cumulative referential extracted from the various school textbooks includes Euclidean geometry, area and volume formulas, metric relations, transformational geometry, analytic geometry and basic vector geometry. We do not attempt to isolate a minimal set of axioms that are used in high school geometry, but instead aim at implementing enough properties and definitions to cover all the material present in high school geometry textbooks. Our ultimate goal is to encode all of this comprehensive set of properties in Prolog.

**Available problems** To learn mathematics, one must solve problems [26]. The tasks of finding, adapting and creating new problems plays an important role in a tutoring system such as QED-Tutrix. Currently, our work base is composed of sixty problems covering a vast array of mathematical topics from high school courses. We divided those in a training set of 19 problems and a validation set of 41 problems. The training set was available during the development of the automated proof generator, and the validation was used after the development to obtain statistics on the coverage of the available proofs.

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



(a) Problem statement.

```

hypotheze(point(a)).
hypotheze(point(b)).
hypotheze(point(c)).
hypotheze(point(d)).

hypotheze(line([a,b])). 
hypotheze(line([b,c])). 
hypotheze(line([c,d])). 
hypotheze(line([d,a])).

hypotheze(isAQuad(quad(a,b,c,d))).

hypotheze(angleValue(angle([d],a,[b]),value(90))).
hypotheze(angleValue(angle([b],c,[d]),value(90))).
hypotheze(angleValue(angle([c],d,[a]),value(90))).

conclusion(rectangle(quad(a,b,c,d))).
usefulAngle([a],b,[c]).
```

(b) Problem encoding.

Figure 6.8 The translation of the rectangle problem from its statement.

#### 6.5.4 Encoding of a problem

After identifying relevant problems, the next step is to translate them in a Prolog file. An example of such a translation is provided in Figure 6.8. The resulting file has several parts.

**Implicit hypotheses** The first eight lines provide names to the geometrical objects present in the problem figure. We refer to these as **implicit hypotheses**, since they are needed for the prover, but are not specified in the problem statement. Furthermore, even though the names of the points are explicit in this case, the Prolog engine needs names for each point, line, etc., even if they are not at all given, neither in the statement nor in the figure. In that case, arbitrary names must be provided (such as  $P_1$ ,  $P_2$ , etc.). Lines 1 to 4 provide the set of points present in the figure. Then, lines 5 to 8 indicate which point are linked by a line. Here, we have the four lines ( $AB$ ), ( $BC$ ), ( $CD$ ), and ( $AD$ ) (we do not differentiate segments and lines). If we wanted to add the line through  $A$  and  $C$ , we would have to add  $hypotheze(line([a,c]))$  to the file. The example here is simple enough, but in more complex problems, the encoding of lines can become a delicate issue. Indeed, we must provide, **as soon as the problem is encoded**, all the points that are on the line. This set of points is, as far as Prolog is concerned, the unique identifier for the line. A direct consequence of this

implementation is that it becomes impossible to add, during a proof, new points to a line.

**Explicit hypotheses and conclusion** The next four lines provide the hypotheses in a more general sense, meaning the ones that are usually given explicitly in the problem statement, hence the name **explicit hypotheses**. Here, the statement of the problem, “Prove that any quadrilateral with three right angles is a rectangle”, with the addition of the figure, provides four hypotheses : there is a quadrilateral named ABCD, and three of its angles,  $\widehat{DAB}$ ,  $\widehat{BCD}$  and  $\widehat{CDA}$ , are right angles. The problem statement also provides the expected conclusion :  $ABCD$  is a rectangle, encoded in the second-to-last line.

**Auxiliary hypotheses** In many problem resolutions, there is, at some step, the need for the construction of additional elements, such as a new line or point. However, the creation of such elements is a difficult issue. Given that our aim in this project is not actually to solve problems in and of itself, but to ease the process of adding new problems to QED-Tutrix, we chose to require the addition of **auxiliary hypotheses**, i.e. elements that are not present on the problem statement directly, but are useful in one or several of its resolutions. For instance, in the rectangle problem, one could write a proof using the diagonals of the rectangle. The fact that  $(AC)$  is a line would therefore enter in this category. Structurally, auxiliary hypotheses are identical to implicit and explicit hypotheses, it is their origin that differs. When adding these auxilliary elements to the figure of the problem, we create the **super-figure**.

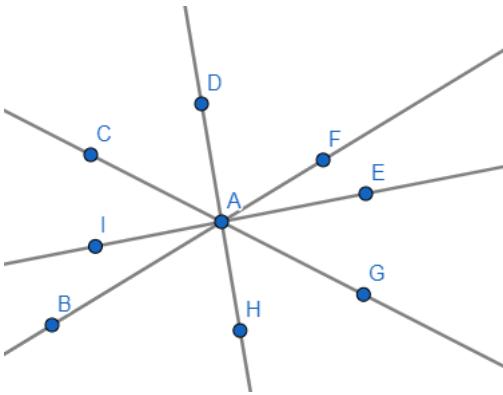


Figure 6.9 A geometrical situation with many angles.

**Additional elements** The last line in our example is neither a hypothesis nor a conclusion, but additional information provided to the prover. It can be of two types : useful angles and dictionary items. The useful angles are required because in some geometrical situations, such

as the one in Figure 6.9, the number of possible angles can become quite huge. Here, there are 8 points around the center A, meaning that there are  $8 \times 7$  angles. When considering inference such as “two angles that share a side are adjacent” and “two angles whose sum measures 180 degrees are supplementary angles”, the number of possible inferences becomes very impractical. For this reason, to limit the combinatorial explosion that plagues many synthetic automated theorem provers, we chose to impose the following requirement : the problem file must contain the list of all angles on which Prolog is allowed to infer results. In other words, if an angle is not explicitly written in the problem file, no result on this angle will be present in the HPDIC graph. Finally, the dictionary allows the user to provide alternative names for geometrical objects. This is useful in problems where lines, angles, circles, or, more rarely, triangles and quadrilaterals, have alternative names, such as “line  $l$ ” or “rectangle  $R$ ”. This is not needed for the solver, but is important for the tutor software, since the students must be able to enter their sentences with any valid name.

One should note that, as required by the Prolog language, each constant must have its first letter in lowercase, otherwise Prolog would consider it as a variable. This could lead to some collisions if there is a point named  $A$  and a point named  $a$ , but, in our experience, the conventions used in the statement of high school geometry problems prevent that situation.

### 6.5.5 Generation of the Complete Set of Proofs

Once the problem is encoded in Prolog, we proceed to the proof generation process. To summarize, we proceed to a construction of the graph by forward chaining. We begin with the hypotheses provided in the problem file, such as the one in Figure 6.8, and put them in the set of known facts. Then, until this set is empty, we take one fact out of the set and ask Prolog : “what new facts can you infer using this one?” and add all the new facts to the set. Every new fact also leads to the storage of the inference that led to the creation of this fact in the proto HPDIC graph.

When the set of known facts is empty, it means that Prolog is not able to find any new fact. Therefore, it has found all the results that could possibly be inferred from the set of hypotheses, using the allowed properties. If the problem’s conclusion is in this set of results, then we have found at least one proof of the problem. If not, there is an error. Otherwise, we explore the generated graph in backward chaining, starting from the conclusion, and marking an element on the graph only if it can be used to infer the conclusion. This last step is necessary, because the inference engine can infer results that are valid, but useless for the resolution of the problem, i.e., not used in any proof that reaches the conclusion. These results are marked, but not removed from the graph. This will be useful when the student

works on the problem and enters such a result. In this situation, the software should not say “This result is false”, but instead “This result is valid, but not useful for the problem. Try something else.” For this behavior to be possible, the “valid but useless” elements must remain in the graph and be identified as such.

Since the generated graph is based on mathematical properties, which typically have a reciprocal, there are usually many cycles in the resulting graph. For example, if  $ABC$  is a right triangle in  $A$ , then we can infer that  $\widehat{BAC}$  is a right angle. And since  $\widehat{ABC}$  is a right angle, we can infer that  $ABC$  is a right triangle in  $A$ . This may be a problem for the calculation of the students’ progress in the proof resolution, but it is solved by a clever exploration of the graph. This algorithm, however, is not part of the proof generation process, and is not detailed in this chapter.

This algorithm is written in Python, and sends a Prolog query every time the program needs to obtain new results from a fact. Because Prolog is a logic programming language, it is not well-suited for handling complex data structures and printing them in a text file, hence the choice of using a more classical language for all the “standard” tasks.

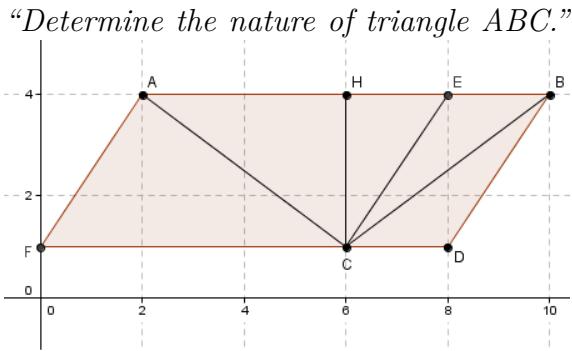
### 6.5.6 Validation

Ideally, the generator should be able to generate all the possible proofs for each problem. However, it is impossible to assess if all the proofs have been found without already knowing all the possible proofs beforehand, which is such a time-consuming process that we dedicated this whole project to avoid it. Therefore, we used a two-step validation process.

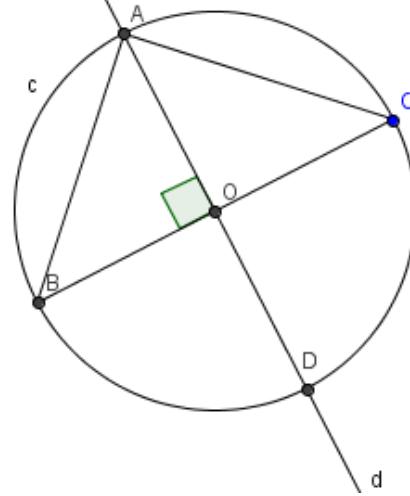
First, we used the four problems for which we had a manually created HPDIC graph. These four problems are not extremely difficult and remain accessible to high school students, but have been carefully crafted by experts in mathematics education to offer a large set of possible proofs that encompass many geometry concepts, and therefore have large HPDIC graphs. Two of these four problems have been used as examples in this chapter, in Figures 6.3 and 6.8. The other two are given in Figure 6.10. We then encoded these problems in Prolog and attempted to reproduce the handcrafted HPDIC graph for each of them. Since these graphs have been created, and therefore validated, by mathematics experts, the simple fact that the proof generator is able to re-generate them confirms that the process in itself is sound, and that the generator is indeed able to infer all possible results using a given set of properties.

Second, we also had to validate that the proof generator is also able to solve simple, more realistic problems. Indeed, these four problems are doable in high school, but are not repre-

*“Let  $c$  be a circle of diameter  $BC$ ,  $d$  the perpendicular bisector of  $BC$  and  $A$  an intersection between  $c$  and  $d$ . What is the nature of triangle  $ABC$ ? ”*



(a) The triangle in a parallelogram problem.



(b) The triangle in a circle problem.

Figure 6.10 The statements of two of the four intial problems.

sentative of the simpler problems used in class. Therefore, we extracted 60 problems from various datasets : 27 from the work of Py [13], 14 from high school manuals, and 19 were created by an expert in mathematics education to complete this set. Among these 60, we randomly selected 41 to be used exclusively for the validation process, and the remaining 19 were available during the development of the proof generator.

Then, we encoded those problems in Prolog, including the creation of the super-figure, and entered them in the proof generator. If at least one proof was found, i.e., the conclusion was successfully inferred by Prolog, we considered this problem as solved. Since those problems are much simpler than the original four, they should not have more than one or two possible solutions, and therefore considering that generating at least one proof is enough for a success was deemed acceptable.

Out of these 41 problems, the proof generator managed to solve 33 of them, or 80%. Then, we analyzed the reason why the remaining 8 were not solved. Five of them need some algebraic manipulation, for instance being able to “remember” that a segment is  $2/3$  of the length of another, and to use that information later in the proof, a process that we chose not to implement in the proof generator at the moment. Four of them failed because they needed

concepts in a domain of geometry that we had not yet implemented, such as the notion of a perimeter or of a polygon with more than four sides. Lastly, two of them failed because of a missing or incorrect property. It should be noted that these reasons are not mutually exclusive.

Overall, among these 8 misses, only two are due to real issues in the set of implemented properties. The remaining six are simply out of the initial scope of this project. Therefore, we can conclude the following :

- the automated proof generator is able to solve a respectable portion, 80%, of the given problems ;
- the initial scope of this project, which does not encompass several geometry concepts such as polygons or perimeters, and excludes the possibility of an algebraic manipulation, is sufficient for 80% of the problems, and by solving the issues of the two missing or incorrect properties, this would reach 85% ;
- the usage of logic programming is a sound solution to generate proofs, and by both expanding the scope of encoded properties and solving implementation issues on existing properties, we would potentially be able to solve all the given problems.

It is important to note that performance has not been a consideration at all in our validation process. Therefore, this validation only confirms that logic programming is a sound approach to generate proofs relevant for high school education, but we make no claim whatsoever about the performance of such a method in the ATP domain as a whole. Nevertheless, we also measured the time spent on proving each of the 33 successful problems, and these results are quite interesting. The average is 259 seconds, but the median is 1 second. This is explained by the fact that one problem in particular is unreasonably long to process, at 8440 seconds, even though it is not more complex than the other : “*ABC* is an equilateral triangle, and *E*, *F* and *G* are the middle points of *AB*, *BC* and *AC* respectively. Prove that *EFG* is also equilateral.” The source of this complexity is a typical example of combinatorial explosion. Indeed, even though this problem is quite simple to solve, there is an incredible number of possible variations of the proof, because this problem has no less than 12 angles measuring 60 degrees, meaning 132 (12·11) results indicating that two angles have the same measure, each obtainable in around a hundred possible inferences. Furthermore, there are nine segments of identical lengths. By combining the two, we obtain around 600 inferences about equal segment lengths, 900 inferences about equal angle measures, and a thousand inferences related to equal triangles. By comparison, the second most time consuming problem, at 43.7 seconds, has a total of around 180 inferences in its HPDIC graph. Therefore, it is not surprising that solving this problem took such a disproportionate amount of time.

If we remove this problem from the calculations, the average goes down to 3.4 seconds and

the median to 0.95. By removing the next two problems, at 14.8 and 43.7 seconds, the average becomes 1.68 seconds and the median 0.85. In other words, this proof generator is overall reasonably fast and could be used in real time for most problems in a future application, but it is also very easy to design a problem that is absolutely in the scope of high school education, but extremely time-consuming for the proof generator.

### 6.5.7 Limitations

Despite the encountered educational challenges, logic programming is indeed quite adapted to generate the proofs of high school geometry problems. However, it is a lengthy task, and our solver still has several limitations whose correction would require work of varying magnitudes : possible in the current implementation ; possible but requiring fundamental changes in the implementation ; and intrinsically difficult or impossible.

#### Possible in the current implementation

As we mentioned previously, the solver currently does not handle algebra. For instance, it is impossible to represent the result “the length of segment  $AB$  is three times the length of the segment  $CD$ ” without knowing either values. This would require the implementation in Prolog of basic arithmetic operations, but it would be possible, since the use of such results remain in the domain of inferences. Furthermore, we do not handle proofs by contradiction, but, similarly, these kinds of proofs still follow an inferential format. The difference is that the hypotheses represent an impossible situation, and the conclusion is “we reach a contradiction”. This would require to implement all the numerous possible inferences that result in a contradiction, such as having a triangle with two parallel sides, but it would fit in our solving process. Those two limitations cause another problem concerning an axiomatic referential. Indeed, Euclid’s axioms generate most of high school’s properties, but most of his work is done by proofs by contradiction which is not currently feasible by our system. On the other hand, Clairaut’s adaptation of Euclid’s work also generates this geometry and without using proof by contradiction. However, Clairaut uses an algebra system to bypass this difficulty, something that, again, our system cannot do. For this reason, our system requires a much more extensive referential that includes those properties that we cannot generate to be able to solve the problems. This limitation adds another layer of complexity concerning the challenge of the level of granularity discussed in [4].

## Possible with some fundamental change

One of the biggest limitations of our solver is the necessity to provide, beforehand, the whole geometrical situation (the super-figure), including the set of elements that could be useful in a proof. This is very difficult to solve, as the construction of new elements in automated theorem proving in geometry is a whole research domain in itself. However, we envisioned a possible solution. Since QED-Tutrix is an online tool, and that the generation of the possible proofs can usually be done in a matter of seconds, it would be possible to wait for the student to construct a new element in the interface. Then, when constructed, QED-Tutrix sends it to the prover, that attempts to infer new results from this new hypothesis. If we can reach the conclusion, then the hypothesis is indeed useful, and is added, in real time, to the HPDIC graph of the problem. This solution, however, would require a complete change of the interactions between QED-Tutrix and the solver, and comes with its share of issues. A direct, less profound consequence of this limitation on the possible proofs is that it is impossible to discover, in the middle of a proof, that a line actually passes through a point. This is because it is necessary to provide the complete set of points the line passes through from the beginning when encoding the problem. This limitation could be solved, but it would require a drastic change to the way we represent elements in Prolog.

## Intrinsically difficult

A crucial issue comes from the floating-point precision of calculations in the machine. For example, let us assume that there is a right triangle  $ABC$ , whose dimensions we know. From these dimensions, we can calculate the cosine, sinus and tangent of the angles. This calculation is done with a certain precision. This creates a first problem : what level of imprecision from the student is acceptable ? If the cosine of the angle has a value of 0.7654321, is 0.8 acceptable ? Is 0.77 ? Is 0.76 ? Still, while delicate, this question concerns the tutor aspect, and not the automated proving, and we will not discuss it further. However, there is another issue concerning approximations. In the previous example, now that we know the cosine of the angle, by using the reciprocal of the property, we can calculate the value of the sides of the triangle. This calculation is also done with a certain precision, and if the resulting value is even imperceptibly different from the already known one, then it is considered a new result, that will lead to the calculation of a slightly different cosine, and so on. Hopefully, we were able to solve this issue by only considering the first result. When we already know the value of something, every inference that results in another measurement of that value does not do any calculation, but uses the already known one. In this example, if we already know that the hypotenuse of the triangle is of length 5, then any inference which results

in the measurement of that line segment will not even calculate it, but reuse the 5 that has been calculated previously. Lastly, the precision can also create problems when using an inference such as “two angles are equal if they have the same measure”. Just checking the equality of their measures is not enough, since they could be very slightly different due to rounding. In that case, we allow for a difference of 1% between the two values. We chose that precision by trial-and-error, with the argument that, in high school geometry problems, two different values that are supposed to be measured will differ by more than 1%. This is not a mathematically ideal solution, but for our purposes, it is enough.

## 6.6 Conclusion

In this chapter, we presented both the tutor software QED-Tutrix and the ancillary automated proof generator we developed to complete its set of available problems. QED-Tutrix provides a space for the students to solve high school geometry proof problems. In accordance to the theory of mathematical working space, it helps the student in the three geneses of mathematical work, i.e. to learn and do mathematics. Furthermore, its interface allows the student to explore the problem, enter mathematical elements as they come to him or her, and redact a mathematically sound proof. Lastly, its virtual tutor is able to provide custom-tailored help to the student in case of an impasse in the resolution, by attempting to identify the very specific step of the proof that he or she is stuck on.

We then presented the four layers that compose the core of QED-Tutrix, and that allow these functionalities. The first is the HPDIC graph, storing in an oriented graph the set of all possible proofs for a problem, which is useful to instantly assess if the sentence entered by the student is relevant to the problem resolution, and serves as a support for the next layers. Then comes the MIA, allowing a representation of the student’s cognitive state by storing all the sentences he or she entered in the resolution. The third layer is the EDOI, aimed at quickly identifying the proof the student is most likely working on, a crucial step for both the redaction part of the resolution and the tutor system. Lastly, the GMD is a finite state machine that determines the series of help messages that are most suited to help the student in any given situation.

The first of these, the HPDIC graph, is a crucial part of the software. Furthermore, unlike the other three layers, it has to be generated beforehand for each geometry problem added to QED-Tutrix. To be able to quickly include new problems, and consequently to generate new HPDIC graph is, necessary for turning QED-Tutrix into a software truly relevant for mathematics education. Therefore, we dedicated a whole branch of the project to creating such an automated proof generator.

For this endeavour, logic programming was a natural choice, because of the closeness between the reasoning behind logic programming and the inference process used in mathematical proofs. Creating such a generator then became a task of implementing each mathematical property, definition and possible result in Prolog. That task was not without difficulties, since human reasoning is very different from a mechanical process of inference, and, despite the apparent rigour of mathematical reasoning, the reality of classes is that many assumptions and shortcuts are taken, with good reason. Therefore, we had to adapt the mechanical process of finding new inferences to the many subtleties of human reasoning. At the end, despite these difficulties, the resulting automated proof generator managed a very reasonable coverage of typical high school problems.

Overall, while QED-Tutrix is by itself a very interesting tool for research purpose, the adjunction of an automated proof generator offers the possibility of encoding as many problems as is needed in the future, opening the possibility of QED-Tutrix becoming a truly useful software in high school classrooms.

## CHAPITRE 7 ARTICLE 4 : CREATION OF A MATHEMATICAL MODEL FOR QED-TUTRIX' AUTOMATED PROOF GENERATOR

**Ludovic Font**

École Polytechnique de Montréal  
Montréal, QC, Canada  
ludovic.font@polymtl.ca

**Sébastien Cyr**

Université de Montréal  
Montréal, QC, Canada  
sebastien.cyr.1@umontreal.ca

**Nicolas Leduc**

École Polytechnique de Montréal  
Montréal, QC, Canada  
nicolas.leduc@polymtl.ca

**Michèle Tessier-Baillargeon**

Université de Montréal  
Montréal, QC, Canada  
michele.tessier-baillargeon@umontreal.ca

**Michel Gagnon**

École Polytechnique de Montréal  
Montréal, QC, Canada  
michel.gagnon@polymtl.ca

**Philippe R. Richard**

Université de Montréal C  
Montréal, QC, Canada  
philippe.r.richard@umontreal.ca

**Reference :** L. Font *et al.*, “Creation of a mathematical model for qed-tutrix’ automated proof generator,” *Proceedings of the Sixth Symposium on Mathematical Work*, 2019, p.335-356.

### 7.1 Mise en contexte

Cet article, présenté au symposium des Espaces de Travail Mathématiques, diffère des trois précédents. En effet, là où ceux-ci ont été publiés dans des conférences ou journaux pour un public du domaine informatique, ce dernier est orienté vers la perspective de didactique des mathématiques. L’on n’y trouvera donc pas de détails sur le fonctionnement de QED-Tutrix ou du générateur de preuves, mais plutôt un compte-rendu, premièrement, de l’importance de la génération de preuves automatisée pour la portée de QED-Tutrix, et, deuxièmement, des difficultés rencontrées lors de son développement, dues aux contraintes imposées par le besoin de créer un outil pertinent pour l’enseignement des mathématiques. Ainsi, les premières sections, 7.3 à 7.5 offrent une présentation du processus de génération de preuves présentée de façon accessible à un non-expert en informatique. Leur lecture est laissée au choix du lecteur. En revanche, les sections 7.6 à 7.8 sont inédites et présentent les obstacles profonds rencontrés lors du développement, découlant du fait que la géométrie est un domaine mathématique

complexe, qui est grandement simplifié lors de son enseignement au secondaire. Cet article a également été relu et accepté, cette fois par le comité d'organisation de la conférence ETM.

## 7.2 Abstract

The intelligent tutor system QED-Tutrix has currently one issue preventing large-scale experimentations, as it has only five implemented problems. To address this issue, we began the process to automatize the generation of proofs. This process requires modeling of properties and results used in high-school that must be computable, i.e. without any approximation or tolerance for errors, which is far from what happens in class. We identified three sources of distance between the computable model and the real-life one : the naming of geometrical objects, the approximations in the statement of a problem, and the level of granularity of proofs. For each, we implemented solutions, both in QED-Tutrix and in the automated proof generator.

## 7.3 Introduction

QED-Tutrix is an intelligent tutor software that offers a mathematical working space [86] to solve high-school geometry problems, in a way that is as close as possible to the way students would solve the same problem without the software. In other words, the student is free to explore the resolution without being restrained by forward or backward chaining. Every proof element he inputs in the software leads to a feedback by the virtual tutor. This constant interaction between the system and the student is one of its main features. At its core, QED-Tutrix has the ability to identify the proof(s) the student is most likely trying to achieve, and, with that knowledge, help the student towards that proof, by helping him find the missing elements. However, this system requires the a priori knowledge of all possible high-school proofs for any given problem, meaning that, to implement a new problem in QED-Tutrix, a list of all the possible solutions must be found and implemented. Currently, this work must be done manually, which is an extremely tedious and time-consuming task. Hence, during the development of QED-Tutrix, three years were necessary to program only five problems [2]. Although sufficient for a preliminary study [3] of the use of the software by students and teachers, this is insufficient for a long-term study of the efficiency of QED-Tutrix in class. Therefore, the automation of the generation of all possible proofs is a priority in the development of our software.

After a careful analysis of the state of the art in Automated Theorem Proving (ATP), we decided to implement a custom automated proof generator, for essentially three reasons.

These are discussed at great length in one of our previous papers [4]. First, provers typically aim at proving a theorem, with no constraint on the proofs, as long as it is valid, and as a consequence are focused on speed and not the readability of the proof. Therefore, most provers rely on algebraic methods, such as converting the geometry problem into an equation and solving it, providing, without a doubt, a valid proof, but this proof is completely irrelevant in the context of high-school geometry education. Second, as mentioned, we want all possible proofs for the problem, in a given axiomatic (more details on this axiomatic in the last section), and provers are usually not focused towards exploring several proof paths. Third, we want to be able to represent the degree of granularity of proofs, i.e. to be able to adapt the degree of rigorousness the student has to follow, depending on several factors, such as the class level, the mathematical topic currently studied, or the teacher's personal preferences.

One of the most crucial part of our work to implement that prover is to identify a way to represent the mathematical knowledge used in class, but in a way that is useable by an automated engine, entirely intolerant to approximations. This created many issues that have to be addressed. In summary, in this project for improving QED-Tutrix, our goal is to provide an interface between the teacher and the tutor software in order to facilitate the process of implementing new problems. As a result, our dynamic in this project is to base the development process on a constant discussion between mathematics education experts and computer science experts, instead of developing the software independently and asking the user to adapt along the way. The purpose of this paper is to summarize and explain the issues encountered, as well as present the choices made by experts of both domains to solve them.

First, we quickly present the inference graphs, or HPDIC graphs, an essential part for the internal representation of proofs in QED-Tutrix. Then, we detail the operation of our solver, with an emphasis on the difficulties created by the underlying educational requirements. We then regrouped the issues encountered in three categories, each explained in its section. First, we present the problematic of object naming, in particular regarding the need for a unique identifier for each object. Second, we explain the issues created by the approximations made both in the statement and in the proof of a problem, such as not taking into account the orientation of an angle. Lastly, we explain the concept of granularity of a proof, and how it is handled by the generator.

#### 7.4 From a problem statement to a solutions graph

One of the fundamental pieces of QED-Tutrix is its ability to store and explore the graph of all possible solutions to a given problem. This allows the software to follow in real-time the

progress of the student, and to identify the proof he or she is likely trying to write. This graph of solutions is called the HPDIC graph. In a few words, it is a graph composed of results and justifications. An inference is the combination of some results or hypotheses (the antecedents), a justification, and another result (the consequent). Antecedents and consequents are the same kind of object, since the consequent of an inference can be the antecedent of another inference. An example of inference is provided in Figure 7.1.

In this representation, a proof is a chain of inferences from the hypotheses of the problem to the conclusion. When merging all the proofs for a problem in a single graph, the result is called the HPDIC graph of the problem. The HPDIC graph rectangle problem, whose statement is given in Figure 7.2, is provided in Figure 7.3. The details of this graph are not important to the understanding of this paper, instead one should focus on its structure, based on the chaining of inferences such as the one provided in Figure 7.1.

Those graphs are encoded in QED-Tutrix using simple encoding, where each line represents an inference, i.e. a justification, one or more antecedents, and a consequent. Currently, as mentioned in the introduction, these files have been written manually, adding the burden of writing in an encoded form to the already tedious task of exploring all accessible solutions to a problem. These two difficulties are the reason why we decided to prioritize the creation of an automated proof generator in the current work on QED-Tutrix.

## 7.5 Overview of the proof generation process

The whole pipeline of the proof generation process aims at streamlining the operation of adding a problem to QED-Tutrix, making it accessible to any teacher, and therefore not only experts in the system. The first step, which is not detailed in this paper, consists in extracting the hypotheses and conclusion of the problem from the figure(s) and statement. The second step, the focus of this paper, exploits that information to automatically generate an HPDIC graph that is immediately usable by QED-Tutrix, containing all proofs in a given axiomatic. The intermediate results between the two steps contain the conclusion, but also a series of hypotheses, that is informally divided in two categories :

- The “high level” hypotheses of the problem, for instance the topmost results (rectangles) in Figure 7.3. These hypotheses are non-trivial, and represent the information that should be given explicitly (typically, in the text) or implicitly (typically, in the figure) to the student.
- The “low level” hypotheses, which are obvious for any student, but still necessary for an automated tool, such as “A, B, C, D are points”, “there is a line, named (AB),

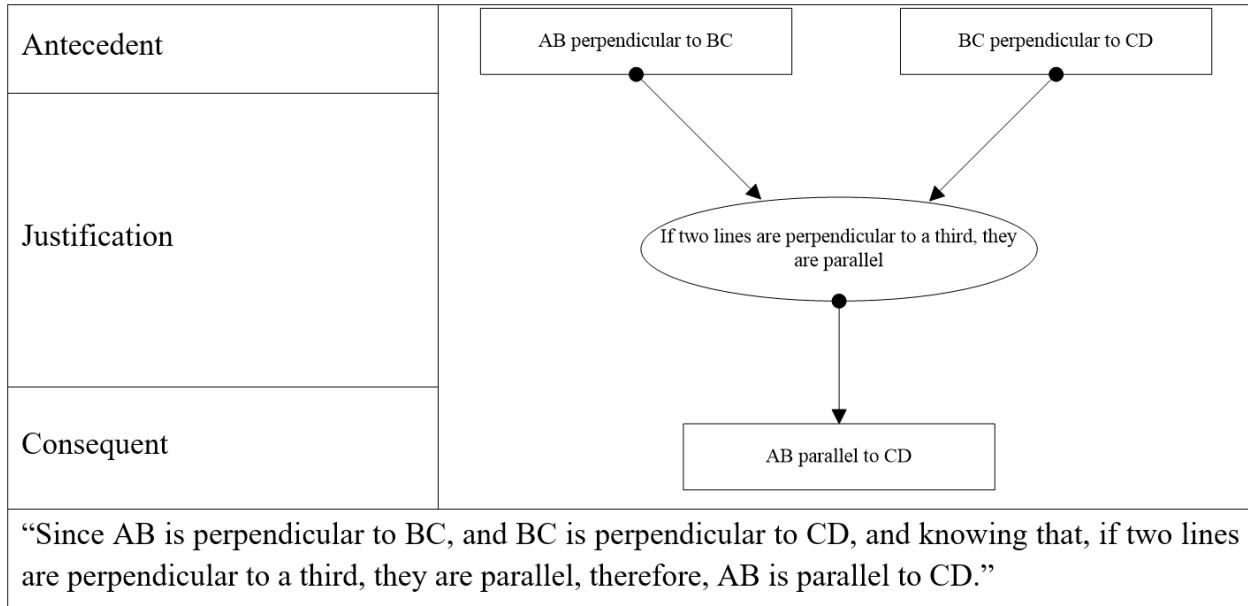
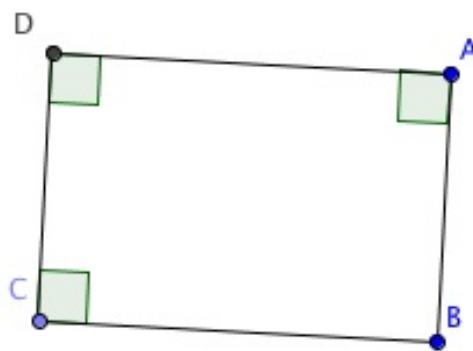


Figure 7.1 A simple inference.



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

Figure 7.2 The rectangle problem.

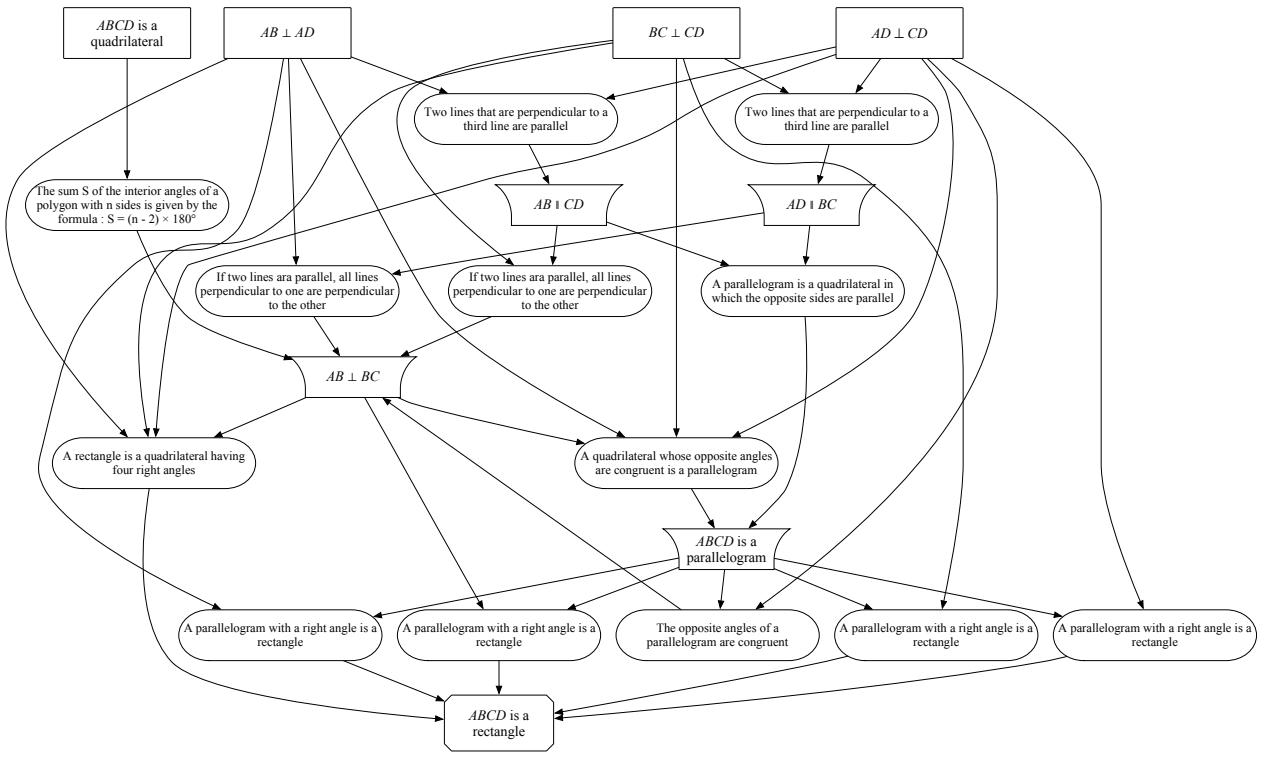


Figure 7.3 The HPDIC graph for the rectangle problem.

that contains the points A and B”, or “there is an angle named ABC that represents the angle between AB and BC”.

A crucial and difficult requirement is that this second category should contain **all geometrical elements** that could be used in **any** proof. In other words, for a proof to be accepted by QED-Tutrix, all intermediate elements, such as triangle heights, median lines, or even useful angles, should be indicated in the “low level” hypotheses. The reason for this requirement is a technical one : first, to impose and unify the name of intermediate objects in the inference graph, and, most importantly, to indicate to the automated proof generator which objects he is allowed to work on. Indeed, the automated proof generator is not able to create new objects, since the automated construction in geometry theorem proving is an extremely complex matter. To the best of our knowledge, the only system attempting to construct new elements is GRAMY [44], and it only allows a small subset of new constructions. Therefore, even though we plan to explore this possibility in the future, we chose, for the time being, to limit the generator’s work to a predefined set of elements.

In other words, in the problem given in Figure 7.4a, no result can be inferred about the line (BD) if it is not present in the list of low-level hypotheses, because, as far as the generator is concerned, this line does not exist (and cannot be constructed).

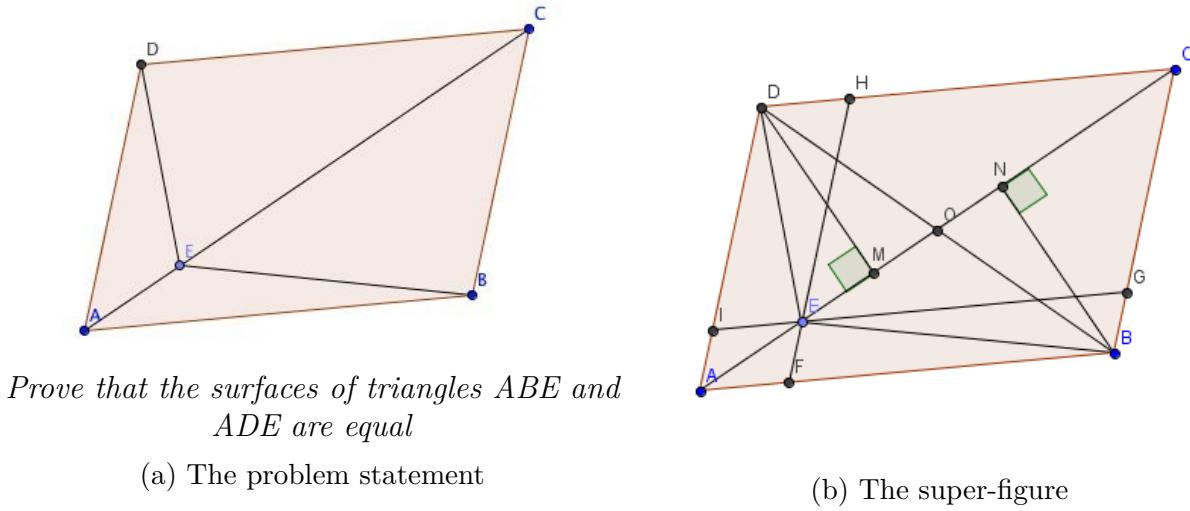


Figure 7.4 The parallelogram problem

In the light of this requirement, we defined the super-figure of a problem as the figure containing all geometrical elements that could be used in any proof. A requirement to add a problem in QED-Tutrix is to provide such a figure in the form of a GeoGebra file. Also, this file can easily be altered later, if new geometrical elements are needed for additional solutions. The super-figure of the parallelogram problem is given in Figure 7.4b. It is important to note that this figure is necessary, but not sufficient, since the generator also needs to know which angles could be useful to solve the problem. This last point is a technical constraint, since automatically constructing all the possible angles would be possible, but costly. Therefore, although a requirement for the time being, work is currently being done to automate this process.

With this set of high-level and low-level hypotheses, the generator is able to infer new results. To do so, it has access to a list of mathematical properties, encoded in Prolog. By combining hypotheses and properties, we obtain a set of new results, that are added to the hypotheses for the next iteration.

This process is repeated until no new results are found. Given that the set of geometrical objects is finite, and that the number of allowed properties is also finite, we can ensure that the process will end at one point. The output is a set of inferences that form a HPDIC graph. A conversion module is then responsible for the encoding into the format accepted by QED-Tutrix. We will not detail the conversion process, since it is simply a question of different encodings, the content remains the same.

This automatization of the process of writing down the HPDIC graph created several issues

that we had to address. In the remainder of this paper, we explain those issues, as well as the solutions we chose.

## 7.6 The many names of geometrical objects

One of the first problems encountered is the question of the naming of geometrical objects. To illustrate this point, let us take the example of the line. In the previous version of QED-Tutrix, where the HPDIC graphs were created manually by a mathematics education expert, the lines were named by a single lower-case letter. It quickly became clear that this system was not appropriate for all the use-cases. Indeed, in many cases, the line is not named with a single letter, but with the name of two points present on it. When the line contains more than two points, any combination of those is a valid name. In Figure 7.5a, the line through A, B and C can be named : “l”, “(AB)”, “(AC)”, “(BC)”, or even “(BA)”, “(CA)”, or “(CB)”. This does not render many combinations in the case of 3 points, but for 5 points, there are already 20 possible names. To be able to store that information, QED-Tutrix was modified. It, however, creates a constraint for the input of the proof generator, as the problem creator (teacher or programmer) must indicate, for each line, the complete set of points on that line. In essence, the problem creator would have to say “there is a line named l in the problem that contains the points A, B and C”.

The same is true for angles, with an additional issue. Indeed, the angle in Figure 7.5a has many names : “alpha”, “EAB”, “EAC”, “DAB”, or “DAC”. However, the problem of the orientation arises. Even though most teachers won’t correct a student that names the angle as “BAE” instead of “EAB”, those are not the same when considering the orientation, as “EAB” is  $\alpha$ , but “BAE” is the complementary angle of  $\alpha$  (whose value is  $360 - \alpha$ ). In the proof generator, it is crucial to distinguish those two, or else the generator would infer incoherent results, such as having two values ( $\alpha$  and  $360 - \alpha$ ) for the same angle. We therefore chose, for the moment, to impose the orientation in the proof generator, but to allow more flexibility in QED-Tutrix, meaning that, even though the proof generator separates EAB and BAE as two completely different angles, the student can identify the angle EAB by both the names “EAB” and “BAE”. The user that wishes to create a problem must indicate all existing and potentially useful angles with all this information : the name (if it has one), all the points on the first (in trigonometric order) semi-line, the central point, and all the points on the second semi-line.

A quick word about polygons. The one in Figure 7.5b will typically be named “ABCD”. However, choosing a different starting point does not change anything : “BCDA”, “CDAB”, “DABC” are equally valid names. In most cases, reversing the order is also valid : the polygon

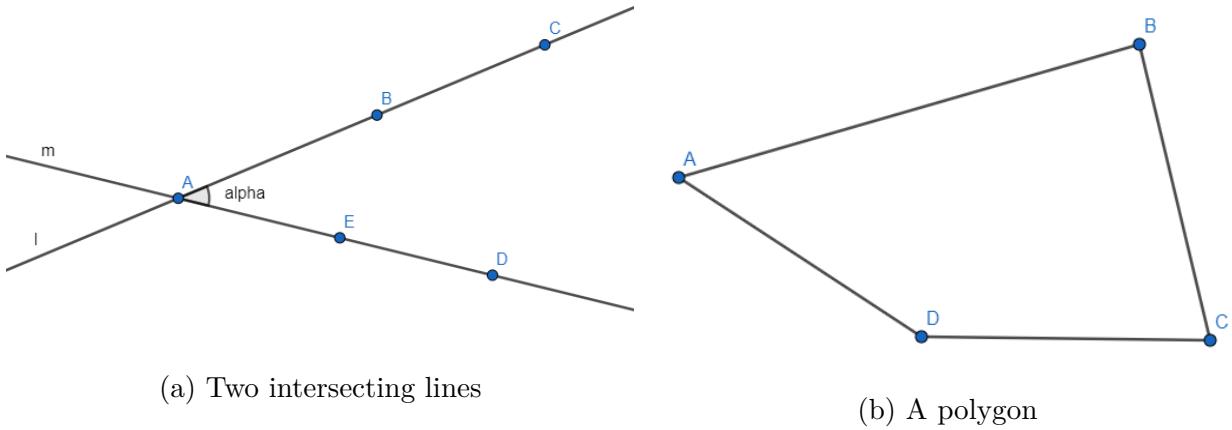


Figure 7.5 Two mathematical situations

could be referred to as “ADCB” by a student. In the proof generator, we want the name to be unique to avoid creating redundant results (such as “ABCD is a rectangle” and “BCDA is a rectangle”), and therefore chose to use an encoding ensuring the unicity of naming. In QED-Tutrix however, any combination of shifting and reversing the points is accepted.

We mentioned previously that lines and angles can have names unrelated to their geometrical properties (for instance, line “m”, with no relation to the points present in the line). This can be true for any geometrical object. Even though it is rare, it could happen that a problem gives a name to a polygon or a segment, and we must be able to handle that case. Therefore, we generalized the possibility of naming objects to each non-point geometrical object.

Lastly, we noticed that, on some complex problem, we would have an issue of point names. Since every point in the figure must have a name (or the proof generator doesn't know it exists), in problems that introduce grids or other complex structures, the limit of 26 letters for points could be reached. Therefore, we allowed QED-Tutrix to use points named with one letter followed by any combination of digits, such as P1, P2, or A256. Points can still be named without a digit, which would be the case in most problems.

## 7.7 The degrees of approximation of a problem

All the issues discussed in the previous section point towards the same global problematic : although in the vast majority of cases, there is no ambiguity, we still must take into account all the possible problems that could be given in a high-school setting, including the ones complex enough to create an ambiguity. For instance, in most problems, the orientation of angles is not relevant, but we must be able to handle the small number of problems where

it is, and therefore must take into account the orientation of angles in all problems. The same can be said for circle arcs. In most problems, such as in Figure 7.6a, there is at most one circle, and no one will correct a student that says “the arc AB” instead of “the arc AB of circle C”. Therefore, in the initial version of QED-Tutrix, arcs were defined by only two points. However, there could be problems with two or more circles, each containing a different arc AB, such as in Figure 7.6b, and in this case, this approximation becomes unacceptable because it creates an important confusion. As a consequence, the generator must account for the circle in every case, even though it could be implicit in most problems, which creates constraints on the sentences the student can input into QED-Tutrix.

This last point illustrates the issue created by the variation of rigor needed to properly state and solve a problem, depending on the content of the problem, on its level in the curriculum, and also on the teacher. Indeed, many problems don’t need a fully rigorous representation of every object, but such a rigorous representation is necessary in the generator. Often, it even becomes a necessity to provide a certain degree of liberty in the statement of the problem : in Figure 7.6a, if A and C is fixed but B can be any point on the line, we cannot explicitly state that the angle alpha is between the semi-lines containing E,D and B,C respectively, since the angle EAB is not the same depending on the position of B (left or right of A).

## 7.8 Level of granularity

We mentioned previously the difficulty created by the different levels of rigor between the work of a high-school student and the low-level proving done by an automated tool. This is only the tip of a much larger issue : the level of granularity of proofs in QED-Tutrix.

We define informally the level of granularity of a proof as follows. First, we define the “high-school axioms”, a set of lowest-level properties, that can always be used without justification regardless of the students’ level. For instance, “a quadrilateral with three right angles is a rectangle” is not a “high-school axiom”, since it can’t be used without justification in early high-school classes, although, depending on teachers, it could be used as such later. However, “if two lines are parallel to a third, then they are parallel to each other” is such an axiom. It is important to note that, in the examples provided in the previous section (such as the obligation to indicate the circle an arc belongs to), the technical constraints are imposing a higher level of granularity.

Then, the level of granularity of a proof represents the gap between the properties used in the actual proof, written by a student (or teacher) attempting to solve the problem, and a fully rigorous proof that uses exclusively the high-school axioms. For instance, in the very

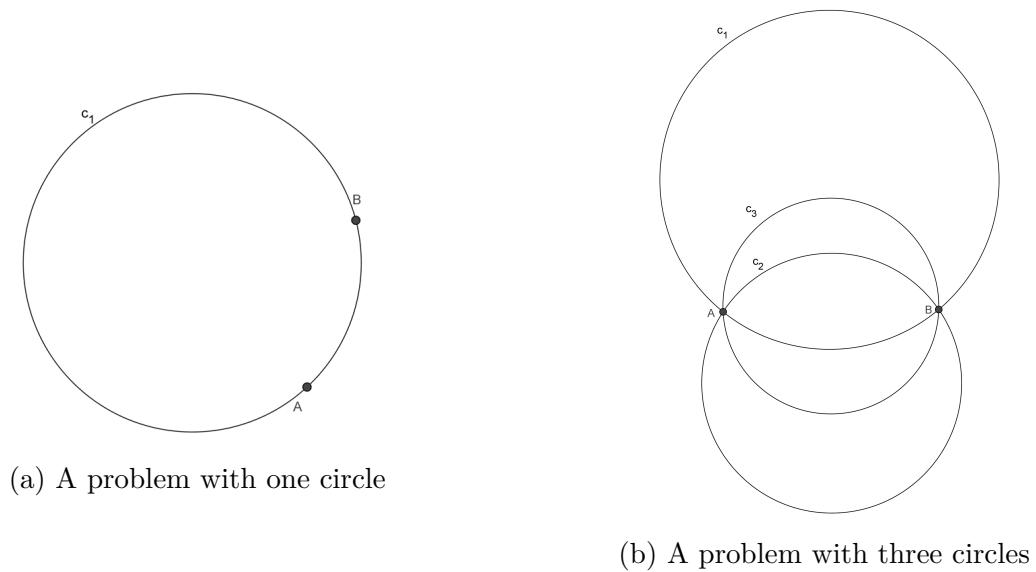
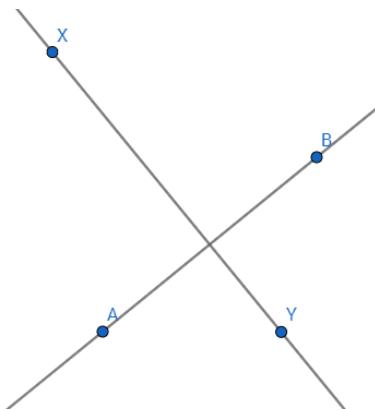


Figure 7.6 Two geometrical situations using circles



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

Figure 7.7 A perpendicular bissector problem.

simple problem given in Figure 7.7, an axiomatic, or high granularity, proof would be :

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. Since Y is equidistant from A and B, Y is on the perpendicular bisector of [AB].
4. Since X and Y are two distinct points, 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].

But a low granularity proof could be :

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].

The HPDIC graphs for both proofs are given in Figure 7.8 and Figure 7.9. We can see in the second proofs that shortcuts have been taken in two ways. The whole red square has disappeared, and the green one has collapsed into a single property. Proofs written by students can contain any number of such shortcuts, and one of the short-terms goals of this project is to understand the way these various levels of granularity are determined and required in class and how they impact the style of proofs the student will produce.

## 7.9 Future work

The automated proof generator is still a work in progress. However, the core is operational, and we have been able to generate the full HPDIC graph for four out of five problems, by implementing 34 out of 46 results and 89 out of 203 previously identified properties, representing a first draft of the set of properties used at a high-school level. The next short-term objective is to implement the remaining relevant properties. One should note that several of the remaining 114 properties are not necessary, since the internal representation of objects in Prolog renders some of them redundant (for instance, “the point A is the intersection of lines L1 and L2” can be bypassed by the simple fact that the point A is in the list of points of both lines L1 and L2).

As mentioned in the introduction, our work on QED-Tutrix is at the moment focused on only five problems, since adding the proofs to new ones was previously extremely time consuming.

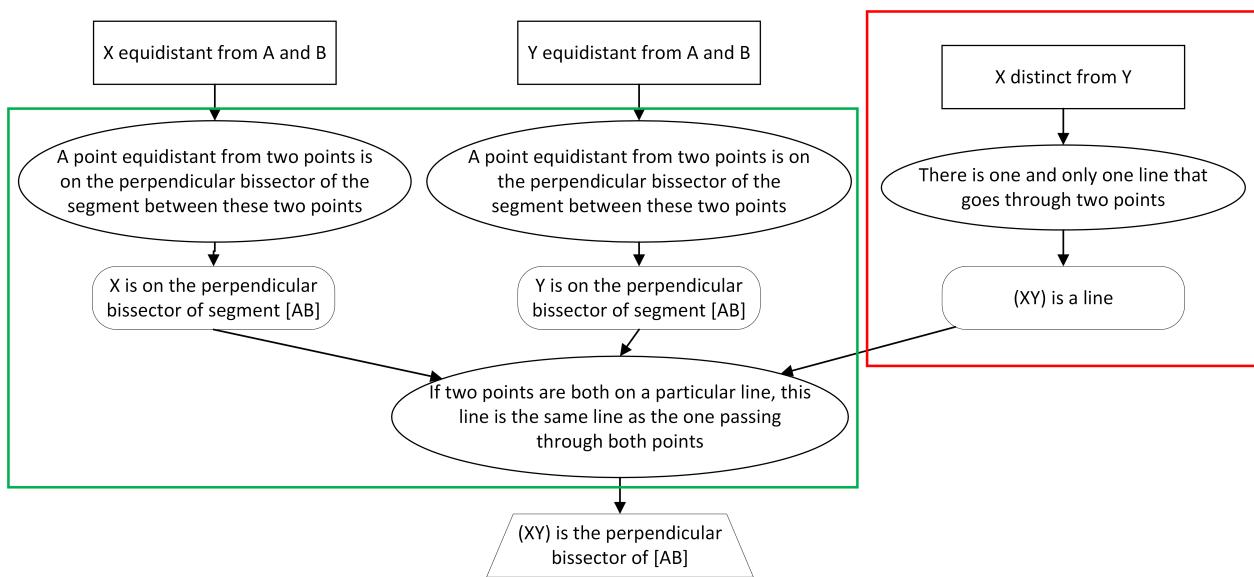


Figure 7.8 The HPDIC graph for a high granularity proof of the perpendicular bisector problem

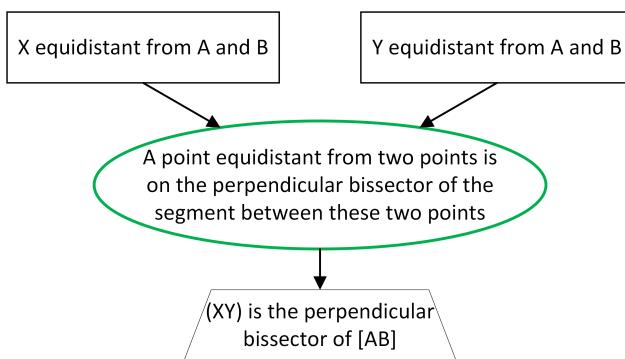


Figure 7.9 The HPDIC graph for a low granularity proof of the perpendicular bisector problem

However, with the help of automated proof generation, the addition of new problems is drastically simplified. Therefore, we have written down 22 additional problem statements. Another short-term task is to identify and implement properties that are useful for those new problems. Then, we plan to explore improvements of the automated proof generator on several aspects.

First, we explained previously that, even though most problems can use a simpler form of geometric objects and properties (for instance, not precising the circle an arc belongs to), the fact that a few cannot use those means that we must use the most general case. A solution we envisioned for that is to create a set of flags to indicate the degree of granularity on several aspects : are there more than one circle (if no, then no need to ask for the circle when using an arc), is the orientation of angles relevant to the problem (if yes, then the software would check the orientation of angles entered by the student), is the orientation of polygons relevant, etc.

Second, the issue of granularity configuration. Indeed, one of the objectives of our team is to allow teachers to configure finely the degree of granularity the student is allowed to use in each particular problem. Currently, the automated proof generator is able to generate a graph encompassing all levels of granularity. The difficulty will be to make the selection process between the levels of granularity for each mathematical topic as easy as possible, in order to make it accessible to teachers unfamiliar with the internal operation of QED-Tutrix.

Third, further from the contents of this paper, is the possibility to semi-automatically extract hypotheses, including the low-level ones, and conclusion from a problem statement and its figure. The need for a manual redaction of all hypotheses is currently one of the weaknesses of our automated proof generator, and the reduction or deletion of the human part of this task would be a major improvement.

Finally, we will address the need to provide a super-figure that, in a way, already contains all possible proofs when encoding a problem. A solution would be to implement a construction tutor that would allow the student to dynamically add new geometrical elements to his or her figure. These new elements would then be given as soon as provided to the automated proof generator, that would then attempt to infer new proofs using these elements.

Concerning future work that are not improvements of the automated proof generator, one of our goals is to validate the model by gathering remarks and opinions from teachers on the generated proofs : are they accessible to a high-school student ? Do the generated proofs really encompass all the possible paths that a student could take ?

## 7.10 Conclusion

In essence, the project of automating the generation of proofs consists in adapting the software, both QED-Tutrix and the associated generator, to the educational requirements, and not the opposite. Still, at the moment, there are some points where it is the student that has to adapt, because of purely technical constraints. Typically, the circle an arc belongs to must be specified at all times, even though, in a classical proof with pen and paper, it would not be necessary. In this paper, we detailed the three main issues encountered while developing the automated proof generator for QED-Tutrix, along with the steps taken or envisioned to solve them. Indeed, although we already determined that the automation of the generation of proofs is the next step for the usability of QED-Tutrix in real situations [4], these limitations remain an important obstacle for the students' learning, and therefore still prevent QED-Tutrix from attaining its full potential as a learning tool.

## CHAPITRE 8 DISCUSSION GÉNÉRALE

Les travaux présentés dans ce mémoire constituent ainsi la seconde phase du développement de QED-Tutrix. Leur portée a été discutée dans les quatre articles, avec un accent sur l'aspect génération de preuves dans les premier et deuxième articles, sur l'aspect didactique dans le quatrième, et sur l'ensemble du projet dans le troisième.

Pour résumer, l'adjonction d'un générateur de preuves automatisé permet d'augmenter drastiquement la portée du logiciel tuteur en automatisant le processus d'écriture des graphes HPDIC, étape qui était auparavant le goulot d'étranglement de l'intégration de nouveaux problèmes. L'apport au domaine de la démonstration automatique de théorèmes est non négligeable. En effet, bien que les méthodes de démonstration synthétiques (i.e. basées sur une succession d'inférence) soient connues et utilisées, ces méthodes sont généralement limitées par de faibles performances. La restriction de l'espace de recherche, par l'implémentation des propriétés utilisées au secondaire uniquement, permet d'éviter en grande partie l'explosion combinatoire. Combiné à l'algorithme utilisé, qui limite encore l'espace de recherche aux propriétés directement applicables à partir des résultats déjà inférés, ceci rend notre outil suffisamment rapide pour notre application. Cependant, comme nous l'avons vu avec l'exemple des triangles équilatéraux imbriqués, ces bonnes performances ne sont pas systématiques, et la structure même d'une preuve mathématique se prête à la possibilité d'avoir un ensemble de preuves possibles démesurément large. De plus, cette restriction de l'espace de recherche a évidemment comme conséquence de limiter considérablement le domaine d'application de notre outil. Enfin, cet outil n'a pas été conçu pour entrer en compétition avec les autres systèmes de démonstration. Typiquement, la nécessité de fournir la super-figure du problème impose un cas d'utilisation bien précis limitant son utilisation au projet QED-Tutrix.

D'un point de vue didactique, une conséquence indirecte de ce développement a été l'encodage systématique des concepts mathématiques, propriétés et définitions appartenant au programme du secondaire dans un format informatique. Ce format impose ainsi d'explorer en détail les raccourcis naturellement faits en classe (par exemple, une justification dont les conditions d'utilisation ne sont pas toutes vérifiées, alors que cela serait nécessaire dans un système informatique). La mise en avant systématique de ces raccourcis est une source d'information particulièrement intéressante pour les chercheurs en didactique.

## CHAPITRE 9 CONCLUSION

### 9.1 Synthèse des travaux

Nous pouvons ainsi regrouper les travaux effectués en trois axes principaux : le développement informatique du générateur de preuves ; la création et l'encodage du référentiel, contenant les propriétés, définitions et résultats mathématiques pouvant apparaître dans la résolution d'un problème de géométrie au secondaire ; et l'encodage de soixante-quatre problèmes en Prolog, incluant la construction de leur super-figure.

Premièrement, nous avons développé le générateur de preuves automatisé, basé sur un noyau inférentiel en Prolog. Les enjeux didactiques au coeur du projet QED-Tutrix, et les contraintes précises qu'ils engendrent, ont rendu impossible l'utilisation des outils standards en démonstration automatique de théorèmes. Par conséquent, la création de ce noyau inférentiel est un travail tout à fait inédit dans le domaine, non pas dans le principe (l'utilisation de chaînes d'inférences est une approche connue, bien que moins répandue que sa contrepartie algébrique), mais dans son utilisation pour représenter une situation d'enseignement réelle. Chaque étape du développement de ce générateur a été construite en prenant en compte son ultime utilité pour l'enseignement.

Notre générateur n'est cependant pas uniquement composé d'un programme Python permettant de chaîner des inférences dans un référentiel donné. Ce noyau est appelé par un programme Python, chargé :

- de l'interprétation du fichier contenant l'encodage du problème,
- du stockage et exploration de l'ensemble des inférences retournées par le noyau,
- du nettoyage de cet ensemble d'inférences pour éliminer celles n'étant pas utiles pour la résolution du problème,
- de la génération du graphe HPDIC lorsque toutes les inférences possibles ont été identifiées.

Le processus de génération de preuve, présenté dans le deuxième et troisième articles (chapitres 5 et 6), se résume ainsi. À l'initialisation, le problème, encodé comme un ensemble d'hypothèses, est chargé en mémoire. Ensuite, de façon itérative, le logiciel parcourt tous les résultats connus (les hypothèses sont informatiquement considérées comme des résultats), et, pour chacun, “demande” à Prolog ce que l'on peut inférer à partir de celui-ci dans le référentiel fourni. Tous les nouveaux résultats obtenus sont ajoutés à l'ensemble des résultats connus, et les inférences ayant permis de les obtenir sont ajoutées au graphe HPDIC. Cette itération se prolonge jusqu'à épuisement des résultats connus non explorés. Par la suite, le

graphe est nettoyé, en éliminant les inférences ne permettant pas d'aboutir à la conclusion. Enfin, le graphe HPDIC nettoyé est imprimé dans un fichier, qui pourra être fourni à QED-Tutrix. Ainsi, le générateur de preuves prend en entrée un problème et un référentiel, et renvoie en sortie le graphe HPDIC de ce problème.

Pour son bon fonctionnement, il a donc été nécessaire d'implémenter le référentiel de la géométrie de secondaire dans un format informatique. Celui-ci est composé d'un ensemble de patrons de résultats mathématiques possibles, chacun adjoint d'un ensemble de paramètres (par exemple, "deux droites sont parallèles", prenant deux droites comme paramètres), et de justifications, chacune permettant de passer d'un ensemble d'antécédents à un conséquent, ceux-ci suivant l'un des patrons. Pour ce travail, la difficulté principale a été la formalisation des justifications, en mariant le besoin informatique d'une définition rigoureuse de la situation dans laquelle cette justification s'applique, et la possibilité de l'utiliser de façon incomplète mais acceptable dans un contexte d'enseignement. Dans l'exemple précédent, le bon fonctionnement de cette simple règle exige que les droites soient distinctes, mais cette vérification est très rarement demandée en classe. Il a par conséquent été nécessaire de construire le logiciel de façon à vérifier implicitement que les deux droites sont distinctes, mais sans exiger que l'élève utilisant QED-Tutrix ne doive fournir cet antécédent. Ce travail a donc été à la fois un défi technique et didactique. Nous avons au total implémenté 183 règles d'inférences Prolog, chacune représentant un type d'inférence possible, constitué d'antécédents, d'une justification, et d'un conséquent. Ces 183 règles implémentent 113 propriétés, définitions, fournies en annexe A. Les nombres diffèrent car il est souvent nécessaire d'implémenter plusieurs règles pour traiter différents cas d'utilisation d'une même propriété ou définition.

Enfin, nous avons encodé quatre des cinq problèmes disponibles initialement dans QED-Tutrix, le cinquième ayant été mis de côté de par la difficulté d'en représenter la conclusion dans un mode inférentiel. De plus, nous avons également encodé les soixante problèmes fournis par nos experts en didactique. Par "encodé", nous entendons non seulement le processus de traduction du problème en soi, mais également la construction de sa super-figure, c'est-à-dire l'ensemble des éléments géométriques tels que des droites, points ou angles (spécifiés ou non dans les hypothèses) pouvant apparaître dans une preuve. Ces soixante problèmes ont servi de méthode d'évaluation. Dix-neuf ont été sélectionnés par nos experts pour aider à l'élaboration du générateur de preuves, les quarante-et-un autres ont été conservés pour l'étape de validation. Une conséquence directe de ce processus de validation est que les graphes HPDIC de ces soixante problèmes ont été générés, et ont donc pu être ajoutés à QED-Tutrix.

En résumé, nous avons automatisé au maximum le processus d'ajout d'un problème à QED-Tutrix, en particulier la phase complexe et démesurément longue, lorsque faite à la main, de

création du graphe HPDIC du problème. Ainsi, à l'aide de cet outil, l'extension de QED-Tutrix par l'ajout de problèmes revient simplement à encoder le problème et sa super-figure en Prolog.

## 9.2 Limitations et améliorations futures

L'outil développé, bien que puissant, a d'importantes limitations. Celles-ci ont été mentionnées dans les articles, en particulier le troisième, chapitre 6. Pour résumer les principales, par ordre croissant d'importance, la première limitation est l'étendue du référentiel implémenté. Certains pans de la géométrie du secondaire n'ont pas encore été implémentés dans la version actuelle. Parmi ceux-ci, nous pouvons mentionner en particulier tout ce qui touche aux polygones au-delà du quadrilatère et la notion de périmètre. Bien que des difficultés non anticipées puissent surgir, nous estimons que la complétion de ces référentiels ne présenterait pas de problème important, et serait donc une des étapes essentielles des travaux futurs.

La deuxième limitation importante est l'absence de moteur algébrique. Bien que le générateur soit tout à fait capable d'effectuer des opérations arithmétiques, il est nécessaire, en l'état actuel, que les valeurs soient connues. Il est par exemple impossible d'inférer  $a = 4b$  à partir de  $a = 2c$  et  $c = 2b$  si la valeur de  $b$  n'est pas initialement connue. L'implémentation de cette simple inférence demanderait le développement d'un moteur algébrique en Prolog, ce qui nécessiterait un travail important. En effet, il faudrait définir explicitement les opérations arithmétiques de base, telles que l'addition et la multiplication, incluant leur propriétés telles que la commutativité, transitivité et priorité. L'ampleur de ce travail comparée au faible nombre de situations où il serait utile a fait que nous avons laissé ce volet dans les améliorations futures.

Enfin, la troisième limitation majeure est la nécessité de fournir la super-figure. En effet, pour ne pas s'engager dans la pente glissante de génération automatisée de nouveaux éléments géométriques lors de la résolution, le générateur de preuves ne peut travailler que sur les éléments ayant été explicitement définis dans le fichier initial. Ainsi, pour pouvoir inférer que la droite  $(AH)$  est la hauteur du triangle  $ABC$ , il faut que la droite  $(AH)$  apparaisse quelque part dans les hypothèses. Le générateur ne sera jamais capable de construire une telle droite. Il est ainsi nécessaire de fournir, dans le fichier du problème, la super-figure, comprenant tous les objets géométriques pouvant servir à la résoudre. Nous avons envisagé plusieurs voies pour pallier à cette limitation. Il serait par exemple possible d'intégrer le générateur de preuves directement à QED-Tutrix, afin de réagir en temps réel aux ajouts faits par l'élève. Pour reprendre l'exemple précédent, si l'élève crée le point  $H$  et la droite  $(AH)$  perpendiculaire à  $(BC)$ , alors un simple appel au générateur de preuves permettrait d'ajouter à la volée

l'information que  $(AH)$  est la hauteur du triangle  $ABC$ , et par la suite à tous les chemins de résolution découlant de ce résultat. Il serait également possible d'explorer les méthodes existantes dans l'état de l'art pour les intégrer au processus de résolution, qui deviendrait alors capable de construire ces nouveaux objets de façon autonome.

Pour conclure cette thèse, les travaux présentés constituent un grand pas en avant dans le développement du logiciel tuteur QED-Tutrix. L'utilisation de la programmation logique est un choix pertinent, rendant le processus de génération automatique de preuves, et par conséquent d'ajout de nouveaux problèmes à QED-Tutrix, rapide et aisément. Ces travaux ont également permis de mettre en avant des points de friction fondamentaux entre la réalité de l'enseignement au secondaire, avec toutes ses subtilités, simplifications, et interactions complexes entre élève et enseignant, et l'utilisation d'un outil informatique, demandant une rigueur absolue et ne disposant pas de la moindre capacité de réflexion autonome. Plusieurs de ces points de friction ont été partiellement résolus dans ce projet, mais la route à parcourir avant de pouvoir concilier intelligence humaine et intelligence artificielle est potentiellement infiniment longue.

## RÉFÉRENCES

- [1] P. R. Richard, F. Venant et M. Gagnon, *Issues and challenges about instrumental proof.* Suisse : Springer, 2019.
- [2] N. Leduc, “Qed-tutrix : système tutoriel intelligent pour l’accompagnement d’élèves en situation de résolution de problèmes de démonstration en géométrie plane,” Thèse de doctorat, École polytechnique de Montréal., 2016.
- [3] M. Tessier-Baillargeon, “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,” Thèse de doctorat, Université de Montréal., 2015.
- [4] L. Font, P. R. Richard et M. Gagnon, “Improving qed-tutrix by automating the generation of proofs,” *arXiv preprint arXiv :1803.01468*, 2018.
- [5] L. Font *et al.*, “Automating the generation of high school geometry proofs using prolog in an educational context,” *arXiv preprint arXiv :2002.12551*, 2020.
- [6] ——, “Intelligence in qed-tutrix : Balancing the interactions between the natural intelligence of the user and the artificial intelligence of the tutor software,” *Mathematics Education in the age of Artificial Intelligence*, 2021.
- [7] ——, “Creation of a mathematical model for qed-tutrix’ automated proof generator,” dans *Proceedings of the Sixth Symposium on Mathematical Work*, 2019, p. 335–356.
- [8] “Mathway | Math Problem Solver.” [En ligne]. Disponible : <https://www.mathway.com/Algebra>
- [9] E. Melis *et al.*, “Activemath—a learning platform with semantic web features,” *The Future of Learning*, p. 159, 2009.
- [10] M. Hohenwarter, “Geogebra 4.4—from desktops to tablets,” *Indagatio Didactica*, vol. 5, n°. 1, 2013.
- [11] D. Py, “Reconnaissance de plan pour la modélisation de l’élève. le projet mentoniezh,” *Recherches en didactique des mathématiques*, vol. 14, n°. 1/2, p. 113–138, 1994.
- [12] ——, “Aide à la démonstration en géométrie : le projet mentoniezh,” *Sciences et techniques éducatives*, vol. 3, n°. 2, p. 227–256, 1996.
- [13] ——, “Environnements interactifs d’apprentissage et démonstration en géométrie,” 2001.
- [14] B. Buchberger, “Applications of gröbner bases in non-linear computational geometry,” dans *Trends in computer algebra.* Springer, 1988, p. 52–80.

- [15] D. Kapur, “Using gröbner bases to reason about geometry problems,” *Journal of Symbolic Computation*, vol. 2, n°. 4, p. 399–408, 1986.
- [16] S.-C. Chou, “An introduction to wu’s method for mechanical theorem proving in geometry,” *Journal of Automated Reasoning*, vol. 4, n°. 3, p. 237–267, 1988.
- [17] H. Wu, “An elementary method in the study of nonnegative curvature,” *Acta Mathematica*, vol. 142, n°. 1, p. 57–78, 1979.
- [18] J. Zhang, L. Yang et M. Deng, “The parallel numerical method of mechanical theorem proving,” *Theoretical Computer Science*, vol. 74, n°. 3, p. 253–271, 1990.
- [19] F. Botana *et al.*, “Automated theorem proving in geogebra : Current achievements,” *Journal of Automated Reasoning*, vol. 55, n°. 1, p. 39–59, 2015.
- [20] P. Boutry, G. Braun et J. Narboux, “From tarski to descartes : formalization of the arithmetization of euclidean geometry,” dans *SCSS 2016, the 7th International Symposium on Symbolic Computation in Software Science*, vol. 39, n°. 14-28. EasyChair, 2016, p. 15.
- [21] R. Manthey et F. Bry, “Satchmo : a theorem prover implemented in prolog,” dans *International Conference on Automated Deduction*. Springer, 1988, p. 415–434.
- [22] M. E. Stickel, “A prolog technology theorem prover : a new exposition and implementation in prolog,” *Theoretical Computer Science*, vol. 104, n°. 1, p. 109–128, 1992.
- [23] A. Kuzniak, “Paradigmes et espaces de travail géométriques.” Thèse de doctorat, Université Paris VII-Denis Diderot, 2003.
- [24] ——, “L’espace de Travail Mathématique et ses genèses,” *Annales de didactique et de sciences cognitives*, vol. 16, p. 9–24, 2011. [En ligne]. Disponible : <https://halshs.archives-ouvertes.fr/halshs-01060043>
- [25] A. Kuzniak et P. R. Richard, “Espacios de trabajo matemático. puntos de vista y perspectivas,” *Revista latinoamericana de investigación en matemática educativa*, vol. 17, n°. 4, 2014.
- [26] G. Brousseau et N. Balacheff, *Théorie des situations didactiques : Didactique des mathématiques 1970-1990*. La pensée sauvage Grenoble, 1998.
- [27] S. Cyr, “Étude des référentiels de géométrie utilisés en classe de mathématiques au secondaire,” Mémoire de maîtrise, Université de Montréal, 2021.
- [28] M. Tessier-Baillargeon *et al.*, “Étude comparative de systèmes tutoriels pour l’exercice de la démonstration en géométrie,” *Annales de didactique et de sciences cognitives*, vol. 22, p. 91–117, 2017.

- [29] P. R. Richard, “L’inférence figurale : un pas de raisonnement discursivo-graphique,” *Educational Studies in Mathematics*, vol. 57, n°. 2, p. 229–263, 2004.
- [30] G. Brousseau, *Theory of didactical situations in mathematics : Didactique des mathématiques, 1970–1990.* Springer Science & Business Media, 2006, vol. 19.
- [31] G. Weber et P. Brusilovsky, “Elm-art : An adaptive versatile system for web-based instruction,” *International Journal of Artificial Intelligence in Education (IJAIED)*, vol. 12, p. 351–384, 2001.
- [32] J.-C. Falmagne *et al.*, “The assessment of knowledge, in theory and in practice,” dans *Formal concept analysis.* Springer, 2006, p. 61–79.
- [33] I. Arroyo *et al.*, “Web-based intelligent multimedia tutoring for high stakes achievement tests,” dans *Intelligent Tutoring Systems.* Springer, 2004, p. 142–169.
- [34] S. Jean-Daubias, “Pépite : un système d’assistance au diagnostic de compétences,” Thèse de doctorat, Université du Maine, 2000.
- [35] V. Aleven et K. R. Koedinger, “Knowledge component (kc) approaches to learner modeling,” *Design Recommendations for Intelligent Tutoring Systems*, vol. 1, p. 165–182, 2013.
- [36] J. R. Anderson, “Act : A simple theory of complex cognition.” *American Psychologist*, vol. 51, n°. 4, p. 355, 1996.
- [37] J. R. Anderson et C. Schunn, “Implications of the act-r learning theory : No magic bullets,” *Advances in instructional psychology, Educational design and cognitive science*, p. 1–33, 2000.
- [38] V. Aleven *et al.*, “Combatting shallow learning in a tutor for geometry problem solving,” dans *Intelligent Tutoring Systems.* Springer, 1998, p. 364–373.
- [39] V. Aleven et K. R. Koedinger, “Limitations of student control : Do students know when they need help ?” dans *Intelligent tutoring systems*, vol. 1839. Springer, 2000, p. 292–303.
- [40] V. A. Aleven et K. R. Koedinger, “An effective metacognitive strategy : Learning by doing and explaining with a computer-based cognitive tutor,” *Cognitive science*, vol. 26, n°. 2, p. 147–179, 2002.
- [41] V. Aleven *et al.*, “Toward meta-cognitive tutoring : A model of help seeking with a cognitive tutor,” *International Journal of Artificial Intelligence in Education*, vol. 16, n°. 2, p. 101–128, 2006.
- [42] I. Roll *et al.*, “On the benefits of seeking (and avoiding) help in online problem-solving environments,” *Journal of the Learning Sciences*, vol. 23, n°. 4, p. 537–560, 2014.

- [43] N. Matsuda et K. VanLehn, “Advanced geometry tutor : An intelligent tutor that teaches proof-writing with construction.” dans *AIED*, vol. 125, 2005, p. 443–450.
- [44] N. Matsuda et K. VanLehn, “Gramy : A geometry theorem prover capable of construction,” *Journal of Automated Reasoning*, vol. 32, n°. 1, p. 3–33, 2004.
- [45] K. Koedinger, “On the design of novel notations and actions to facilitate thinking and learning,” dans *Proceedings of the International Conference on the Learning Sciences*, 1991, p. 266–273.
- [46] K. R. Koedinger et J. R. Anderson, “Effective use of intelligent software in high school math classrooms,” 1993.
- [47] ———, “Abstract planning and perceptual chunks : Elements of expertise in geometry,” *Cognitive Science*, vol. 14, n°. 4, p. 511–550, 1990.
- [48] N. Balacheff *et al.*, “Baghera assessment project, designing an hybrid and emergent educational society,” 2003.
- [49] C. Webber *et al.*, “The baghera project : a multi-agent architecture for human learning,” dans *Workshop-Multi-Agent Architectures for Distributed Learning Environments. In Proceedings International Conference on AI and Education. San Antonio, Texas*, 2001.
- [50] Y. Baulac, “Un micromonde de géométrie, cabri-géomètre,” Thèse de doctorat, Université Joseph-Fourier-Grenoble I, 1990.
- [51] M. Kordaki et A. Mastrogianis, “The potential of multiple-solution tasks in e-learning environments : Exploiting the tools of cabri geometry ii,” dans *E-Learn : World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education*. Association for the Advancement of Computing in Education (AACE), 2006, p. 97–104.
- [52] V. Luengo et N. Balacheff, “Contraintes informatiques et environnements d'apprentissage de la démonstration en mathématiques.” *Sciences et Techniques Educatives*, vol. 5, p. 15–45, 1998.
- [53] V. Luengo, “Cabri-euclide : Un micromonde de preuve intégrant la réfutation,” *These de doctorat, INPG, France*, 1997.
- [54] ———, “Some didactical and epistemological considerations in the design of educational software : the cabri-euclide example,” *International Journal of Computers for Mathematical Learning*, vol. 10, n°. 1, p. 1–29, 2005.
- [55] “Géométrix.” [En ligne]. Disponible : <http://geometrix.free.fr/site/>
- [56] S. El-Khoury *et al.*, “Development of an intelligent tutorial system to enhance students' mathematical competence in problem solving,” dans *E-Learn : World Conference on*

- E-Learning in Corporate, Government, Healthcare, and Higher Education.* Association for the Advancement of Computing in Education (AACE), 2005, p. 2042–2049.
- [57] P. R. Richard et J. M. Fortuny, “Amélioration des compétences argumentatives à l'aide d'un système tutoriel en classe de mathématique au secondaire,” dans *Annales de didactique et de sciences cognitives*, vol. 12, 2007, p. 83–116.
  - [58] P. Janicic, “Gclc-a tool for constructive euclidean geometry and more than that,” dans *ICMS*. Springer, 2006, p. 58–73.
  - [59] S.-C. Chou, X.-S. Gao et J.-Z. Zhang, *Machine proofs in geometry : Automated production of readable proofs for geometry theorems*. World Scientific, 1994, vol. 6.
  - [60] ———, “Automated production of traditional proofs for constructive geometry theorems,” dans *Logic in Computer Science, 1993. LICS'93., Proceedings of Eighth Annual IEEE Symposium on*. IEEE, 1993, p. 48–56.
  - [61] G. Braun et J. Narboux, “A synthetic proof of pappus’ theorem in tarski’s geometry,” *Journal of Automated Reasoning*, vol. 58, n°. 2, p. 209–230, 2017.
  - [62] J. Narboux, “Mechanical theorem proving in tarski’s geometry,” dans *International Workshop on Automated Deduction in Geometry*. Springer, 2006, p. 139–156.
  - [63] “Geogebra api.” [En ligne]. Disponible : <http://dev.geogebra.org/trac/wiki/WikiStart>
  - [64] P. Richard, M. Gagnon et J. M. Fortuny, “Connectedness of problems and impass resolution in the solving process in geometry : a major educational challenge,” dans *International Perspectives on the Teaching and Learning of Geometry in Secondary Schools*, P. Herbst *et al.*, édit. Springer, 2018, p. 311–327.
  - [65] H. Gelernter, J. R. Hansen et D. W. Loveland, “Empirical explorations of the geometry theorem machine,” dans *Papers presented at the May 3-5, 1960, western joint IRE-AIEE-ACM computer conference*. ACM, 1960, p. 143–149.
  - [66] A. J. Nevins, “Plane geometry theorem proving using forward chaining,” *Artificial Intelligence*, vol. 6, n°. 1, p. 1–23, 1975.
  - [67] E. Elcock, “Representation of knowledge in geometry machine,” *Machine Intelligence*, vol. 8, p. 11–29, 1977.
  - [68] J. G. Greeno, “Constructions in geometry problem solving.” 1979.
  - [69] H. Coelho et L. M. Pereira, “Automated reasoning in geometry theorem proving with prolog,” *Journal of Automated Reasoning*, vol. 2, n°. 4, p. 329–390, 1986.
  - [70] P. R. Richard *et al.*, “Didactic and theoretical-based perspectives in the experimental development of an intelligent tutorial system for the learning of geometry,” *ZDM*, vol. 43, n°. 3, p. 425–439, 2011.

- [71] P. Rabardel, *Les hommes et les technologies ; approche cognitive des instruments contemporains*. Armand Colin, 1995.
- [72] R. Duval, *Sémiosis et pensée humaine : registres sémiotiques et apprentissages intellectuels*. Peter Lang Berne, 1995.
- [73] S.-C. Chou, X.-S. Gao et J.-Z. Zhang, “Automated generation of readable proofs with geometric invariants, I. multiple and shortest proof generation,” *Journal of Automated Reasoning*, vol. 17, p. 325–347, 1996.
- [74] P. Janičić, J. Narboux et P. Quaresma, “The Area Method : a recapitulation,” *Journal of Automated Reasoning*, vol. 48, n°. 4, p. 489–532, 2012.
- [75] K. Wang et Z. Su, “Automated geometry theorem proving for human-readable proofs,” dans *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.
- [76] C. Houdement et A. Kuzniak, “Paradigmes géométriques et enseignement de la géométrie,” dans *Annales de didactique et de sciences cognitives*, vol. 11, 2006, p. 175–193.
- [77] M. Niss et T. Højgaard, “Mathematical competencies revisited,” *Educational Studies in Mathematics*, vol. 102, n°. 1, p. 9–28, 2019.
- [78] F. S. Tanswell et C. J. Rittberg, “Epistemic injustice in mathematics education,” *ZDM*, vol. 52, n°. 6, p. 1199–1210, 2020.
- [79] S. Coutat, C. Laborde et P. R. Richard, “L’apprentissage instrumenté de propriétés en géométrie : propédeutique à l’acquisition d’une compétence de démonstration,” *Educational Studies in Mathematics*, vol. 93, n°. 2, p. 195–221, 2016.
- [80] L. Trouche, “Construction et conduite des instruments dans les apprentissages mathématiques : nécessité des orchestrations,” 2003.
- [81] S. A. Papert, *Mindstorms : Children, computers, and powerful ideas*. Basic books, 2020.
- [82] S. Gousseau Coutat et P. Richard, “Les figures dynamiques dans un espace de travail mathématique pour l’apprentissage des propriétés géométriques,” dans *Annales de didactique et de sciences cognitives*, vol. 16, 2011, p. 97–126.
- [83] P. R. Richard, M. Gagnon et J. M. Fortuny, “Connectedness of problems and impasse resolution in the solving process in geometry : A major educational challenge,” dans *International perspectives on the teaching and learning of geometry in secondary schools*. Springer, 2018, p. 357–375.
- [84] J.-P. Corbeil, M. Gagnon et P. R. Richard, “Probabilistic approaches to detect blocking states in intelligent tutoring system,” dans *International Conference on Intelligent Tutoring Systems*. Springer, 2020, p. 79–88.

- [85] O. Farid, “Extraction des connaissances en géométrie plane à partir d’énoncés de problèmes,” Thèse de doctorat, Polytechnique Montréal, 2020.
- [86] M. Tessier-Baillargeon *et al.*, “Conception et analyse de geogebratutor, un système tutoriel intelligent : genèse d’un espace de travail géométrique idoine,” *Revista latinoamericana de investigación en matemática educativa*, vol. 17, n°. 4, p. 303–326, 2014.

## ANNEXE A PROPRIÉTÉS IMPLÉMENTÉES

Cette annexe contient la liste des 113 propriétés et définitions implémentées en Prolog.

- Dans deux triangles congrus, les angles et les côtés homologues sont congrus.
- Dans un cercle, deux angles au centre congrus interceptent deux arcs congrus.
- Dans un cercle, deux angles inscrits congrus interceptent deux arcs congrus.
- Dans un cercle, deux arcs de même mesure sont sous-tendus par deux cordes de même mesures.
- Dans un cercle, l'angle au centre et l'arc intercepté par (compris entre) les côtés de l'angle ont la même mesure en degrés.
- Dans un cercle, l'arc intercepté par (compris entre) les côtés de l'angle inscrit mesure le double de ce dernier.
- Dans un cercle, si deux arcs ont la même mesure, alors les angles inscrits les interceptant sont congrus
- Dans un repère orthonormé, l'équation d'une droite passant par deux points A(X1,Y1) et B(X2,Y2) peut s'écrire  $y = a * x + b$ , où  $a = (y2-Y1)/(x2-x1)$  et  $b = (x2*y1 - x1*y2) / (x2-x1)$ .
- Dans un triangle rectangle, la tangente d'un angle aigu est le rapport du côté opposé à l'angle aigu donné sur la longueur du côté adjacent à cet angle.
- Des angles adjacents dont les côtés extérieurs sont en ligne droite sont supplémentaires.
- Des angles adjacents dont les côtés extérieurs sont perpendiculaires sont complémentaires.
- Des angles alternes-internes sont congrus si les deux droites coupées par la sécante formant ces angles sont parallèles.
- Des angles alternes-internes sont des angles formés par deux droites coupées par une sécante si ceux-ci sont situés de part et d'autre de cette sécante, entre les deux droites et ne sont pas adjacents.
- Des angles correspondants sont congrus si les deux droites coupées par la sécante formant ces angles sont parallèles.
- Des angles correspondants sont des angles formés par deux droites coupées par une sécante si ceux-ci sont situés du même côté de cette sécante, un à l'extérieur et l'autre à l'intérieur des deux droites et ne sont pas adjacents.
- Des angles opposés par le sommet sont les angles non adjacents formés par deux droites sécantes.

- Des sécantes coupées par des parallèles sont partagées en segments de longueurs proportionnelles.
- Deux angles congrus et complémentaires mesurent 45 degrés chacun.
- Deux angles congrus sont de même mesure.
- Deux angles sont adjacents s'ils ont le même sommet, un côté commun et s'ils sont situés de part et d'autre du côté commun.
- Deux angles sont complémentaires si la somme de leurs mesures est 90 degrés.
- Deux angles sont congrus s'ils sont de même mesure.
- Deux angles sont supplémentaires si la somme de leurs mesures est 180 degrés.
- Deux arcs congrus sont de même mesure.
- Deux arcs sont congrus s'ils sont de même mesure
- Deux droites parallèles à une troisième sont parallèles entre elles
- Deux droites perpendiculaires à une même troisième sont parallèles entre elles
- Deux droites sont perpendiculaires lorsqu'elles se coupent à angle droit.
- Deux figures semblables ont la même forme.
- Deux points A et B sont symétriques par rapport à une droite  $d$  lorsque  $d$  est la médiatrice du segment  $[AB]$ .
- Deux segments congrus sont de même mesure.
- Deux triangles ayant des bases et des hauteurs homologues congrues ont la même aire.
- Deux triangles ayant trois côtés congrus sont nécessairement congrus (cas d'isométrie CCC).
- Deux triangles congrus ont des hauteurs congrues.
- Deux triangles congrus ont la même aire.
- Deux triangles qui ont deux angles homologues congrus sont semblables (cas de similitude AA).
- Deux triangles qui ont un angle congru compris entre des côtés homologues congrus sont congrus (cas d'isométrie CAC).
- Deux triangles qui ont un côté congru compris entre des angles homologues congrus sont congrus (cas d'isométrie ACA).
- Deux triangles rectangles ayant des cathètes homologues congrues sont congrus (cas d'isométrie CAC).
- Deux triangles semblables qui ont aussi une paire de côtés homologues congrus sont congrus.
- L'égalité est transitive, c'est-à dire si  $A=B$  et  $B=C$ , alors  $A=C$ .
- La bissectrice d'un angle est une demi-droite issue du sommet qui partage l'angle en deux angles adjacents et congrus.

- La congruence de segments est transitive, c'est-à-dire si le segment AB est congru à CD et CD est congru à EF, alors AB est congru à EF.
- La congruence des angles est transitive, c'est-à-dire si l'angle a est congru à b et b est congru à c, alors a est congru à c.
- La demi-droite passant par le point de rencontre de deux bissectrices et un sommet est la bissectrice du troisième angle.
- La droite passant par le point de rencontre de deux médiatrices et un sommet est aussi une médiatrice.
- La mesure d'un angle obtenu par l'union d'angles adjacents est égale à la somme des mesures de ces angles.
- La médiane coupe tout segment parallèle au côté où elle tombe et compris entre les côtés du triangle en leur point milieu.
- La médiane d'un triangle est un segment qui relie un sommet d'un triangle et le milieu du côté opposé.
- La médiatrice d'un segment est la droite perpendiculaire à ce segment et passant par son milieu.
- La somme des amplitudes des angles intérieurs d'un quadrilatère est égale à 360 degrés.
- La somme des amplitudes des angles intérieurs d'un triangle est égale à 180 degrés.
- La symétrie axiale conserve les longueurs.
- Le carré de la longueur de l'hypoténuse d'un triangle rectangle est égal à la somme des carrés des longueurs des deux cathètes.
- Le milieu d'un segment est le point qui appartient à ce segment et qui est équidistant de ses extrémités.
- Le segment passant par le point de rencontre de deux médianes et un sommet est aussi une médiane.
- Les angles consécutifs d'un parallélogramme sont supplémentaires.
- Les angles opposés d'un parallélogramme sont congrus.
- Les angles opposés d'un parallélogramme sont la paire d'angles qui ne partagent pas de côtés du parallélogramme.
- Les angles opposés par le sommet sont toujours congrus.
- Les angles à la base d'un triangle isocèle sont congrus. (le triangle isocèle a deux angles congrus).
- Les complémentaires de deux angles congrus sont congrus.
- Les côtés opposés d'un parallélogramme sont de même longueur.
- Les diagonales d'un parallélogramme se coupent en leurs milieux.
- Les médiatrices d'un triangle sont concourantes en un point qui est le centre du cercle

circonscrit.

- Les trois angles intérieurs d'un triangle équilatéral sont congrus et mesure chacun 60 degrés.
- Les trois hauteurs d'un triangle sont concourantes en un point appelé orthocentre.
- Les trois médianes d'un triangle sont concourantes en un point appelé centre de gravité du triangle.
- Pour deux points  $A(x_1,y_1)$  et  $B(x_2,y_2)$  la longueur du segment  $AB$  est la racine de la somme des carrés de  $x_2-x_1$  et  $y_2-y_1$ .
- Pour un cercle donné, le diamètre  $d$  mesure le double de la mesure du rayon  $r$ .
- Si  $AI = IB = AB/2$  alors  $I$  est le milieu de  $[AB]$ .
- Si  $I$  est le milieu de  $[AB]$  alors  $AI = IB = AB/2$ .
- Si des angles alternes-internes sont congrus, alors les deux droites coupées par la sécante formant ces angles sont parallèles.
- Si des angles correspondants sont congrus, alors les deux droites coupées par la sécante formant ces angles sont parallèles.
- Si deux droites sont parallèles, alors toute perpendiculaire à l'une est perpendiculaire à l'autre.
- Si deux points distincts sont équidistants des extrémités d'un segment, alors la droite passant par ces points est la médiatrice de ce segment.
- Si deux triangles ont une même aire et des bases congrues, alors les hauteurs associées à ces bases sont elles aussi congrues.
- Si le point  $P$  appartient au cercle de centre  $O$  et de rayon  $r$ , alors  $OP = r$ .
- Si on ajoute ou enlève une même quantité à deux quantités égales, on obtient encore deux nombres réels égaux.
- Si, pour deux angles aigus homologues alpha et beta provenant de deux triangles rectangles, les rapports trigonométriques sin, cos et tan (tg) sont égaux, alors alpha et beta sont congrus
- Tous les diamètres du même cercle sont congrus.
- Tous les rayons du même cercle sont congrus.
- Tout diamètre d'un cercle coupe ce dernier en deux arcs de 180 degrés chacun.
- Tout point de la médiatrice d'un segment est équidistant des extrémités de ce segment.
- Tout triangle dont deux angles intérieurs sont congrus est isocèle.
- Tout triangle dont deux angles sont congrus et mesurent 45 degrés chacun est isocèle-rectangle.
- Tout triangle inscrit dont deux de ses sommets coïncident avec les extrémités d'un diamètre d'un cercle est un triangle rectangle.

- Tout triangle où pour au moins un sommet, les segments remarquables (médiatrice, bissectrice, médiane, hauteur) sont confondus est un triangle isocèle.
- Un carré possède toutes les propriétés du rectangle et du losange.
- Un cercle circonscrit au triangle passe par ses sommets et son centre est le point de rencontre des 3 médiatrices.
- Un diamètre de cercle est toute corde qui passe par le centre d'un cercle.
- Un losange est un quadrilatère dont les quatre côtés sont de même longueur.
- Un parallélogramme ayant deux côtés consécutifs de même longueur est un losange.
- Un parallélogramme ayant un angle droit est un rectangle.
- Un parallélogramme dont les diagonales sont de même longueur est un rectangle.
- Un parallélogramme dont les diagonales sont perpendiculaires est un losange.
- Un parallélogramme est un quadrilatère dont les côtés opposés sont parallèles.
- Un quadrilatère ayant les côtés opposés congrus est un parallélogramme.
- Un quadrilatère ayant une paire de côtés opposés parallèles et congrus est un parallélogramme.
- Un quadrilatère dont les angles opposés sont congrus est un parallélogramme.
- Un quadrilatère dont les côtés opposés sont congrus est un parallélogramme.
- Un quadrilatère dont les diagonales se coupent en leurs milieux est un parallélogramme.
- Un quadrilatère qui possède toutes les propriétés du losange et du rectangle est un carré.
- Un rayon de cercle est tout segment qui a pour extrémité le centre O et un point du cercle.
- Un rectangle est un quadrilatère ayant quatre angles droits.
- Un triangle isocèle est un triangle ayant deux côtés congrus.
- Un triangle isocèle-rectangle est un triangle rectangle dont les deux cathètes sont de même mesure. On peut aussi dire, un triangle isocèle avec un angle droit.
- Un triangle qui est isocèle et rectangle est dit isocèle-rectangle.
- Un triangle rectangle est un triangle ayant un angle droit.
- Un triangle équilatérale est un triangle ayant trois côtés congrus.
- Une diagonale de polygone est un segment qui relie deux sommets non consécutifs.
- Une hauteur d'un triangle est une droite passant par un sommet et perpendiculaire au côté opposé. Une hauteur peut aussi être le segment porté par cette droite et compris entre le sommet et le côté opposé.
- Une médiane d'un triangle coupe ce triangle en deux triangles d'aires égales.

## ANNEXE B UNE INFÉRENCE ENCODÉE EN PROLOG

Considérons une inférence très simple : “Chaque angle d'un triangle équilatéral mesure 60 degrés.” Il s'agit tout simplement de la réciproque de la définition du triangle équilatéral (ou, plus exactement, de la définition du triangle équiangle combiné avec la propriété d'équivalence entre équilatéral et équiangle, mais ces deux concepts sont souvent confondus). Cette inférence s'encode sous la forme de la règle Prolog suivante.

```

newInference(angleValue(angle(Pb,A,Pc),value(60)),
  'equiAngVal',
  [equilateral(triangle(A,B,C))]):-  
  

  equilateral(triangle(A,B,C)),
  angle(Pb,A,Pc),
  in(C,Pc),
  in(B,Pb).

```

Toutes les règles Prolog suivent le même format. Les trois premières lignes constituent le “patron” de la règle. La première ligne représente le conséquent de l'inférence : l'angle nommé *angle(Pb, A, Pc)* mesure 60 degrés. Il est important de noter qu'en Prolog, tout élément commençant par une majuscule est une variable, ici Pb, A et Pc. La seconde contient un code unique permettant, par l'utilisation d'un dictionnaire, de faire le lien avec différents énoncés en langue naturelle pour cette propriété, telles que celui donné précédemment, “Chaque angle d'un triangle équilatéral mesure 60 degrés.” La troisième, enfin, contient la liste des antécédents nécessaires à l'application de cette inférence, ici constituée d'un seul élément, le fait que le triangle nommé *triangle(A,B,C)* soit équilatéral.

Par la suite, le corps de la règle énonce les conditions nécessaires pour que cette règle s'applique. Ces conditions sont différentes de la liste des antécédents. En effet, la liste des antécédents contenue sur la troisième ligne n'est absolument pas utilisée par Prolog, mais sert uniquement par la suite pour construire la structure du graphe HPDIC. En revanche, le corps de la règle, lui, est utilisé par Prolog. Sa première ligne indique qu'il doit exister un triangle équilatéral, dont les extrémités sont rangées dans les variables A, B et C. La deuxième indique qu'il doit exister un angle, que le générateur connaît, dont le point central est nommé B (ce B devant correspondre au deuxième sommet du triangle), et dont les deux demi-droites sont nommées Pb et Pc. Enfin, les troisième et quatrième lignes indiquent que le point C

utilisé dans la première ligne doit faire partie de la demi-droite  $Pc$ , et de même pour  $B$  et la demi-droite  $Pb$ .

En d'autres termes, ces conditions peuvent se formuler en langue naturelle : s'il existe un triangle équilatéral, et un angle entre deux de ses côtés, alors cet angle mesure 60 degrés.

Le lecteur attentif aura sans doute remarqué que cette propriété ne donne de résultat que sur le deuxième angle du triangle. Il aurait été possible de tripler cette règle pour les trois angles, mais cette solution n'est pas viable sur le long terme lorsque l'on considère l'ensemble des règles à implémenter. À la place, la magie réside dans le fait Prolog *equilateral(triangle(A,B,C))*. En effet, ce fait n'est pas la conséquence directe d'une inférence Prolog au format que nous venons de voir. À la place, il est inféré par une autre règle intermédiaire :

```
equilateral(triangle(A,B,C)):-  
    inference(equilateral(triangle(A,B,C)),_,_);  
    inference(equilateral(triangle(A,C,B)),_,_);  
    inference(equilateral(triangle(B,A,C)),_,_);  
    inference(equilateral(triangle(B,C,A)),_,_);  
    inference(equilateral(triangle(C,A,B)),_,_);  
    inference(equilateral(triangle(C,B,A)),_,_).
```

Cette deuxième règle encode toutes les permutations possibles du nom du triangle. Ainsi, lorsque Prolog obtient le résultat que *triangle(q,e,d)* (en minuscules pour ne pas confondre avec les variables) est équilatéral, alors, par cette seconde règle, l'on obtient les faits *equilateral(triangle(q,e,d))*, *equilateral(triangle(e,d,q))* et toutes leurs permutations. Combinant cela à la règle d'inférence initiale, nous allons ensuite pouvoir inférer que les angles *qed* et *edq* mesurent tous deux 60 degrés, à condition que ceux-ci existent dans le problème.

Enfin, concernant la différence entre les faits *inference* et *newInference*, ceci est nécessaire pour éviter que Prolog n'entre dans une boucle infinie : pour inférer que le triangle est équilatéral, il faut que ses angles mesurent 60 degrés, mais pour inférer qu'un angle mesure 60 degrés, il faut (parmi d'autres possibilités) qu'un triangle formé par cet angle soit équilatéral, ce qui n'est possible que si l'angle mesure 60 degrés... et ainsi de suite. À la place, les inférences déjà existantes sont nommées *inference*, et, à chaque intégration, les nouveaux faits, nommés *newInference*, ne sont transformés en *inferences* qu'une fois l'itération terminée.

## ANNEXE C    ENCODAGE D'UN PROBLÈME EN PROLOG

Reprends le problème présenté dans l'introduction, dont la figure est fournie en C.1). Celui-ci s'énonce de la façon suivante : “Démontrer que le triangle  $DEF$  formé par les milieux des côtés du triangle équilatéral  $ABC$  est lui aussi équilatéral.”

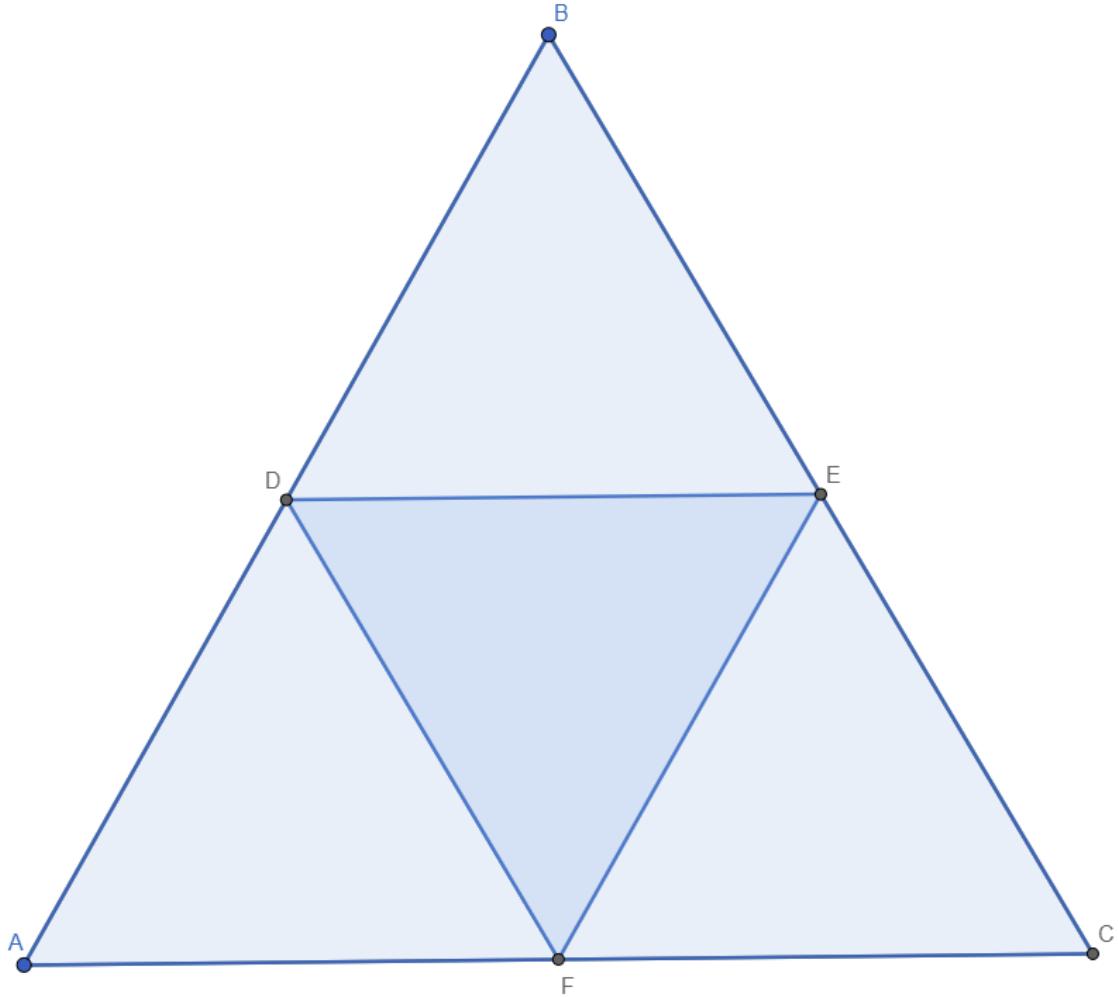


Figure C.1 Un triangle équilatéral et le triangle formé par les milieux de ses côtés

Le code Prolog représentant l'encodage de ce problème, et qui est fourni en entrée au générateur de preuves, est le suivant.

```
:consult('Properties.pl').
hypothese(equilateral(triangle(a,b,c))).
```

```
hypothesize(line([a,b,d])).  
hypothesize(line([a,c,f])).  
hypothesize(line([c,b,e])).  
hypothesize(line([d,e])).  
hypothesize(line([d,f])).  
hypothesize(line([e,f])).  
  
hypothesize(midpoint(d,segment(a,b))).  
hypothesize(midpoint(e,segment(b,c))).  
hypothesize(midpoint(f,segment(a,c))).  
  
hypothesize(point(a)).  
hypothesize(point(b)).  
hypothesize(point(c)).  
hypothesize(point(d)).  
hypothesize(point(e)).  
hypothesize(point(f)).  
  
hypothesize(segment(a,b)).  
hypothesize(segment(a,c)).  
hypothesize(segment(a,d)).  
hypothesize(segment(a,f)).  
hypothesize(segment(b,c)).  
hypothesize(segment(b,e)).  
hypothesize(segment(d,b)).  
hypothesize(segment(d,e)).  
hypothesize(segment(d,f)).  
hypothesize(segment(e,c)).  
hypothesize(segment(e,f)).  
hypothesize(segment(f,c)).  
  
hypothesize(triangle(a,b,c)).  
hypothesize(triangle(d,e,f)).  
  
conclusion(equilateral(triangle(d,e,f))).
```

```

usefulAngle([c,f],a,[e,b]).  

usefulAngle([a],d,[g]).  

usefulAngle([d],f,[a]).  
  

usefulAngle([a,d],b,[c,e]).  

usefulAngle([b],e,[d]).  

usefulAngle([e],d,[b]).  
  

usefulAngle([b,e],c,[f,a]).  

usefulAngle([c],f,[e]).  

usefulAngle([f],e,[c]).  
  

usefulAngle([f],d,[e]).  

usefulAngle([d],e,[f]).  

usefulAngle([e],f,[d]).
```

Cet ensemble de faits Prolog donne la situation de départ, avec l'ensemble des segments, points, droites et triangles existants, ainsi que les hypothèses de départ (points milieux, et équilatéralité du triangle  $ABC$ ). Notons également que la conclusion à atteindre,  $DEF$  est équilatéral, est également présente. Ce fait n'est pas utilisé pour la génération, mais est utile par la suite pour ne conserver dans le graphe HPDIC final que les inférences faisant partie d'une chaîne permettant d'atteindre cette conclusion. Enfin, il est nécessaire de spécifier quels angles le moteur inférentiel a le droit d'utiliser afin d'éviter, la plupart du temps, l'explosion combinatoire du nombre d'angles possibles.

Une étape de résolution serait de commencer par choisir une hypothèse, telle que la première ( $ABC$  est équilatéral), et de demander à Prolog ce qu'il est capable d'inférer à partir de ce fait. En combinant ceci avec l'information que l'angle  $\widehat{ABC}$  existe dans le cadre de ce problème (par le biais du fait `usefulAngle([a,d],b,[c,e])`), Prolog va répondre, entre autres, que  $\widehat{ABC} = 60$ . Ce nouveau fait rejoint ensuite la liste des faits connus. Cette liste des faits connus continue à être explorée jusqu'à ce qu'il ne soit plus possible d'obtenir aucun nouveau résultat.