

Étude et réalisation d'un système tuteur pour la construction de figures géométriques

Cyrille Desmoulins

▶ **To cite this version:**

Cyrille Desmoulins. Étude et réalisation d'un système tuteur pour la construction de figures géométriques. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I, 1994. Français. NNT : . tel-00005086

HAL Id: tel-00005086
https://theses.hal.science/tel-00005086

Submitted on 25 Feb 2004

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THESE

Présentée par

Cyrille Desmoulins

Pour obtenir le titre de

Docteur de l'Université
Joseph Fourier - Grenoble I
(Arrêtés ministériels du 5 juillet 1984
et du 30 mars 1992)

(Spécialité : INFORMATIQUE)

Étude et réalisation d'un système tuteur pour la construction de figures géométriques.

Thèse soutenue le 2 février 1994

Composition du jury :

Yves CHIARAMELLA	Président
Michel VAN CANEGHEM	Rapporteurs
Martial VIVET	
Jean-Marie LABORDE	Examinateurs
Dominique PASTRE	
Jean-Pierre PEYRIN	
Laurent TRILLING	

Thèse préparée au sein du **Laboratoire de Génie Informatique.**

Remerciements

Je tiens à remercier en premier lieu les membres du jury :

Yves Chiaramella, professeur à l'Université Joseph Fourier, directeur du Laboratoire de Génie Informatique, pour le soutien qu'il m'a apporté durant mon travail au LGI et pour l'honneur qu'il m'accorde en acceptant de présider le jury de cette thèse.

Michel Van Caneghem, Professeur à l'Université d'Aix-Marseille pour avoir accepté d'être rapporteur de cette thèse. J'apprécie l'intérêt qu'il y a porté, en particulier quant aux systèmes informatiques pour l'enseignement avec lesquels il n'était pas familier.

Martial Vivet, professeur à l'Université du Maine, pour s'être intéressé depuis longtemps déjà à mon travail et pour avoir accepté d'être rapporteur de cette thèse. Ses critiques constructives et ses encouragements répétés m'ont été très précieux.

Laurent Trilling, professeur à l'Université Joseph Fourier, qui a été le directeur de cette thèse. Je voudrais lui exprimer toute ma reconnaissance pour la confiance sans cesse renouvelée qu'il m'a accordée, pour le soutien efficace qu'il m'a apporté sans jamais ménager sa peine. Je le remercie pour l'énergie qu'il a su me communiquer pour aborder sans cesse de nouveaux rivages informatiques. Je le remercie encore de m'avoir proposé ce sujet et tiens à lui rappeler ce qu'il me disait alors : "Vérifier qu'une figure est correcte n'est pas si simple que cela en a l'air!". J'ose espérer que cette thèse illustre suffisamment la justesse *a posteriori* de ses propos.

Dominique Pastre, professeur à l'Université Paris V, pour les remarques très pertinentes qu'elle a faites sur mon travail. Sa compétence en résolution de problèmes en géométrie m'a apporté une plus grande précision dans mon travail.

Jean-Pierre Peyrin, professeur à l'Université Joseph Fourier, pour la lecture en spécialiste du domaine qu'il a faite de ce document, sans oublier le sympathique travail d'enseignement que j'ai effectué sous sa férule.

Jean-Marie Laborde, directeur de recherche au CNRS, responsable du projet Cabri-géomètre, pour l'inlassable ardeur qu'il prodigue pour diriger ce projet. J'ai beaucoup profité des groupes de travail qu'il y a dirigés et je tiens en particulier à le remercier des efforts qu'il a soutenus pour me fournir une version serveur de Cabri-géomètre.

Je tiens aussi à remercier tous ceux qui m'ont entouré dans ce travail :

Les membres de l'équipe PLIAGE, avec qui j'ai passé d'agréables moments et de qui j'ai appris de jolis morceaux de science, et quelques jeux de mots : Paul Amblard, Denis Bouhineau, Jeanne Idt, Stéphane Leman, Michel Levy et Pierre Ostier.

Les membres du Laboratoire de Structures Discrètes et Didactique, avec lesquels j'ai beaucoup et fructueusement collaboré : Nicolas Balacheff et Colette Laborde pour leur autorité en didactique des mathématiques, Bernard Capponi pour son expérience irremplaçable des classes, Salima Tahri et Vanda Luengo pour leur connaissance des tuteurs de géométrie, Yves Baulac, Franck Bellemain et Sylvie Tessier pour leurs talents de

développeurs de Cabri-géomètre. Je remercie en particulier Sylvie pour le travail qu'elle a fourni pour mettre au point la version serveur de Cabri-Géomètre et répondre à temps à mes besoins particuliers.

Mes voisins au sens très large, pour le cadre de travail sympathique qu'ils ont contribué à constituer, au labo comme dans nos réunions d'enseignants : la liste serait longue ...

Richard Allen, pour nos longues discussions amicales pendant son congé sabbatique. Sa vaste connaissance des tuteurs de géométrie et de leur utilisation en classe m'a beaucoup enrichi. De même Dominique Py pour son accueil toujours souriant à Rennes et pour les collaborations amicales que nous avons pu amorcer.

Et si la part qui revient à ceux qui me sont chers dans ce travail ne se voit pas, c'est parce qu'elle est essentielle. Je serais heureux si mes parents pouvaient voir dans cette thèse le fruit de leur affection, ma femme Domitille et mes enfants Sarah et Thomas l'effet bienfaisant de leur présence quotidienne à mes côtés.

Introduction..... 9
Le cadre..... 9
La problématique..... 11
Plan..... 12

Chapitre Premier.

Les Environnements Interactifs d'Apprentissage par Ordinateur.. 15

1. Des environnements génériques aux environnements dédiés..... 16
2. Des micro-mondes aux tuteurs..... 19
2.1. Micro-mondes..... 19
2.1.1. LOGO..... 19
2.1.2. Euclide..... 19
2.1.3. Cabri-géomètre..... 20
2.1.4. GEOSPECIF..... 22
2.1.5. CHYPRE..... 24
2.1.6. RMG..... 24
2.2. Tuteurs Intelligents..... 25
2.2.1. Geometry Tutor..... 25
2.2.2. MENTONIEZH..... 26
2.2.3. APLUSIX..... 27
2.3. Tuteurs de découverte guidée..... 28
2.3.1. DEFI-CABRI..... 28
2.3.2. HYPERCABRI..... 29
3. Intégration de capacités de raisonnement..... 31
3.1. EIAO intégrant des capacités de raisonnement..... 33
3.1.1. Cabri-géomètre..... 33
3.1.2. Geometry Tutor..... 34
3.1.3. Archimède..... 34
3.1.4. APLUSIX..... 34
3.2. EIAO intégrant des capacités de raisonnement complètes..... 36
3.2.1. DEFI..... 36
3.2.3. CHYPRE..... 37
4. Intégration de théories..... 39
4.1. EIAO intégrant des théories fixes..... 39
4.2. EIAO intégrant des théories modifiables..... 40
5. EIAO en géométrie..... 41
5.1. Panorama des EIAO en géométrie..... 41
5.2. Les objectifs de l'enseignement de la géométrie et comment les EIAO actuels y répondent..... 42
5.3. Intérêt didactique d'un système diagnostiquant la correction d'une construction..... 43

Chapitre Deuxième.

Définition de TALC, tuteur de découverte guidée pour la construction de figures géométriques satisfaisant une spécification 47

1. Objectifs et exigences 47

1.1. Un contrat didactique à la sémantique claire 48

1.2. Un degré de liberté maximal dans la formulation du contrat. 49

1.3. Des capacités de déduction complètes 50

1.4. Des performances garantissant un niveau d'interactivité suffisant pour un EIAO 50

2. Définition du langage logique LDL et des langages d'interfaces CDL et SCL. 53

2.1. LDL, un langage logique pour la vérification de la correction. 53

2.1.1. Présentation du langage. 53

2.1.2. Exemple d'énoncé décrit par une formule LDL. 55

2.1.3. Conséquences des choix effectués. 56

2.2. CDL, un langage d'interface pour l'expression de spécifications géométriques. 58

2.2.1. Présentation informelle de CDL 58

2.2.2. Syntaxe de CDL. 60

2.2.3. Exemple d'énoncé représenté en CDL 62

2.2.4. Sémantique de CDL 63

2.2.4.1. Traduction des littéraux. 64

2.2.4.2. Traduction des atomes 64

2.2.4.3. Traduction des termes représentant des objets géométriques 66

2.2.4.4. Traduction complète. 67

2.2.5. Synthèse des connaissances nécessaires pour l'utilisateur 67

2.3. SCL, un langage d'interface pour l'expression d'une construction géométrique. 69

2.3.1. Méthodologie de définition de SCL. 69

2.3.1.1. Utilisation d'un langage existant. 69

2.3.1.2. Notion d'"extension faible". 70

2.3.2. Traduction des primitives graphiques dans le langage d'énoncé de Cabri-géomètre 70

2.3.2.1. Primitives graphiques de Cabri-géomètre. 70

2.3.2.2. Traduction des primitives graphiques. 72

2.3.2.3. Conditions d'utilisation des primitives de Cabri-géomètre. 74

2.3.3. Traduction d'énoncés de Cabri-géomètre en CDL 75

2.3.3.1. Traduction simple d'un énoncé. 75

2.3.3.2. Conditions d'utilisation des primitives de SCL. 77

2.3.4. Exemple de traduction d'une construction en SCL. 79

3. Principes de formulation d'une théorie logique TIG. 81

3.1. Choix du langage utilisé pour représenter les axiomes 81

3.2. Choix de la forme des axiomes. 82

3.2.1. Restriction à des formules universelles sans symboles de fonctions. 82

3.2.2. Restriction à des clauses de Horn 83

3.2.3. Problèmes posés par le traitement de la négation. 84

3.3. Méthodologie de définition du contenu de TIG. 87

3.3.1. L'axiomatisation d'Euclide 88

3.3.2. L'axiomatisation de Hilbert. 88

3.3.3. L'axiomatisation de Tarski. 89
3.3.4. Définition d'une théorie initiale TIG_0 89
4. Expression de la partie prédéfinie du contrat didactique. *91*
4.1. Satisfaction par la construction de toutes les hypothèses de la spécification et seulement ces hypothèses. 91
4.1.1. Satisfaction par la construction de toutes les hypothèses de la spécification. 92
4.1.1.1. Formulation logique. 92
4.1.1.2. Objets à construire explicitement 94
4.1.1.3. Correspondance entre les noms de la construction de l'élève et les noms donnés par le professeur. 96
4.1.2. Satisfaction par la construction de seulement les hypothèses de la spécification. 98
4.1.2.1. Formulation logique. 99
4.1.2.2. Formulation faisant intervenir un ensemble d'axiomes d'extension. 99
4.1.2.3. Formulation faisant intervenir une fonction d'extension. 103
4.1.2.3.1. Définition de la fonction d'extension. 103
4.1.2.3.2. Définition opérationnelle de la fonction d'extension. 104
4.2. Cohérence de la spécification et de la construction vis-à-vis de la théorie. 108
4.3. Existence d'une construction répondant à la spécification. 109
4.4. La spécification doit être satisfiable par une unique construction. 109
4.4.1. Restriction du langage CDL à des conjonctions. 110
4.4.2. Restriction de la formulation des axiomes de TIG. 111
4.5. Synthèse des exigences du contrat didactique. 112

Chapitre troisième.

Mise en œuvre de TALC....113
1. Mise en œuvre du démonstrateur.....113
1.1. Mise en œuvre de base du démonstrateur 114
1.1.1. Représentation des axiomes et des faits par des règles Prolog.....114
1.1.2. La question des boucles.....116
1.1.3. Mise en œuvre d'un mécanisme de détection des boucles.....117
1.1.3.1. Première proposition de mise en œuvre.....117
1.1.3.2. Seconde proposition de mise en œuvre. 119
1.1.4. Mise en œuvre permettant des preuves sur des ensembles de faits variables. 120
1.1.5. Mise en œuvre permettant des preuves sur une théorie variable.....122
1.1.6. Mise en œuvre générique à l'aide d'un interpréteur.....124
1.1.6.1. Démonstrateur générique sans modification dynamique des faits et de la théorie.....124
1.1.6.2. Démonstrateur générique avec modification dynamique des faits. 125
1.1.6.3. Démonstrateur avec modification dynamique des faits et de la théorie.....126
1.1.6.4. Prise en compte de l'hypothèse de nom unique. .126
1.2. Améliorations des performances:.....127
1.2.1. Amélioration du mécanisme de détection de bouclages.....127
1.2.2. Amélioration de la version "faits variables" par recherche indexe sur les prédicats LDL pour les faits.....127
1.2.3. Amélioration par retardement de l'instanciation.....128
1.2.4. Amélioration par utilisation de lemmes 130
1.2.4.1. Intérêt de l'utilisation de lemmes.....131
1.2.4.2. Mise en œuvre utilisant les lemmes.....132
1.2.4.3. Problèmes posés par cette mise en œuvre.....134
1.2.4.4. Caractérisation de l'introduction d'anti-lemmes...134
1.2.4.5. Complétude d'un algorithme basé sur cette caractérisation de l'ajout d'anti-lemmes.....136
1.2.4.6. Une mise en œuvre utilisant cette caractérisation. 137
1.3. Utilisation d'un contre-modèle. 139
2. Mise en œuvre de TALC-Enseignant.....141
2.1. Mise en œuvre de l'interface de TALC-Enseignant. 141
2.1.1. Edition des éléments du contrat.....141
2.1.1.1. Édition de TIG 141
2.1.1.2. Édition de la spécification de la figure.....142
2.1.1.3. Édition des outils de construction.....142
2.1.1.4. Édition de TEXT. 142
2.1.2. Création d'un exercice.....142
2.2. Mise en œuvre de la traduction simple de CDL. 143
2.3. Réduction de la traduction simple de CDL pour obtenir une traduction complète.....143
3. Mise en œuvre de TALC-Élève 145
3.1. Mise en œuvre de l'interface de TALC-Élève. 145
3.1.1. Fonctionnalités explicites.....145

3.1.2. Fonctionnalités transparentes nécessaires. 148
3.2. Mise en œuvre de la communication avec Cabri-géomètre..... 149
3.3. Mise en œuvre de la traduction de SCL. 149
3.3.1. Représentation et manipulation des faits..... 149
3.3.2. Mise en œuvre de la traduction simple d'une phrase d'énoncé..... 149
3.3.3. Mise en œuvre de la traduction complète d'un énoncé. 151
3.3.4. Fonctionnalités modifiant la traduction autres que les primitives de construction 151
4. Mise en œuvre de la vérification de la correction..... 153
4.1. Vérification que la construction satisfait toutes les hypothèses de la spécification..... 154
4.2. Mise en œuvre de la fonction d'extension. 155
4.3. Vérification de la nécessité d'une construction. 156
5. Diagnostics en vue d'une explication. 157
5.1. Diagnostics..... 157
5.1.1. Construction du diagnostic. 157
5.1.2. Exemple illustrant les diagnostics possibles. 158
5.1.3. Affinement du diagnostic..... 160
5.1.3.1. Diagnostic dans le cas où des propriétés manquent. 161
5.1.3.2. Diagnostic dans le cas où la construction contient des propriétés particulières. 161
5.2. Proposition d'explications..... 162
6. Résultats..... 165
6.1. Évaluation de la mise en œuvre. 165
6.1.1. Importance de la mise en œuvre..... 165
6.1.2. Utilisation de Prolog comme outil de génie logiciel. 166
6.2. Les problèmes traités par TALC. 167
6.2.1. Des spécifications requérant une construction simple..... 168
Exemple 6.2.1.1. 168
Exemple 6.2.1.2. 169
Exemple 6.2.1.3. 169
Exemple 6.2.1.4. 170
6.2.2. Des spécifications exprimant un problème de construction. 171
6.2.2.1. Exemples où aucun objet n'est donné..... 172
6.2.2.2. Exemples où les objets de base sont donnés. 172
Exemple 6.2.2.2.1. 172
Exemple 6.2.2.2.3. 173
Exemple 6.2.2.2.4. 174
6.3. Performances..... 174

Conclusion **177**
Les problèmes informatiques posés. *177*
Méthodologie de spécification logique 177
Techniques de l'intelligence artificielle. 180
La programmation logique comme outil de génie logiciel. 181
Perspectives *183*

Bibliographie **185**

Annexe A

TIG$_0$ exemple d'une théorie de la géométrie.. ..... **191**
A. Axiomes portant sur des égalités d'objets. *191*
1. Egalité de points 191
2. Egalité de droites 193
3. Egalité de demi-droites 193
4. Egalité de segments 193
5. Egalité de cercles 194
6. Egalité de distances. 194
B. Axiomes portant sur les propriétés de sens. 194
7. Sens égaux 194
8. Sens inverses. 195
C. Axiomes portant sur les propriétés d'appartenance. *196*
9. Appartenance à une droite. 196
10. Appartenance à une demi-droite 196
11 . Appartenance à un segment 197
12. Appartenance à un cercle 197
D. Axiomes portant sur les droites *197*
13. Droites parallèles. 197
14. Droites perpendiculaires. 198
E. Axiomes portant sur les distances *199*
15. Demi-distance. 199
16. Distances inférieures 199
17. Distance entre deux points. 200
18. Somme de distances. *201*

Annexe B.

Exemples d'axiomes de TEXT **203**
Constructions sans degré de liberté 203
Constructions avec un degré de liberté. 203
Constructions avec deux degrés de liberté 204

Annexe C.

Sémantique complète de CDL **205**
1. Définition de la traduction, règle par règle. *205*
2. Sémantique de CDL, règle par règle *206*
3.Définition des fonctions de composition. *209*
4.Mots réservés de CDL. *214*
5.Syntaxe abstraite de CDL *215*

Annexe D.

Le langage d'énoncés de Cabri-géomètre.. 217

Annexe E.

Preuve de la satisfaction de la définition logique de la multifonction d'extension par sa définition opérationnelle..... 219

Annexe F.

Mise en œuvre de la traduction simple de CDL..... 221
1. Analyse lexicale. *221*
2. Analyse des identificateurs. *221*
2.1. Construction de l'environ. 221
2.2. Construction de l'ensemble des termes identifiés. 224
3. Traduction simple à partir de l'environ et de l'ensemble des termes identifiés. *224*

Annexe G.

Mise en œuvre de la communication avec Cabri-géomètre..... 227
1. Première mise en œuvre *227*
2. Seconde mise en œuvre. *227*
3. La version de Cabri-géomètre serveur d'une autre application. *228*

Introduction

Le cadre.

L'ambition placée dans l'usage de l'ordinateur pour l'enseignement fut initialement sans doute exagérée. Il s'agissait de définir des systèmes informatiques qui puissent remplacer l'enseignant. Aujourd'hui l'idée est plutôt d'utiliser le système informatique non pas comme un substitut de l'enseignant, mais comme son représentant. Dans cette optique, l'usage de l'ordinateur dans une situation d'apprentissage peut avoir plusieurs intérêts :

• Il permet à l'enseignant de déléguer certaines tâches d'enseignement automatisables pour lesquelles sa présence n'est pas nécessaire. Cela lui permet de focaliser son attention sur des problèmes plus délicats ou sur les élèves en difficulté. L'idée est que bien souvent, un simple "coup de pouce" [Carrière et al 91] suffit à l'élève pour continuer la résolution du problème. L'ordinateur est aussi utile pour vérifier la justesse des productions de l'élève.

• Il fournit à l'apprenant une approche différente. Cela permet à certains élèves ayant des difficultés en suivant la méthode classique d'apprendre à un rythme plus personnalisé, avec une présentation des connaissances différentes et un rapport à cette connaissance détachée de l'influence du côté affectif de sa relation avec l'enseignant. Dans ce cadre, le système informatique est vu comme un nouveau moyen didactique permettant d'enrichir la panoplie des moyens proposés.

• offre des situations d'apprentissage jusque là impossible. Il simule par exemple un éventail d'expérimentations bien plus grand que ne le permettrait un dispositif physique, à moindre frais. Il permet aussi de présenter des situations d'apprentissage nécessitant de fastidieux calculs ou encore nécessitant l'accès à des données de taille conséquente.

• Il introduit aussi des problématiques nouvelles. Avec Cabri-géomètre [Baulac 90] (voir chapitre premier, 2.1.3), par exemple, un nouveau type de problème est apparu : construire une figure géométrique résistant au déplacement dynamique de ses composants. Dans ce cas, le système informatique introduit un savoir diffèrent, inconnu jusqu'alors (la géométrie "qui bouge").

Pour que le système informatique puisse valablement représenter l'enseignant, il faut le doter de capacités similaires. C'est dans cet objectif que les études en intelligence artificielle représentent un intérêt évident pour la conception de systèmes d'enseignement. Les méthodes et les outils de l'intelligence artificielle permettent en particulier d'introduire des capacités de raisonnement dans les systèmes informatiques pour l'enseignement. Ces capacités de raisonnement peuvent être utilisées de diverses manières :

- pour vérifier la cohérence du travail de l'apprenant.
- pour vérifier que la solution qu'il propose est correcte.
- pour résoudre automatiquement des problèmes, ce qui permet d'aider l'apprenant dans sa résolution.
- pour expliquer à l'apprenant ses erreurs en déduisant de son comportement les raisons qui l'ont amené à se tromper.

Notre travail se situe dans ce cadre des systèmes informatiques pour l'enseignement intégrant des capacités de raisonnement. Il concerne les systèmes dont l'objectif est l'apprentissage par la résolution de problèmes.

De notre point de vue, fournir un système d'enseignement possédant des capacités de raisonnement est une bonne chose à condition qu'il puisse représenter suffisamment fidèlement l'enseignant. Dans cette optique, trois qualités nous semblent primordiales.

La première qualité d'un système intégrant des capacités de raisonnement est que ces capacités répondent à des exigences minimales. La première exigence sur les raisonnements du système est qu'ils soient cohérents. En effet, il serait extrêmement préjudiciable pour l'apprentissage de l'élève que le système puisse lui affirmer à la fois une propriété et son contraire. La seconde exigence est que les raisonnements du système soient complets. Dans le cadre de la résolution de problème, cela signifie qu'il faut que le système puisse reconnaître toutes les solutions proposées par l'apprenant. En effet, si une solution correcte du point de vue des connaissances du domaine enseigné n'est pas reconnue comme telle, l'apprenant risque d'acquérir des conceptions erronées, ce qui est évidemment en contradiction avec les objectifs d'un système d'enseignement. De plus, la fidélité du système dans sa représentation de l'enseignant n'est alors pas assurée, si tant est que l'on puisse affirmer que les raisonnements d'un enseignant soient complets dans la matière qu'il enseigne.

La deuxième qualité d'un système est que l'enseignant puisse avoir suffisamment de liberté pour définir les situations qui lui semblent appropriées. En effet, quel système informatique pourrait se vanter de pouvoir intégrer a priori l'ensemble riche et varié des situations que l'enseignant peut vouloir mettre en œuvre? Si l'ensemble des activités proposées par le système est figé à l'avance, il est clair que le choix proposé à l'enseignant risque de ne pas toujours lui convenir. Il est donc essentiel que l'enseignant puisse spécifier lui-même le plus possible les éléments définissant la situation qu'il propose. Ces éléments constituent alors les degrés de liberté que lui offre le système.

La troisième qualité d'un système est que la définition de son comportement soit sans ambiguïté. Concernant les capacités de raisonnement du système par exemple, il est souvent dit à ce sujet qu'il suffit de représenter le raisonnement à l'aide de règles de production et de confier ces règles à un "système expert". Cela nous semble trop simpliste et insuffisant dans la mesure où cette affirmation ne dit rien de précis sur les déductions vraiment effectuées par le système.

Décrire précisément le comportement du système revient à donner une définition précise de ce que nous appelons le "contrat didactique". Le contrat didactique est l'ensemble composé de l'objectif donné à l'apprenant et du contexte qui y est attaché. Nous estimons que la qualité d'un système d'enseignement tient à la sémantique claire et précise du contrat didactique proposé par l'enseignant par son intermédiaire. En effet, comment évaluer précisément la solution décrite par un apprenant si l'objectif du problème et son contexte ne sont pas eux-mêmes précis? Et sans évaluation précise, comment lui proposer des explications de qualité lui permettant d'améliorer ses connaissances? Les moyens d'expression du contrat didactique fournis à l'enseignant, en plus de posséder une sémantique claire et précise, se doivent aussi d'être les plus généraux possibles. Cela signifie que les cas particuliers, dont l'expression est souvent plus ardue que le cas général, doivent pouvoir être exprimés directement plutôt que d'obliger l'enseignant à utiliser des expressions intermédiaires produisant des interférences dans la compréhension du problème. Par exemple, si l'on ne dispose que de l'opérateur de multiplication, décrire des polynômes nécessite de représenter l'opérateur de puissance par plusieurs multiplications. Ainsi notre objectif est de donner à l'enseignant un moyen d'expression du constrat didactique suffisamment général tout en possédant une sémantique précise.

La problématique.

Notre problématique est consacrée à la conception et à la réalisation d'un système pour l'aide à l'enseignement dans un domaine précis quoique vaste : la géométrie. Son objectif est de diagnostiquer la correction d'une construction géométrique d'un apprenant vis-à-vis de la spécification qu'en a donné un enseignant. Nous avons dénommé ce système TALC, pour Tuteur d'Aide Logique à la Construction.

Vérifier qu'une figure est correcte a au moins trois intérêts didactiques :

• Le premier intérêt concerne les activités dites de recherche, point de départ de nombreuses activités dans l'enseignement actuel. Par exemple, l'enseignant peut demander la construction d'une figure sur laquelle des propriétés doivent être étudiées. Il est clair que si la figure ne possède pas toutes les propriétés demandées, les propriétés étudiées risquent d'être fausses, induisant des conclusions erronées. Un tuteur vérifiant automatiquement que la construction est satisfaisante permet à l'élève de passer immédiatement à la phase suivante et de libérer le professeur de la fastidieuse tâche de correction.

• le second intérêt est qu'un tel tuteur permet d'éviter l'aspect numérique et particulier des exercices donnés par des enseignants, du fait de considérations nullement pédagogiques mais purement correctionnelles. Par exemple, la contrainte de taille imposée aux segments dans de nombreux exercices a rarement pour objet de bien utiliser la règle graduée mais plutôt de faciliter la correction en série par l'enseignant des diverses solutions des élèves à un problème, en utilisant astucieusement un calque. Les propriétés particulières introduites pour cette raison peuvent induire l'élève en erreur.

• Le troisième intérêt est plus fondamental. Les enseignants ont remarqué qu'une des plus grandes difficultés des élèves pour construire des démonstrations vient de la confusion qu'ils font entre hypothèses et conclusions. Demander à un élève de construire la figure correspondant aux hypothèses d'un problème permet de vérifier qu'elles sont bien comprises.

Toutes ces raisons d'utiliser un système diagnostiquant la correction d'une figure se confondent en une seule : avant toute activité géométrique il est important de savoir si la figure support de l'activité est correcte.

Historiquement, notre travail est issu du Projet Mentoniezh [Py 90b], qui distinguait quatre composantes pour un système d'enseignement en géométrie :

• acquisition de la figure : l'élève construit pour commencer une figure conforme à la spécification des hypothèses désirées par le professeur. La partie construction est une des fonctionnalité actuelle de Cabri-géomètre. La partie vérification constitue la fonctionnalité principale de TALC.

• appropriation de la figure : il peut ensuite faire évoluer graphiquement la figure dont les propriétés logiques sont conservées. Cela lui permet, d'une part de détecter les invariants intéressants, d'autre part d'observer l'impact de la suppression d'une hypothèse. Cette phase correspond à ce qu'offre Cabri-géomètre.

• exploration des propriétés : l'élève donne son avis sur d'éventuelles propriétés suggérées par le système à l'aide de théorèmes fournis par l'enseignant. C'est ce que propose DEFI-CABRI aujourd'hui [Baulac et al. 91] (voir chapitre premier, 2.3.1).

• construction de la preuve : l'élève construit sa démonstration, vérifiée par le système, en s'inspirant des découvertes faites aux étapes précédentes et en utilisant des théorèmes fournis par l'enseignant. C'est la composante réalisée par D. Py [Py 90a] (voir chapitre premier, 2.2.2), qui a gardé le nom du projet initial pour son logiciel.

À l'heure actuelle et depuis 1990, nous travaillons dans le cadre du projet IMAG Cabri-géomètre, dirigé par Jean-Marie Laborde. Les nombreux échanges et collaborations auxquels ont donné lieu ce projet sont à la source de nos réflexions. Notre réalisation fut dès le départ destinée à fonctionner en communication avec le remarquable logiciel Cabri-géomètre qui dispose maintenant d'une diffusion mondiale et dont la prochaine version constitue une avancée importante.

Le propos que nous soutenons ici est que la définition d'un système d'aide à l'enseignement et en particulier celui que nous présentons ici, pose des problèmes informatiques intéressants et neufs.

En premier lieu, la définition de TALC pose des problèmes de solution automatique en géométrie qui sont spécifiques et non abordés dans d'autres domaines géométriques comme la CAO par exemple.

En second lieu, cette définition constitue un champ d'expérimentation permettant de mesurer l'adéquation et d'améliorer les méthodes de spécification logique, les techniques de l'intelligence artificielle et la méthodologie d'utilisation de la programmation logique comme outil de génie logiciel.

En ce qui concerne la spécification de problèmes, nous soutenons que l'expression par la logique du premier ordre du contrat didactique est nécessaire pour en obtenir une définition qui soit claire et explicite, et pour permettre de fournir de bonnes explications à l'apprenant. Dans cette optique, nous montrons ici à quel point cette attitude est exigeante et fructueuse. Elle nous amène en particulier à concevoir un processus d'extension logique d'une formule par rapport à une autre. Ce processus permet ici de prendre en compte des objets non présents dans une spécification et introduits dans une construction.

Sur le plan des techniques de l'intelligence artificielle, nous devons faire face ici à la mise en œuvre de plusieurs démonstrateurs. La définition opérationnelle du processus d'extension nous amène en particulier à proposer une nouvelle méthode, intéressante dans les cas où la complétude est exigée.

Concernant la programmation logique, la mise en œuvre de ce système montre l'intérêt de Prolog comme outil de haut niveau. Il permet en particulier des mises en œuvre avec plusieurs meta-niveaux de solution et nous permet de proposer une méthodologie d'utilisation de la notion de module de PrologII+. Elle a permis de dégager des méthodes améliorant les performances d'un démonstrateur dans la logique des propositions avec l'hypothèse du monde clos. Ainsi, nous proposons plusieurs techniques d'optimisation des démonstrateurs dont l'une, typiquement adaptée à PrologII+, gère des lemmes et des "anti-lemmes" (lemmes non prouvables).

Plan.

Le plan adopté dans ce document est basé sur trois chapitres.

Dans le premier chapitre, nous présentons le domaine de l'informatique consacré à la conception et à l'usage d'ordinateurs pour l'enseignement. Cette présentation s'articule autour de quatre distinctions. Chaque distinction nous permet d'une part de dégager les caractéristiques de chacun des EIAO (Environnements Interactifs d'Apprentissage par Ordinateur) les plus marquants que nous présentons et d'autre part de les classifier entre eux. Nous présentons ensuite la place que vient prendre TALC parmi les EIAO pour la géométrie dans cette classification. Cette présentation permet de montrer, nous l'espérons, l'importance du domaine.

Dans le second chapitre, nous donnons la définition de TALC. Pour cela nous présentons dans un premier temps les objectifs et les exigences que la définition d'un EIAO (et donc en particulier de TALC) doit remplir. A partir de ceux-ci nous définissons le langage logique LDL et les langages d'interface CDL et SCL. Nous présentons ensuite les principes de la formulation d'une théorie logique dite TIG (pour Théorie Instrumentale de la Géométrie) représentant les connaissances supposées de l'élève en géométrie. Enfin

nous établissons, en procédant par étapes de dérivation successives, l'expression du contrat didactique dans TALC.

Dans le troisième chapitre, nous présentons la mise en œuvre de TALC. Pour cela nous décrivons tout d'abord les différentes mises en œuvre du même démonstrateur de base que nous avons conçues pour chacune des démonstrations nécessaires à la vérification de la correction et les optimisations que nous leur avons apportées. Nous présentons ensuite la mise en œuvre de TALC-Enseignant, qui est le système avec lequel l'enseignant définit le contrat qu'il propose à l'élève. La présentation de la mise en œuvre de ce système nécessite principalement de définir l'interface proposée à l'enseignant et la traduction des éléments du contrat en un exercice. Nous présentons la mise en œuvre de TALC-Élève, qui est le système avec lequel l'élève répond au contrat proposé par un enseignant. Pour cela, nous décrivons la mise en œuvre de l'interface proposée à l'élève, puis la mise en œuvre de la communication avec le logiciel Cabri-géomètre et enfin la traduction de la construction de l'élève qui permet d'effectuer le diagnostic. Enfin, nous présentons la mise en œuvre de la correction, en particulier celle concernant la construction de l'extension.

Chapitre Premier.
Les Environnements Interactifs d'Apprentissage par Ordinateur.

Le domaine de l'informatique consacré à la conception et à l'usage d'ordinateurs pour la formation peut être désigné par de nombreuses appellations□EAO (Enseignement Assisté par Ordinateurs), EIAO (Enseignement Intelligemment Assisté par Ordinateurs), ITS (Intelligent Tutoring Systems), CAI (Computer Aided Instruction), ICAI (Intelligent Computer Aided Instruction), CAL (Computer Aided Learning), CBT (Computer Based Teaching), TI (Tuteurs Intelligents). Le sens exact de toutes ces abréviations peut sembler bien confus au néophyte et, bien que constituant un des domaines de l'informatique, aux informaticiens eux-mêmes. Nous avons pour notre part choisi la dénomination d'EIAO pour désigner le domaine.

Voici comment nous entendons ce sigle, en suivant en cela [Baron et al. 93]□

Le sigle EIAO présente deux acceptions complémentaires. D'une part, il représente un système informatique: un EIAO (*Environnement Interactif d'Apprentissage avec Ordinateur*) est un système informatique qui a pour objet de favoriser l'apprentissage d'un domaine de connaissances par un apprenant. D'autre part il représente un domaine situé au confluent de l'informatique et des sciences de l'éducation: l'EIAO est le domaine de recherche portant sur les systèmes d'EIAO. Il regroupe des travaux de recherches fondamentales et appliquées dont les objectifs sont de formaliser les processus d'apprentissage humains, de concevoir des modèles de connaissances qui soient à la fois cognitifs et computationnels, de réaliser des EIAO et d'étudier leur insertion dans la formation.

Cette acception du terme EIAO a pour avantage par rapport à *Enseignement Intelligemment Assisté par Ordinateur* de couvrir toute l'étendue du domaine, sans exclure les environnements ne comportant pas de faculté de raisonnement [Balacheff 90].

La présentation du domaine que nous proposons est basée sur les distinctions suivantes :

-□a distinction entre les environnements spécialisés, dits "dédiés" et les environnements génériques;

- la distinction entre micro-mondes et tuteurs;

- la distinction suivant la complétude du raisonnement proposé;

-□a distinction entre les systèmes permettant de modifier les théories représentant les connaissances utilisées et ceux manipulant des théories fixées une fois pour toutes.

Ces distinctions nous permettent à la fois de classer les environnements existants et d'exposer les raisons des choix effectués pour leur définition. Nous donnons par la même occasion ainsi une idée de l'étendue de ce domaine.

Nous présentons dans un premier temps ces distinctions en développant les caractéristiques des EIAO les plus marquants dans chacune, pour ensuite proposer une classification générale des environnements de géométrie existants suivant chacun de ces axes. Cela nous permet de dégager quelles questions nous semblent centrales à l'étude des EIAO et la place de notre tuteur vis-à-vis des environnements existants.

On remarquera que parmi les EIAO que nous présentons se trouvent un grand nombre d' EIAO en géométrie. Il y a à cela deux raisons. D'une part la géométrie est le sujet d'un nombre important d'EIAO comparativement aux autres domaines. Ceci est sans doute dû au fait que deux représentations des données, graphique et textuelle sont

disponibles. D'autre part la géométrie est le domaine qui nous intéresse et la présentation d'EIAO de ce type nous permet par la suite de fructueuses comparaisons.

1. Des environnements génériques aux environnements dédiés.

La question centrale est la suivante : *Faut-il s'attacher à concevoir des architectures d'EIAO indépendantes des contenus d'enseignement, ou seuls des environnements dédiés à l'enseignement d'une discipline sont-ils pertinents?*

Cette question distingue les pédagogues des didacticiens.

Les recherches des premiers tendent à détacher les méthodes pédagogiques de leurs contenus. ETOILE [Dillenbourg 93], par exemple, est un outil logiciel pour la création d'EIAO. Il a pour objet de permettre d'une part de définir comment appliquer un "style d'enseignement" (ou stratégie pédagogique) à plusieurs contenus et d'autre part de définir plusieurs styles d'enseignement applicables à un même contenu. Un style d'enseignement est décrit par des paramètres pédagogiques (comme la vitesse d'adaptation du tuteur qui décrit à partir de quand le système réagit aux erreurs de l'élève ou encore son type de raisonnement - analogique, déductif ou inductif). Pour un style donné, certains paramètres sont fixés, les autres obéissent à des "règles didactiques" définissant le style. Ces règles ont pour objectif de représenter une manière d'enseigner, par exemple, à la Papert, ou encore à la Pavlov.

Exemple :

Pour le paramètre dit de directivité du système, dans un style d'enseignement où la directivité est maximale, le tuteur joue le rôle d'un présentateur de connaissance alors que dans un style à directivité faible le système laisse une part importante à l'initiative de l'apprenant.

Pour l'instant, ETOILE ne constitue pas un produit fini. Il a donné lieu à la définition d'un seul EIAO par ses auteurs, MEMOLAB, environnement d'acquisition des compétences de base en psychologie expérimentale, par la création et la simulation d'expériences sur la mémoire.

D'autres travaux ont pour objet de concevoir un formalisme de représentation des connaissances permettant de définir une architecture générique et modulaire pour les EIAO, à base de système à objets [Bournaud et al, 1993]. Leur premier objectif est de disposer d'un système pouvant être enrichi progressivement. Leur second objectif est de fournir ainsi un moyen de comparaison des systèmes.

Pour les didacticiens, même si des méthodes pédagogiques générales sont utiles concernant la forme de l'enseignement (méthodes de communication, supports d'enseignements, …) l'apprentissage ne peut être général du fait qu'il dépend étroitement du fond.

Exemple :

Les méthodes d'enseignements des mathématiques, discipline formelle, ne peuvent être directement transposées à la physique, discipline expérimentale.

Le but des didacticiens est de définir des situations didactiques favorables à l'acquisition de telle ou telle connaissance. Vis-à-vis de l'EIAO, leur but est la définition et l'étude de systèmes informatiques mettant en place de telles situations, comme l'explique Balacheff [Balacheff 94] sur les travaux duquel nous nous appuierons dans la suite de ce chapitre.

Remarques conclusives

Notre point de vue sur cette distinction est qu'elle est trop simpliste. La véritable question à notre avis est double. Elle se pose d'une part vis-à-vis de l'informatique :

Qu'est-ce qui distingue un système EIAO d'un autre système informatique?. Elle se pose d'autre part vis-à-vis des sciences de l'éducation□ *Qu'est-ce qui distingue la situation d'apprentissage habituelle de la situation d'apprentissage utilisant un EIAO?*. C'est à partir des réponses à ces deux questions - et ces réponses sont aujourd'hui encore embryonnaires - que l'on pourra définir la part du générique de celle dépendant de la discipline.

□

Les distinctions que nous proposons dans ce document sont de notre point de vue un début de réponse à ces deux questions. Ces distinctions tournent toutes in fine autour de deux points que nous développons concernant notre tuteur et qui nous semblent caractéristiques des EIAO□ le contrat didactique (voir deuxième chapitre, quatrième partie) et l'explication (voir chapitre troisième, partie 5).

Une autre question développée dans ce document nous semble suffisamment générale pour donner lieu à des formalisations similaires permettant des comparaisons enrichissantes□ il s'agit des possibilités d'introduction d'objets auxiliaires, soit automatiquement, soit par l'apprenant. Avec l'introduction d'objets supplémentaires, la résolution du problème nécessite d'établir la preuve de l'équivalence de deux représentations ne portant pas sur les mêmes objets. Certaines résolutions de problèmes doivent faire appel à des objets non décrits dans l'énoncé du problème.

Exemple :

L'introduction d'objets supplémentaires peut consister en :

- un changement de variable pour l'intégration de fonctions dans ELISE [Delozanne et al. 89]
- l'introduction automatique d'objets géométriques supplémentaires dans PROGE (nommés *objets passerelles*) [Schreck 92] ou dans GEOMUS [Bazin 93].
- l'introduction par l'utilisateur d'objets géométriques supplémentaires dans le résolveur générique de [Pintado 91].

Dans notre cas, il s'agit de donner à l'élève la possibilité de construire des objets géométriques non décrits par le professeur.

Tout au long de ce document, notre objectif est ainsi de dégager ce qui nous parait relever de questions génériques aux EIAO de ce qui nous paraît caractériser la définition de TALC. Les distinctions que nous présentons dans ce qui suit sont ainsi de notre point de vue des questions génériques permettant de bien caractériser un EIAO.

2. Des micro-mondes aux tuteurs

Cette distinction permet de localiser les systèmes existant sur une échelle à peu près continue en fonction de l'initiative laissée à l'apprenant, des environnements à initiative inexistante aux environnements ouverts [Balacheff 94]. Aux deux extrémités de cette échelle se trouvent les tuteurs et les micro-mondes, quelque part entre les deux les systèmes dits "à découverte guidée".

Remarquons que pour être plus précis, il faudrait placer au bout de l'échelle les systèmes laissant très peu d'initiative à l'apprenant. Dans ces systèmes se situent de notre point de vue les EAO. Un EAO est caractérisé par le fait qu'il propose à l'apprenant un système de questions avec réponses préenregistrées. A partir des réponses données par l'apprenant, l'EAO change d'état et propose une nouvelle question à l'apprenant. Au contraire le sigle EIAO était fondé à l'origine sur l'ajout d'un I (pour Intelligemment) au sigle EAO. Ainsi les questions précises auxquelles peut répondre un système d'EIAO dans ce sens du sigle ne sont pas connues à l'avance, du fait des capacités de raisonnement introduites [Nicaud et al. 88]. Avec l'adoption de la signification *Environnement Interactif d'Apprentissage avec Ordinateur,* les environnements anciennement appelés EAO ont leur place dans la mesure où ils disposent de capacités d'interaction évoluées avec l'apprenant, principalement en lui proposant des simulations de phénomènes réels. Nous les évoquons en même temps que les micro-mondes car leurs auteurs eux-mêmes les placent dans cette catégorie.

Pour chaque type d'EIAO (micro-monde, tuteur intelligent, tuteur de découverte guidée) nous proposons tout d'abord une définition générale. Nous présentons ensuite un ensemble d'EIAO représentatifs, en expliquant les choix qui distinguent entre eux et les raisons de ces choix.

2.1. Micro-mondes.

Il est difficile de donner une définition exacte de ce qu'est un micro-monde. Nous nous basons dans un premier lieu sur la définition des inventeurs du terme, Minsky et Papert. L'idée de base est de proposer un univers restreint qui puisse être considéré de façon isolée du reste du monde, dans lequel les objets et leurs relations sont simplifiés [Dreyfus 81]. L'apprenant, en créant et manipulant ces objets, construit lui-même sa connaissance (paradigme d'apprentissage exploratoire). L'objectif de Minsky et Papert [Minsky et al. 70] est de construire des micro-mondes dont la suggestivité et le pouvoir prédictif dépassent le fait que les énoncés qu'ils produisent sur les objets puissent être littéralement faux dans le monde réel.

2.1.1. LOGO.

LOGO, conçu par les mêmes Minsky et Papert [Papert 80], est l'exemple type d'un micro-monde. A partir de quelques primitives simples de dessin géométrique dans le plan, l'apprenant peut construire des procédures de plus en plus complexes lui permettant de répondre à de nouveaux objectifs. En quelque sorte le micro-monde évolue en même temps que la connaissance de l'apprenant. C'est cette capacité d'évolution qui marque la différence entre les micro-mondes et les environnements de simulation.

2.1.2. Euclide.

Euclide [Allard et al. 86] est un langage informatique inspiré de LOGO permettant de construire des figures géométriques. Il fournit à l'utilisateur un ensemble de primitives pour dessiner et conjecturer et lui permet de définir ses propres procédures de

dessin. À la différence de LOGO, conçu à partir d'une représentation en coordonnées relatives du plan, Euclide ne fait référence à aucun système de coordonnées. Une fois une procédure définie, l'élève l'utilise en désignant par le biais d'une souris la place géographique des objets. Implanté directement en LOGO, il reprend ses fonctionnalités et ses inconvénients□il offre les possibilités d'interactivité et de modularité de tout langage interprété et en même temps impose la gestion informatique par l'apprenant des procédures construites (commandes de sauvegarde de fichiers, de visualisation des procédures).

Une de ses propriétés majeures par rapport aux micro-mondes comme Cabri-géomètre (voir ci-dessous) est d'imposer à l'apprenant une formulation syntaxique rigoureuse. C'est une qualité pour les uns [Artigue 90], qui voient là un apprentissage important de la formulation en géométrie, un défaut pour les autres [Matiussi 90], qui considèrent que l'ergonomie des commandes est très lourde et ajoute un apprentissage supplémentaire inutile.

Exemple :

Soit "A INTDD□D3□D2 signifie que le point A est l'intersection des droites D2 et D3

Tous cependant s'accordent pour juger cette rigueur syntaxique comme un bon révélateur des conceptions géométriques déficientes des apprenants.

Une autre propriété intéressante, que l'on retrouve dans les autres micro-mondes de géométrie, est de faire passer l'apprenant d'une conception de la géométrie "en gestes" (*poser une équerre*) à une conception "en objets et relations" (*construire une droite qui soit perpendiculaire à une autre*).

Une dernière propriété, elle aussi commune aux autres micro-mondes de géométrie, est de permettre une démarche expérimentale en géométrie, par l'observation de l'effet de la variation de certains paramètres sur des propriétés de constructions.

2.1.3. Cabri-géomètre.

Cabri-géomètre (CAhier de Brouillon Interactif de géométrie) [Laborde 86] [Baulac 90] [Bellemain 92] est un micro-monde de construction de figures géométriques. A la différence d'Euclide, l'interface est l'un des points forts de Cabri-géomètre. Cette interface évoluée issue conjointement de recherches en didactique et informatique a permis la grande diffusion que connaît aujourd'hui Cabri-géomètre. D'une part il utilise des menus déroulant et la souris pour définir des objets géométriques, d'autre part il permet leur modification dynamique en intégrant le concept de manipulation directe [Nanard 90]. Ce concept, issu des problématiques d'interface homme-machine, stipule que les objets de l'environnement doivent pouvoir être manipulés à travers un minimum d'intermédiaires.

Exemple :

L'utilisateur peut modifier l'heure d'une horloge disponible dans l'environnement directement en bougeant la place des aiguilles de l'horloge.

Poussé à la perfection, l'application de ce concept permet d'envisager par exemple la manipulation de solides (polyèdres, sphères) par l'intermédiaire d'un gant reproduisant à la fois les actions sur les objets sur un système de vision 3D et les sensations dues au poids, à l'élasticité, etc., des objets. Ce principe est important dans un système d'EIAO plus que dans d'autres systèmes informatiques, dans la mesure où il permet à l'apprenant d'éviter de lier ses connaissances à telle ou telle représentation et facilite une prise en main rapide du système.

A partir de la manipulation d'une souris, Cabri-géomètre permet la construction de quatre types d'objets dans le plan : des points, des droites, des segments et des cercles. Les autres objets disponibles (voir figure I.1) sont des composés d'un ou plusieurs de ces objets primitifs munis de propriétés particulières (triangle par exemple). Cabri-géo-

mètre distingue deux catégories d'objets : les objets "de base" dont la construction ne dépend d'aucun autre objet et les objets construits à partir d'autres objets.

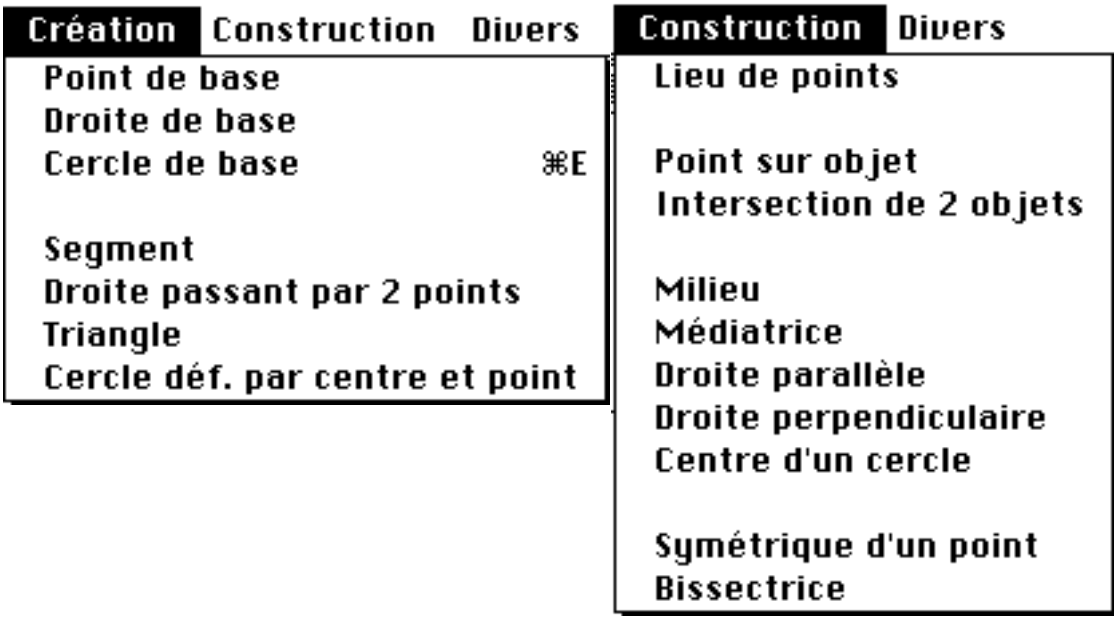


Figure I.1

La manipulation directe permet, à n'importe quel stade d'une construction, de modifier la position géographique des objets dont le degré de liberté est non nul tout en conservant toutes les propriétés géométriques données durant la construction. Seuls les points et les cercles de base possèdent un degré de liberté 2. Les droites de base et les points sur objets possèdent un degré de liberté 1. Les points d'intersection n'ont aucun degré de liberté.

Exemple :

Pour obtenir la figure I.2, l'élève a construit les deux médiatrices d'un triangle et un cercle passant par l'un des sommets et dont le centre est l'intersection des deux médiatrices. Il peut "saisir" un des sommets et lui donner différentes positions, le reste de la construction est alors rédessiné dynamiquement. Cette manipulation conserve par exemple la propriété visuelle (et géométrique dans ce cas) que le cercle passe par les deux autres sommets.

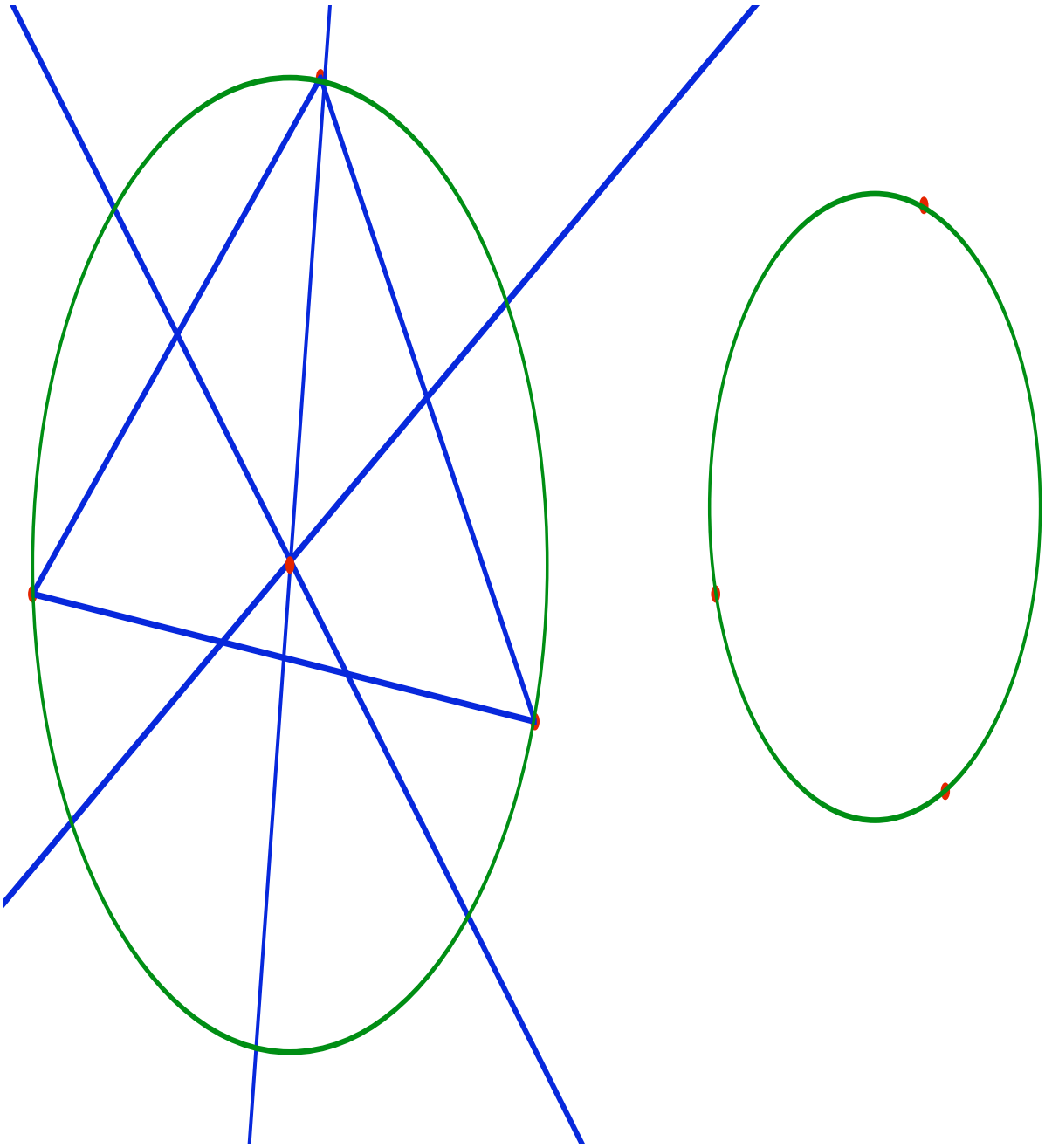


Figure I.2

Cabri-géomètre propose aussi la possibilité de créer des macro-constructions.

Exemple :
A partir de la figure I.2, il peut définir la macro-construction *Cercle passant par trois points* qui a pour objets initiaux les trois sommets et pour objet résultat le cercle (voir figure n). Il étend ainsi le micro-monde, la fonctionnalité *Cercle passant par trois points* devenant alors un nouvel objet de relation applicable à trois points quelconques.

Ces deux capacités, macro-constructions et fidélité par rapport au monde réel, permettent de désigner Cabri-géomètre comme un micro-monde. Cette fidélité n'est pas totale car les nécessités de la manipulation directe introduisent des propriétés nouvelles, par exemple une orientation du plan, mais cela n'est en rien contradictoire avec la définition de Papert et Minsky, bien au contraire.

Il faut insister sur la qualité de la réalisation de ce logiciel, érpouvé par de très nombreuses applications réalisées par des équipes très dynamiques de toutes provenances. Il est maintenant un des logiciels les plus répandus mondialement et en passe de devenir littéralement un standard.

2.1.4. GEOSPECIF

GEOSPECIF est un système encore au stade des recherches dans notre équipe. Ces recherches ont pour objet la construction d'un micro-monde de géométrie où les possibilités de manipulation directe soit les plus générales possibles [Allen et al. 93]. GEOSPECIF est construit à partir du langage PROLOGIII et manipule des équations sur les coordonnées des points.

Dans les micro-mondes de construction géométrique, comme Euclide, les possibilités de déplacement des objets sont limitées très naturellement par l'ordre de construction de ces objets.

Exemple :

Un point construit comme point d'intersection ne peut être déplacé car sa construction dépend entièrement de celles des objets dont il est l'intersection.

Géométriquement cela signifie que certaines relations géométriques entre les objets peuvent être construites de manières différentes selon l'animation escomptée.

Exemple :

L'appartenance de deux points A et B à une droite L est définissable par deux constructions, l'une construisant la droite L passant par A et B, l'autre posant les points A et B sur L. Dans le premier cas on peut déplacer A et B mais pas L, dans le second cas L mais ni A ni B.

Cette limitation des manipulations due à l'ordre de construction est surprenante pour l'apprenant. En effet, pour un apprenant non familier de ces micro-mondes, quoi de plus naturel que de vouloir manipuler un objet sans se soucier de la manière dont il a été construit mais simplement des relations géométriques qui les lient. On retrouve ici la différence entre le monde réel (les relations géométriques) et la représentation qu'en donne le micro-monde.

Définition :

Comme dans [Allen et al. 1993], nous dénommons les micro-mondes pour lesquels la manipulation de la figure dépend de l'ordre de construction des micro-mondes de programmation géométrique "impérative".

Les micro-mondes pour lesquels l'ordre de construction n'a pas d'influence sur les manipulations possibles sont appelles micro-mondes de programmation "déclarative".

En quelque sorte, on retrouve pour les micro-mondes la distinction usuelle entre langage impératif et langage déclaratif.

L'objectif de GEOSPECIF est de disposer d'un tel micro-monde déclaratif.

Il est clair bien sur que toutes les "envies de bouger" de l'apprenant n'ont pas forcément de sens géométrique immédiat. Pour clairement définir les mouvements possibles des objets d'une construction, et s'approcher plus encore des relations géométriques, GEOSPECIF distingue deux catégories de points du point de vue de la manipulation : des points "de base" et des points "liés".

Un point de base est un point dont les coordonnées sont fixées (provisoirement) pour la manipulation.

Un point lié est un point dont la position n'est pas fixée et dépend de ses relations avec les autres objets géométriques. Étant donné un ensemble donné de points de base

et de points liés, la manipulation de la figure est possible par déplacement d'un des points de base. Cette manipulation fait évoluer les positions des points liés et des autres objets de la figure.

Exemple :

La figure I.3 permet l'expérimentation des propriétés (géométriques) d'une lentille en optique. La droite L représente un horizon, le segment [A B] représente par exemple un arbre (A les racines, B le sommet) , [M N] est la lentille, |OF| sa distance focale et [A' B'] l'image de [A B] par la lentille. L'apprenant peut fixer comme point de base A, B et A' et décider de déplacer A' , les autres point étant implicitement désignés comme liés. Il peut alors observer l'effet de la modification de la focale, comme il l'aurait fait dans un micro-monde constructif. Mais il peut aussi décider que B n'est plus un point de base et que F est fixé. Il observe maintenant en déplaçant B' l'effet de la modification de la hauteur de l'arbre image sur l'arbre réel. Dans un micro-monde constructif, il lui aurait fallu définir une autre construction pour obtenir cette dernière animation.

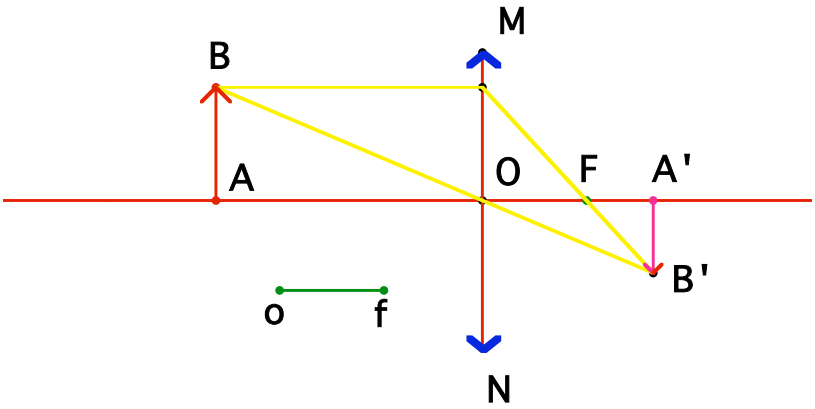


Figure I.3

En résumé, pour permettre des animations non liées à l'ordre de construction, les fonctionnalités attendues de GEOSPECIF sont :

- la possibilité de donner une spécification logique d'une figure géométrique.
- la capacité de construire automatiquement la figure correspondant (si elle existe et qu'elle est unique) à partir de points de base donnés.
- la capacité de proposer toutes les animations possibles sur cette figure.

On ne pourra sans doute pas attendre de GEOSPECIF des performances comparables à celle d'un micro-monde comme Cabri-géomètre.

2.1.5. CHYPRE.

CHYPRE [Bernat 93] (acronyme pour **C**onjecture **HY**pothèse **PRE**uve) est un système en cours d'élaboration se réclamant comme un "micro-monde pour l'apprentissage du raisonnement". Le prototype actuel concerne les preuves en géométrie, mais les idées qu'il met en œuvre se veulent applicables à des preuves sur d'autres types de domaines.

L'originalité de CHYPRE tient à la manipulation automatique du statut de propriétés géométriques qu'il propose. L'utilisateur de CHYPRE introduit dans le système des faits géométriques (comme *I milieu de [A C]*), issus de la construction qu'il élabore, auquels il attrribue un statut : soit hypothèse, soit conjecture. Le système manipule d'une part un ensemble de théorèmes, d'autre part un ensemble d'implicites pour chaque type de fait. Au moment de l'introduction d'un fait, CHYPRE associe au fait introduit les implicites

qui lui sont associés, puis cherche à appliquer un des thèorèmes dont les premisses soient les faits précédemment existants et donc la conclusion est le nouveau fait introduit. Si un théorème est appliqable à partir de premisses dont le statut est prouvé, le statut du fait introduit devient lui aussi prouvé. Concrétement, si la démonstration d'une propriété lui est demandée par un enseignant, alors l'objectif de l'élève est de donner à cette propriété le statut de prouvé dans CHYPRE.

En l'état actuel de CHYPRE, l'ensemble des implicites et l'ensemble des théorèmes manipulés par le système n'est pas accessible ni à l'apprenant, ni à l'enseignant. En intégrant cette possibilité dans une version future, CHYPRE constituera rééllement un micro-monde alors que la version actuelle ne permet qu'en partie à l'apprenant de construire sa propre connaissance. Plutôt qu'un micro-monde pour l'apprentissage du raisonnement, nous estimons que CHYPRE dans son état actuel constitue un micro-monde de manipulation de faits et de leur statut. Avec la possibilité de manipuler et de créer des théorème, CHYPRE correspondra complétement à la définition de micro-monde que nous avons donnée.

2.1.6. RMG.

RMG (Real time Measurement Graphics) est un environnement de simulation de phénomènes physiques simple ayant pour objet d'aider l'apprenant à comprendre des phénomènes difficiles à comprendre dans un environnement traditionnel d'enseignement [Bonnaire et al. 90]. Par exemple, cet environnement simule le mouvement des particules dans des champs de potentiel. Deux modes d'utilisation sont possibles: mode démonstration par l'enseignant en projection vidéo et mode exploration par l'apprenant sur console individuelle.

Dans le cas de RMG, même si les auteurs parlent d'une plate-forme de développement de micro-mondes, la principale exigence des micro-mondes - son évolutivité construite par l'apprenant - est absente. C'est sans doute cette absence qui permet un réel usage en mode démonstration. Les qualités de micro-monde de RMG doivent donc être considérées dans le cadre de la découverte collective, où une classe entière remplace l'apprenant des micro-mondes classiques.

Remarques conclusives.

Par la présentation de ces divers micro-mondes, nous voulons dégager l'évidence que laisser l'initiative à l'utilisateur d'un système n'est pas suffisant pour en faire un réel apprenant [Ruthven 94]. Comme le dit Vivet [Vivet 90], "la logique de transmission de connaissances utilisée seule ne fonctionne pas, pas plus d'ailleurs que la logique de construction personnelle de connaissances pour un élève largué seul dans un micro-monde". Balacheff précise que si "il est évident que sans contrainte extérieure, au moins une tâche *donnée*, il est improbable que l'élève apprenne quoi que ce soit" [Balacheff 94]. Les autres types d'EIAO que nous présentons maintenant proposent une tâche à l'apprenant.

2.2. Tuteurs Intelligents.

Les Tuteurs Intelligents (TI, ou encore ITS, Intelligent Tutoring Systems en anglais) sont situés à l'autre bout de l'échelle en terme d'initiative laissée à l'apprenant. Contrairement aux micro-mondes, ils lui proposent une tâche très précise. Ils se basent sur des capacités de raisonnement propres pour guider plus ou moins fortement l'élève dans sa tâche de résolution de problème.

2.2.1. Geometry Tutor.

Geometry Tutor [Anderson et al. 85] est l'exemple historique et typique d'un tuteur intelligent. C'est un produit commercial utilisée aujourd'hui aux États-Unis. La tâche de l'élève est de construire la preuve d'une propriété géométrique. Il dispose d'une figure représentant l'énoncé des hypothèses, qui est agrémentée au cours de la construction de symboles représentant certaines propriétés effectivement démontrées (Figure I.4).

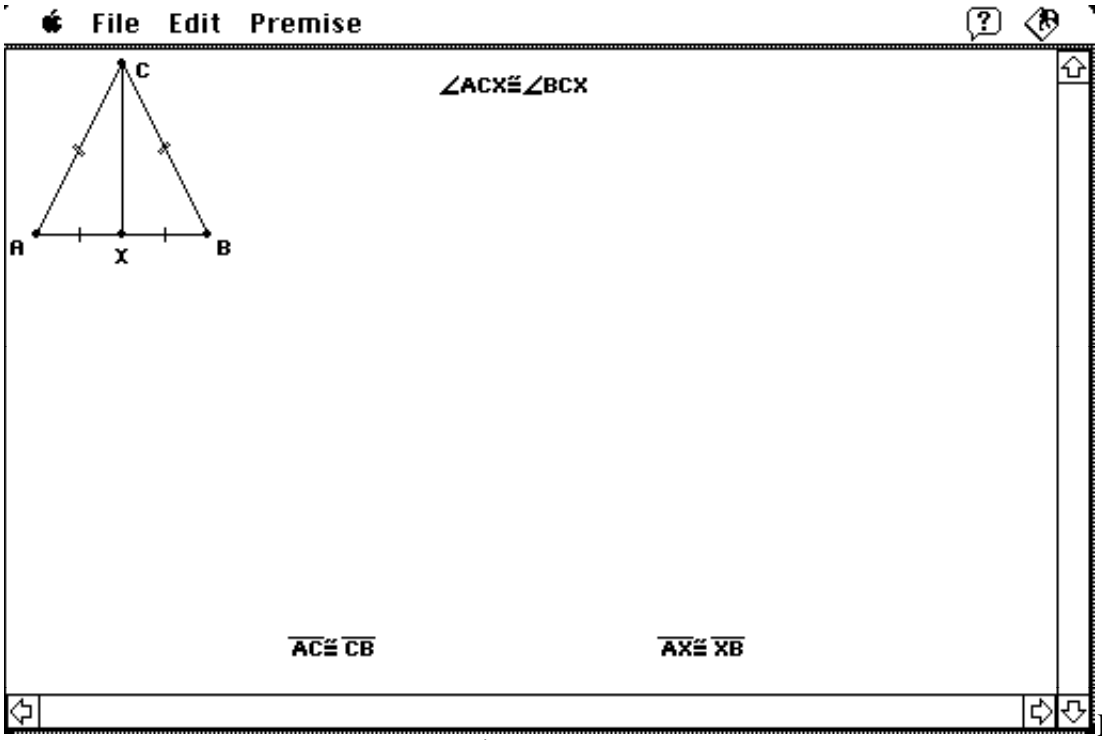


Figure I.4. Écran de Geometry Tutor.

Les principes de ce tuteurs sont issus d'une théorie de la cognition, ACT*. La géométrie n'est qu'un domaine d'application de cette théorie. Il existe d'ailleurs un autre tuteur fondé sur les mêmes principes, LISP-TUTOR, qui comme son nom l'indique a pour objet l'apprentissage du langage LISP. C'est donc ici un exemple de systèmes dont les caractéristiques se veulent génériques.

Du fait de l'originalité de l'approche de Geometry Tutor et son état de produit commercial répandu aux États Unis, il nous paraît intéressant de donner les principes les plus marquants de cette théorie ACT*.

• Le premier principe est que les fonctions cognitives peuvent être représentées par des règles de production. Sont ainsi à la fois représentées les règles correctes d'un domaine et leurs correspondantes incorrectes.
Exemple :
En LISP, *Pour fusionner deux listes en une seule utiliser la fonction APPEND* est une règle correcte et *Pour fusionner deux listes en une seule utiliser la fonction LIST* est la règle incorrecte correspondante).

Dans ce cadre, l'apprentissage se fait par *compilation* soit par procéduralisation de connaissances déclaratives en une règle de production, soit par composition de plusieurs règles en une seule.

• Le troisième principe est que l'élève n'apprend efficacement que lorsqu'il agit.
• Le sixième principe (paradigme du "model tracing") consiste à réagir immédiatement aux erreurs, en donnant une étape "réparatrice" à l'élève au bout d'un nombre

donné d'essais infructueux de l'élève. Ce principe est celui qui fait de Geometry Tutor l'archétype des tuteurs intelligents. Il exprime que l'élève est supposé rester très proche du modèle du système. Cette directivité est aussi le point faible de ce tuteur, car si elle convient bien aux novices, elle est mal appréciée des apprenants plus aguerris.

Conclusion

La directivité impliquée par le sixième principe vient contredire le septième principe d'ACT*, qui stipule que l'enseignement doit être adapté à l'expertise de l'élève [Anderson et al 87].

De plus, entre la version commerciale et sa description dans la littérature, *il y a un monde*, comme l'indique Cuppens [Cuppens 90]. Il explique, entre autres choses, que les démonstrations sont préenregistrées et en conclut qu'*on ne peut parler de tuteur intelligent, une condition minimale étant que la machine ait des connaissances suffisantes pour faire elle-même la démonstration.*

2.2.2. MENTONIEZH.

MENTONIEZH est lui aussi un tuteur pour la construction de preuves en géométrie [Py 90a]. Il est plus souple que GEOMETRY TUTOR□ d'une part il permet de construire la preuve par morceaux non nécessairement reliés aux hypothèses ou au but, d'autre part il autorise l'apprenant à emprunter plusieurs voies, en particulier celles que l'expert n'aurait pas empruntées (ce que GEOMETRY TUTOR n'autorise pas).

Le principe à la base de MENTONIEZH est la *reconnaissance de plan* [Kautz 87]. Partant d'une description incomplète des actions effectuées par un sujet, la reconnaissance de plan s'attache à retrouver le but recherché par le sujet, ainsi que le plan qui sous-tend et relie ces actions. MENTONIEZH cherche ainsi à identifier quelle démonstration parmi l'ensemble des possibles l'apprenant est en train de construire pour l'aider en cas d'erreur.

Comme GEOMETRY TUTOR, MENTONIEZH propose des explications à l'élève sur ses erreurs. La différence est qu'ici les règles incorrectes potentiellement utilisables par l'élève ne sont pas explicitement décrites : comme dans DEBUGGY [Burton 82], le système comporte des méta-règles permettant de construire une règle incorrecte en fonction de l'ensemble des règles correctes.

Une autre différence réside dans une plus grande souplesse du guidage : une aide n'est donnée qu'à la demande de l'élève. Dans le cas où un plan a été reconnu, elle consiste à indiquer la propriété suivante à démontrer dans le plan. Dans le cas contraire, l'élève est encouragé à explorer le problème depuis le but, en proposant des décompositions possibles en sous-buts.

2.2.3. APLUSIX.

APLUSIX [Nicaud 89] est un tuteur pour les factorisations d'expressions polynomiales. Son objectif principal est l'acquisition de stratégies de résolution par l'apprenant, le système le libérant des calculs pour mieux l'inviter à se concentrer sur le raisonnement. Il s'appuie sur un modèle général de raisonnement. Ce modèle de raisonnement est basé sur une analyse d'observations pédagogiques expérimentales et représente différents niveaux de connaissances.

APLUSIX comporte deux modes d'interaction□le mode apprentissage par l'exemple et le mode apprentissage par l'action [Nicaud et al 90].

Dans le mode apprentissage par l'exemple, l'élève observe le résolveur dans sa tâche. Le résolveur effectue le raisonnement pas à pas, un pas de calcul étant une opération de réduction, de développement, de développement-réduction ou de factorisation. Le rai-

sonnement est représenté par un arbre d'inférence - y compris les phases de raisonnement menant à un échec - dont les nœuds sont des expressions et les arcs la description des transformations appliquées. A chaque pas, l'élève choisit de demander soit plus d'explications, soit la suite du raisonnement.

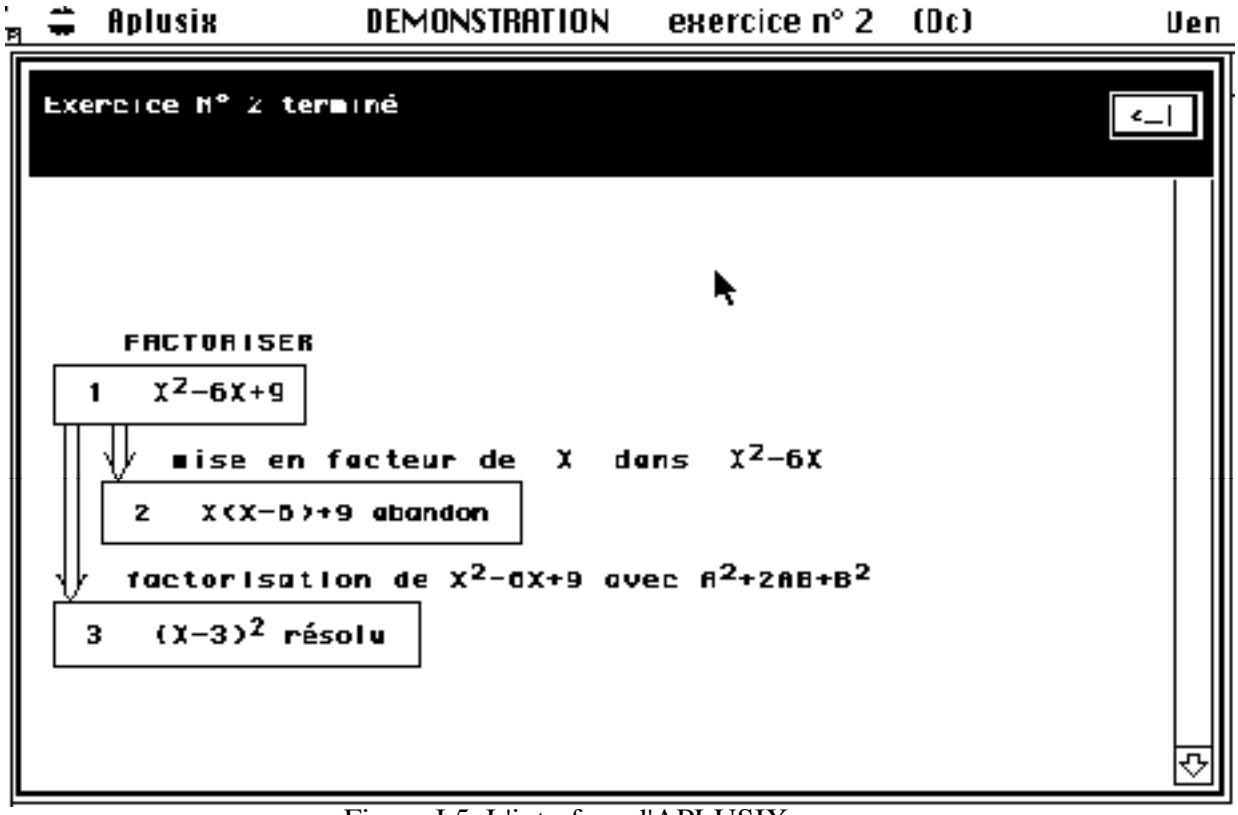


Figure I.5. L'interface d'APLUSIX.

Dans le mode apprentissage par l'action, l'élève développe sur sa propre initiative l'arbre de raisonnement en choisissant parmi les transformations possibles proposées par le système (voir figure I.5). Le choix de ces transformations ainsi que celui des explications à proposer à l'élève sont déterminés par des règles expertes et pédagogiques. Les règles expertes sont des règles de transformation d'expressions algébriques (comme l'utilisation des produits remarquables). Des règles pédagogiques tirées a priori de l'expérience pédagogique d'enseignants sont regroupées par profil pédagogique (débutant, moyen, expert). Elles décrivent le comportement du tuteur face à chacun de ces profils. Comme Geometry Tutor, APLUSIX intègre des règles erronées permettant d'analyser les comportements incorrects de l'élève et ayant pour but de lui expliquer ses erreurs. Par contre, le modèle du domaine ne comporte pas de plans explicites comme dans MENTONIEZH, l'évaluation de la situation de l'élève étant élaborée localement à partir de règles évaluant la qualité des choix effectués par l'élève.

Pour conclure, indiquons qu'APLUSIX est à l'heure actuelle à l'état d'un prototype sur lequel de nombreuses experiences didactiquess ont été menées, mais dont l'état d'achévement n'est pas encore suffisant pour en faire un produit commercial.

Remarques conclusives.

Du point de vue de l'intelligence artificielle, on ne peut discriminer les micro-mondes des tuteurs sur la simple base de l'absence ou de la présence dans le système de capacités de raisonnement. Même si aujourd'hui seuls des "presque-micro-mondes" (CHYPRE et GEOSPECIF, car ne comportant pas de macro-construction) possèdent ces capacités, le principal obstacle à l'intégration de raisonnement dans les micro-

mondes mis en avant - les performances du système - n'est pas forcément plus gênant que d'autres capacités déjà introduites et coûteuses elles aussi. Ce qui nous parait clair, par contre, c'est que la philosophie des micro-mondes est contradictoire avec l'introduction de capacités de vérification d'une solution à un problème donnée par un apprenant, la notion de problème n'ayant pas sa place dans un micro-monde.

Pour notre part, nous proposons donc de caractériser les tuteurs intelligents par cette capacité à évaluer la réalisation d'une solution à un problème. La caractéristique d'un micro-monde serait ainsi la limitation de ses capacités de raisonnement à la vérification de la cohérence des constructions possibles. Par exemple, les capacités de vérification concernant l'égalité d'objets géométriques présentes dans notre tuteur peuvent être rapidement intégrées dans Cabri-géomètre (ou dans tout autre micro-monde de géométrie). En considérant isolément la partie saisie de la figure de TALC, on obtient immédiatement un tel micro-monde permettant aujourd'hui à Cabri-géomètre en particulier de disposer d'une vérification géométrique de l'égalité d'objets à la place de la vérification syntaxique actuelle (voir deuxième chapitre, 2.3.2.3).

2.3. Tuteurs de découverte guidée.

Deux reproches principaux ont été faits aux tuteurs intelligents [Balacheff 94]:

• le premier est que les réponses du tuteur à l'apprenant font plus référence aux connaissances de référence qu'aux connaissances de l'élève et à leur évolution.

• le second est que l'apprenant peut obtenir une évaluation excellente du point de vue du tuteur sans acquérir réellement la connaissance en jeu dans la tâche proposée. Il s'agit là d'un détournement de tuteur, comme la méthode consistant à toujours demander l'aide disponible et ainsi résoudre le problème.

D'un autre côté, on reproche aux micro-mondes leur neutralité vis-à-vis d'un objectif didactique.

C'est en partant de ces reproches faits à ces deux approches extrêmes que les *tuteurs de découvertes guidées* ont vu le jour. Un tuteur de découverte guidée [Elsom-Cook 90] est un système où la directivité est adaptée à l'état de connaissance de l'apprenant, pouvant passer d'un comportement de micro-monde à celui d'un tuteur intelligent.

Les deux tuteurs de découverte guidée que nous présentons ici utilisent le même micro-monde de géométrie, Cabri-géomètre.

2.3.1. DEFI-CABRI

DEFI-CABRI [Baulac et al. 91] est un système en cours d'expérimentation utilisant à la fois Cabri-géomètre (voir 2.1.3) et DEFI (voir 3.2.1).

Après que l'apprenant ait construit une figure dans Cabri-géomètre, CABRI-DEFI l'évalue et guide l'apprenant dans sa tâche de preuve. Le comportement du système est donc celui d'un micro-monde pour la construction d'une figure correspondant à l'énoncé et celui d'un tuteur pour l'élaboration d'une preuve, ce qui nous amène à le désigner comme un tuteur de découverte guidée.

Comme DEFI-CABRI ne possède pas de capacités de raisonnement, ses capacités en sont d'autant limitées. D'une part le diagnostic de correction de la figure construite est rudimentaire, obligeant l'élève à construire "mot à mot" les objets demandés. D'autre part les capacités d'explorations sont limitées à l'ensemble des possibilités prévues par les concepteurs de DEFI. En effet, comme DEFI ne permet pas la création par l'enseignant de ses propres exercices, seuls ceux définis par les concepteurs sont disponibles, ce qui en fait un système très peu ouvert.

2.3.2. HYPERCABRI

HYPERCABRI [Capponi 91] est un prototype organisant une séquence didactique pour le problème de la construction d'un carré. L'objectif pour l'élève est la construction d'une figure géométrique ayant les propriétés logiques d'un carré. L'objectif sous-jacent est l'acquisition de la notion de cercle comme ensemble de points équidistants d'un point donné.

HYPERCARRE est construit à partir d'une pile HYPERCARD faisant appel à Cabri-géomètre. Après que l'élève ait construit une figure censée répondre à la définition d'un carré, sa construction est évaluée pour, en cas d'insuccès, lui proposer soit un sous-problème ayant pour objet de lever sa difficulté, soit une aide directe en rapport avec sa production. La décision de la réaction à adopter par le tuteur est basée sur une analyse préalable de l'ensemble des situations possibles et des conceptions de l'apprenant sur ces situations.

Au vu des nombreux problèmes rencontrés par la mise en œuvre d'un prototype pour une situation d'apprentissage extrêmement restreinte, HYPERCABRI montre la complexité des problèmes à résoudre, d'une part du côté didactique, pour que les réactions du système soient adaptées à l'élève, d'autre part du point de vue informatique, pour obtenir un diagnostic correct et rapide sur la correction de la construction.

L'objectif d'HYPERCABRI peut être compris comme un sous-objectif de TALC. En effet, dans TALC, il suffit de définir une spécification représentant un carré pour être placé dans la même situation. La différence entre TALC et HYPERCABRI tient surtout à l'objectif□alors que dans HYPERCABRI il s'agit de valider des hypothèses didactiques sur les EIAO, l'objectif premier de TALC est de fournir un diagnostic complet dans le cadre d'un système général de construction de figures.

Remarques conclusives.

Le travail que nous présentons dans cette thèse aboutit lui aussi à un tuteur de découverte guidée. Comme les précédents, il associe des capacités de micro-monde (en l'occurrence Cabri-géomètre) et des capacités de tuteur.

Cette intégration d'un micro-monde préexistant dans un tuteur de découverte guidée est la source de nouvelles questions pour l'EIAO, sur le plan informatique aussi bien que sur le plan didactique. Elles vont sans doute mener à la définition de moyens de communication génériques entre micro-mondes et tuteurs [Senet 93]. De tels moyens de communication permettront d'utiliser dans un système les fonctionnalités d'un autre sans nécessiter la mise en œuvre de lourdes programmations.

3. Intégration de capacités de raisonnement

Comme nous venons de l'exposer, un nombre important d'EIAO aujourd'hui intègre des capacités de raisonnement.

En effet il est fondamental que les EAIO, que ce soient des micro-mondes ou des tuteurs, puissent décider si une propriété est valide ou non. Comme nous l'avons vu dans le chapitre précédent, pour un micro-monde cela permet de valider la cohérence d'une étape de construction relativement aux propriétés intrinsèques du micro-monde. Pour un tuteur cela consiste majoritairement à valider les solutions proposées par l'apprenant au problème qui lui est soumis. Un système ne disposant pas de capacités de raisonnement voit le champ de ses interactions avec l'apprenant fortement restreint. Dans le cas des micro-mondes, un tel système ne peut vérifier que des propriétés locales, très partielles et fortement liées à ses structures internes. Alors que dans un EAO on peut donner *a priori* l'ensemble des solutions à un problème, dans un EIAO, on ne peut en général donner *a priori* l'ensemble des solutions à un problème. L'ajout de capacités de raisonnement est indispensable pour vérifier *a posteriori* qu'une solution est acceptable, même si elle était imprévisible. Cela permet de faire passer l'éventail des solutions admissibles à un problème donné d'un ensemble restreint à un ensemble bien plus vaste et plus réaliste.

Intégrer des capacités de raisonnement n'est pas aisé. Cela peut sembler une évidence et pourtant il y a peu d'EIAO pour lesquels les capacités de raisonnement sont clairement définies. Les auteurs s'en tiennent souvent à une simple description en termes de "règles de productions" manipulées par un "système expert". Si l'on veut dépasser cette vue opératoire, il faut pouvoir définir la cohérence et surtout la complétude de ces raisonnements.

Disposer de raisonnements cohérents, cela signifie que si le système décide qu'une propriété est vérifiée, il ne peut décider en même temps que son contraire l'est. Même si le raisonnement humain n'est pas forcément cohérent, l'attente minimale d'un apprenant face à un système automatique (sinon même vis à vis d'un enseignant humain!) est qu'il soit cohérent.

Disposer de raisonnements complets, cela signifie que le système est capable pour n'importe quelle propriété exprimable, de décider si elle est valide ou non. Dans le cas des tuteurs, la complétude signifie que pour toute solution correcte donnée par l'apprenant, le système la reconnaît comme telle. La complétude est donc une qualité évidemment attendue de tout tuteur mais difficile à obtenir d'une part parce que sa définition peut être ardue et d'autre part parce qu'algorithmiquement le problème est ardu.

Commentons quelques points de vue et affirmations à partir d'une des rares réflexions sur la complétude des EIAO [Bruillard 90]□

• *On peut sans conteste affirmer que les programmes se limitant aux opérations numériques sont complets*. Cette affirmation est malheureusement fausse en général. Elle n'est vraie que si la façon de faire les calculs est complète et ces calculs sont exacts. Les calculs sur les nombres flottants sont par exemple inexacts. Bruillard précise *On peut simplement rencontrer des problèmes de précision, mais les machines dépassent largement les capacités humaines*. Ceci est démenti par le fait que la simple construction d'un polygone régulier en LOGO pose rapidement des problèmes de précisions, le premier côté ne joignant pas exactement le dernier. Même des micro-mondes plus récents ont de graves problèmes vis-à-vis de la précision des calculs, dès que le nombre d'objets est important.

Exemple :

La figure I.6, produite à partir de Geometer's Sketchpad [Jackwick 89] montre comme la précision des calculs peut amener des problèmes didactiques importants. Bien que Geometer's Sketchpad soit un logiciel de principes similaires à ceux de Cabri-géomètre, les problèmes ces problèmes ne se retrouvent pas dans Cabri-Géomètre. Cette différence importante montre aussi le rôle esentiel des informaticiens dans la conception de tels logiciels.

D'après la construction faite pour obtenir cette figure, il est prouvé que les cercles de centre A et F respectivement sont tangents (c'est d'ailleurs le but de l'activité proposée). Le problème est que l'élève a du mal à être convaincu de la validité de la propriété qu'il doit démontrer, puisqu'elle est fausse expérimentalement sur cette figure. La non-tangence des cercles de centre A et F (respectivement) est due ici à l'accumulation d'erreurs de calcul, la construction comportant environ 100 objets intermédiaires.

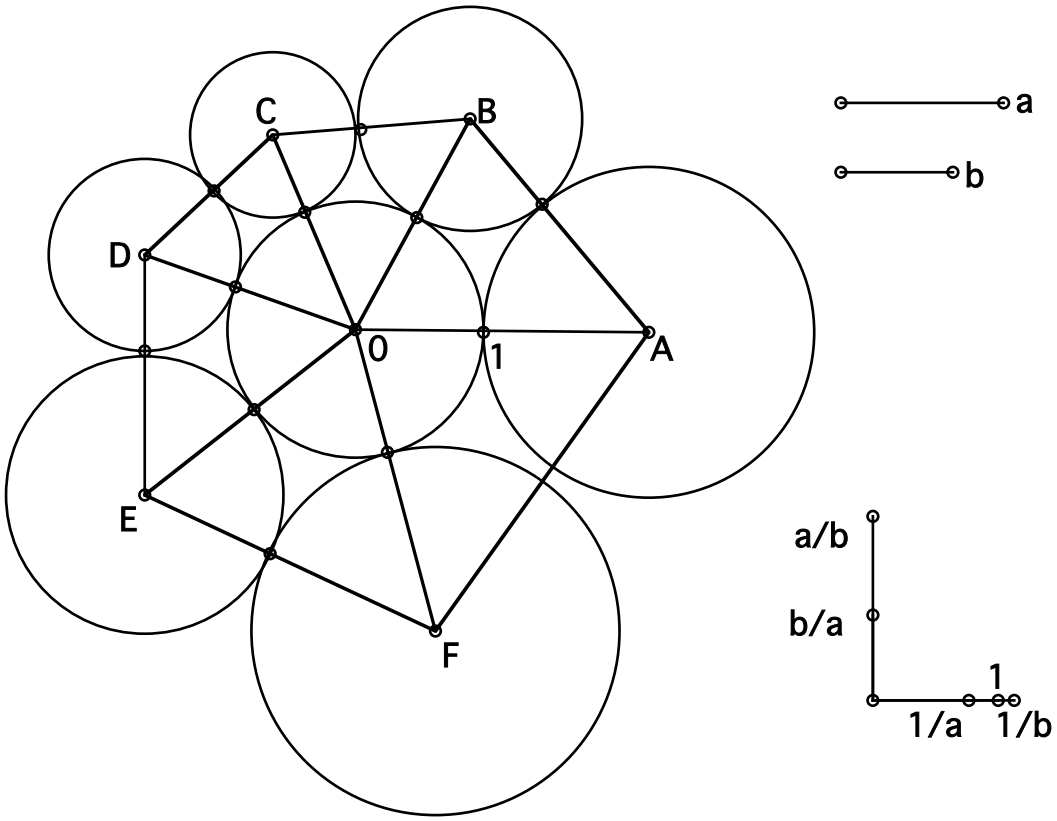


Figure I.6

La même figure construite dans Cabri-Géomètre est exacte et permet de contrôler expérimentalement que les cercles de centres respectifs A et F sont bien tangents, quelles que soient les conditions de manupulation.

• *Un système peut ne pas être capable de résoudre un problème parce qu'il nécessite l'emploi de méthodes non connues du système*.

Remarquons tout d'abord que le plus souvent, on se rend compte qu'un EIAO est incomplet quand il refuse une solution correcte de l'apprenant. On voit alors que la complétude est une propriété dont on ne peut parler sans faire référence aux connaissances d'un domaine (désignées par *théorie* en logique), par rapport auxquelles on peut détecter les lacunes du système. En particulier, une propriété peut être vraie par rapport à une théorie particulière sans l'être dans une autre (la géométrie en est un exemple typique, suivant que l'on se place dans la géométrie d'Euclide ou celle de Riemann).

Si l'on se place dans la théorie que décrivent les "méthodes" du système, alors inévitablement tout système est complet. Le véritable problème est donc de savoir si les méthodes contenues dans le système traduisent fidèlement les méthodes du domaine. Du point de vue didactique, il s'agit de savoir si les raisonnements du système sont bien conformes aux raisonnements que lui prête l'apprenant. Deux situations mènent à une non conformité□ soit les conceptions de l'apprenant sont erronées par rapport à la théorie et l'EIAO a pour tâche d'amener l'apprenant à les reconsidérer, soit l'EIAO ne possède pas les capacités qu'il déclare plus ou moins clairement posséder. C'est dans ce dernier cas que nous parlerons d'incomplétude.

L'objet de ce sous-chapitre est d'examiner les EIAO selon cette distinction entre systèmes complets et ceux n'assurant pas la complétude. Nous exposons ainsi d'une part des systèmes que nous jugeons ne pas assurer la complétude et les justifications invoqués par leurs auteurs quand ils s'en sont préoccupés. D'autre part nous présentons les systèmes que nous jugeons complets et les restrictions qu'ils présentent en vue d'obtenir cette complétude.

3.1. EIAO intégrant des capacités de raisonnement.

Il faut noter tout d'abord que pour un certain nombre d'EIAO, la question de la complétude ne se pose pas, simplement parce qu'ils n'intègrent aucune capacité de déduction. C'est ainsi le cas de LOGO et d'EUCLIDE, qui peuvent être vus comme des langages de programmation géométriques de style impératif et possédant à ce titre des capacités de vérification purement syntaxiques. Parmi les autres micro-mondes que nous avons déjà présentés, RMG et Cabri-géomètre non plus ne comportent aucune capacité de raisonnement, même concernant la cohérence des objets manipulés. Parmi les tuteurs de découverte guidée, HYPERCABRI ne possède pas lui non plus de capacités de raisonnement. L'évaluation de la figure construite qu'il effectue est basée sur une analyse préalable des situations possibles.

3.1.1. Cabri-géomètre.

Bien qu'il ne dispose pas de capacité de raisonnement stricto sensu, Cabri-géomètre détecte un certain nombre d'incohérences syntaxiques et numériques.

Il propose en outre à l'apprenant un item de *Vérification de propriété*, appelé *oracle*. Cet oracle propose la vérification de cinq propriétés (appartenance, parallélisme, perpendicularité, alignement, longueurs égales). Il donne quatre types de réponses□

- la propriété est visiblement fausse.
- □a propriété est visiblement vraie mais fausse dans le cas général. Suit la proposition d'un contre-exemple.
- □a propriété est visiblement vraie, est fausse dans le cas général mais Cabri ne peut donner de contre-exemple où elle soit visiblement fausse. Ce cas se produit par exemple quand la construction illustre un phénomène de limite.
- □a propriété est vraie dans le cas général.

C'est sur cette dernière réponse principalement que la complétude doit être précisée. Cette complétude ne doit pas être entendue au sens courant de la géométrie euclidienne, mais au sens de la géométrie particulière définie dans Cabri-géomètre, en particulier vis-à-vis de la notion d'animation de la figure. En fait, les concepteurs de Cabri-géomètre se réfèrent à un contexte probabiliste: l'oracle ne s'appuie pas sur un raisonnement (La documentation du logiciel stipule d'ailleurs clairement que *Cabri-géomètre ne peut pas renseigner l'utilisateur sur la manière dont la réponse est obtenu*e), mais donne sa réponse après des tests fondés sur des probabilités. Cet oracle donne de bons résultats en pratique et il faut lui présenter des cas très particuliers pour le mettre en défaut.

C'est la raison pour laquelle, bien qu'on ne puisse considérer les réponses de l'oracle de Cabri-géomètre comme complètes vis-à-vis de la géométrie euclidienne, l'utilisation de cet oracle est didactiquement fondée.

3.1.2. Geometry Tutor

Le raisonnement dans Geometry Tutor est basé sur un ensemble de règles de production représentant les connaissances d'un expert et leur variante erronée. A chaque étape de la construction d'une démonstration par l'apprenant, le tuteur génère toutes les possibilités d'application d'une règle dont il dispose. Il compare ensuite le pas de démonstration de l'apprenant avec celles-ci. S'il ne correspond à aucune possibilité, Geometry Tutor demande la modification du pas de démonstration. Au bout d'un certain nombre d'essais infructueux, il construit lui-même le pas par rapport à ce qu'il considère être la meilleure règle.

Ce comportement de Geometry Tutor assure une complétude locale. Il est appelé *Model Tracing* par ses auteurs, et modélise le fait que le comportement de l'élève est supposé rester très proche du modèle du système. Par contre, il n'assure pas une complétude globale : le système ne cherche pas à trouver une solution en revenant sur les choix antérieurs.

3.1.3. Archimède

Archimède est un projet de recherche consacré à la réalisation d'un tuteur de géométrie élémentaire [Chouraqui et al. 90] fondé sur un raisonnement par analogie.

Le raisonnement par analogie fonctionne sur le principe suivant : il s'agit de mettre en relation d'analogie une situation cible, c'est à dire une situation où l'apprenant a à résoudre un but B sous les hypothèses H, avec une situation source, c'est à dire une situation où le but B1 est résolu sous les hypothèses H1. Cela correspond à établir le fait que B est à H ce que B1 est à H1. Si H1 est reconnu "similaire" à H alors on peut dire que B est établi. La similarité est établie entre deux objets si le "rapport de similarité" entre B et B1 dépasse un certain seuil (0.7 dans Archimède).

Archimède dispose au départ d'un ensemble de théorèmes qu'il enrichit en fonction des problèmes (ou sous-problèmes) que l'apprenant a déjà résolus. Il utilise une représentation à objets des figures géométriques et des théorèmes.

Il est clair que la complétude au regard d'une théorie de la géométrie n'est pas assurée dans Archimède, du simple fait de l'utilisation d'un seuil dans le calcul de la similarité entre objets. Cela tient au fait que la complétude ne fait pas partie des préoccupations des auteurs. Leurs objectifs sont surtout d'obtenir une modélisation de l'activité cognitive de l'apprenant, lui permettant de résoudre de nouveaux problèmes par rapport à des expériences acquises. La complétude peut aussi difficilement être définie car aucune définition de l'enrichissement des théorèmes ne peut être donnée dans ce cas, sinon en termes opérationnels.

3.1.4. APLUSIX

Dans APLUSIX, les capacités de raisonnement sont représentées dans des règles de transformation d'expressions algébriques : règles de production et règles de réécriture. Elles expriment les possibilités données à l'apprenant de transformer des sous-termes d'une expression.

En mode apprentissage par l'exemple, pour une expression donnée, le système :

- détermine les règles potentiellement applicables.
- choisit une de ces règles de façon heuristique en fonction de règles didactiques et l'applique.

•décide si l'expression est "factorisée au maximum ".
Définition :
Est "factorisé au maximum" :
- un polynôme de degré inférieur ou égal à un;
- un polynôme AP+ BQ tel que A et B sont des constantes, P de degré 2, Q une expression constante et AB>0.
Exemple :
$3 (2x^2 - x^2) + 2 (4+1)$
- un produit de polynômes factorisés au maximum.

Si l'expression est factorisée au maximum, le problème est résolu sinon le système reprend le processus sur la nouvelle expression. Le système s'arrête aussi dans le cas où soit il n'y a plus de règles à appliquer, soit un nombre borné (fixé par APLUSIX) de règles a déjà été utilisé (12 exactement).
En mode apprentissage par l'action, le système propose à l'apprenant un choix de transformations et le guide à la demande, en lui indiquant des transformations dont la qualité est indiquée (*moyen, bien, très bien*).

Le fait que le système choisisse des règles uniquement en fonction d'heuristiques sans les essayer toutes signifie en soi que APLUSIX n'est pas complet par rapport à cet ensemble de règles. Le fait qu'il y ait abandon après utilisation d'un certain nombre de règles le montre aussi.
Exemple :
En prenant comme théorie les factorisations et développements simples sur les nombres entiers et les identités remarquables de base, l'expression x^2-x-1 est déclarée non entièrement factorisée par le système. Pourtant il est clair qu'elle ne peut être plus factorisée qu'elle n'est. Le problème dans ce cas est que l'expression ne rentre pas dans un des cas où APLUSIX décide de la factorisation réussie. Comme des règles sont heuristiquement applicables, alors le système décide localement qu'il est meilleur de continuer, alors qu'il est globalement équivalent de continuer ou non.

Les auteurs donnent trois justifications principales à cette incomplétude:
•La première est historique. Elle tient à la priorité accordée à la modélisation de plusieurs niveaux de compétences tirées d'une analyse d'observations pédagogiques. Les auteurs d'APLUSIX se sont concentrés sur cette modélisation et n'ont pas envisagé la question de la complétude dans sa définition.
•La seconde est liée aux contraintes de l'interactivité : du fait du grand nombre de règles, l'utilisation d'heuristique permet de limiter l'explosion combinatoire de la recherche. De plus il est possible qu'une exploration systématique des règles applicables ne finisse pas, du fait de l'utilisation simultanée de règles de factorisation et de développement (problème de la terminaison de la réécriture).
•La troisième vient de l'objectif principal du système qui est de favoriser l'apprentissage des heuristiques elles-mêmes. Le système est donc construit autour de recherches heuristiques et ne vise pas à la complétude a priori.

Remarques conclusives.

Vis-à-vis de la question de la complétude, ce qui nous semble essentiel est la définition précise du contrat didactique que propose l'enseignant à l'apprenant (voir chapitre deuxième, quatrième partie). Si cette définition est claire, l'évaluation de la complétude du système est aisée.

Exemple :

La définition d'une expression factorisée que manipule APLUSIX peut être difficilement comparée avec celle de l'enseignant, car il n'en donne pas une expression explicite, pas plus que les manuels scolaires!

La question de la complétude montre une des conséquences de la définition et de l'utilisation de EIAO : comme cela a lieu dans d'autres domaines, l'introduction d'un système informatique oblige a définir précisement l'objet manipulé par le système, alors que sa définition était jusque là implicite. C'est d'autant plus crucial dans un problématique d'enseignement, car si les bons élèves arrivent à deviner les règles et les définitions implicites, l'explicitation serait très profitable pour les autres.

Nous verrons dans la deuxième partie de ce document que cette question de la définition précise du contrat didactique est aussi une des questions centrales de la définition de TALC.

L'exemple d'APLUSIX illustre d'ailleurs bien une difficulté importante de la définition et de la mise en œuvre des EIAO : la liberté offerte par un système de définir des exercices à volonté rend délicate la réalisation d'un système complet. En effet, il n'est pas certain que les exercices proposés aient toujours une solution du point de vue du système. Le problème est qu'implictement le fait d'exprimer un problème signifie habituellement pour l'élève qu'il en existe une solution. La position du système informatique dans la situation didactique comme représentant de l'enseignant fait que l'élève s'attend au même présupposé alors que cela est loin d'être garanti. Dans APLUSIX, par exemple, si l'on pouvait rejeter les énoncés d'exercice pour lesquels le système considère qu'il n'y a pas de solution, la complétude interne serait mieux assurée.

La question de savoir si un exercice admet une solution au sens du système est aussi une de nos préoccupations dans la définition de TALC (voir chapitre deuxième, 4.3).

3.2. EIAO intégrant des capacités de raisonnement complètes.

Nous présentons dans ce sous-chapitre des EIAO dont la définition s'appuie explicitement ou non sur une certaine complétude. Nous présentons pour chacun tout d'abord la définition des capacités de raisonnement pour ensuite expliquer de quelle sorte de complétude il est question.

3.2.1. DEFI

DEFI [Gras 88] est un tuteur d'aide à la démonstration de géométrie en classe de quatrième qui propose deux activités par rapport à une figure donnée:

• exploration de la figure. DEFI propose à l'élève plusieurs façons de décomposer le problème en sous-problèmes. L'élève choisit celle qui lui semble convenir. Il doit déclarer ensuite savoir ou non démontrer chacun des sous-problèmes de la décomposition choisie. S'il déclare ne pas savoir démontrer un sous-problème, celui-ci devient le problème courant. Le processus est réitéré jusqu'à ce que l'élève ait déclaré savoir démontrer le problème initial.

• démonstration. L'élève construit effectivement la démonstration du problème, de but en sous-buts.

La connaissance dans DEFI est représentée par un ensemble de théorèmes prédéfinis pour la classe de quatrième. Par rapport à cette théorie, le système est a priori complet car, par définition, toutes les possibilités de démonstration sont proposées à l'élève. En réalité ce n'est pas le cas comme l'explique Luengo [Luengo 93], car les possibilités de démonstration ont été prédéfinies par les concepteurs du système, ce qui ne garantit pas toujours la complétude.

3.2.2. MENTONIEZH

MENTONIEZH dispose comme DEFI d'un ensemble de théorèmes utilisables. Pour un exercice défini par l'enseignant, le système effectue les calculs préalables suivants:

• production de toutes les propriétés déductibles à partir des hypothèses du problème par application des théorèmes valides sous la forme d'un graphe.

• énumération des preuves possibles du problème, c'est à dire de l'ensemble des chemins du graphe menant des hypothèses à la conclusion. Seules sont retenues les preuves dont la longueur n'excède pas une borne raisonnable fixée par l'enseignant.

La complétude est donc assurée si on considère qu'un théorème n'est pas démontrable par une preuve trop longue au sens précédent. La justification de l'utilisation d'une borne dans les longueurs de preuves est ici aussi liée à une question d'explosion combinatoire. Elle se justifie aussi par le fait que les preuves de plus de trois pas ne sont pas choisies par les élèves du fait de leur complexité.

En tout état de cause, la complétude est formellement assurée si le professeur fixe une limite plus grande que la profondeur maximale des preuves possibles à partir des hypothèses. Ce nombre est en effet fini, car MENTONIEZH, comme la plupart des tuteurs de preuve en géométrie que nous avons présentés, n'introduit pas d'objets autres que ceux présents dans les hypothèses et manipule donc un nombre fini de propriétés potentielles sur les objets géométriques. Le nombre de nœuds de l'arbre de profondeur étant fini, sa profondeur maximum aussi.

De notre point de vue, la complétude proposée par MENTONIEZH, même si elle est restreinte, a l'avantage d'être à la disposition de l'enseignant. Il peut ainsi, selon l'exercice qu'il pose, définir une complétude totale ou restreinte. Reste la question difficile de savoir quels moyens l'enseignant a pour évaluer l'impact des choix qu'il retient sur l'interaction du tuteur avec l'élève. C'est là aussi une question que nous développons dans la seconde partie de ce document.

3.2.3. CHYPRE

Malgré l'affirmation de son auteur qu'il ne comporte *par principe aucun résolveur*, la version actuelle de CHYPRE, basée sur un ensemble de théorèmes figés, gère pourtant automatiquement le passage du statut conjecture au statut prouvé d'un fait (*module de résolution*). Comme l'interface ne fournit aucun moyen à l'utilisateur de connaître ces théorèmes, il faut supposer que l'enseignant a accès (dans le manuel de référence par exemple) à l'ensemble de ces théorèmes pour pouvoir parler de complétude.

Les déductions effectuées par CHYPRES sont limitées à un seul pas. En effet, à l'introduction d'un fait, CHYPRE essaie de relier ce fait à des faits déjà présents par une théorème. L'ensemble des implicites reliés aux faits joue une rôle important dans la construction de ce pas de preuve, car l'algorithme ne procéde pas de la même façon suivant qu'un fait est un implicite ou un explicite. C'est la sémantique assez complexe de la manipulation des implictes dans CHYPRE qui ne permet pas de savoir par rapport à quoi définir une complétude. Le fait qu'un seul pas de preuve soit appliqué à chaque fait ajouté rend aussi difficile l'expression d'une complétude, généralement comprise par rapport à un preuve au sens général. Il noter au passage que l'ordre d'introduction des faits joue un grand rôle quand au statut conjecture ou prouvé de ces faits. En particulier, si l'utilisateur considère qu'un fait devrait être prouvé du fait des faits précédemment introduits, il doit comprendre que l'ordre de ses introductions ne permet pas d'attribuer un statut prouvé à ce fait et "réintroduire" les faits dans un ordre satisfaisant.

Remarques conclusives.

Ces trois exemples montrent que l'exigence de la complétude est difficile. Elle nécessite non seulement une définition claire et précise des déductions que fait le système (comme ce n'est pas le cas pour CHYPRE par exemple) mais aussi de faire des choix assurant au système de performances correctes (comme dans DEFI et MENTONIEZH), du fait de la combinatoire induite par la complétude.

Pour assurer cette complétude, nous avons ainsi dû définir précisément les déductions possibles dans TALC et les choix que nous avons retenus (voir chapitre deuxième, quatrième partie).

4. Intégration de théories

Si l'idée que les EIAO pourraient se substituer au professeur a pu avoir un certain crédit il y a quelques années, il n'en est aujourd'hui plus question. La vision qui substituait l'EIAO au professeur doit être remplacée par un autre plus complexe où est définie la place de chacun dans la situation didactique. Définir précisément les responsabilités de chacun consiste à décrire d'une part les choix que le système impose et d'autre part les degrés de liberté dont dispose l'enseignant.

Un premier degré de liberté est la possibilité de construire ses propres exercices. La plupart des EIAO offrent cette liberté aujourd'hui. Un deuxième degré de liberté, pour les EIAO disposant de capacités de raisonnement, est la possibilité de modifier la théorie. Ce deuxième degré de liberté nous semble être un moyen de distinction important des EIAO actuels. D'une part les conséquences didactiques qu'il entraîne sont importantes : ne pas pouvoir faire varier la théorie, c'est fournir un système dont le corpus de connaissances est figé une fois pour toutes, c'est proposer un système dont l'adaptation à l'évolution de l'élève est limitée (si l'on exclut les micro-mondes). D'autre part la présence ou non de cette liberté est un choix fondamental pour un EIAO, qui entraîne de nombreuses conséquences sur le plan de la mise en œuvre et qui ne peut donc être remis en cause facilement. Cette distinction nous permet ainsi de toucher du doigt de nouvelles justifications capitales des choix effectués pour chaque EIAO.

Nous présentons en premier lieu des EIAO pour lesquels la théorie est fixée une fois pour toutes et ceux autorisant la modification de la théorie.

4.1. EIAO intégrant des théories fixes

Parmi les EIAO n'autorisant pas la modification de la théorie, nous trouvons tout d'abord les premiers historiquement développés. Cela vient du fait qu'il est plus simple pour mettre à l'épreuve une réalisation de restreindre le cadre de son application. Ainsi Geometry Tutor, dont l'objectif premier était la mise en œuvre d'une théorie cognitive, propose à l'apprenant la résolution d'un ensemble d'exercices préétablis à l'aide d'un ensemble de théorèmes prédéfinis, exercices et théorèmes ne pouvant être modifiés par l'enseignant (ils sont issus d'un programme scolaire bien précis). Nous avons déjà souligné la contradiction de Geometry Tutor avec cette théorie de la cognition, où un des principes est précisément l'adaptation du tuteur à l'apprenant.

La plupart des micro-mondes présentés n'ont pas non plus cette faculté, en considérant que la théorie est celle sous-jacente aux vérifications de cohérence. Cela n'enlève pas leurs qualités sur le plan de l'apprentissage, du fait de leurs capacités d'adaptation à l'apprenant via les procédures ou macro-constructions.

Dans GEOSPECIF, la théorie est la théorie générale de la géométrie, du fait de la représentation des relations géométriques par des relations entre coordonnées. Ce système de représentation ne permet ni d'augmenter ni d'affaiblir la théorie. Par contre l'incomplétude du résolveur concernant les relations non-linéaires peut être améliorée par l'utilisation d'axiomes existentiels pour ajouter des objets à la construction. Cet ensemble d'axiomes existentiels pourrait être lui modifié. La problématique de la manipulation de cet ensemble d'axiomes est très proche de celle que nous développons au chapitre deuxième, 4.1.2.3, pour l'ajout d'objets non prévus par le professeur.

Pour CHYPRE, on s'attend naturellement à voir la théorie évoluer en fonction des connaissances acquises par l'apprenant. Nous avons vu que ce n'est pas le cas, car la théorie de CHYPRE est fixée et que les macro-constructions de théorèmes ne sont pas possibles.

DEFI ne permet pas non plus de modifier la théorie, à moins de demander à ses auteurs une version spéciale comprenant des axiomes voulus. Mais alors il est évident que cette modification lourde ne peut-être utilisée pour s'adapter à chaque nouveau type de problèmes. Le point de vue de DEFI est avant tout de couvrir le programme de quatrième.

Remarques conclusives.

L'argument le plus fort en faveur d'une théorie fixe tient aux nécessités de l'interactivité. Il semble clair qu'on peut garantir les performances d'un système conçu avec une théorie ad hoc. Notre point de vue est qu'il faut chercher d'autres moyens que celui-ci, très restrictif sur le plan didactique, de proposer des performances acceptables. Il nous apparaît préférable de ne restreindre l'adaptabilité d'un système à l'apprenant qu'au minimum nécessaire.

4.2. EIAO intégrant des théories modifiables.

Parmi les EIAO que nous avons déjà présentés, la part de la théorie qui est modifiable est plus ou moins grande.

APLUSIX permet aujourd'hui principalement de modifier les capacités de factorisation de carrés.
Exemple :
Pour utiliser la factorisation de $A^2 - B^2$ en (A-B)(A+B), trois possibilités sont offertes suivant le niveau de l'élève : soit il décrit la factorisation en donnant A, B, un coefficient C (pour appliquer $CA^2 - CB^2$ -> C(A-B)(A+B)) et la règle à appliquer; soit il donne A, B et la règle à appliquer, soit il donne simplement la règle à appliquer.
Le reste de la théorie n'est pas modifiable aujourd'hui mais il est envisagé de rendre possible son affaiblissement concernant les règles de transformations. Par contre les heuristiques de choix ne peuvent être modifiées, sans doute parce que leur apprentissage est l'objectif général de APLUSIX.

MENTONIEZH permet dès à présent la modification de la théorie. Cette modification est restreinte à un ensemble d'axiomes fournis par le système, que le professeur choisit d'inclure ou d'enlever. L'amélioration par rapport à APLUSIX sur ce point est que n'importe quel axiome peut être enlevé. De plus la modification de la théorie n'est concrètement possible que pour les carrés dans APLUSIX alors qu'elle est déjà mise en œuvre pour MENTONIEZH.

C'est la définition même d'Archimède qui permet a priori à la théorie d'évoluer en fonction des problèmes résolus par l'apprenant. Malheureusement comme c'est le système qui détermine cette évolution, il est difficile d'évaluer la qualité didactique des choix retenus. Nous n'avons d'ailleurs eu connaissance d'aucune étude importante menée dans cette voie.

Remarques conclusives.

L'exemple d'APLUSIX montre que l'on peut disposer d'un système réellement interactif même en permettant une modification de la théorie. Mais il reste encore beaucoup à faire pour que les restrictions apportées par cette possibilité ne soient pas trop contraignantes, comme le montre par exemple le traitement de la négation que nous avons introduit pour TALC (voir chapitre deuxième, 3.2.3).

5. EIAO en géométrie.

Au cours de l'exposé des trois distinctions précédentes, nous avons proposé comme exemple un bon nombre d'EIAO de géométrie. L'objet de ce chapitre est de situer chacun d'entre aux par rapport aux autres et d'indiquer ainsi où se situent nos travaux.

Nous proposons dans un premier temps une synthèse des classifications que nous avons proposées.

Dans un deuxième temps, nous présentons les objectifs de l'enseignement de la géométrie et comment ces objectifs motivent la définition d'EIAO en géométrie.

Enfin, nous exposons l'intérêt du système diagnostiquant la correction d'une figure qu'est TALC pour répondre à ces objectifs d'enseignement.

5.1. Panorama des EIAO en géométrie.

La figure I.7 a pour but de donner une vue d'ensemble des EIAO de géométrie que nous avons exposés.

Cette figure permet de faire la synthèse des classifications que nous avons proposées en situant chacun des EIAO présentés par rapport à ces trois axes.

On retrouve par exemple, dans la dimension "directivité", que le micro-monde Logo est à directivité nulle, ou encore que MENTONIEZH peut être défini comme étant un système :

- moyennement directif;
- permettant certains choix sur la théorie.
- à la complétude fortement définie.

Nous y avons placé TALC comme un système :

- moyennement directif (tuteur de découverte guidée).
- permettant de définir librement la théorie.
- disposant de capacités de raisonnement complètes.

Nous avons placé sur cette figure un système que nous avons nommé Cabri-TALC. On le définit soit comme TALC auquel les capacités de diagnostic de corrections ont été enlevées, soit comme Cabri-géomètre auquel les capacités de vérification de cohérence des constructions ont été ajoutées.

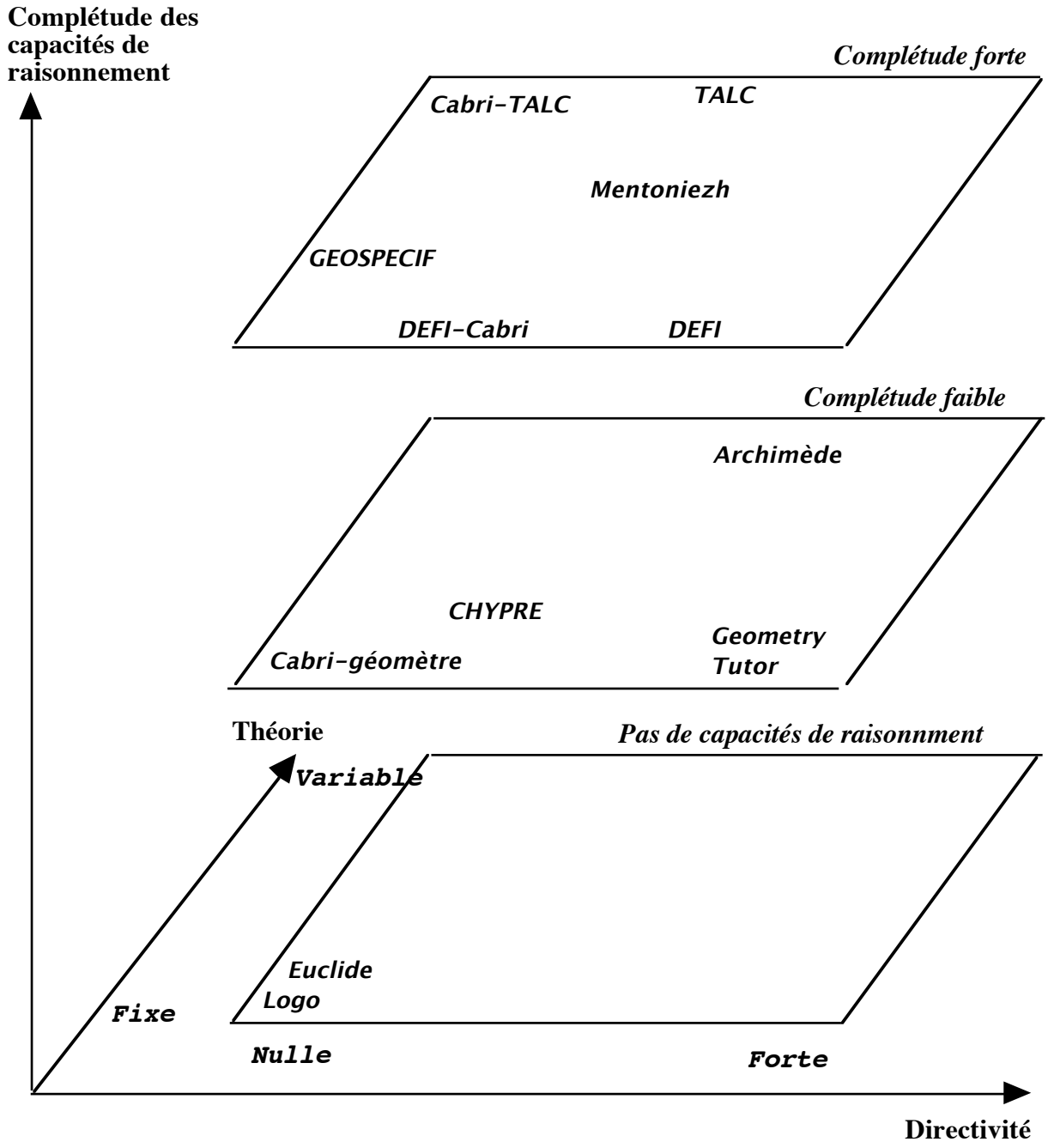


Figure I.7

5.2. Les objectifs de l'enseignement de la géométrie et comment les EIAO actuels y répondent.

Les objectifs de l'enseignement de la géométrie sont de plusieurs ordres⊡

•⊡Le premier est de familiariser l'élève avec le tracé à la règle et au compas.

•⊡Le second, plus général, est d'apprendre aux élèves à construire et formuler des raisonnements logiques.

• le troisième est de donner à l'élève une approche concrète des outils de calculs (aires, distances, angles) utilisés dans d'autres domaines, mathématiques ou non.

C'est dans le but de satisfaire un ou plusieurs de ces objectifs qu'ont été définis les EIAO présentés ici.

Considérons un hypothétique "environnement informatisé d'apprentissage de la géométrie", proposant à un apprenant l'acquisition de toutes les facettes de cette discipline et utilisant toutes les capacités développées dans ces EIAO.

En supposant tous les EIAO dont nous avons parlé finalisés, cet apprenant pourrait :

- Construire une figure avec LOGO, Euclide, Cabri-géomètre, Geometer's Sketchpad, GEOSPECIF ou Chypre.
- la manipuler dynamiquement avec Cabri-géomètre, Geometer's Sketchpad ou GEOSPECIF.
- Explorer l'espace des propriétés qui sont attachées à une figure avec DEFI.
- Expérimenter graphiquement des conjectures avec LOGO, Euclide, Cabri-géomètre, Geometer's Sketchpad ou GEOSPECIF.
- Construire des démonstrations de théorèmes avec Geometry Tutor, Archimède, Mentoniezh, DEFI.

Le point de départ de notre recherche vise à combler un manque indubitable dans cet ensemble : la capacité de vérifier qu'une construction est correcte au regard d'un énoncé fourni par un enseignant.

L'utilisation de cette capacité se situe en aval des activités de construction de figures et en amont des autres activités.

5.3. Intérêt didactique d'un système diagnostiquant la correction d'une construction.

Vérifier qu'une figure est correcte a au moins trois intérêts didactiques :

- Le premier intérêt concerne les activités de recherche, point de départ de nombreuses activités dans l'enseignement actuel.

Exemple :

L'enseignant introduit la notion de cosinus dans un triangle par l'intermédiaire de la situation représentée par la figure I.8 une fois une figure construite, il est demandé à l'élève, pour différents angles Δ (obtenus en modifiant la position du point P) de modifier la position de C, de consigner dans un tableau les longueurs |AH| et |AC|, et de calculer |A H|/|A C|; enfin il doit tirer des conclusions de ses observations. Cette situation de découverte du cosinus prend place avant le cours théorique sur cette notion.

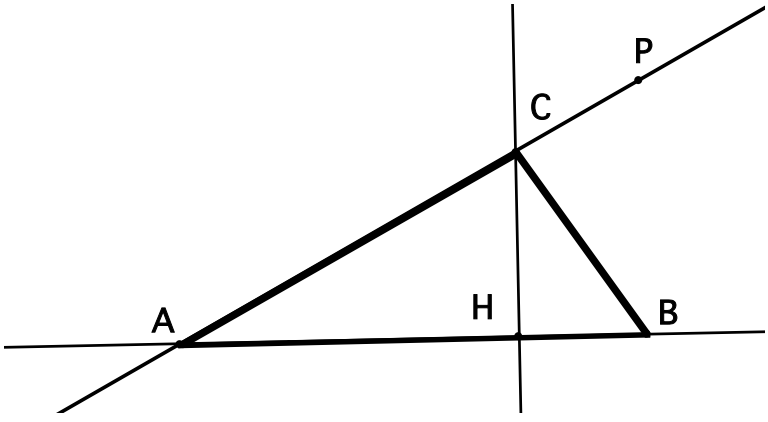


Figure I.8

Il est clair dans cet exemple que si la figure (simple pourtant) ne possède pas toutes les propriétés demandées, la situation perd tout son sens. L'enseignant qui utilise cette situation m'a d'ailleurs expliqué que la consigne donnée à l'élève pour s'en assurer était de l'appeler pour qu'il vérifie lui-même la correction. La classe utilisait Cabri-géomètre

et l'enseignant utilisait comme méthode de vérification le déplacement d'un des points. La correction était acquise si les propriétés visuelles attendues étaient conservées. Il est évident que si cette méthode de vérification par déplacement est plutôt efficace dans ce cas, dans le cas général elle ne peut être employée sans risque d'erreur.

Un tuteur intégrant des déductions complètes peut dans ces situations permettre à l'élève de passer immédiatement à la phase suivante et de libérer le professeur de la fastidieuse tâche de correction.

• le second intérêt est qu'un tel tuteur permet d'éviter l'aspect numérique et particulier des exercices donnés par des enseignants, du fait de considérations nullement pédagogique mais purement correctionnelles. Un professeur de mathématiques m'a ainsi expliqué que la contrainte de taille imposée aux segments dans de nombreux exercices a rarement pour objet de bien utiliser la règle graduée mais plutôt de faciliter la correction en série des diverses solutions des élèves à un problème, en utilisant astucieusement un calque.

• Le troisième intérêt est sans doute le plus fondamental. Les enseignants ont remarqué qu'une des plus grandes difficultés des élèves pour construire des démonstrations vient de la confusion qu'ils font entre hypothèses et conclusions. Demander à un élève de construire la figure correspondant aux hypothèses d'un problème permet de vérifier qu'elles sont bien comprises.

Toutes ces raisons d'utiliser un système diagnostiquant la correction d'une figure se confondent en une seule : avant toute activité géométrique il est important de savoir si la figure support de l'activité est correcte.

Notre travail est issu du Projet Mentoniezh [Py 90b], qui distinguait quatre composantes pour un système d'EIAO en géométrie :

• acquisition de la figure : l'élève construit, pour commencer, une figure conforme à la spécification des hypothèses désirées par le professeur. La partie construction est une des fonctionnalité actuelle de Cabri-géomètre. La partie vérification constitue la fonctionnalité principale de TALC.

• appropriation de la figure : il peut ensuite faire évoluer graphiquement la figure dont les propriétés logiques sont conservées. Cela lui permet, d'une part de détecter les invariants intéressants, d'autre part d'observer l'impact de la suppression d'une hypothèse. Cette phase correspond aussi à ce qu'offre Cabri-géomètre.

• exploration des propriétés : l'élève donne son avis sur d'éventuelles propriétés suggérées par le système à l'aide de théorèmes fournis par l'enseignant. C'est ce que propose DEFI aujourd'hui.

• construction de la preuve : l'élève construit sa démonstration, vérifiée par le système, en s'inspirant des découvertes faites aux étapes précédentes et en utilisant des théorèmes fournis par l'enseignant. C'est la composante réalisée par D. Py, qui a gardé le nom du projet initial pour son logiciel.

P. Nicolas a jeté les bases de nos travaux [Nicolas 89]. Son principal travail a été la réalisation d'une première maquette d'une interface de saisie à partir d'une tablette graphique, traduisant les outils de la table à dessin. Malheureusement cette première maquette, réalisée sur un matériel non compatible, n'a pas été portée sur d'autres systèmes. Notre participation au projet Cabri-géomètre [Laborde et al. 89] nous a conduits tout naturellement à chercher à intégrer les capacités offertes par ce logiciel.

De façon synthétique, on peut définir la place de TALC dans l'ensemble des EIAO de géométrie par les caractéristiques suivantes :

• c'est un tuteur de découverte guidée : il intègre les capacités du micro-monde de géométrie Cabri-géomètre et a pour objectif d'aider l'élève à obtenir une construction correcte.

• il utilise des capacités de raisonnement complètes.

• les capacités de raisonnement sont basées sur une théorie modifiable de la géométrie.

Si l'on restreint TALC à ses capacités de micro-mondes, ce nouvel EIAO (repéré par TALC-CABRI sur notre schéma) trouve sa place comme un micro-monde constructif pour la géométrie, utilisant des capacités de raisonnement complètes vis-à-vis d'une théorie modifiable de la géométrie pour garantir la cohérence géométrique des constructions.

Pour conclure, remarquons qu'au vu des systèmes proposés, il apparaît que les problématiques des EIAO pour la géométrie posent de nombreux problèmes scientifiques difficiles et ont amené à de nombreuses approches différentes.

On s'aperçoit aussi qu'il y a souvent loin de la définition à la réalisation : les projets sont ambitieux, les logiciels très restreints (voir 3.1.2 Geometry Tutor, et 3.2.3 Chypre). Sur le plan de la mise en œuvre, une remarque courante sur les EIAO est que pour les mettre véritablement entre les mains d'apprenants, il faut qu'ils soient complètement réalisés. Les défauts d'un prototype introduisent un biais trop important dans la situation didactique et masquent les véritables enjeux. Une réalisation finalisée est donc un atout majeur d'un EIAO, comme le montre bien le succès très important de Cabri-géomètre, qui constitue aujourd'hui à lui seul un tiers des ventes de logiciels éducatifs en France.

Chapitre Deuxième. Définition de TALC, tuteur de découverte guidée pour la construction de figures géométriques satisfaisant une spécification.

Le chapitre précédent a montré l'intérêt de disposer de capacités de vérification de correction dans un environnement informatisé de la géométrie. Ce chapitre a pour objet d'exposer la définition de TALC.

TALC est un EIAO dont l'objectif principal est la vérification de la correction d'une construction d'un élève vis-à-vis de l'énoncé donné par le professeur. L'énoncé (ou spécification) du professeur est exprimé dans un premier langage d'interface nommé CDL (pour Classroom Description Language). La construction est exprimée dans un second langage d'interface SCL (pour Student Construction Language). Le principe de l'établissement du diagnostic de correction est visualisée par le diagramme dit de Lookwood-Moris (cf figure II.1). Il exprime le fait que pour pouvoir comparer la construction de l'élève à la spécification du professeur, il faut les traduire dans une formule, respectivement F et S, d'un langage logique nommé LDL (pour Logical Description Language). La comparaison requise s'exprime entre F et S.

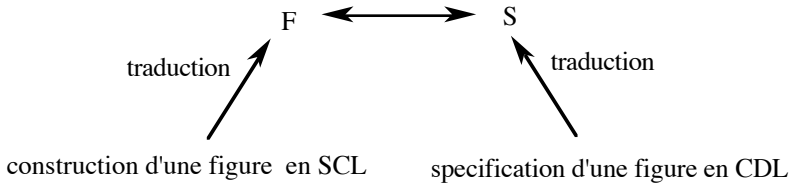


Figure II.1

Nous présentons dans un premier temps les objectifs et les exigences que la définition d'un EIAO (et donc en particulier de TALC) doit remplir. A partir de ceux-ci nous définissons le langage LDL et les langages d'interface CDL et SCL. Nous présentons ensuite les principes de la formulation d'une théorie logique dite TIG (pour Théorie Instrumentale de la Géométrie) représentant les connaissances supposées de l'élève en géométrie. Enfin nous définissons comment le contrat didactique est exprimé dans TALC.

1. Objectifs et exigences.

Les objectifs d'un EIAO sont à la fois issus d'exigences didactiques et d'exigences informatiques. Dans ce qui suit, nous présentons ces exigences que nous avons classées en quatre volets: exigence d'un contrat didactique à la sémantique claire et précise; exigence d'une liberté maximale donnée à l'enseignant dans la formulation de ce contrat; exigence de capacités de déduction complètes; exigence de performances garantissant un niveau d'interactivité suffisant pour un système d'EIAO.

1.1. Un contrat didactique à la sémantique claire.

Cette première exigence tient à la place de l'EIAO dans la "situation didactique".
Précisons tout d'abord la définition du terme "situation didactique".

Définition :
Une situation est dite "didactique" quand elle a été organisée en vue de l'apprentissage de connaissances par un sujet (nommé l'apprenant) [Brousseau 86]. Au contraire une situation "a-didactique" ne comporte pas de but d'apprentissage, bien que pouvant produire indirectement des connaissances.
Exemple :
Remplir correctement des formulaires administratifs est une connaissance qui s'acquiert alors qu'aucune situation d'apprentissage n'a été organisée.

De part l'objectif même du domaine de l'EIAO ("la conception et de l'usage d'ordinateurs pour la formation", voir chapitre premier), un EIAO est par définition inscrit dans une situation didactique. Dans la situation didactique mise en place autour d'un EIAO, on trouve au moins deux autres partenaires□□enseignant et l'apprenant. L'introduction de l'EIAO dans la traditionnelle relation enseignant/apprenant impose de redéfinir cette relation.
L'idée (simple) à l'origine des EIAO était la suivante□□EIAO devait remplacer l'enseignant et l'introduction de capacités dites "intelligentes" devait contribuer à rendre cette substitution la plus fidèle possible, reproduisant en particulier l'intelligence de l'enseignant. Cette idée a vite été contredite par les premières réalisations effectives. La définition de ces relations est aujourd'hui un sujet de recherche important pour les didacticiens dont la thématique est la didactique computationnelle [Balacheff 93]. Ce qui est clair à leurs yeux, c'est que l'EIAO n'est qu'un intermédiaire entre un enseignant et un apprenant. Mais cet intermédiaire ne possède pas les mêmes caractéristiques que l'enseignant. Les caractéristiques sur lesquelles ils diffèrent sont entre autres les suivantes :
- il est capable d'effectuer des opérations que l'enseignant refuserait de réaliser pour l'apprenant (car fastidieuses);
- il nécessite une rigueur de langage plus importante;
-□a confiance en son verdict est plus grande qu'en celle de l'enseignant.
Cette situation d'intermédiaire entre l'enseignant et l'apprenant nécessite la définition précise des possibilités d'expression du "contrat didactique" passé entre l'enseignant et l'apprenant dans un EIAO.

Définition.
Pour les didacticiens, le "contrat didactique" est l'ensemble des présupposés explicites ou implicites de la situation d'enseignement [Brousseau 90]. Ces présupposés évoluent au cours des rapports entre les enseignants et les apprenants, par des renégociations successives du contrat initial.

Exemple.
Dans une tâche de résolution de problème, les apprenants du secondaire considèrent que si l'enseignant propose un problème alors implicitement ce problème a une solution, de surcroît unique. Un présupposé peut aussi avoir été explicité par l'enseignant, comme l'obligation de fournir tous les calculs intermédiaires.

Pour remplir son rôle d'intermédiaire de l'enseignant, un EIAO se doit donc de posséder une représentation du contrat didactique. Cette représentation informatique oblige le contrat à être explicité.
Notre définition du contrat didactique dans le cadre des systèmes d'EIAO est alors la suivante :

Définition.
Le "contrat didactique" dans un système d'EIAO est l'ensemble composé de l'objectif donné à l'apprenant et du contexte qui y est attaché.

Nous estimons que la qualité d'un EIAO, dont la motivation principale est l'acquisition de connaissances par l'élève, tient à la sémantique claire et précise du contrat didactique conclu par l'enseignant par son intermédiaire. En effet, comment évaluer précisément la solution décrite par un apprenant si l'objectif du problème et son contexte ne sont pas eux-mêmes précis? Et sans évaluation précise, comment lui proposer des explications de qualité lui permettant d'améliorer ses connaissances? Les moyens d'expression du contrat didactique fournis à l'enseignant, outre de posséder une sémantique claire et précise, se doivent aussi d'être les plus généraux possibles. Cela signifie que les cas particuliers, dont l'expression est souvent plus ardue que le cas général, doivent être évités le plus possible.
Exemple **:**
Le langage de spécification CDL permet au professeur de spécifier qu'un segment est parallèle à une demi-droite, plutôt que de l'obliger à exprimer le parallélisme entre les droites supports du segment et de la demi-droite.

Pour préciser notre définition du contrat didactique, nous proposons de séparer cette définition en deux parties dont nous donnons ici les définitions.
Définitions
La "partie disponible" du contrat didactique est celle que l'enseignant a la liberté de définir avec les outils langagiers que lui propose le système.
La "partie prédéfinie" du contrat didactique est celle qui découle des choix effectués dans la conception du système et qui ne peut être modifiée par l'enseignant.
Exemple:
Dans APLUSIX, la partie disponible du contrat est la faculté pour le professeur de définir des exercices. La partie prédéfinie est importante : elle comporte entre autres les méthodes heuristiques du système et les évaluations qu'il donne sur la validité de la solution.

L'exigence de clarté et de précision du contrat se traduit ainsi :
- pour la partie disponible du contrat, il s'agit de donner la sémantique exacte des langages fournis à l'enseignant pour décrire le contrat. Nous présentons la définition des langages utilisés pour définir le contrat didactique dans TALC dans la deuxième partie de ce chapitre.
- pour la partie prédéfinie du contrat, il s'agit de décrire exactement les conséquences des choix de conception effectués sur l'évaluation que donne le système du respect du contrat par l'apprenant. En particulier il s'agit d'expliquer le plus clairement possible les clauses du contrat induites par l'introduction de l'intermédiaire informatique dans la situation didactique par rapport à la situation didactique classique. L'apprenant doit pouvoir retrouver les raisons d'un comportement a priori inattendu à partir d'une description statique, c'est à dire sans avoir à examiner le comportement opérationnel du système. L'expression de la partie prédéfinie du contrat didactique dans TALC est donnée dans la quatrième partie de ce chapitre.

1.2. Un degré de liberté maximal dans la formulation du contrat.

Bien qu'on ne considère plus aujourd'hui que l'EAIO puisse entièrement remplacer l'enseignant dans la situation didactique, il doit toutefois le représenter le plus fidèlement possible. Cela veut dire à nos yeux qu'il faut proposer un nombre maximal de degrés de liberté dans la description de la partie disponible du contrat. Nous dégageons ainsi quatre degrés de liberté donnés à l'enseignant :
- celui de construire ses propres énoncés de problèmes et/ou de modifier les énoncés existants (voir 2.2).

- celui de définir lui même la théorie représentant les connaissances requises pour la résolution d'un exercice (ou d'une séquence d'exercices) et supposées acquises par l'élève (voir 3.).
- celui de définir les outils disponibles à l'apprenant pour résoudre l'exercice et les conditions de validité de leur usage (voir 2.3).
- celui de spécifier le comportement du tuteur dans le cas où le contrat n'a pas été respecté en vue de produire des explications appropriées (voir chapitre troisième, §5).

Si l'idée de donner un maximum de degrés de liberté au professeur est essentielle pour façonner le système, elle peut aussi apparaître comme une difficulté supplémentaire pour lui. Notre point de vue est qu'il faut lui donner en même temps que ces libertés les moyens de faciliter sa tâche dans leur utilisation. L'interface du système avec le professeur ne doit donc pas être un simple éditeur de ses choix, mais un véritable tuteur de définition de situations didactiques. Les EIAO qui ont été jusqu'ici définis uniquement par rapport à leur utilité pour l'apprenant doivent ainsi intégrer aussi la tâche de l'enseignant [Major 93]. Comme l'apprenant, l'enseignant doit disposer d'un système qui lui facilite le travail. C'est l'objectif de l'interface pour le professeur que nous avons réalisée dans TALC [Desmoulins 90][Desmoulins et al 91] (voir Annexe F).

1.3. Des capacités de déduction complètes.

Dans le cadre d'un contrat didactique précis que le professeur a défini en donnant à la fois l'énoncé et la théorie, il est indispensable de disposer de capacités de déduction complètes par rapport à cette théorie. Si ce n'était pas le cas, le système pourrait amener l'apprenant à remettre en cause des connaissances exactes. Par exemple, dans la résolution d'une équation polynomiale, si le système refuse une solution de l'élève alors qu'elle est acceptable vis-à-vis de la théorie, l'élève va soit élaborer une théorie (fausse) prenant en compte la fausseté de cette solution, soit perdre confiance dans le système. Ces deux possibilités sont l'une et l'autre préjudiciables à l'évolution de ses connaissances.

Comme nous l'avons vu au premier chapitre, troisème partie, certains EIAO disposent seulement de déductions complètes vis-à-vis de l'ensemble des déductions produites avec la théorie en restreignant les déductions (en fonction de la profondeur ou du nombre de nœuds de l'arbre de déduction). Nous pensons que cette complétude est difficilement explicable tant à l'élève qu'à l'enseignant, pour qui la complétude a un sens beaucoup plus exigeant. Il nous semble qu'un bon apprentissage utilisant un EIAO doit pour ces raisons être basé sur des déductions complètes dans le sens le plus strict.

C'est pour cette raison que nous avons choisi d'exprimer le contrat didactique à l'aide de la logique du premier ordre, formalisme habituel pour lequel la complétude est simple à comprendre.

1.4. Des performances garantissant un niveau d'interactivité suffisant pour un EIAO.

Il est clair que pour qu'un EIAO puisse être utilisé de façon réaliste, il faut que le dialogue qu'il engage avec l'apprenant puisse être qualifié d'interactif. La notion de dialogue interactif est toute relative. Au vu des expérimentations menées avec Cabri-géomètre sur des tâches de construction de figure, on peut dire qu'un élève reste en moyenne 30 secondes à ne rien faire (du point de vue de son interaction avec le système évidemment!) entre la construction de deux objets, pour un niveau de classe de quatrième. C'est la limite que nous nous sommes fixée en première analyse pour définir un dialogue interactif.

L'exposé de ces quatre exigences majeures que nous proposons pour un EIAO montre qu'elles sont contradictoires. En effet, l'exigence d'interactivité est difficilement compatible avec les exigences d'une liberté maximale donnée à l'enseignant et de capacités de déduction complètes. Pour assurer les performances exigées par l'interactivité , une solution simple (et souvent adoptée, voir DEFI, chapitre premier 3.2.1 ou Geometry Tutor, chapitre premier 2.2.1), consiste à ce que le système ne propose que des exercices prédéfinis, que le concepteur a pu longuement analyser auparavant. Quand une certaine liberté est donnée au professeur dans les systèmes existants, c'est au détriment de la complétude des déductions (comme dans APLUSIX, voir chapitre premier 2.2.3). Et si l'on met en œuvre des déductions complètes et performantes, la compréhension de ces déductions est ardue et les explications sont difficilement mises en rapport avec le contrat didactique.

Ces contradictions montrent que l'objectif de répondre à toutes ces exigences à la fois est ambitieux. Dans les trois sous-chapître qui suivent, nous montrons comment nous résolvons ces contradictions par le choix de restrictions adéquates.

2. Définition du langage logique LDL et des langages d'interfaces CDL et SCL.

Notre objectif principal est de définir des moyens d'expression du contrat didactique dont la sémantique soit claire et précise. Les moyens d'expression d'une solution répondant à ce contrat doivent eux aussi posséder une sémantique claire et précise. Pour y arriver, notre méthode consiste à définir la sémantique des deux langages d'interface (CDL et SCL) à partir de la sémantique du langage logique (LDL) dans lequel ils sont traduits. LDL étant un langage de la logique du premier ordre, cette façon de faire permet d'obtenir la précision et la clarté de la logique du premier ordre pour la sémantique de CDL et de SCL.

Nous présentons en premier lieu la sémantique de LDL. A partir de celle-ci, nous présentons successivement les sémantiques de CDL et SCL.

2.1. LDL, un langage logique pour la vérification de la correction.

Les formules du langage LDL ont pour objet de représenter des spécifications de figures géométriques dans un formalisme permettant des déductions logiques sur ces figures. LDL est un langage du premier ordre dans lequel peuvent être exprimés les axiomes de la théorie TIG représentant les connaissances supposées de l'élève.

Nous présentons tout d'abord le langage et donnons un exemple de description d'une figure et d'axiomes en LDL. Nous discutons ensuite des répercussions des choix effectués pour la définition du langage sur les déductions possibles.

2.1.1. Présentation du langage.

LDL comprend des prédicats divisés en deux catégories⊡les prédicats de typage et les prédicats de propriété. Les prédicats de typage expriment le type et les composants d'un objet géométrique. Les prédicats de propriétés expriment les relations géométriques liant des objets. Le tableau suivant donne l'ensemble des prédicats LDL et la signification des atomes leur correspondant.

Prédicats	**Atomes**	**Signification**
Typages		
point	*point(p)*	p est un point
droite	*droite(l)*	l est une droite
demi-droite	*demi-droite(h,p,l)*	h est une demi-droite dont l'origine est le point p et le support est la droite l.
segment	*segment(s,p1,p2,l)*	s est un segment dont les extrémités sont les points p1 et p2 et le support est la droite l
cercle	*cercle(c,p,d)*	c est un cercle dont le centre est le point p et le rayon est la distance d.
distance	*distance(d)*	d est une distance.

Propriétés		
appdr	*appdr(p,l)*	le point p appartient à la droite l.
appdd	*appdr(p,h)*	le point p appartient à la demi-droite h.
appseg	*appseg(p,s)*	le point p appartient au segment s.
appcc	*appcc(p,c)*	le point p appartient au cercle c.
par	*par(l1,l2)*	les droites l1 et l2 sont parallèles.
perp	*perp(l1,l2)*	les droites l1 et l2 sont perpendiculaires.
invsens	*invsens(h1,h2)*	les demi-droites h1 et h2 ont sens inverse.
memesens	*memesens(h1,h2)*	les demi-droites h1 et h2 ont même sens.
distancep	*distancep(d,p1,p2)*	d est la distance entre les points p1 et p2
infdist	*infdist(d1,d2)*	la distance d1 est inférieure à d2
demidist	*demidist(d1,d2)*	la distance d1 est la moitié de la distance d2
sommdist	*sommdist(d,d1,d2)*	la distance d est la somme des distances d1 et d2
egalpt	*egalpt(p1,p2)*	les points p1 et p2 sont égaux
egaldr	*egaldr(l1,l2)*	les droites d1 et d2 sont égales
egaldd	*egaldd(h1,h2)*	les demi-droites h1 et h2 sont égales
egalseg	*egalseg(s1,s2)*	les segments s1 et s2 sont égaux
egalcc	*egalcc(c1,c2)*	les cercles c1 et c2 sont égaux
egaldist	*egaldist(d1,d2)*	les distances d1 et d2 sont égales

Une formule bien formée représentant une figure géométrique répond aux contraintes suivantes :

- Une formule est une conjonction de formules de base. Une formule de base est soit un atome, soit la négation d'un atome de propriété.
- Toutes les formules sont closes.
- le type de tout identificateur est défini par un et un seul atome de typage.

Ces contraintes correspondent à des choix effectués dans la partie prédéfinie du contrat, que nous exposons dans la quatrième partie de ce chapitre.

Une formule bien formée ne doit comporter qu'un atome de typage pour un identificateur donné. Cependant plusieurs choix d'atome de typage sont possibles pour cet identificateur, suivant les choix effectués sur les paramètres des prédicats de typage.

Prenons par exemple une demi-droite h d'origine `o` passant par `p1, p2, …, pn`. Si le prédicat de typage d'une demi-droite comporte comme paramètre, outre son nom et son point d'origine, un second point lui appartenant (`demi-droite(h,o,p)` exprimant que la demi-droite `h` a pour origine `o` et passe par le point `p`), cette demi-droite peut être représentée par les n atomes de typages `demi-droite(h,o,p1), demi-droite(h,o,p2), …, demi-droite(h,o,pn).`

Pour répondre simplement à la contrainte qu'une formule bien formée ne doive comporter qu'un atome de typage pour un identificateur donné, nous nous sommes fondés pour chaque prédicat sur les propriétés géométriques de ses paramètres : nous avons choisi les paramètres des prédicats de typage de façon à ce qu'un seul atome de typage puisse être associé à un objet géométrique au regard de ses propriétés géométriques. Par exemple, le prédicat `demi-droite` comporte deux argument outre l'identificateur de la demi-droite : son point d'origine et sa droite support car il est géométriquement incohérent qu'une demi-droite ait deux origines ou deux droites supports.

Pour les prédicats `point`, `droite` et `distance`, nous avons défini un seul paramètre, le nom de l'objet, ces objets ne nécessitant pas de références à d'autres objets pour être définis.

Pour le prédicat cercle, l'introduction du centre et du rayon correspond à notre exigence, vérifiant

`cercle(c1,p1,d1) ⇒ cercle(c2,p2,d2) ⇒ (c1=c2 ⇔ p1=p2 ⇔ d1=d2).`

De même pour le prédicat demi-droite l'introduction du point origine et de la droite support vérifie que

```
demi-droite(h1,p1,l1) ⇒ demi-droite(h2,p2,l2) ⇒
     (h1=h2 ⇔ p1=p2 ⇔ l1=l2).
```

La question est un peu plus délicate pour le prédicat segment. Pour qu'il vérifie que

```
segment(s1,p1,p2,l1) ⇒ segment(s2,p3,p4,l2) ⇒
     (s1=s2 ⇔ p1=p3 ⇔ p2=p4 ⇔ l1=l2)
```

il faut ajouter la contrainte suivante sur les extrémités du segment : un segment `[p1 p2]` est représenté par `segment(s,p1,p2,l)` ssi `p1` ≤ `p2` dans l'ordre lexicographique usuel des identificateurs. Ainsi les segments `[A B]` et `[B A]` ont la même représentation `segment(s,A,B,l).`

2.1.2. Exemple d'énoncé décrit par une formule LDL.

Le tableau suivant présente la correspondance entre un énoncé en langue naturelle et sa description en une formule de LDL. La figure correspondant à l'énoncé est la figure II.2.

Énoncé en langue naturelle	Formule LDL correspondante
(1) Soit ABCD un parallélogramme.	point(A) ⇔ point(B) ⇔ point(C) ⇔ point(D) ⇔ segment(s1,A,B,l1) ⇔ segment(s2,B,C,l2) ⇔ segment(s3,C,D,l3) ⇔ segment(s3,D,A,l4) ⇔
(2) Soit O le milieu de ses diagonales.	segment(s5,A,C,l5) ⇔ distance(d1) ⇔ distancep(d1,A,C) ⇔ distance(d2) ⇔ distancep(d2,A,O) ⇔ demidist(d2,d1) ⇔ appseg(O,s5) ⇔ segment(s6,D,B,l6) ⇔ appseg(O,s6) ⇔
(3) Soit F le point d'intersection entre la droite parallèle à (DB) et passant par C et (AD).	droite(l7) ⇔ appdr(C,l7) ⇔ par(l7,l6) ⇔ appdr(F,l7) ⇔ appdr(F,l4) ⇔
(4) Soit G le point d'intersection entre la droite parallèle à (DB) et passant par A et (BC).	droite(l8) ⇔ appdr(A,l7) ⇔ par(l8,l6) ⇔ appdr(G,l8) ⇔ appdr(G,l2) ⇔
(5) Soit K le cercle de centre O passant par F.	cercle(K,O,r) ⇔ distance(r) ⇔ appcc(F,K)

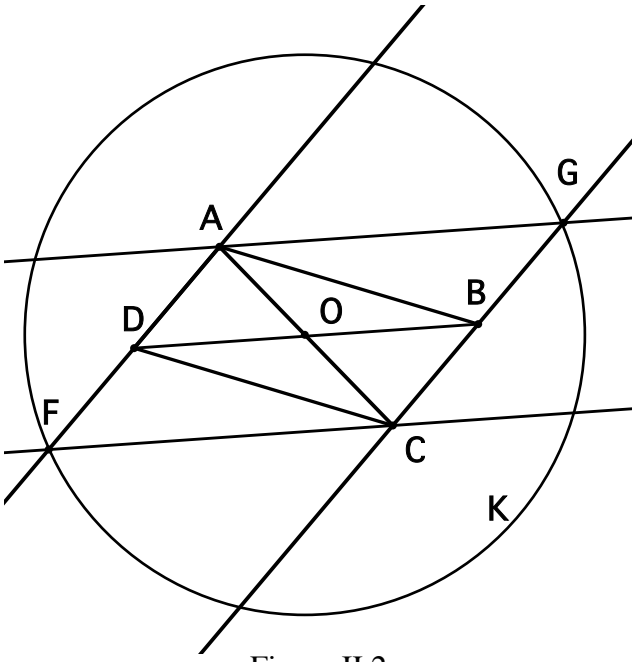


Figure II.2

2.1.3. Conséquences des choix effectués.

Les choix effectués sur les composants des atomes de typage peuvent avoir une incidence sur les déductions.

Considérons par exemple le prédicat segment. Nous pouvons lui donner plus ou moins d'arguments. Par exemple un segment `[A B]` peut être représenté par l'atome `segment(s,l)` en ajoutant un prédicat de propriété décrivant qu'un point est extrémité d'un segment, ou encore par `segment(s,p1,p2,h1,h2,d,l)` exprimant que `s` est un segment d'extrémités `p1` et `p2`, de support `l`, dont la distance entre les extrémités est `d` et intersection des deux demi-droites `h1` et `h2`. Supposons que nous ayions effectué ce dernier choix et que soit seul présent l' axiome exprimant que "tout point appartenant à un segment appartient à chacune des demi-droites issues d'une des extrémités du segment et passant par l'autre". Dans ce cas, l'appartenance d'un point `P` à la demi-droite `[A B)` est déductible de la formule LDL exprimant que `P` appartient au segment `[A B]`. Elle ne l'est pourtant pas avec le choix que nous avons retenu (`segment(s,p,p1,p2,l)`). Cette bizarrerie aux yeux d'un géomètre averti s'explique simplement par le fait que la représentation que nous avons choisie n'introduit aucune demi-droite comme composant du prédicat *segment*. Comme aucun autre axiome n'est présent que celui cité précédemment, en particulier aucun axiome existentiel permettant de déduire l'existence d'une demi-droite, la déduction est impossible faute d'existence de la demi-droite en question. Mais si l'introduction de nombreux composants permet plus de déductions, elle entraîne aussi une explosion du nombre d'objets préjudiciables aux performances.

Les choix que nous proposons à ce sujet reflètent ce qui nous semble un compromis raisonnable. Ils sont basés sur notre opinion de ce que l'on désigne implicitement lorsque l'on fait référence à un objet géométrique.

Ainsi avons nous considéré

- qu'en désignant une demi-droite on faisait implicitement référence à la droite qui la supporte;

- qu'en désignant un segment on faisait implicitement référence à ses deux extrémités et à la droite qui la supporte;
- qu'en désignant un cercle on faisait implicitement référence à son centre et son rayon.

Ces choix ont une importance fondamentale car ils déterminent le contrat didactique. Par exemple, si le professeur a demandé la construction d'une droite `(A B)`, alors la construction d'un segment `[A B]` est suffisant, car la droite `(A B)` est alors implicitement présente dans la construction.

Les prédicats choisis peuvent aussi avoir une influence sur les déductions.

Par exemple, la distance entre deux points peut être considérée soit comme une propriété reliant deux points, soit comme un objet en relation avec deux points. Si l'on suppose l'absence d'axiomes existentiels (ce qui est notre cas pour des raisons exposées en 3.2.2), alors la première possibilité permet certaines déductions que la seconde ne permet pas, car nécessitant l'existence d'une distance. En effet, un axiome qui serait existentiel avec la seconde possibilité (et donc exclus de ce fait) ne le serait pas avec la première et toutes les déductions découlant de cet axiome seraient seulement valides dans le premier cas.

Exemple :

L'axiome "Pour tout parallélogramme les longueurs de ses côtés sont deux à deux égales" se traduit par

```
∀L1,L2,L3,L4,A,B,C,D
droite(L1) ∧ droite(L2) ∧ droite(L3) ∧ droite(L4) ∧
point(A) ∧ point(B) ∧ point(C) ∧ point(D) ∧
par(L1,L2) ∧ par(L3,L4) ∧
appdr(A,L1) ∧ appdr(B,L1) ∧ appdr(B,L2) ∧ appdr(C,L2) ∧
appdr(C,L3) ∧ appdr(D,L3) ∧ appdr(D,L4) ∧ appdr(A,L4)
⇒
egaL_dist(A,B,C,D)
```

dans le premier cas et

```
∀L1,L2,L3,L4,A,B,C,D
droite(L1) ∧ droite(L2) ∧ droite(L3) ∧ droite(L4) ∧
point(A) ∧ point(B) ∧ point(C) ∧ point(D) ∧
par(L1,L2) ∧ par(L3,L4) ∧
appdr(A,L1) ∧ appdr(B,L1) ∧ appdr(B,L2) ∧ appdr(C,L2) ∧
appdr(C,L3) ∧ appdr(D,L3) ∧ appdr(D,L4) ∧ appdr(A,L4)
⇒
∃d distance(d) ∧ distancep(d,A,B) ∧ distancep(d,B,C)
```

dans le second cas.

Nous avons pourtant choisi la seconde possibilité. D'une part cela permet de désigner une distance sans qu'il soit fait référence à deux points (par exemple le rayon d'un cercle), d'autre part cela nous semble plus cohérent avec les usages qui considèrent la distance comme un objet, notamment concernant ce qu'on nomme le "report de distance".

2.2. CDL, un langage d'interface pour l'expression de spécifications géométriques.

Un des degrés de liberté que TALC propose à l'enseignant est la capacité de définir ses propres énoncés, que nous nommons "spécifications". Ces spécifications de figures géométriques sont exprimées en CDL.

CDL a été défini de façon à être proche des langages de spécification généralement utilisés dans les manuels scolaires en France [Jullien et al. 88]. Cela permet de proposer un langage d'interface dont l'apprentissage est rapide et la sémantique facile à comprendre, tout en permettant une expression précise de l'énoncé de la figure.

Nous donnons tout d'abord une présentation informelle de CDL. Nous présentons ensuite la syntaxe du langage et un exemple de spécification en CDL. Nous définissons alors la sémantique formelle de CDL. Enfin nous résumons l'ensemble de ce que l'utilisateur du langage doit retenir sur sa syntaxe et sa sémantique : cela nous permet en particulier de préciser les aspects négatifs : ce que la syntaxe ne permet pas, les faux sens qu'on pourrait lui attribuer.

2.2.1. Présentation informelle de CDL.

CDL permet à l'enseignant de décrire des spécifications comportant les objets géométriques suivants : point, droite, demi-droite, segment, cercle et distance.

Ces objets sont représentés de deux manières :

- soit par des identificateurs. Le langage permet à l'enseignant de signifier quels sont les objets qu'il veut retrouver avec un nom identique dans la construction de l'élève.

Définitions :
- les objets "nommés" d'une spécification en CDL sont les objets que l'enseignant veut retrouver avec un nom identique dans la construction de l'élève;
- les objets "non-nommés", sont les autres objets de la spécification.

Les objets nommés sont représentés en CDL par des identificateurs commençant par une majuscule. Ainsi, le nom de tout objet désigné par un identificateur commençant par une minuscule est laissé à l'initiative de l'élève dans sa construction.

La possibilité de donner un nom à des objets non-nommés est utile d'une part quand l'objet ne peut être décrit par une expression (par exemple un cercle), d'autre part si le professeur veut éviter de répéter l'expression d'un objet en lui associant un identificateur (par exemple pour des expressions complexe comme `1/2 |A centre(Q)|`).

- soit par des expressions. Elles permettent de faire référence à un objet sans lui donner de nom, en décrivant les relations qu'il entretient avec d'autres objets. Il s'agit par exemple d'expressions dénotant une droite passant par deux points (`(A B)`), un segment (`[A B]`).

Définitions :
- les objets "identifiés" sont les objets que l'enseignant a désigné par un identificateur dans sa spécification en CDL;
- les objets "non-identifiés" sont les autres objets de la spécification.

Exemple :

Dans la spécification

```
(A B ) // (C D),
(B C) // (D A),
s1 = [A C], s2 = [B D].,
p e s2, p e s2
```

`A,B,C,D` sont des objets nommés (et donc identifiés);

`p, s1, s2, [A C]` et `[B D]` sont des objets non-nommés et identifiés;

`(A B), (C D), (B C)` et `(D A)` sont des objets non-nommés et non identifiés.

L'enseignant peut aussi décrire avec CDL des relations géométriques liant les objets de la spécification. Ces relations peuvent être des appartenances, des parallélismes, des comparaisons, etc. Une catégorie particulière de relation permet de préciser le type d'un objet. Ce sont les relations de typage. Elles sont indispensables dans certaines spécifications où les relations géométriques ne suffisent pas à définir le type d'un objet. Par exemple la relation d'appartenance peut lier un point à une droite, une demi-droite, un segment ou un cercle.

Le tableau suivant présente les termes représentant des objets et les atomes représentant des relations entre objets. Les identificateurs en italique dénotent des termes représentant des objets (y compris des identificateurs du langage), les identificateurs non en italique dénotent seulement les identificateurs du langage.

Définitions :

Deux types d'objets sont définis :
Ens-Points-Alignés = Droite ∪ Demi-droite ∪ Segment;
Ens-Points = Cercle ∪ Ens-Points-Alignés.

Expression	Sémantique	Typage induit
TERMES		
(*a b*), (*a,b*)	droite passant par a et b	∈
[*a b*], [*a,b*]	segment passant par a et b	∈
[*a b*), [*a,b*),	demi ∪ droite d' origine a	a et b sont des points
(*b a*], (*b,a*]	et contenant b	∈
|*a b*|, |*a,b*|	distance entre a et b	∈
2*d*	deux fois d	d est une distance
1/2*d*	un demi d	d est une distance
centre(c)	centre de c	c est un cercle
rayon(c)	rayon de c	c est un cercle

ATOMES		
Typage		
point(p)	p est un point	
droite(d)	d est une droite	
demi_droite(h)	h est une demi-droite	
segment(s)	s est un segment	
cercle(c)	c est un cercle	
distance(d)	d est une distance	
Propriétés		
o1 = o2	égalité des objets a1 et o2	type(o1) = type(o2)
p e *E*	p appartient à e	E est un Ens-Points
ea1 // ea2	x parallèle à y	ea1 et ea2 sont des
ea1 !⊥ ea2	x perpendiculaire à y	Ens-Points-Alignés
d1 < d2	d1 inférieur à d2	d1 et d2 sont des distances
memesens(*h1*,*h2*)	h et h' ont même sens	
invsens(*h1*,*h2*)	h et h' ont sens inverse	h1 et h2 sont des demi-droites

Pour les mêmes raisons que pour LDL (voir 4.1.1.2), un énoncé en CDL est une conjonction de formules de base. Une formule de base est soit un atome positif, soit la négation d'un atome de propriété.

On remarque que ne sont pas admises les expressions dénotant des demi-droites de la forme `[A x)`, où `x` représente l'un des sens de la droite, comme c'est l'usage dans certains manuels. Pour que cela soit possible, il faudrait pouvoir décrire un objet de type "sens d'une droite". Dans ce cas on pourrait écrire une spécification comme `h = [A⊥x), sens(h,x).` L'inconvénient de cette solution est qu'elle manipule le sens comme un objet alors que le sens est conçu comme une propriété dans les manuels. C'est la raison de notre choix, qui impose de donner un second point différent de l'origine pour exprimer une demi-droite.

2.2.2. Syntaxe de CDL.

La syntaxe de CDL est la suivante, exprimée sous forme BNF.

```
<CDL>                    ::= <LISTE_LITTERAUX>

<LISTE_LITTERAUX>        ::= <LITTERAL> , <LISTE_LITTERAUX>
<LISTE_LITTERAUX>        ::= <LITTERAL> .

<LITTERAL>               ::= <ATOME>
                         ::= ¬ ( <ATOME_PROPRIETE> )

<ATOME>                  ::= <ATOME_PROPRIETE>
                         ::= <ATOME_TYPAGE >
                         ::= <ATOME_EGALITE>
```

<ATOME_TYPAGE > ::= point (<identificateur>)
::= droite (<identificateur>)
::= demi_droite (<identificateur>)
::= segment (<identificateur>)
::= cercle (<identificateur>)
::= distance (<identificateur>)

<ATOME_EGALITE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>
::= <OBJET_TERME> = <OBJET_TERME>

<ATOME_PROPRIETE> ::= <POINT> e <ENS_POINTS>
::= <ENS_PTS_ALIGNES> // <ENS_PTS_ALIGNES>
::= <ENS_PTS_ALIGNES> !- <ENS_PTS_ALIGNES>
::= <DISTANCE> '<' <DISTANCE>
::= memesens (<DEMI-DROITE>, <DEMI-DROITE>)
::= invsens (<DEMI-DROITE>, <DEMI-DROITE>)

<OBJET_TERME> ::= <POINT_TERME>
::= <ENSPT_TERME2>
::= <DISTANCE_TERME>

<ENS_POINTS> ::= <identificateur>
::= <ENS_POINTS_TERME>
<ENS_POINTS_TERME> ::= <ENS_PTS_ALIGNES>

<ENSPT_TERME2> ::= <ENS_PTS_ALIGNES_TERME2>

<ENS_PTS_ALIGNES> ::= <identificateur>
::= <ENS_PTS_ALIGNES_TERME>
<ENS_PTS_ALIGNES_TERME> ::= <DROITE_TERME>
::= <DEMI-DROITE_TERME>
::= <SEGMENT_TERME>

<ENS_PTS_ALIGNES_TERME2> ::= <DROITE_TERME>
::= <DEMI-DROITE_TERME>
::= <SEGMENT_TERME2>

<POINT> ::= <identificateur>
::= <POINT_TERME>
<POINT_TERME> ::= centre (<CERCLE>)

<DROITE_TERME> ::= (<POINT> <POINT>)
::= (<POINT> , <POINT>)

<DEMI-DROITE> ::= <identificateur>
::= <DEMI-DROITE_TERME>
<DEMI-DROITE_TERME> ::= [<POINT> <POINT>)
::= [<POINT> , <POINT>)
::= (<POINT> <POINT>]
::= (<POINT> , <POINT>]

<SEGMENT_TERME> ::= [<POINT> <POINT>]
::= [<POINT> , <POINT>]

<SEGMENT_TERME2> ::= [<identificateur> <identificateur>]
::= [<identificateur> , <identificateur>]

<CERCLE> ::= <identificateur>

<DISTANCE> ::= <identificateur>
::= <DISTANCE_TERME>
<DISTANCE_TERME> ::= 2 <DISTANCE>
::= 1/2 <DISTANCE>
::= | <POINT> <POINT> |
::= | <POINT> , <POINT> |
::= rayon (<CERCLE>)

La définition des identificateurs est basée sur la syntaxe des identificateurs de PrologII+ [Prologia 92], en définissant <identificateur> par :
<identificateur_prolog> ::= <alpha> <identificateur>
en excluant les cas où <identificateur_prolog> commence par un symbole non alphabétique.

Cela permet de passer facilement d'un identificateur à sa représentation en Prolog. La syntaxe de <identificateur> est alors la suivante :
<identificateur> ::= <alpha> <alpha_numérique> <prim>
<alpha_numérique> ::= <alpha> <alpha_numérique>
::= "_" <alpha_numérique>
::= <chiffre> <alpha_numérique>
::= □
<prim> ::= " ' " <prim>
::= □
<alpha> est l'ensemble des caractères de l'alphabet et <chiffre> un chiffre entre 1 et 9.

2.2.3. Exemple d'énoncé représenté en CDL.

Le tableau suivant présente la correspondance entre un énoncé en langue naturelle et sa description en une formule de CDL. La figure correspondant à l'énoncé est la figure II.2 du sous-chapître 2.1.2.

Énoncé en langue naturelle	Formule CDL correspondante

(1) Soit ABCD un parallélogramme.
(2) Soit O le milieu de ses diagonales.
(3) Soit F le point d'intersection entre la droite parallèle à (DB) et passant par C et (AD).
(4) Soit G le point d'intersection entre la droite parallèle à (DB) et passant par A et (BC).
(5) Soit K le cercle de centre O passant par F.
	`(A B) // (D C), (A D) // (B C),`
	`O  [A C], O  [B D],`
	`|A O| = 1/2 |A C| ,`
	`droite(l1), l1 // (D B), C  l1,`
	`F  l1, F  (A D),`
	`droite(l2), l2 // (D B), A  l2,`
	`G  l2, G  (B C),`
	`O = centre(K), F  K.`

Les phrases (1) et (2) se traduisent assez naturellement en CDL. Le langage autorise d'exprimer `|A O| = |O C|` à la place de `|A O| = 1/2 |A C|.`

Les phrases (3) et (4) montrent l'utilité des atomes de typage. Comme le parallélisme peut-être décrit entre plusieurs types d'objets, l'atome `droite(l1)` permet de desambiguïser l'atome `l1 //(D B).`

La phrase (5) permet de montrer l'intérêt des prédicats décrivant des composantes d'objets, comme ici le centre d'un cercle.

2.2.4. Sémantique de CDL.

La sémantique de CDL est définie par référence à celle du langage LDL définie en 2.1. Définir la sémantique de CDL correspond donc à définir la traduction de CDL en LDL.

La traduction de CDL en LDL est définie en suivant le principe de composition sur la grammaire de CDL, à l'aide d'un ensemble de fonctions. Les plus grandes difficultés de définition viennent du fait que CDL est un langage déclaratif acceptant des typages implicites. La traduction suppose donc de définir des déductions de type sans tenir compte de l'ordre de la spécification.

La définition complète de cette traduction étant donnée en annexe C, le but de ce sous-chapitre est de présenter les points principaux de cette définition.

Pour cela, nous expliquons en même temps les traductions définies à partir des non-terminaux les plus significatifs et les définitions des fonctions permettant l'application du principe de composition.

Afin de présenter cette traduction sans nous préoccuper des détails dus aux différentes formes syntaxiques des termes et à la transformation des identificateurs, nous avons défini une syntaxe abstraite des termes (voir l'annexe C). Le passage d'une spécification en CDL à la spécification correspondante dans la syntaxe abstraite de CDL étant immédiat, nous supposons que nous disposons d'une traduction de la spécification dans cette syntaxe, que nous nommons Spec. Cela nous permet de définir les fonctions portant sur l'ensemble de la spécification d'une façon plus élégante.

Nous expliquons la sémantique de CDL en présentant successivement :
- la traduction des littéraux.
- la traduction des atomes.
- la traduction des termes représentant des objets géométriques.
La traduction que nous définissons ainsi est nommée "traduction simple".

Nous définissons ensuite la traduction complète d'une spécification en CDL. Cette traduction est la traduction simple pour laquelle aucun objet n'est égal à un autre selon la théorie TIG.

2.2.4.1. Traduction des littéraux.

La traduction du non-terminal principal est simple. Il s'est la composition par la fonction @ des traductions des littéraux, qui consiste en la fusion des ensembles de littéraux LDL des traduction des littéraux.

<CDL> ::= <LISTE_LITTERAUX>
Trad =Trad1 **Trad1**

<LISTE_LITTERAUX> ::= <LITTERAL> , <LISTE_LITTERAUX>
Trad =Trad1 @ Trad2 **Trad1** **Trad2**

La traduction d'un littéral négatif est une fonction de la traduction de l'atome et du littéral LDL exprimant la propriété principale exprimée par l'atome. La fonction **NOT** permet de transformer le littéral de l'atome négativé en remplaçant l'atome **Prop** sur lequel porte la négation par l'atome opposé correspondant.

<LITTERAL> ::= ¬ (<ATOME_PROPRIETE>)
Trad = NOT (Trad1 ,Prop) **Trad1 Prop**

2.2.4.2. Traduction des atomes.

Atomes de typage.

La traduction des atomes de typage pour les demi-droites, segments et cercles nécessite l'utilisation de fonctions définissant l'identificateur dans Spec de :
- du point origine d'une demi-droite (ORIGINE)
- de la droite support d'un segment ou d'une demi-droite (SUPPORT)
- la première et la seconde extrémité d'un segment (PREMIERE_EXTREMITE, SECONDE_EXTREMITE).
- le centre et le rayon d'un cercle (CENTRE, RAYON).

<ATOME_TYPAGE > := demi_droite (<identificateur>)
Idf
Trad = demi_droite(Idf, ORIGINE(Idf,Spec), SUPPORT(Idf,Spec)) point(ORIGINE(Idf,Spec)), droite(SUPPORT(Idf,Spec))

::= segment (<identificateur>)
Idf
Trad =
segment(Idf , PREMIERE_EXTREMITE(Idf,Spec), SECONDE_EXTREMITE(Idf,Spec),SUPPORT(Idf ,Spec)) point(PREMIERE_EXTREMITE(Idf,Spec)), point(SECONDE_EXTREMITE(Idf,Spec)), droite (SUPPORT(Idf,Spec))

::= cercle (<identificateur>)
Idf

Trad =
cercle(Idf , CENTRE(Idf,Spec),RAYON(Idf,Spec)), point(CENTRE(Idf,Spec)), distance(RAYON(Idf, Spec))

Les fonctions ORIGINE, CENTRE et RAYON sont définies de façons similaires. Leurs définitions s'appuient sur la fonction NOM. Le principe de la définition est basé d'une part sur l'application de la transitivité de l'égalité sur les atomes d'égalité contenant un terme d'une forme donnée, d'autre part sur la fonction NOM. Si aucun identificateur de Spec ne représente l'objet, l'identificateur est un identificateur nouveau et unique défini par la fonction CREATION.

Exemple :
ORIGINE(Idf,Spec) est l'identificateur de l'origine de la demi-droite identifiée par Idf dans la spécification Spec, défini par
s'il existe un atome d'égalité de Spec de la forme [T1 T2) = Term' ou Term' = [T1□T2) tel que NOM(Term',Spec)= Idf' est un identificateur de Spec et T1 et T2 sont des termes
alors
ORIGINE(Idf,Spec) = Idf'
sinon
ORIGINE(Idf,Spec) = est un identificateur unique attribué par la fonction CREATION

La fonction NOM est définie elle aussi en s'appuyant sur la transitivité de l'égalité sur les atomes d'égalité. Si par transitivité un terme est égal à un identificateur, le nom du terme est cet identificateur, sinon l'identificateur est un identificateur nouveau et unique défini par la fonction CREATION.

Les fonctions PREMIERE_EXTREMITE et SECONDE_EXTREMITE dont les définitions sont assez similaires, utilisent deux fonctions auxiliaires (PREMIER et SECOND) permettant de définir un ordre sur les extrémités d'un segment. Cela est dû à la contrainte de LDL sur l'ordre lexicographique des extrémités d'un segment (voir 2.1.1.).

Atomes d'égalité.

La traduction des atomes d'égalité est très simple car la composition utilise uniquement la fonction @. Aucune substitution n'est nécessaire. En effet la définition des traductions des termes composant l'atome utilise les fonctions précédemment décrites qui déduisent l'identificateur d'un terme de l'ensemble de la spécification Spec.

<ATOME_EGALITE> ::= <identificateur> = <OBJET_TERME>
Trad = Trad1 **Idf1** **Trad1 Idf2**
::= <OBJET_TERME> = <identificateur>
Trad = Trad1 **Trad1 Idf2** **Idf1**
::= <OBJET_TERME> = <OBJET_TERME>
Trad1 Idf1 **Trad2 Idf2**
Trad = Trad1 @ Trad2

Atomes de propriété.

La traduction des atomes de propriété consiste à composer par la fonction @ les traductions des termes de l'atome et un atome représentant la propriété liant ces termes.

Exemple :

<ATOME_PROPRIETE> ::= <DISTANCE> '<' <DISTANCE>
Trad1 Idf2 Trad2 Idf2

Trad = infdist(Idf1,Idf2)) @ Trad1 @ Trad2

Pour les atomes d'appartenance d'un point à un objet, il faut utiliser la fonction **PREDAPP** pour que la traduction utilise le prédicat d'appartenance correspondant au type de l'objet.

<ATOME_PROPRIETE> ::= <POINT> e <ENS_POINTS>
Trad1 Idf2 Trad2 Idf2

Trad = app(Idf1,Idf2) @ Trad11 @ Trad2 avec app= PREDAPP(Idf2,Spec)

Pour les atomes de perpendicularité (resp. de parallélisme), la sémantique de CDL est induite par celle de LDL : si les objets sur lesquels porte le parallélisme (resp. la perpendicularité) ne sont pas des droites, la traduction doit porter sur les droites supports des objets car en LDL seules deux droites peuvent être perpendiculaires (resp. parallèles).

<ATOME_PROPRIETE> ::= <ENS_PTS_ALIGNES> !- <ENS_PTS_ALIGNES>
Trad1 Idf2 Trad2 Idf2

Trad = perp(SUPPORT(Idf1),SUPPORT(Idf2)) @ Trad1 @ Trad2

2.2.4.3. Traduction des termes représentant des objets géométriques.

Traduction des termes dérivant sur un identificateur.

La traduction des termes dérivant sur un identificateur est l'atome de typage de cet identificateur défini par la fonction TYPAGE. La définition de cette fonction dépend de la définition du type d'un identificateur par la fonction TYPE qui déduit de la spécification Spec le type de l'identificateur. La déduction du type d'un identificateur i est basée sur l'analyse de l'ensemble des non-terminaux dérivant sur i.

Notons que l'identificateur LDL correspondant à un identificateur CDL est cet identificateur auquel soit un 'a' soit un 'A' a étét ajouté pour correspondre à la syntaxe de LDL (voir annexe C). Par la suite, nous simplifions pour plus de lisibilité, dans les exemples de traduction de spécification ou de construction que nous donnons, les identificateurs des traductions en leur ôtant le premier a (ou A).

Traduction des autres termes.

La définition de la traduction des termes ne dérivant pas sur un identificateur ne pose pas de problème particulier. Cette traduction est une composition des traductions des termes composant le terme à traduire et d'atomes des propriétés et de typage.

Exemple :

<DISTANCE_TERME> ::= 2 <DISTANCE>
Trad1 Idf1 Term1

Trad = distance(Idf) demidist(Idf1,Idf) @ Trad1, Idf= NOM(Term,Spec), Term = 2 Term1

::= 1/2 <DISTANCE>
Trad1 Idf1 Term1

Trad = distance(Idf) demi_dist(Idf,Idf1) @ Trad1, Idf= NOM(Term,Spec), Term = 1/2 Term1

::= | <POINT> <POINT> |

::= | <POINT> , <POINT> |
Trad1 Idf1 Term1 Trad2 Idf2 Term2

Trad = distance(Idf) distancep(Idf,Idf1,Idf2) @ Trad1

2.2.4.4. Traduction complète.

Intuitivement, la traduction complète est la traduction simple pour laquelle aucun objet n'est égal à un autre selon TIG.

Soit S la traduction simple d'une spécification.

Formellement, la traduction complète de cette spécification est une formule S' telle que

- S' est la formule S dans laquelle des identificateurs ont été substitués par d'autres et les atomes doublons supprimés.
- ∀ o1,o2 ∈ S' TYPE (o1) = TYPE (o2)
  TIG, HYPNU, S' ⊢ egal(o1,o2)

où egal est le prédicat d'égalité correspondant au type de o1 et de o2

HYPNU est un ensemble d'axiomes exprimant l'hypothèse de nom unique sur les objets identifiés de S'.

Concrètement, si un objet non-identifié de S est égal à un autre objet de S, alors il est représenté dans S' par un seul identificateur.

Il y a deux raisons à la définition d'une telle traduction, équivalente à la traduction simple vis-à-vis de TIG.

Du point de vue didactique, il parait intéressant que l'enseignant puisse savoir quels sont exactement les objets de la spécification qui sont différents du point de vue de la théorie TIG et qui sont donc tous à construire par l'élève pour répondre au contrat (voir 4.1.1.2) : comme l'hypothèse de nom unique n'est appliquée qu'aux objets identifiés, il dispose ainsi d'un moyen de contrôler les objets non-identifiés dont il demande la construction. De plus, comme il est intéressant d'empêcher l'élève de construire deux fois le même objet, la traduction de sa construction doit aussi vérifier que tous les objets sont différents deux à deux. De ce fait, il est plus cohérent que la traduction de la spécification vérifie aussi cette propriété.

Du point de vue de la réalisation informatique, il est préférable de calculer une fois pour toutes les égalités plutôt que de les démontrer à chaque recherche de preuve d'une propriété.

2.2.5. Synthèse des connaissances nécessaires pour l'utilisateur.

Du point de vue de l'utilisateur, il est important de retenir les caractéristiques principales et quelques points particuliers retenant l'attention.

Les caractéristiques principales à retenir sont les suivantes :

- CDL est un langage déclaratif. Cela permet à l'enseignant de décrire sa spécification sans se soucier d'un ordre. Cela a deux avantages. D'une part le fait de pouvoir faire référence à un objet sans l'avoir défini préalablement éloigne si besoin la spécification de la description d'une construction. D'autre part, cela lui permet de préciser la définition d'un objet en cours d'écriture de la spécification. Par exemple, s'il a oublié de préciser qu'un point appartient à une droite, il peut l'ajouter à la fin de sa spécification.

Néanmoins la déclarativité a un désavantage. Comme aucun ordre n'est imposé, il se peut que le professeur n'arrive pas très bien à savoir précisément ce qu'il a spécifié, en

particulier du point de vue de l'égalité des objets. C'est pour cela qu'il est sans doute utile de lui préciser quels sont les objets effectivement différents qui sont demandés par sa spécification.

• CDL permet des typages implicites.
Exemple :
Le prédicat d'appartenance relie toujours un point p à un autre objet o sans qu'il soit nécessaire de préciser le type de p.
Comme il permet de définir des égalités entre objets, le type d'un objet est donc déduit conjointement des typages implicites et des égalités posées.
Les incohérences sémantiques possibles d'une spécification viennent uniquement d'erreurs sur les types des objets définis :
- le type d'un identificateur n'est pas connu.
- un identificateur représente deux objets de types différents.
- un identificateur représente deux objets différents de même type.
- une égalité a été posée entre des termes de types différents.

• La spécification désigne des objets implicites. D'une part chaque terme représente un objet (`(A B)` signifie qu'il existe une droite passant par `A` et `B`); D'autre part la référence à certains objets est implicitement faite par la définition d'autres objets :
- la définition d'une demi-droite fait implicitement référence à une origine et une droite support.
- la définition d'un cercle fait implicitement référence à son centre et son rayon.
- la définition d'un segment fait implicitement référence à ses deux extrémités et à sa droite support.
Cela vient du choix des prédicats LDL représentant ces objets.

Par contre, CDL ne fait pas de référence implicite à des objets comme la longueur et les deux demi-droites d'un segment. Il faut donc que l'enseignant les décrive explicitement s'il veut les voir figurer dans la construction de l'élève.

• Chaque terme ne représente qu'un unique objet. Cela peut paraître une évidence, pourtant cela signifie que CDL ne permet pas d'exprimer des énoncés de certaines géométries non-euclidiennes.
Exemple :
Dans certaines géométries, par deux points il passe une infinité de droites.
Notre choix qu'un terme ne représente qu'un seul objet fait qu'il n'est pas possible que le terme `(A B)` puisse représenter deux objets différents.

Pour pouvoir définir des spécifications utiles dans n'importe quelle géométrie en gardant la contrainte assez naturelle qu'un terme ne représente qu'un seul objet, il faudrait que le professeur puisse définir lui-même les termes de CDL, comme le propose Schreck [Schreck 92].

Deux points particuliers retiennent l'attention :

• CDL permet que deux descriptions différentes de demi-droites désignent le même objet.
• le parallélisme et la perpendicularité sont uniquement définis sur des droites, bien que syntaxiquement on puisse exprimer par exemple qu'un segment est parallèle à une demi-droite. Dans le cas où l'expression ne porte pas syntaxiquement sur des droites, le parallélisme est défini sur les droites supports des objets décrits.

L'ensemble des choix que nous avons faits pour la sémantique de CDL nous paraît raisonnable au vu de l'objectif visé : fournir un langage proche de ceux utilisés dans les manuels scolaires et simple à comprendre.

2.3. SCL, un langage d'interface pour l'expression d'une construction géométrique.

Pour que l'élève exprime sa solution, TALC propose un langage de constructions géométriques dénommé SCL (Student Construction Language). Ce langage est défini par la sémantique des primitives qu'il offre et leurs conditions de validité.

Le second degré de liberté dont dispose le professeur dans l'expression du contrat didactique consiste à définir le sous-ensemble de primitives de SCL qu'il laisse à la disposition de l'élève.

Nous donnons en premier lieu notre méthodologie de définition de SCL, basée sur l'utilisation du micro-monde Cabri-géomètre et sur l'adoption de ce que Nicolas appelle "l'extension faible" [Nicolas 89].

Nous présentons ensuite la traduction des primitives du langage graphique de constructions géométriques de Cabri-géomètre en un langage textuel appelé "langage d'énoncé de Cabri-géomètre". Cela nous permet de définir la sémantique de SCL par la traduction d'une formule du langage d'énoncés de Cabri-géomètre en une formule de CDL, en nous référant à la sémantique précédemment donnée de CDL.

Enfin nous proposons un exemple de traduction d'une construction en SCL en une formule de LDL.

2.3.1. Méthodologie de définition de SCL.

2.3.1.1. Utilisation d'un langage existant.

Les premiers travaux sur la question de la correction d'une construction géométrique par rapport à une spécification ont mis l'accent sur la définition d'un tel langage [Nicolas 89].

Notre approche méthodologique pour définir un langage de construction pour l'élève est identique à celle de Nicolas. Elle consiste à définir le langage à partir de langages déjà existants, en les adaptant pour qu'ils correspondent à nos exigences.

Dans les travaux de Nicolas, la définition du langage de construction est basée sur les capacités offertes par une table à dessin. Cela permet d'atteindre un des objectifs de l'enseignement de la géométrie au collège : l'apprentissage de l'utilisation des outils de tracé géométrique (règle, équerre, compas).

Notre définition est basée sur celle du langage de construction proposé par le micro-monde Cabri-géomètre. Le choix de Cabri-géomètre, outre les facilités que nous offre notre participation au Projet Cabri-géomètre, est basé sur la très large diffusion dont dispose aujourd'hui ce micro-monde (on compte environ cinquante mille licences vendues, aussi bien en Europe, qu'en Amérique, en Afrique et au Sud-Est asiatique). Cette diffusion représente un atout important d'une part pour les campagnes d'expérimentation, où la prise en main de l'outil est plus courte pour des élèves maîtrisant Cabri-géomètre, d'autre part pour la diffusion de TALC. Bien entendu la diffusion de Cabri-géomètre n'est pas son seul atout. Ces qualités autant informatiques que didactiques sont nombreuses et il apparaît aujourd'hui comme la référence en matière de micro-monde de construction géométrique. Geometer Sketchpad (son concurrent le plus diffusé), nous l'avons déjà fait remarquer, ne présente pas d'aussi bonnes qualités scientifiques et techniques.

2.3.1.2. Notion d'"extension faible".

Dans sa thèse, Nicolas distingue deux types de langages de construction géométriques : les langages basés sur l'extension fortes et ceux basés sur l'extension faible.

Définitions :

Avec le principe de l'"extension forte", la construction d'un nouvel objet est possible seulement si son existence est assurée par les constructions précédentes vis-à-vis de la théorie.

Exemple :

La construction de l'intersection de deux droites n'est possible que si on peut déduire de la construction courante et de TIG qu'elles ne sont pas parallèles. C'est le cas si elles sont perpendiculaires et que TIG comprend l'axiome :

```
Toute droite perpendiculaire à une seconde ne lui est pas parallèle.
```

Avec le principe de l'"extension faible", la construction d'un objet est possible même si les conditions de son existence ne sont pas remplies. Dans ce cas, la construction du nouvel objet ajoute implicitement les propriétés supplémentaires non déductibles de la construction précédente.

L'extension faible est le principe choisi par Nicolas comme représentant la manière de construire une figure sur une table à dessin. Nous l'adoptons pour les mêmes raisons pour SCL : cela permet à l'élève des constructions proches de celles réalisées dans le cadre habituel (papier et crayon). L'extension forte peut être adoptée quand les objectifs sont différents, par exemple dans le cadre de l'exploration des propriétés d'une figure à l'aide d'animation de construction automatisées, comme dans GEOSPECIF (voir chapitre premier, 2.4.1).

2.3.2. Traduction des primitives graphiques dans le langage d'énoncé de Cabri-géomètre .

Nous présentons dans ce sous-chapître les primitives que propose Cabri-géomètre et leur traduction dans ce que nous appelons le "langage d'énoncés" de Cabri-géomètre ainsi que les conditions de leur validité.

2.3.2.1. Primitives graphiques de Cabri-géomètre.

Les objets géométriques manipulés par Cabri-géomètre sont constructibles à partir des deux menus que propose le logiciel à l'utilisateur (cf figure II.3).

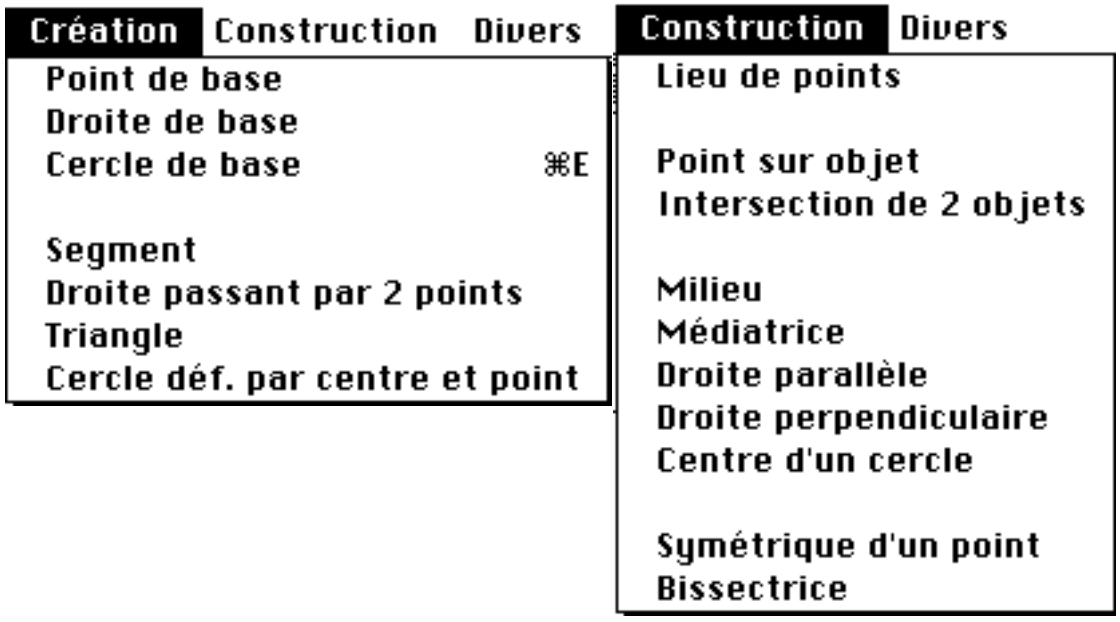


Figure II.3

A l'aide de ces menus, Cabri-géomètre permet à l'utilisateur de créer les cinq types d'objets suivants : point, droite cercle, segment, triangle. Chacun des items de menu représente la construction d'un objet de ce type, en spécifiant les relations géométriques le liant aux objets précédemment construits.

Exemple :

L'item "droite perpendiculaire" permet de construire une droite perpendiculaire à une droite précédemment construite et passant par un point donné.

Les primitives de constructions offertes par ces items sont activées en deux temps :

• dans un premier temps, l'utilisateur choisit une primitive de construction en sélectionnant un item d'un menu.

• dans un second temps, il désigne à l'aide de la souris les paramètres effectifs de la primitive qui sont :

- soit un objet préalablement construit, auquel cas un message signale à l'utilisateur que l'objet près duquel est positionné le curseur de la souris est candidat à être paramètre, car de type correspondant.

- soit un objet non encore construit, auquel cas la construction de l'objet paramètre est effectuée "à la volée". Cette seconde possibilité est offerte uniquement pour les items *Segment*, *Droite passant par 2 points* , *Triangle* et *Cercle déf. par centre et point*.

Bien entendu, comme dans tout langage, l'ordre des paramètres doit être respecté.

Entre chaque utilisation d'une primitive de construction, l'utilisateur peut donner des noms aux objets de type point, droite ou cercle qu'il a construits. Cabri-géomètre attribue systématiquement un nom à chaque objet de ce type construit, respectant la grammaire suivante :

P# <entier> pour les points

D# <entier> pour les droites

C# <entier> pour les cercles.

L'utilisateur peut nommer les objets en utilisant l'ensemble du jeu de caractère disponible sur la machine.

2.3.2.2. Traduction des primitives graphiques.

Au cours de la construction, l'ensemble des objets construits et des relations les liant (et non nécessairement représentées graphiquement) est visualisable à la demande de l'utilisateur dans une forme textuelle.

Définitions :
Nous appelons le langage qu'utilise cette forme textuelle le "langage d'énoncés de Cabri-géomètre".
Nous appelons un texte en langage d'énoncés de Cabri-géomètre un "énoncé de Cabri-géomètre".

NB : nous nous référons à la version du langage d'énoncés décrivant les objets sans leurs caractéristiques numériques.

Exemple :
L'énoncé correspondant à la construction de la figure II.4, répondant à la spécification :
`s = [A B], A = centre(K), B e K, B = centre(K2), A e K2, |A I|=1/2|A B|, I e S, P e K1, P e K2, L=(I P),`
décrivant la médiatrice `(I P)` de `[A B]` est le suivant :
A point de base
B point de base
segment [A B]
I milieu des deux points A et B
K1 cercle de centre A passant par B
K2 cercle de centre B passant par A
P intersection du cercle K1 et du cercle K2
P#6 intersection du cercle K1 et du cercle K2 (P est l'autre point)
L droite passant par I et P

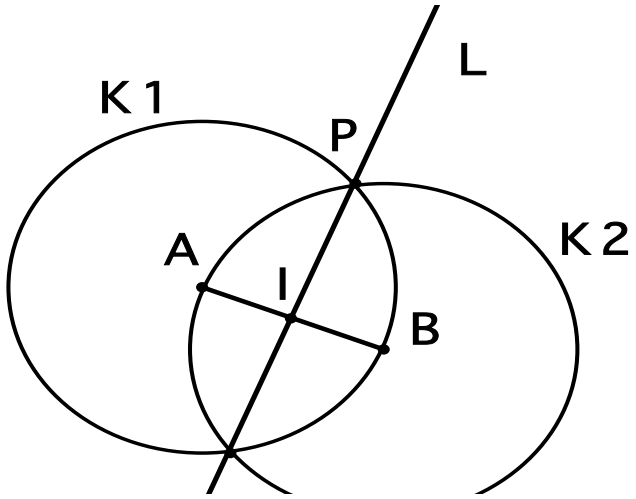


Figure II.4.

L'ensemble du langage d'énoncés est défini en annexe D.

Dans le tableau suivant, nous donnons les primitives de construction possibles pour chaque type d'objet, les paramètres requis et la traduction que Cabri-géomètre donne de chaque primitive dans le langage d'énoncés.

Type d'objet construit	Primitive de construction	Type des paramètres	Traduction en langage d'énoncés
point	Point de base	*pas de paramètre*	P#n : point quelconque
	Point sur objet	droite	P#n : point de la droite D#n
		segment	P#n : point du segment [P#n1 P#n2]
		cercle	P#n : point du cercle C#n
	Intersection de 2 objets	droite, droite	P#n : intersection des 2 droites D#n1 et D#n2
		droite, segment segment, droite	P#n : intersection du segment [P#n1 P#n2] et de la droite D#n
		droite, cercle cercle, droite	P#n1 : intersection de la droite D#n et du cercle C#n P#n2 : intersection de la droite D#n et du cercle C#n (P#n1 est l'autre point)
		segment, segment	P#n : intersection du segment [P#n1 P#n2] et du segment [P#n3 P#n4]
		segment, cercle cercle, segment	P#n1 : intersection du segment [P#n3 P#n4] et du cercle C#n P#n2 : intersection du segment [P#n3 P#n4] et du cercle C#n (P#n1 est l'autre point)
		cercle, cercle	P#n1 : intersection du cercle C#n1 et du cercle C#n2 P#n2 : intersection du cercle C#n1 et du cercle C#n2 (P#n1 est l'autre point)
	Milieu	point, point segment	P#n : milieu des 2 points P#n1 et P#n2
	Centre d'un cercle	cercle	P#n : centre du cercle C#n
	Symétrique	point, droite	P#n : symétrique du point P#n1 par rapport au point P#n2
			P#n : symétrique du point P#n1 par rapport à la droite D#n
droite	Droite de base	pas de paramètre	D#n : droite quelconque
	Droite passant par 2 points	point, point	D#n : droite passant par P#1 et P#2
	Droite parallèle	point, droite	D#n : droite passant par P#n et parallèle à D#n1
		point, segment	D#n : droite passant par P#n et parallèle au segment [P#n1 P#n2]

	Droite perpendiculaire	point, droite	D#n : droite passant par P#n et perpendiculaire à D#n1
		point, segment	D#n : droite passant par P#n et perpendiculaire au segment [P#n1 P#n2]
	Médiatrice	point, point segment	D#n : médiatrice de P#n1 et P#n2
	Bissectrice	point, point, point	D#n : bissectrice de l'angle formé par P#n1 P#n2 P#n3 % ENLEVEE
segment	Segment	point, point	segment [P#n1 P#n2]
cercle	Cercle de base	pas de paramètre	C#n : cercle quelconque
	Cercle déf. par centre et point	point, point	C#n : cercle de centre P#n1 passant par P#n2
triangle	Triangle	point, point, point	triangle P#n1 P#n2 P#n3

Le sens de deux primitives doit être précisé :

• *Symétrique* est une primitive construisant soit le symétrique orthogonal d'un point donné par rapport à une droite donnée, soit le symétrique d'un point par la symétrie centrale de centre un point donné.

• *Bissectrice* est une primitive construisant la bissectrice du plus petit angle formé par les droites (non obligatoirement construite) passant respectivement par les premier et deuxième points et par les deuxième et troisième points désignés comme paramètres.

2.3.2.3. Conditions d'utilisation des primitives de Cabri-géomètre.

Une construction utilisant les primitives de Cabri-géomètre doit vérifier des conditions de validité pour être acceptées. Ces conditions sont principalement de trois types :

• les paramètres des primitives doivent respecter le type attendu. L'utilisateur ne peut enfreindre cette condition du fait de l'interface : l'interface ne propose que des objets dont le type a été reconnu adéquat comme candidat à être paramètre d'une primitive. La condition est donc vérifiée a priori.

• l'objet construit doit pouvoir exister.

Exemple :

Si l'utilisateur définit un cercle de base de rayon nul, c'est à dire en désignant le même point comme centre du cercle et comme point appartenant au cercle, l'existence du cercle n'est pas assurée.

Dans ce cas, la condition est vérifiée a posteriori et la construction n'est pas affichée.

• l'objet construit ne doit pas être égal à un objet précédent. Mais cette condition d'identité est limitée à une identité syntaxique.

Exemples :

Si l'utilisateur a déjà construit la droite passant par les deux points A et B, la construction de la droite passant par B et A est rejetée, de même que la répétition de la construction de la droite passant par A et B.

Par contre, si l'utilisateur a déjà construit la droite passant par les deux points A et B et un point C sur cette droite, la construction de la droite passant par A et C (ou par B et C) n'est pas rejetée.

2.3.3. Traduction d'énoncés de Cabri-géomètre en CDL.

Nous présentons en premier lieu la sémantique d'une construction en SCL par sa traduction en LDL. En second lieu nous précisons les conditions de validité des primitives de SCL.

2.3.3.1. Traduction simple d'un énoncé.

Définition :
Nous appelons sémantique de SCL la traduction de l'énoncé Cabri-géomètre représentant une construction faite à partir de SCL en une formule de LDL.

La traduction d'un énoncé en une formule LDL est comparativement plus simple que celle d'une formule CDL du fait que cet énoncé traduit une construction. Une construction est un cas particulier de spécification telle que la définition d'un objet ne se réfère qu'à des objets précédemment définis. De ce fait, la sémantique de SCL peut être exprimée en fonction de celle de CDL (voir 2.3.3).

Définitions :
Nous appelons "phrase" du langage d'énoncé de Cabri-géomètre une suite de caractères comprise entre deux retours-chariot (voir définition du langage en Annexe D).
Un énoncé de Cabri-géomètre est alors défini comme une suite de phrases.

Nous donnons dans le tableau suivant la traduction de chaque phrase d'énoncé type (correspondant à l'application d'une primitive graphique : voir tableau précédent) de Cabri-géomètre en une formule CDL.
Les identificateurs apparaissant dans une phrase représentent un identificateur de Cabri-géomètre (soit un identificateur crée par Cabri-géomètre, soit un identificateur choisi par l'utilisateur).

Phrase du langage d'énoncés	traduction en CDL
p : point quelconque	point(p).
p : point de la droite d	p e d.
p : point du segment [p1 p2]	p e [p1 p2].
p : point du cercle c	p e c.
p : intersection des 2 droites d1 et d2	p e d1, p e d2.
p : intersection du segment [p1 p2] et de la droite d	p e [p1 p2], p e d.
p1 : intersection de la droite d et du cercle c	p1 e d, p1 e c.
p2 : intersection de la droite d et du cercle c (p1 est l'autre point)	p2 e d, p2 e c.
p : intersection du segment [p1 p2] et du segment [p3 p4]	p e [p1 p2], p e [p3 p4].
p1 : intersection du segment [p3 p4] et du cercle c	p1 e [p3 p4], p e c.
p2 : intersection du segment [p3 p4] et du cercle c (p1 est l'autre point)	p1 e [p3 p4], p e c.
p1 : intersection du cercle c1 et du cercle c2	p1 e c1, p1 e c2.
p2 : intersection du cercle c1 et du cercle c2 (p1 est l'autre point)	p2 e c1, p2 e c2.

p : milieu des 2 points p1 et p2	|p1 p|=|p p2|, p e (p1 p2).
p : centre du cercle c	p = centre(c)
p : symétrique du point p1 par rapport au point p2	|p p2|=|p2 p1|, p e (p1 p2).
p2 : symétrique du point p1 par rapport à la droite d	(p1 p2) !- d, p e (p1 p2), p e d, |p□2|=|p2□1|.
d : droite quelconque	droite(d).
d : droite passant par p1 et p2	d = (p1 p2).
d1 : droite passant par p et parallèle à d2	d1 !- d2, p e d1.
d : droite passant par p1 et parallèle au segment [p2 p3]	d !- [p2 p3], p1 e d.
d1 : droite passant par p et perpendiculaire à d2	d1 // d2, p e d1.
d : droite passant par p1 et perpendiculaire au segment [p2 p3]	d // [p2 p3], p1 e d.
d : médiatrice de p1 et p2	d !- (p1 p2), p e d, |p1 p|=|p p2|, p e (p1 p2).
segment [p1 p2]	s = [p1 p2]
c : cercle quelconque	cercle(c)
c : cercle de centre p1 passant par p2	p1 = centre(c), p2 e c.
triangle p1 p2 p3	*RIEN*

La seule difficulté pour traduire une phrase d'énoncé de Cabri-géomètre en une formule en CDL réside dans la traduction des segments. Comme le langage d'énoncés ne nomme pas les segments (car le langage graphique ne le permet pas), il faut ajouter un identificateur CDL pour chaque segment.

Exemple :

L'énoncé

```
segment[A B]
segment[C D]
```

est traduite en CDL par

```
s1 = [A B],
s2 = [C D].
```

Il faut aussi noter que la traduction de l'énoncé

```
triangle p1 p2 p3
```

est vide, car d'une part le type triangle n'existe pas en CDL (ni en LDL) et d'autre part le fait de désigner un triplet de points comme un triangle ne lui ajoute pas de propriétés géométriques.

Notons encore que nous n'avons pas retenu la primitive de construction de bissectrices. Ce choix provient de la complexité et de la diversité des définitions que l'on peut en donner. Nous avons préféré proposer des primitives à la sémantique simple, quitte à proposer par la suite au professeur des moyens d'étendre le langage par l'utilisation de macro-constructions. Ainsi il pourra définir lui même de façon exacte et claire le sens qu'il donne à des objets aux constructions complexes et multiples. Pour lui éviter ce travail qui peut être complexe de son point de vue, le système pourra aussi lui proposer un ensemble de macro-constructions prédéfinies correspondant par exemple à la définition donnée par tel ou tel manuel scolaire.

En effet, la primitive de construction de bissectrices que propose Cabri-géomètre est une primitive ayant trois points comme paramètres.

Exemple :
La figure II.5 représente la bissectrice des trois points A B et C construite par Cabri-géomètre.

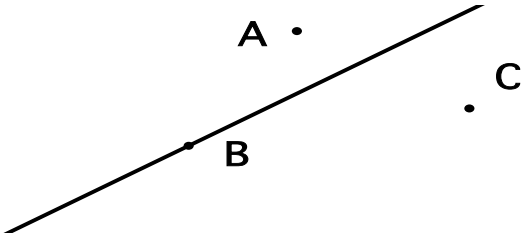


Figure II.5.

Pour traduire cette figure en terme des types d'objets primitifs de Cabri-géomètre (points, droites, cercles et segments), il faut passer par un nombre considérable d'objets intermédiaires, comme le montre la figure II.6.

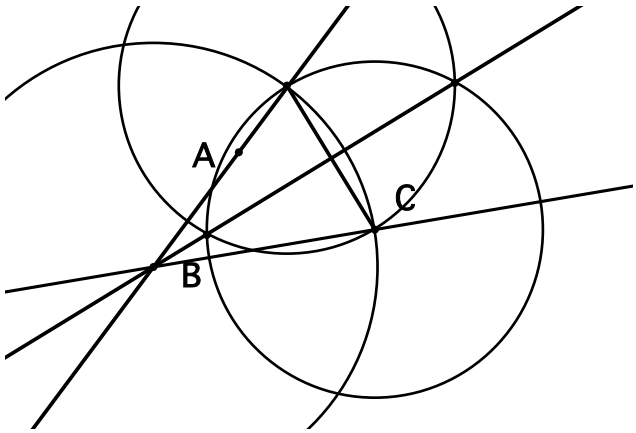


Figure II.6.

Le tableau que nous venons de donner définit la traduction simple d'une phrase d'énoncé de Cabri-géomètre. Nous avons fait le choix de ne pas considérer de traduction simple de l'ensemble d'un énoncé de Cabri-géomètre du fait des conditions d'utilisation des primitives du point de vue de l'égalité. Nous exposons cette question dans ce qui suit.

2.3.3.2. Conditions d'utilisation des primitives de SCL.

Une construction utilisant les primitives de SCL doit vérifier bien évidemment les conditions de validité des primitives de Cabri-géomètre. Cependant les conditions exigées par SCL sont plus exigeantes concernant l'égalité. En effet, puisque l'on connaît la théorie géométrique représentant les connaissances de l'élève, il paraîtrait incohérent de le laisser construire deux objets identiques vis-à-vis de cette théorie.

Cela signifie que tout objet construit en SCL doit être différent de chaque objet précédemment construit vis à vis de la théorie TIG..
Exemple :
Si l'élève a déjà construit la droite passant par les deux points A et B et un point C sur cette droite, si TIG contient le deuxième postulat d'Euclide (`Pour tout point P et tout point Q non egal à P la droite L, telle que P et`

`Q sont sur L, est unique`) la construction de la droite passant par `A` et `C` (ou par `B` et `C`) est rejetée sinon elle est acceptée.

Cette condition sur l'égalité des objets de la construction est sensiblement différente de celle définie pour le langage SCL.

Une sémantique similaire à celle de CDL serait de considérer que tous les objets de la traduction de la construction doivent être différents deux à deux.

Une traduction complète serait alors définie par :
Soit F la traduction simple d'une construction.
Formellement, la traduction complète de cette construction est une formule F' telle que
• F' est la formule F dans laquelle des identificateurs ont été substitués par d'autres et les atomes doublons supprimés.
• ∀ o1,o2 ∈ F' TYPE (o1) = TYPE (o2)
TIG, F' ⊢ egal(o1,o2)
où egal est le prédicat d'égalité correspondant au type de o1 et de o2

Cependant, du fait que SCL est un langage utilisé interactivement, chaque construction est traduite avant que la suivante puisse être effectuée. Dans ce contexte, il est naturel de considérer que l'introduction d'un nouvel objet ne remet pas en cause le fait que tous les objets précédemment construits sont différents deux à deux.

Une traduction complète est alors définie formellement par :
soit P1, P2, …, Pn une suite de phrases décrivant un énoncé Cabri-géomètre.
• la traduction complète F1' de P1 est telle que :
- F1' est la formule F1 dans laquelle des identificateurs ont été substitués par d'autres et les atomes doublons supprimés.
-∀ o1,o2 ∈ F1' TYPE (o1) = TYPE (o2)
TIG, F1' ⊢egal(o1,o2)
avec F1 la traduction simple de P1.
• ∀ i>1, la traduction complète Fi' de P1, …, Pi est telle que :
Soit Fi = Fi-1 @ EAi
où EAi est l'ensemble des atomes de la traduction de la phrase Fi.
- Fi' est la formule Fi dans laquelle des identificateurs ont été substitués par d'autres et les atomes doublons supprimés.
-∀ o1 ∈ Fi-1, ∀ o2 ∈ EAi TYPE (o1) = TYPE (o2)
TIG, Fi' ⊢egal(o1,o2)

Concrètement, cela signifie qu'au cours de l'interaction, si la condition n'est pas vérifiée pour une phrase donnée, le pas de construction est refusé à l'élève qui doit reprendre au pas précédent.

C'est ce choix que nous avons fait pour la sémantique de SCL. Il nous semble raisonnable, bien qu'il soit possible en théorie que l'introduction d'un nouvel objet permettre de déduire une égalité sur des objets précédemment construits. En effet, l'adoption de l'extension faible (voir 2.3.1.2) a pour conséquence l'instabilité des déductions possibles sur les objets déjà construits.
Exemple :
Si la construction comprend deux droites et que l'on construit leur point d'intersection, alors la propriété qu'elles sont non parallèles devient démontrable alors que ce n'était pas le cas auparavant.

Si en théorie cette situation est possible, aucun exemple concret n'a pu être exhibé dans notre système. Nous présumons qu'il n'est pas possible d'en trouver du fait des propriétés des constructions dans le système, sans avoir pu le montrer.

Notons de plus, qu'à la différence de CDL, l'hypothèse de nom unique n'est pas retenue. Comme tout objet (sauf les segments) d'une construction reçoit un nom dans l'énoncé Cabri-géomètre, son adoption signifierait tous les objets sont différents a priori, ce qui n'est pas le cas évidemment.

2.3.4. Exemple de traduction d'une construction en SCL.

Soit la construction permettant d'obtenir la médiatrice `(I P)` de `[A B]` représenté graphiquement par la figure II.7.

Son énoncé Cabri-géomètre et la traduction correspondante en LDL sont représentés dans le tableau suivant :

Énoncé Cabri-géomètre	Traduction en LDL
A point de base	point(A)
B point de base	point(B)
segment [A B]	segment(var1,A, B,var2) droite(var2)
I milieu des deux points A et B	point(I) distance(var3) distancep(var3,A,I) distancep(var3,I,B) appdr(I,var2)
K1 cercle de centre A passant par B	cercle(K1,A,var4) distance(var4) appcc(B,K1)
K2 cercle de centre B passant par A	cercle(K2,B,var5) distance(var5) appcc(B,K2)
P intersection du cercle K1 et du cercle K2	appcc(P,K1) appcc(P, K2)
P#6 intersection du cercle K1 et du cercle K2 (P est l'autre point)	appcc(VarP6,K1) appcc(VarP6, K2)
L droite passant par I et P	droite(AL) appdr(AI,AL) appdr(AP,AL)

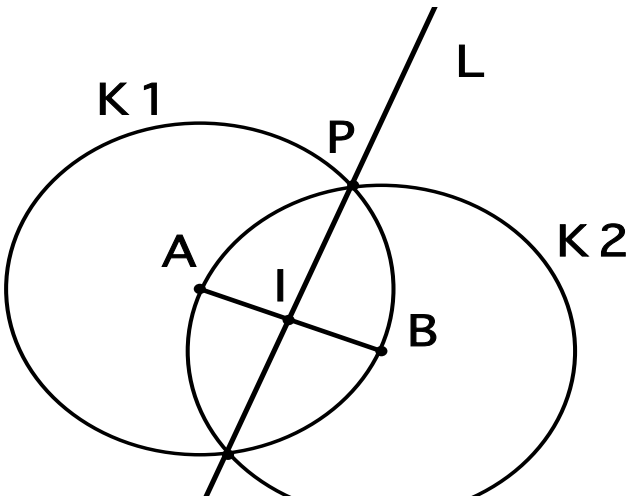


Figure II.7.

3. Principes de formulation d'une théorie logique TIG.

Aux différentes étapes de son apprentissage, la connaissance de l'élève en géométrie évolue. Cette connaissance est représentée dans TALC par une "Théorie Instrumentale de la Géométrie" (TIG). TIG est décrite par un ensemble d'axiomes. Cela signifie que l'ensemble des formules valides selon TIG est engendré par l'application des règles de déduction de la logique du premier ordre à cet ensemble d'axiomes.

L'objectif de ce chapitre est de décrire la méthode que nous avons définie pour la formulation de théories TIG. Nous présentons en premier lieu le choix du langage utilisé pour représenter les axiomes de TIG. Nous présentons ensuite la forme des axiomes que nous avons retenue et les raisons de ce choix. La forme des axiomes étant définie, une question reste à résoudre : comment choisir le contenu de la théorie TIG? Nous présentons pour répondre à cette question une méthodologie de définition de TIG et proposons une première théorie, TIG_0, à partir de laquelle l'enseignant peut définir les théories qui lui conviennent par ajout et retrait d'axiomes.

3.1. Choix du langage utilisé pour représenter les axiomes.

Pour représenter les axiomes d'une théorie géométrique, une première idée consiste à se baser sur les langages utilisés par les axiomatisations classiques.

Euclide est le premier à avoir donné une formulation de la géométrie sous forme axiomatique. Hilbert, puis Tarski ont successivement amélioré cette formulation axiomatique, le premier en précisant un certain nombre de termes indéfinis et en s'appuyant sur la théorie des ensembles, le second en s'appuyant uniquement sur la logique du premier ordre.

Le langage d'expression des axiomes qu'ils utilisent est sensiblement le même. Outre les connectives et les quantificateurs de la logique du premier ordre, ils utilisent trois prédicats :

- le prédicat binaire ÉGAL. ÉGAL(o1,o2) exprime que o1 et o2 sont identiques.
- le prédicat ternaire ENTRE. ENTRE(o1,o2,o3) exprime que le point o2 est entre les points o1 et o3.
- le prédicat quaternaire CONGRUENTS. CONGRUENTS(o1,o2,o3,o4) exprime que la distance entre les points o1 et o2 est la même que celle entre les points o3 et o4.

Du point de vue mathématique, le petit nombre de prédicats utilisés permet une sémantique à la fois claire et précise de ce qu'est la géométrie dans son ensemble. Ce langage permet ainsi de représenter la géométrie sous la forme la plus synthétique possible.

Du point de vue d'un système d'enseignement, le langage choisi doit permettre de représenter une connaissance de la géométrie qui évolue. Il ne s'agit pas alors de disposer d'un langage le plus synthétique possible, mais plutôt d'un langage adapté à l'évolution de la connaissance et dont les explications qui l'utilisent soient compréhensibles par l'élève.

De ce dernier point de vue, le langage utilisé par ces mathématiciens n'est pas adapté à la description d'une théorie TIG sous-ensemble de la géométrie. Un langage adapté à notre contexte est un langage de la logique du premier ordre utilisant des prédicats représentant directement les connaissances enseignées à l'élève.

À partir des critères que nous venons de définir, nous choisissons de représenter les axiomes de TIG avec le langage LDL : ses prédicats sont proches des connaissances manipulées par l'élève et c'est un langage de la logique du premier ordre.

On remarque de plus que, puisque la sémantique du langage CDL est définie à partir de celle de LDL, le langage utilisé en pratique peut être indifféremment l'un ou l'autre de ces deux langages.

3.2. Choix de la forme des axiomes.

À partir du langage que nous avons choisi pour représenter les axiomes, il importe de définir quelles sont les formes acceptées pour ces axiomes. LDL étant un langage du premier ordre, toute formule du premier ordre est possible a priori.

Nous présentons dans ce sous chapitre les raisons de notre choix de deux restrictions sur la forme des formules acceptées pour représenter les axiomes :

- formules universelles sans symboles de fonctions.
- clauses de Horn.

Nous définissons ensuite la manière dont nous traitons la négation par rapport à la restriction à des clauses de Horn.

3.2.1. Restriction à des formules universelles sans symboles de fonctions.

Si l'on veut réellement disposer d'une mise en œuvre de TALC, il faut pouvoir s'assurer que les problèmes à résoudre sont décidables. Or vérifier qu'une propriété est valide dans la logique du premier ordre n'est pas décidable en général. Comme la théorie TIG est un des degrés de liberté mis à la disposition du professeur, on ne peut compter sur une théorie qui ait la propriété particulière (pour une théorie exprimée dans la logique du premier ordre) d'être décidable a priori.

Un remède à ce problème est possible : il s'agit de restreindre les formules permettant d'exprimer des axiomes de TIG de façon à contraindre l'univers de Herbrand à être fini.

La solution que nous avons adoptée pour garantir cette finitude consiste à restreindre les formules à la forme dite de Bernays-Schönfinkel [Ackerman 84]. Cette restriction impose aux formules de ne contenir ni termes fonctionnels, ni quantification existentielle.

Exemple :

L'axiome

```
∀ p1, p2 point(p1)∧ point(p2) ⇒ ∀d droite(d)∧ p1 ∈ d ∧ p2 ⇒∈d.
```

exprimant le deuxième postulat d'Euclide ne respecte pas la forme de Bernays-Schönfinkel.

En effet, la finitude de l'univers de Herbrand n'est pas garantie si les formules contiennent un seul terme fonctionnel. Mais la restriction à des formules sans termes fonctionnels n'est pas suffisante : l'usage de quantificateurs existentiels revient à l'utilisation de termes fonctionnels, par skolémisation.

Concrètement, cela signifie que les formules du type $\forall x \exists y P(x,y)$ ou $\forall x P(x,f(x))$ ne sont pas autorisées.

Il n'y a pas lieu de restreindre les formules de la logique du premier ordre utilisant LDL concernant les symboles de fonction car ce langage a été préalablement défini au regard de ce problème. De ce fait, il ne contient pas de symboles de fonction.

Par contre, la restriction concernant l'usage des symboles existentiels n'est pas sans importance car les axiomatisations classiques de la géométrie comportent un nombre important d'axiomes ne respectant pas la forme de Bernays-Schönfinkel. Par exemple, sur les cinq postulats d'Euclide (voir 3.3.1) quatre sont existentiels.

Exemple :

L'axiome

```
∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧ p1
∈ d ∧ p2 ∈ d ∧ p1=p2   d1 =d2
```

est une version restreinte du deuxième postulat d'Euclide respectant la forme de Bernays-Schönfinkel mais supposant l'existence de deux droites et de deux points.

Cet exemple montre que la restriction que nous adoptons permet a priori d'exprimer une propriété importante des théories TIG descriptibles de cette façon :

TIG ne peut être qu'au mieux incluse strictement dans toute théorie exprimée en utilisant des formes existentielles, comme celles d'Euclide, d'Hilbert ou de Tarski.

3.2.2. Restriction à des clauses de Horn

La restriction à des formules de Bernays-Schönfinkel n'est pas la seule que nous adoptons pour la forme des axiomes de TIG. Nous restreignons aussi les formules à des clauses de Horn comportant exactement un littéral positif (que nous désignons plus simplement par la suite par "clauses de Horn").

Les raisons de cette restriction sont les suivantes :

• un ensemble de clause de Horn peut être vu comme un ensemble de paquets de clauses, chaque paquet définissant un prédicat. Il suffit de représenter chaque clause sous la forme `T ∨ Q1 Q2 … Qn,` où `T, Q1, … Qn` sont des atomes du langage. Cela permet de donner une description facilement compréhensible de la théorie, prédicat par prédicat.

Exemple :

Considérons les axiomes suivants concernant le prédicat par (parallélisme), exprimés en CDL :

1) Symétrie

```
∀ l1,l2
l1 // l2 ⇒ l2 // l1
```

2) Transitivité

```
 ∀ l1,l2,l3
l1 // l2 ∧ l1 // l3 ⇒ l3 // l2
```

3) Droite des milieux

```
∀ l1,l2,A,B,C,D,E
l1 // l2 ∧ l1 = (D E) ∧ l2 = (B C) ∧ D e [A B] ∧ E e [A C] ∧
|A∈D|=|D∈B| ⇒ |A E|=|E C|
```

Suivant que les axiomes 1, 2 et 3 sont présents ou non dans la théorie, on peut décrire précisément les connaissances de l'élève sur le parallélisme.

• les clauses de Horn permettent de fournir des explications par un raisonnement facile à comprendre (de buts en sous-buts). Un exemple contraire est celui des déductions faites par la règle de résolution sur des clauses générale : ce type de déductions est diffcile à expliquer simplement à un élève.

• elles apportent une solution au problème de l'expression d'un contrat didactique ne requérant pas la construction de plusieurs figures exclusives (voir 4.4).

Exemple d'axiomes :

Les trois axiomes de l'exemple précédent sont exprimés en LDL par :

1) Symétrie

```
∀ l1,l2
par(l1,l2) ⇒ par(l2,l1)
```

2) Transitivité

```
∀ l1,l2,l3
par(l1,l2) ∧ par(l1,l3) ⇒ par(l3,l2)
```

3) Droite des milieux

```
∀ l1,l2,A,B,C,D,E,s1,s2,l1,l2,d1,d2
par(l1,l2) ∧ appdr(D,l1) ∧ appdr(E,l1) ∧ appdr(B,l2) ∧ appdr(A,l2) ∧
segment(s1,A,B,l3) ∧ segment(s2,A,C,l4) ∧ appseg(D,s1) ∧ appseg(E,s2)
∧ distancep(d1,A,D) ∧ distancep(d1,D,B) ∧ distancep(d2,A,E) ∧
distancep(d2,E,C)
```

De même que pour la restriction de Bernays-Schönfinkel, la restriction à des clauses de Horn ne permet pas d'exprimer certains axiomes des théories classiques de la géométrie.

3.2.3. Problèmes posés par le traitement de la négation.

La restriction de la forme des axiomes de TIG aux clauses de Horn ne permet d'exprimer que des clauses ne possédant qu'un seul littéral positif. Autrement dit, un axiome de TIG doit être de la forme

$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$

avec $n \geq 0$ et $\forall i$ p_i littéral positif et q littéral positif.

Malheureusement, même des axiomes très simples de la géométrie ne peuvent s'exprimer de cette façon.

Exemple :

L'axiome

```
∀ p1,p2,d1,d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧
appdr(p1,d1) ∧ appdr(p2,d1) ∧ appdr(p1,d2) ∧ appdr(p2,d2)
∧ egal_dr(d1,d2)   egal_pt(p1,p2)
```

exprimant le second postulat d'Euclide (par deux points différents ne passe qu'une droite) ne peut être formulé avec une clause de Horn.

Pour exprimer cet axiome avec une implication, on peut donner les deux formes équivalentes suivantes :

```
1)∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧
p1 ∈ d ∧ p2 ∈ d ∧ ¬egal_pt(p1,p2) ⇒ egal_dr(d1,d2)

2)∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧
p1 ∈ d ∧ p2 ∈ d ∧ ¬egal_dr(d1,d2) ⇒ egal_pt(p1,p2)
```

Le problème des clauses de Horn est qu'elles interdisent d'avoir plus d'un littéral négatif. Si on considère la forme utilisant une implication, cela signifie que les formules de la forme :

$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$

où p_i peut être un littéral négatif

ne sont pas acceptées.

Ce problème est très gênant car il est clair que se passer d'axiomes si simple est très difficile dans notre contexte.

Pour pouvoir accepter des axiomes de cette forme, une idée consiste à utiliser la SLD-résolution [Lloyd 87] à laquelle une règle de négation par échec pour les sous-but négatifs est ajoutée.

Informellement, cette règle s'énonce ainsi :

si le sous-but à prouver est un littéral négatif de la forme ¬ p(x)

alors si p(x) peut être prouvé, ¬p(x) n'est pas prouvé

si p(x) ne peut être prouvé, ¬p(x) est prouvé

Cette SLD-résolution "avec échec" est complète si :

- la théorie est sous forme de clauses de Horn
- la question posée (i.e. une formule telle que sa négation est ajoutée à l'ensemble de clauses pour prouver une inconsistance) contient des littéraux négatifs clos.

Si la question n'est pas close, cette méthode n'est pas complète.

Exemple :

Considérons l'ensemble d'axiomes du premier ordre

- riche(Pierre)
- homme(Pierre)
- homme(Paul)

La formule ¬ riche(x) homme(x) n'est pas déductible de cet ensemble en utilisant la règle de négation par échec, car ¬ riche(x) n'est jamais prouvé. Par contre la formule homme(x) ¬ riche(x) est déductible avec x = Paul.

Malheureusement, ce cas de complétude ne correspond pas à notre cas, car les axiomes de la géométrie ne sont pas tous sous forme de clause de Horn.

Une seconde manière d'assurer la complétude de cette méthode consiste à vérifier que la théorie est stratifiée [Apt & al. 88].

Définitions :

Une "fonction de niveaux" d'un ensemble de formules est une fonction de l'ensemble des symboles de prédicats vers les entiers naturels. L'image d'un symbole de prédicat par la fonction est appelle son "niveau".

Un ensemble de clauses est dit "stratifié" s'il existe une fonction de niveau f telle que pour toute clause de la forme

$p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q$

$\forall$ i $f(\text{Pred}(p_i)) \leq f(\text{Pred}(q))$ si p_i est un littéral positif

et $f(\text{Pred}(p_i)) < f(\text{Pred}(q))$ si p_i est un littéral négatif

avec Pred(l) le symbole de prédicat du littéral l.

De façon intuitive cela signifie que dans un ensemble de clauses stratifié il ne peut y avoir de définition récursive de la négation d'un prédicat, c'est à dire de définition du prédicat en fonction de sa négation.

Malheureusement, on s'aperçoit dans ce cas aussi que les axiomes les plus simples de la géométrie ne sont pas stratifiés.

Exemple :

Les axiomes

1) $\forall$ p1, p2, d1, d2 point(p1) $\wedge$ point(p2) $\wedge$ droite(d1) $\wedge$ droite(d2) $\wedge$ p1 $\in$ d $\wedge$ p2 $\in$ d $\wedge$ ¬egal_pt(p1,p2) $\rightarrow$ egal_dr(d1,d2)

2) $\forall$ p1, p2, d1, d2 point(p1) $\wedge$ point(p2) $\wedge$ droite(d1) $\wedge$ droite(d2) $\wedge$ p1 $\in$ d $\wedge$ p2 $\in$ d $\wedge$ ¬egal_dr(d1,d2) $\rightarrow$ egal_pt(p1,p2)

ne constituent pas un ensemble de clauses stratifiées.

En effet, le premier axiome permet de déduire que si f(egal_dr) = n alors f(egal_pt) = n-1 et le second axiome permet de déduire que si f(egal_pt) = m alors f(egal_dr) = m-1, ce qui peut être ramené au système d'équations

m = n -1

n = m -1

qui est contradictoire.

Exemple :

Avec f(egal_dr) = 1 alors f(egal_pt) = 1, d'où la contradiction avec f(egal_dr) = f(egal_pt) -1 = 0.

La solution que nous avons adoptée pour disposer d'une théorie TIG sous forme de clause de Horn et permettant d'exprimer des clauses ayant plus d'un littéral négatif consiste à transformer tout littéral négatif des axiomes de TIG en un littéral positif [Aida 83].

Pour présenter cette transformation, considérons chaque littéral négatif p_i=non(p(x1,…,xn)) ($1 \le i \le n$) d'un axiome de TIG de la forme

$p_1 \wedge p_2 \wedge \ldots \wedge p_n \rightarrow q$

La transformation consiste à créer un nouveau prédicat "non_p" et à remplacer non(p(x1,…,xn)) par le littéral positif non_p(x1,…,xn) dans l'axiome.

Exemple :

Les axiomes

1) ∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧ p1 ∈ d ∧ p2 ∈ d ∧ ¬egal_pt(p1,p2) → egal_dr(d1,d2)

2) ∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧ p1 ∈ d ∧ p2 ∈ d ∧ ¬ egal_dr(d1,d2) → egal_pt(p1,p2)

sont transformés en

1) ∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧ p1 ∈ d ∧ p2 ∈ d ∧ non_egal_pt(p1,p2) → egal_dr(d1,d2)

2) ∀ p1, p2, d1, d2 point(p1) ∧ point(p2) ∧ droite(d1) ∧ droite(d2) ∧ p1 ∈ d ∧ p2 ∈ d ∧ non_egal_dr(d1,d2) → egal_pt(p1,p2).

Cette solution a plusieurs défauts vis à vis de nos exigences de départ.

Le premier défaut est qu'a priori les propriétés p(x1,…,xn) et non_p(x1,…,xn) ne sont pas exclusives. Ce défaut peut être corrigé si l'on suppose que chaque axiome de TIG est conséquence logique de la géométrie en son entier (que nous désignons par la suite par "La Géométrie", pour plus de commodité). En effet, cette hypothèse signifie que les déductions de p(x1,…,xn) dans TIG sont incluses dans celles de p(x1,…,xn) dans La Géométrie, de même que celles de non_p(x1,…,xn) dans TIG sont incluses dans celles de non(p(x1,…,xn)) dans La Géométrie. Comme les déductions de p(x1,…,xn) et celles de non(p(x1,…,xn)) sont exclusives dans La Géométrie, alors celles de p(x1,…,xn) et de non_p(x1,…,xn) dans TIG le sont aussi.

Le deuxième défaut est que certaines déductions ne sont plus possibles.

Exemple:

Considérons l'ensemble d'axiomes suivant

$P \vee Q$

$\neg P \vee Q$.

La résolution permet de dire que Q est déductible de cet ensemble d'axiome.

Par contre, la SLD-résolution appliquée à la transformation de cet ensemble en l'ensemble de formules suivantes :

P ⇒ Q
non_P ⇒ Q.

ne le permet pas.

Cela vient du fait qu'il n'y a plus de lien entre une propriété et sa négation (ce qui signifie par exemple que ¬(¬P) n'est pas équivalent à P).

Le troisième défaut est que la sémantique de la négation ainsi définie est difficilement explicable.

L'explication que l'on peut donner est la suivante, en supposant les axiomes conséquences logiques de La Géométrie : si on dit qu'une propriété P est vraie, c'est qu'elle l'est dans l'interprétation habituelle; si on dit qu'elle est fausse, c'est soit que la propriété non_P a pu être montrée et donc que P est vraie dans l'interprétation habituelle, soit que ni la propriété ni son contraire n'ont pu être montrées. Dans ce dernier cas, P et non_P sont considérées comme fausses.

Exemple :

Soit la spécification :

droite(l1), droite(l2).

par(l1,l2) et non_par(l1,l2) sont faux.

Cette solution nous amène aussi à modifier la traduction de CDL en LDL en redéfinissant la fonction NOT (voir annexe C) par :

• NOT(T,A) est la fonction transformant la traduction LDL d'un atome de propriété en sa négation, définie par :

NOT(T,A) = OPPOSE(A) @ MOINS(T,A)

• OPPOSE(A) est la fonction transformant d'un atome LDL en son opposé, définie par⊡

OPPOSE(A) = A' tel que

- les arguments de A' sont les mêmes que ceux de A;
- le prédicat de A' est l'"opposé" de A, c'est à dire que A' = concat(non_, A)

3.3. Méthodologie de définition du contenu de TIG.

Définir une théorie TIG représentant les connaissances requises pour un exercice et supposées à l'élève est un des degrés de liberté que le système offre à l'enseignant pour l'expression du contrat didactique qu'il propose à l'élève. On pourrait donc imaginer que le professeur ait à inventer lui-même ses théories. Ce n'est pas chose aisée et il nous semble donc préférable de lui proposer une première théorie, nommée TIG_0, à partir de laquelle il pourra définir les théories qui lui conviennent, par ajout et retrait des axiomes de son choix. Cette première théorie peut aussi être pour lui un guide utile quant aux possibilités d'expression de théories qui lui sont offertes.

Pour définir une théorie TIG (la première comme les autres), nous proposons une méthodologie simple, basée sur la représentation sous la forme de clauses de Horn des axiomes. Elle consiste à considérer successivement chacun des prédicats de LDL pour décrire des paquets d'axiomes de la forme :

p1 ⇒ p2 ⇒ ... ⇒ pn ⇒ q

où q est un littéral formé sur le prédicat considéré.

De plus, pour disposer d'un moyen de comparaison avec les d'autres axiomatisations, nous proposons de baser la théorie décrite sur des théories existantes : il s'agit alors de retrouver pour chaque prédicat de LDL quels sont les axiomes de la théorie existante permettant de le prouver et de transformer ces axiomes en des clauses impliquant ce prédicat.

Pour définir TIG_0 nous avons ainsi examiné les théories axiomatiques d'Euclide, de Hilbert et de Tarski.

3.3.1. L'axiomatisation d'Euclide.

Euclide est le premier à avoir donné une formulation de la géométrie sous la forme axiomatique, à l'aide de cinq postulats :

Premier postulat.
```
Pour tout point P et tout point Q non egal à P il existe une unique
droite L telle que P et Q sont sur L.
```

Deuxième postulat.
```
Pour tout segment AB et tout segment CD il existe un point unique E
tel que B est entre A et E et le segment CD est égal au segment BE.
```

Troisième postulat.
```
Pour tout point O et tout point A non egal à O il existe un cercle de
centre O et de rayon OA.
```

Quatrième postulat.
```
Tous les angles droits sont égaux entre eux.
```

Cinquième postulat.
```
Pour toute droite L et pour tout point P qui n'est pas sur L il existe
une unique droite M telle que P est sur M et M est parallèle à L.
```

Malheureusement, ces axiomes sont assez mal adaptés pour être utilisés par système formel, du fait de l'utilisation par Euclide d'implicites qui devraient être représentés par des axiomes dans une axiomatisation formelle.

Exemple :

Comme l'a montré Hilbert, la preuve du critère de congruence des triangles (deux cotés et un angle congruents) est basée sur une supposition non explicitée (le principe de superposition) qui devrait être représenté par un axiome [Greenberg 74].

De plus, la théorie des ensembles que ne connaissait pas Euclide rend le troisième postulat inutile. En effet, elle permet d'affirmer que l'ensemble des points P tels que |OP| = |OA| existe. Le postulat d'Euclide a pour but essentiellement de préciser qu'il est possible de *dessiner* le cercle de centre O passant par A.

3.3.2. L'axiomatisation de Hilbert.

Hilbert a repris les axiomes d'Euclide en précisant les termes indéfinis et en ajoutant des axiomes implicitement utilisés par Euclide (par exemple, le critère de congruence des triangles).

Il a classé les axiomes en cinq groupes : axiomes d'appartenance, axiomes d'ordre, axiomes de congruence, axiome des parallèles et axiomes de continuité qui forment un total de 20 axiomes.

Ce que Hilbert apporte par rapport à Euclide, c'est d'une part une axiomatisation plus précise, notamment grâce à l'utilisation de la théorie des ensembles et d'autre part un travail sur la compatibilité et l'indépendance des axiomes qu'il propose.

Comme le fait remarquer Rossier [Hilbert 71], les preuves que donne Hilbert sur la compatibilité et l'indépendance des axiomes doivent être remises en cause. La preuve de la compatibilité qu'il donne est basée sur la solidarité de la géométrie avec l'arithmétique. Hilbert pensait pouvoir montrer la compatibilité de l'arithmétique, espoir qui s'est évanoui à la suite des travaux des logiciens postérieurs à 1930. La preuve de l'indépendance que donne Hilbert est liée à un ordre sur les axiomes (indépendance dite "successive") : tout axiome est indépendant des précédents. Par contre, les axiomes pris en bloc ne sont pas indépendants.

Exemple :

L'axiome d'unicité du report du segment (III-1) est une conséquence des autres axiomes de congruence.

3.3.3. L'axiomatisation de Tarski.

L'objectif de Tarski est de donner une axiomatisation sans l'aide de la théorie des ensembles. L'axiomatisation qu'il donne est formulée à l'aide de la logique élémentaire, c'est à dire à partir du calcul des prédicats du premier ordre. Il propose ainsi 13 axiomes n'utilisant que les prédicats ÉGAL, ENTRE et CONGRUENTS. Seul le 13ème axiome (axiome de continuité) n'est pas un axiome du premier ordre : cet axiome représente en fait un ensemble infini d'axiomes.

Cette axiomatisation basée sur la logique a pour intérêt, en plus de son indépendance vis-à-vis de la théorie des ensembles, d'être utilisée telle que dans des systèmes de déduction [Balbiani et al. 92][Quaife 89].

3.3.4. Définition d'une théorie initiale TIG_0 .

La définition que nous donnons d'une première théorie TIG_0 est inspirée des trois axiomatisations précédentes, et plus particulièrement sur celle de Hilbert. Nous nous inspirons aussi de celle donnée dans [Lelong-Ferrand 85].

Exemple :

Pour illustrer la méthode de formulation de TIG que nous avons utilisée, nous présentons ici la partie de cette théorie définie en considérant le prédicat egal_dr (égalité entre droites) et son opposé non_egal_dr.

Chacun des axiomes est numéroté en fonction du prédicat impliqué. Au prédicat egal_dr étant associé le numéro 2, les axiomes impliquant ce prédicat et son opposé sont numérotés successivement 2.1, 2.2, …

a) Deux points distincts définissent une seule droite.

```
2.1 appdr(p1,l1) appdr(p1,l2) appdr(p2,l1) appdr(p2,l2)
non_egal_pt(p1,p2)
 □ egal_dr(l1,l2)
```

b) Par un point, il ne passe qu'une droite parallèle à une autre.

```
2.2 appdr(p,l1) appdr(p,l2) par(l1,l) par(l2,l) □ egal_dr(l1,l2)
```

c) Par un point, il ne passe qu'une droite perpendiculaire à une autre.

```
2.3 appdr(p,l1) appdr(p,l2) perp(l1,l) perp(l2,l)  egal_dr(l1,l2)
```

d) Deux demi-droites égales (resp. segments égaux) ont même droite support.

```
Assuré par 1.1, 3.1 et 7.5 (resp 1.1 et 4.1)
```

e) Deux droites perpendiculaires sont différentes.

```
2.4 perp(l,l')  non_egal_dr(l,l')
```

e) Deux droites parallèles sont différentes.

```
2.5 par(l,l')  non_egal_dr(l,l')
```

Pour caractériser TIG_0, il serait intéressant de pouvoir en donner une comparaison par rapport à l'axiomatique de Tarski, basée elle aussi sur la logique du premier ordre. Cette comparaison est cependant rendue difficile du fait des langages différents employés.

La question de l'indépendance des axiomes de TIG par exemple n'est pas aussi importante que dans les systèmes automatiques de démonstration géométriques ou du point de vue des logiciens. En effet, dans ces systèmes, le fait de disposer d'un ensemble d'axiomes indépendants permet une représentation minimale de la théorie qui peut être importante du point de vue des performances du système.

Du point de vue didactique, l'ambiguïté introduite par des axiomes non indépendants est plus une qualité qu'un défaut. En effet, si un ensemble d'axiomes n'est pas indépendant, c'est que des propriétés sont exprimées par d'au moins deux façons, d'une part par un axiome, d'autre part par un ensemble d'autre axiomes impliquant ce premier axiome. Cette ambiguïté permet donc d'une paer des explications plus riches en présentant plusieurs points de vue à l'apprenant, d'autre part elle rend mieux compte du fonctionnement du raisonnement humain, qui se sert à son gré d'un ensemble d'axiomes ou d'un autre suivant la situation.

Enfin notons que du point de vue didactique, il serait sans doute plus judicieux de parler de théorèmes et de définitions plutôt que d'axiomes, principalement du fait justement de la non-indépendance desdits axiomes. Concrètement, il semble aussi qu'il faille distinguer parmi les axiomes les axiomes "de base", que n'importe quel enseignant désigne comme toujours acquis (comme par exemple la symétrie de l'égalité) de ceux "modulables", acquis ou non suivant la progression de l'apprenant.

4. Expression de la partie prédéfinie du contrat didactique.

L'objectif de ce chapitre est de donner une sémantique précise à la partie prédéfinie du contrat didactique que nous avons définie pour TALC. La sémantique de la partie disponible est définie par la sémantique des langages que le professeur utilise et qui constituent les degrés de liberté que lui propose le système. Elle a été décrite aux chapitres précédents. Il s'agit ici d'exposer les exigences du contrat didactique sur les rapports que doivent entretenir les différents éléments du contrat - la spécification, les outils de construction disponibles et la théorie TIG de référence définis par le professeur d'une part et d'autre part la solution proposée par l'élève - pour que le contrat soit réalisé.

Ces exigences ont pour but d'assurer la correction des paramètres fournis par l'enseignant et l'apprenant, la plus importante étant bien sur, la correction de la construction par rapport à la spécification. Nous abordons successivement les exigences suivantes :

- une exigence par rapport à la spécification donnée par l'enseignant : il faut d'une part que la construction satisfasse toutes les hypothèses de la spécification et d'autre part qu'elle ne satisfasse que ces hypothèses.
- une exigence par rapport à la théorie TIG : il faut que la spécification (et la construction) soit cohérente vis-à-vis de la théorie.
- une exigence par rapport aux outils de construction disponibles : il faut qu'une construction répondant à la spécification donnée par l'enseignant soit possible avec les outils qu'il propose.
- une exigence liée à la construction : il faut que la spécification puisse être réalisée par une construction à elle seule.

Pour conclure, nous proposons une synthèse de l'ensemble de ces exigences.

Remarque : dans ce chapître, nous désignons par S la traduction en LDL (définie en 2.2.4) de la spécification donnée en CDL par le professeur et par F la traduction en LDL (définie en 2.3.2) de la construction donnée en SCL par l'élève pour répondre au contrat.

4.1. Satisfaction par la construction de toutes les hypothèses de la spécification et seulement ces hypothèses.

Nous exposons cette exigence en deux temps.

Dans un premier temps, nous demandons que la construction satisfasse toutes les hypothèses de la spécification. Nous donnons une formulation logique de cette exigence. Cela nous amène à définir précisément l'exigence du contrat d'une part sur les objets que l'élève doit construire explicitement et sur ceux dont la construction peut être déduite de la construction d'autres objets et d'autre part sur la correspondance entre les noms de la construction de l'élève et les noms donnés par le professeur.

Dans un second temps, nous demandons que la construction satisfasse uniquement les hypothèses de la spécification et donnons une formulation logique de cette exigence. Le fait que TIG soit restreinte à formules de Bernays-Schönfinkel nous amène alors à définir une multifonction que nous nommons "fonction d'extension" pour tenir compte des cas où la construction comporte des objets non décrits dans la spécification.

4.1.1. Satisfaction par la construction de toutes les hypothèses de la spécification.

Nous présentons en premier lieu une formulation logique de cette exigence. Nous affinons ensuite cette formulation en précisant les objets à construire par l'élève. Pour cela, nous répondons aux questions suivantes :

- Quels sont les objets que l'élève doit construire explicitement et quels sont ceux dont la construction peut être déduite de la construction d'autres objets?
- Quelle doit être la correspondance entre les noms de la construction de l'élève et les noms donnés par le professeur?

4.1.1.1. Formulation logique

Cette exigence signifie intuitivement que l'enseignant désire que la construction contienne les objets qu'il a mentionnés et que ces objets entretiennent les relations géométriques définies dans la spécification.

Prise au pied de la lettre, cette exigence signifierait concrètement qu'il attend que la traduction de la construction F contienne les littéraux de définition et les littéraux de propriétés de la traduction de la spécification S, ce qui s'exprime par la formule

(a) S □ F

Exemple:

Soit la spécification

```
l1 = (a b), l2 = (b c), l3 = (c a), l1 !- l3.
```

définissant un triangle rectangle en a. Elle est traduite par la formule LDL S1 suivante :

```
point(a), point(b), point(c), droite(l1), droite(l2), droite(l3),
perp(l1,l3).
```

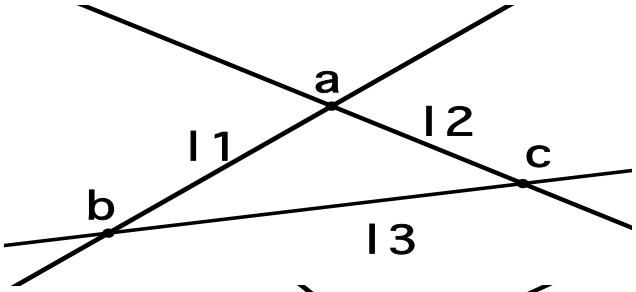


Figure II.8

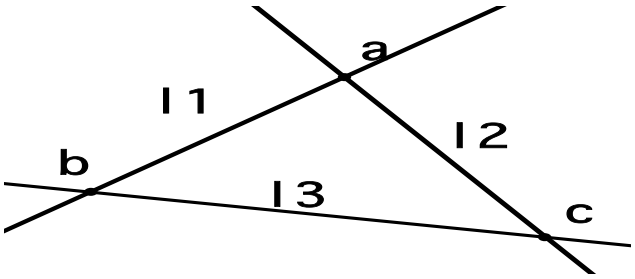


Figure II.9

Les constructions des figure II.8 et II.9 sont traduites par les énoncés Cabri-géomètre E1 et E2 suivants :

E1	E2
a : point de base	a : point de base
b : point de base	b : point de base
c : point de base	l1 : droite passant par a et b
l1 : droite passant par a et b	l2 : droite passant par a et
l2 : droite passant par a et c	perpendiculaire à l1
l3 : droite passant par b et c	c : point de la droite l2
	l3 : droite passant par b et c

La construction de la figure II.8 est traduite par la formule F1 suivante
`point(a), point(b), point(c), droite(l1), droite(l2), droite(l3), appdr(a,l1), appdr(b,l1), appdr(b,l2), appdr(c,l2), appdr(c,l3).`
et la figure II.10 par la formule F2 suivante
`point(a), point(b), point(c), droite(l1), droite(l2), droite(l3), appdr(a,l1), appdr(b,l1), appdr(b,l2), appdr(c, l2), appdr(c,l3), perp(l1,l3).`
Seule F2 répond à l'exigence que nous venons de décrire. En effet, l'élève a construit `(a b)` perpendiculaire à `(c a)`, ce qui est traduit par le littéral `perp(al1,al3)` dans F2 alors que ce n'est pas le cas pour la construction de la figure II.8. La traduction F1 par contre ne contient pas le littéral attendu.

Cette première caractérisation d'une construction satisfaisant toutes les hypothèses d'une spécification est trop grossière. En effet, il s'avère que les exigences de l'enseignant ne sont pas aussi contraignantes : il accepte en général que des objets ou des propriétés ne soient pas construits en reprenant mot pour mot la spécification.
Exemple :
La construction de la figure II.10, dont la traduction est
`point(a), point(b), point(c), segment(var1,a,b,var2) droite(var2) segment(var3,b,c,var4) droite(var4) segment(var5,c,a,var6) droite(var6).`
n'est pas conforme à l'exigence d'appartenance.

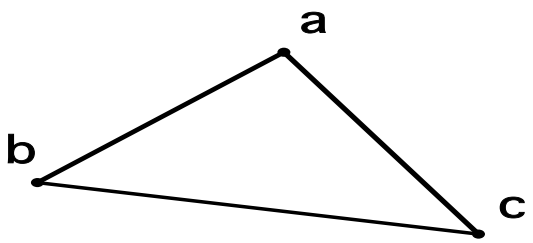


Figure II.10

En effet, la traduction ne contient pas les littéraux `appdr(aa,al1)`, `appdr(ab,al1)`, `appdr(ab,al2)`, `appdr(ac, al2)` et `appdr(ac,al3)`, car les points `a`, `b` et `c` n'ont pas été construits appartenant aux droites `l1`, `l2` et `l3` mais comme extrémités des segments. Pourtant, dans le cas où la théorie TIG contient les axiomes suivants :
• Toute extrémité d'un segment lui appartient.
• Tout point appartenant à un segment appartient à sa droite support.
cela signifie que l'élève sait que si un point appartient à un segment il appartient à la droite qui le supporte. Dans ce cas, il est clair que l'enseignant accepte la construction fournie par l'élève.

On peut noter que dans certains cas l'enseignant peut vouloir une construction qui suive pas à pas l'énoncé donné, fournissant un processus de construction trivial. Dans cette optique, il ne cherche pas à ce que l'élève montre sa bonne compréhension des hypothèses, ce qui est l'objectif du système que nous définissons. Son objectif est simplement que l'élève et le système puissent disposer par la suite d'une figure sur laquelle appuyer la résolution d'un problème. Cette démarche est par exemple celle de DEFI (voir 3.2.1).

La définition logique que nous donnons ici pour définir notre exigence est la suivante :

(b) TIG / F □ S

L'exigence est moins forte que celle de (a), permettant d'accepter une construction où chaque propriété de la spécification peut être déduite par rapport à TIG, sans nécessairement qu'elle soit contenue dans F.

4.1.1.2. Objets à construire explicitement

Avec la formulation logique que nous venons de donner, nous seulement la construction explicite de chaque propriété de la spécification n'est pas exigée, puisqu'une propriété de la spécification peut être déduite de la construction et de la théorie, mais aussi la construction explicite de chaque objet, car son exsitence peut être démontrée à l'aide de la théorie.

Exemple :

Soit la spécification

```
p e [A B].
```

La construction d'une droite passant par les points A et B, et d'un point p milieu de A et B est acceptée si la théorie TIG comporte l'axiome suivant :

```
□ p1, p2 point(p1) □ point(p2) □ non_egalpt(p1,p2)
□ □s segment(s,p1,p2,l).
```

Cet axiome signifie que pour tout couple de points différents il existe un segment dont ils sont les extrémités et des axiomes permettant de déduire que le milieu d'un segment lui appartient.

Cet axiome permet à l'apprenant de ne pas construire explicitement le segment [A□B].

De façon générale, on peut de la même façon supposer présents dans TIG d'autres axiomes permettant à l'élève de ne pas construire effectivement certains objets. On peut alors se demander si cette possibilité donnée à l'élève ne conduit pas à réduire la tâche qui lui incombe à la portion congrue, l'ensemble de la figure pouvant se déduire de la construction de quelques objets. Au pire la construction entière pourrait être déduite ex nihilo, en supposant que TIG rend compte de La Géométrie, ce qui évidemment n'a plus aucun intérêt didactique.

Une autre préoccupation didactique invalide cette interprétation concernant les objets à construire. En vérifiant la correction d'une figure, un des objectifs de TALC est de préparer l'activité géométrique qui va suivre, en assurant que l'élève a bien compris l'énoncé de la figure sur laquelle est basée la situation. L'activité suivante peut-être par exemple la construction d'une preuve d'égalité d'objets. Dans ce cas, il est clair que le professeur requiert la construction explicite de chacun de ces objets. Par exemple, si la spécification décrit les trois points d'intersections des hauteurs d'un triangle (voir figure II.11) et que l'activité suivante consiste à donner la preuve que ces trois points sont égaux, il est clair que le professeur attend la construction des trois points d'intersection et non d'un seul.

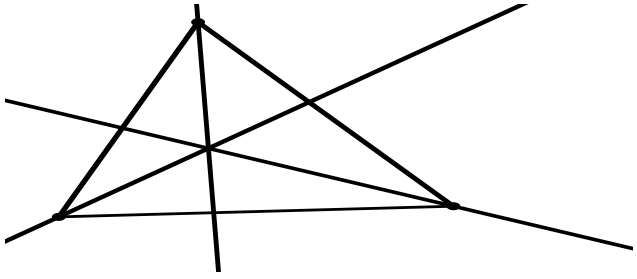


Figure II.11

Il nous semble que l'objectif d'assurer que l'élève a bien compris l'énoncé de la figure amène assez naturellement à demander que l'élève construise explicitement au moins tous les objets cités dans la spécification pour que le contrat didactique soit respecté. Du fait de la restriction de la forme des axiomes à des formules de Bernays-Schönfinkel que nous avons adoptée (voir 3.2.1), la théorie TIG ne permet pas de déduire l'existence d'un objet qui n'a pas été explicitement construit. De ce fait la formulation donnée correspond à notre exigence.

Cependant il nous reste à définir ce que nous entendons par "objets cités dans la spécification". En effet, il est possible que la spécification fasse référence à plusieurs objets dont on peut déduire l'égalité vis-à-vis de la théorie. Il s'agit alors de définir quel est l'ensemble d'objets auquel fait référence la spécification.

Exemple :
Soit l'énoncé suivant, en langue naturelle□"Soit un cercle K dont le centre est l'intersection des droites L1 et L2" à laquelle correspond la figure II.12.

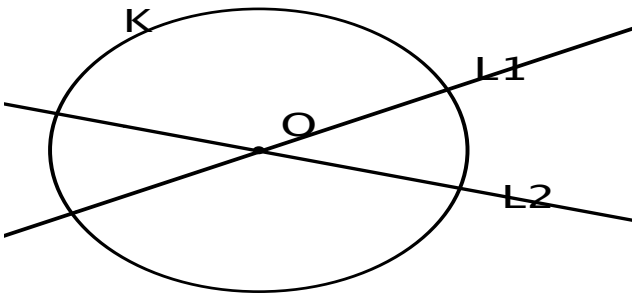


Figure II.12

Il est clair ici que l'enseignant demande que l'élève construise deux droites différentes et non pas une seule. La définition de l'exigence du système par contre ne le demande pas. L'ensemble d'objets cités par la spécification comporte donc ici un cercle, un point et les deux droites L1 et L2.
Mais le problème est moins simple si la spécification énonce de plus que deux points différents appartiennent aux deux droites L1 et L2 et que TIG contient un axiome exprimant le premier postulat d'Euclide (`Pour tout point P et tout point Q non egal à P il existe une unique droite L telle que P et Q sont sur L`). Dans ce cas, bien qu'il soit possible de déduire de la construction et de la théorie que L1 et L2 sont les mêmes, on désire tout de même que l'élève construise L1 et L2. L'ensemble d'objets cités par la spécification comporte ici aussi L1 et L2, bien qu'ils soient égaux vis-à-vis de TIG. Au contraire de l'exemple précédent des trois hauteurs, où il n'est pas évident que la théorie permette de déduire l'égalité des trois intersections, il est clair que la théorie risque en général de comporter ce postulat.

Pour que le système puisse prendre en compte cette exigence, nous adoptons une règle radicale sur les objets d'une spécification en CDL, couramment admise en base de données, l'"hypothèse de nom unique" [Thaysse et al 89]. Cette hypothèse signifie que l'on suppose que toutes les constantes sont différentes. Dans notre cas, elle signifie que l'on suppose que tous les objets identifiés (c'est à dire nommés par l'enseignant, voir 2.2.1) sont différents les uns des autres. Autrement dit, si le professeur a désigné un objet par un nom dans une spécification, c'est qu'il le suppose différent des autres objets identifiés. S'il n'a pas identifié un objet, c'est que cet objet est représenté par une dénotation dans la spécification.

Notons que l'hypothèse de nom unique n'empêche en rien que les objets représentés par une dénotation puissent représenter un autre objet.

Exemple :

`(A B)` représente la droite `L1` dans la spécification

`C e (A B), L1 = (B C).`

si le premier postulat d'Euclide est présent dans TIG.

Comme d'après sa définition (voir 2.2.4), la traduction de la spécification ne peut comporter que des objets identifiés égaux vis-à-vis de TIG, la définition logique de l'exigence qu'une construction réalise toutes les hypothèses d'une spécification est alors la suivante :

$$(c)\ TIG, HYPNU / F \square S$$

où HYPNU représente l'hypothèse de nom unique sur les identificateurs de la spécification.

Notons enfin que pour obtenir une définition des objets à construire par l'élève dans la pratique, il convient de tenir compte de la définition de la traduction d'une construction que nous avons donnée en 2.3.3. En effet, la construction de certains objets est implicitement admise par la construction d'un autre.

Exemple :

Si l'élève construit un segment sans avoir préalablement construit la droite passant par ses deux extrémités, alors cette droite est implicitement construite, bien qu'elle ne soit pas visuellement présente dans l'affichage de la construction ni mentionnée dans l'énoncé produit par Cabri-géomètre.

Concrètement, cela signifie qu'il faut que l'élève ait connaissance des constructions implicites effectuées en utilisant chacun des outils de construction de SCL pour pouvoir répondre au contrat en connaissance de cause.

4.1.1.3. Correspondance entre les noms de la construction de l'élève et les noms donnés par le professeur.

Une exigence de l'enseignant est que certains objets soient présents dans la construction avec les noms qu'il a choisis, et que d'autres objets soient laissés dans l'énoncé à la discrétion de l'apprenant.

Exemple :

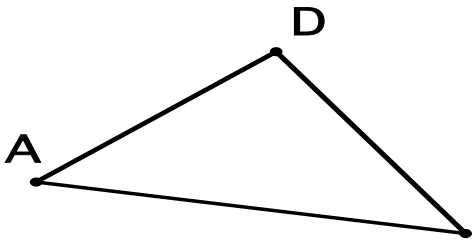
Pour la spécification en langage naturel "Soit un triangle rectangle en A", dont l'énoncé fournit en CDL pourrait être

`s1 = [A b], s2 = [b c], s3 = [c A], s1 !- s3.`

il est clair que le seul nom que l'enseignant veut retrouver dans la construction est `A`. Les autres points et les segments peuvent être désignés par n'importe quel autre nom. C'est la raison de la division des identificateurs de CDL en deux catégories représentant des objets dits "nommés" (dont il veut retrouver le nom dans la construction de l'élève) et les objets dits "non-nommés". Concrètement, les objets nommés sont ceux commençant par une majuscule.

Exemple :

La construction de la figure II.13 ne répond pas à l'attente de l'enseignant alors que la construction de la figure II.14 y répond.



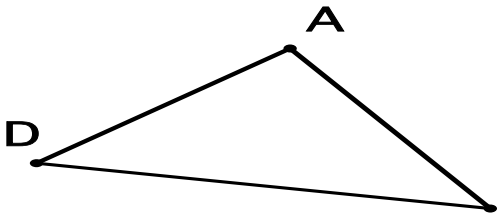


Figure II.13 Figure II.14

Les énoncés des figure II.13 et II.14 sont respectivement les suivantes:

E1	E2
A : point de base	D : point de base
D : point de base	A : point de base
D#1 : droite passant par A et D	D#1 : droite passant par D et A
D#2 : droite passant par D et perpendiculaire à D#1	D#2 : droite passant par A et perpendiculaire à D#1
P#3 : point de la droite D#2	P#3 : point de la droite D#2
D#3 : droite passant par A et P#3	D#3 : droite passant par D et P#3

Laisser certains noms à l'initiative de l'élève peut cependant créer une ambiguïté dans la correspondance établie entre les noms de l'élève et les noms des objets non-nommés du professeur. En effet, deux objets de même type de l'élève peuvent être candidats à correspondre à un objet non-identifié de la spécification. L'attitude d'un enseignant confronté à cette difficulté est de choisir la correspondance la plus favorable, c'est à dire la correspondance qui permet à la construction d'être correcte.

Exemple :

Pour la spécification en langue naturelle "Soit un triangle rectangle", traduite en CDL par

```
s1 = [a b], s2 = [b c], s3 = [c a], s1 !- s3.
```

chacune des deux figures II.13 et II.14 sont acceptées, l'enseignant choisissant la correspondance la plus favorable en prenant a <-> D dans le premier cas et a <-> A dans le second.

Nous avons choisi que l'exigence du système ne soit pas plus forte que celle que l'enseignant sur ce point. Cela signifie que si pour une construction donnée, il existe un renommage des objets non-identifiés de la spécification telle que cette construction satisfasse toutes les hypothèses de la spécification, cette construction réalise le contrat.

Exemple :

Pour la spécification en langue naturelle "Soit un triangle rectangle en A", traduite en CDL par

```
l1 = (A b), l2 = (b c), l3 = (c A), l1 !- l2.
```

avec TIG comportant les axiomes cités dans l'exemple de la figure II.10, la figure n2 répond au contrat, avec l'un des renommages suivants

σ_1 = { b -> D, c -> P#1, l1 -> D#1, l2 -> D#2, l3 -> D#3}
σ_2 = { b -> P#1, c -> D, l1 -> D#1, l2 -> D#2, l3 -> D#3}
NB : P#1, D#1, D#2, D#3 sont les noms donnés par Cabri-géomètre aux objets sans nom.

Pour exprimer logiquement cette exigence sur les noms choisis par l'élève, nous représentons la correspondance entre les noms de la spécification et ceux de la construction par une substitution injective σ définie ainsi :
σ : OSNI -> OF
avec
OF l'ensemble des objets de F.
OS l'ensemble des objets de S.
OSNI l'ensemble des objets non-identifiés de S.

Et nous affinons la formulation du contrat en exigeant qu'il existe une substitution σ telle que :

$$(d)\ TIG, HYPNU \ / F \subseteq \sigma(S)$$

Pour résumer, l'exigence du contrat pour que la construction de l'apprenant satisfasse toutes les hypothèses de la spécification signifie intuitivement :
- que l'élève doit construire tout objet décrit dans la spécification, en tenant compte des objets implicitement construits par les primitives de construction. Les objets identifiés de la spécification sont considérés différents entre eux. Un objet non-identifié de la spécification peut être égal à un autre vis-à-vis de la théorie TIG : dans ce cas la construction d'un des deux objets seulement est requise.
- que l'élève a la latitude de choisir les noms qui lui conviennent pour les objets que le professeur a désignés comme non-nommés.
- que toute propriété de la spécification doit être déductible de la construction au regard de la théorie TIG.

4.1.2. Satisfaction par la construction de seulement les hypothèses de la spécification.

A y regarder de plus près, l'exigence du professeur est plus forte que la seule satisfaction par la construction des hypothèses de la spécification. Il exige de plus que la construction soit la plus générale possible, autrement dit qu'elle ne constitue pas un cas particulier par rapport à la spécification. En effet, si la construction donnée par l'élève est un cas particulier et que le professeur l'accepte, l'élève risque de faire des fausses déductions dans l'activité qui suit la construction de la figure, déductions induites par les propriétés particulières de la figure. Ainsi, il faut non seulement que la construction réalise toutes les hypothèses de la spécification, mais aussi qu'elle n'en réalise pas plus.

Dans ce sous-chapître, nous définissons tout d'abord une première relation logique liant la spécification, la construction et la théorie et exprimant le fait que la construction doit réaliser toutes les hypothèses et seulement ces hypothèses. À partir du constat que cette formulation ne permet pas de prendre en compte les situations où l'élève construit des objets non décrits par la spécification, nous donnons ensuite une nouvelle formulation faisant intervenir un ensemble d'axiomes dits d'"extension" qui définissent l'ensemble des objets non décrits par la spécification dont le professeur permet à l'élève la construction. Cette formulation logique se révèle en fait trop lâche par rapport aux exigences du professeur car elle permet à l'élève de ne pas construire explicitement certains objets de la spécification. Nous donnons donc une formulation logique plus précise faisant intervenir ce que nous appelons une "fonction d'extension". Enfin nous pré-

sentons une définition de cette fonction d'extension par une formulation logique puis par une formulation opérationnelle dont nous montrons qu'elle satisfait la formulation logique.

4.1.2.1. Formulation logique.

Une première façon de définir le fait que la construction réalise seulement les hypothèses de la spécification consiste à exprimer que toute propriété de la construction doit être déductible de la spécification. Logiquement, cela s'exprime par :

TIG / F □ S

En tenant compte de l'exigence que la construction réalise toutes les hypothèses de la spécification, formulée par (d) (voir 4.1.1.3), l'exigence est alors qu'il existe □ tel que :

(e) TIG, HYPNU / F □ □(S)

Exemple :

Si la spécification décrit un triangle isocèle et que la construction fournie est un triangle isocèle rectangle ou équilatéral, la construction ne répond pas à cette formulation du contrat.

Pour plus de simplicité, nous notons S_0 = □(S) dans la suite de ce document.

4.1.2.2. Formulation faisant intervenir un ensemble d'axiomes d'extension.

Un problème surgit avec la formulation précédente du fait que les constructions répondant au contrat sont restreintes à celles faisant intervenir uniquement les objets décrits dans la spécification, dits "objets spécifiés".

Définitions :

Nous appelons "objets spécifiés" les objets d'une construction qui correspondent à un objet d'une spécification par la substitution □ et tels que
TIG, HYPNU / F □ □(S).

Nous appelons "objets non-spécifiés" les autres objets, c'est à dire les objets ne pouvant correspondre à aucun objet de la spécification.

Intuitivement, on peut dire que les objets non-spécifiés sont des objets d'une construction qu'une spécification donnée ne décrit pas.

En effet, la formulation (e) permet d'accepter une classe de constructions plus petite que celle admise par les enseignants. Au pire la classe de constructions répondant au contrat est vide, alors qu'il est clair que le professeur décrit une spécification pour laquelle il envisage au moins une construction possible.

Exemples :

Si la spécification décrit un triangle `(A B C)` rectangle en `A` par

```
s1 = (A B),
s2 = (B C),
s3 = (C A),
(A C) !- (A B).
```

il est possible de construire une figure répondant à la spécification, en supposant disponibles tous les outils de SCL (voir construction de la figure II.15).

Par contre, il est impossible pour l'élève de construire une figure répondant à la spécification suivante, décrivant un triangle isocèle en A :

```
s1 = [A B],
s2 = [B C],
s3 = [C A],
```

`|A B| = |A C|.`

sans construire d'objets non-spécifiés comme le cercle de la construction de la figure II.16.

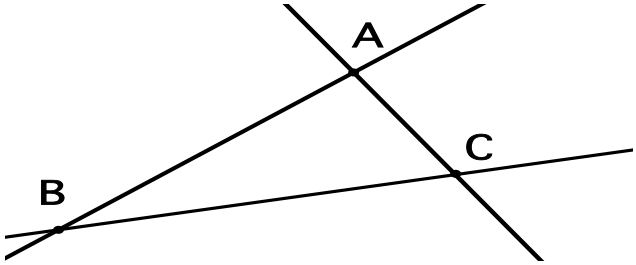


Figure II.15

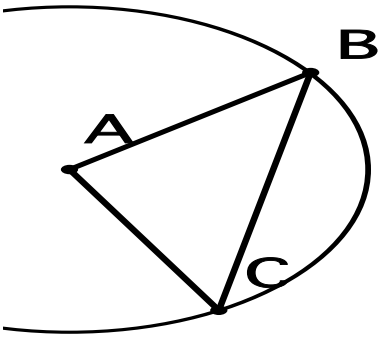


Figure II.16

```
B : point quelconque
A : point quelconque
D#1 : droite passant par B et A
D#2 : droite passant par A et
perpendiculaire à D#1
C : point de la droite D#2
D#3 : droite passant par B et C
```

```
A : point quelconque
B : point quelconque
segment [A B]
C#1 : cercle de centre A passant
par B
C : point du cercle C#1
segment [A C]
segment [C B]
```

Plus concrètement on peut dire que l'expression du contrat didactique proposée limite l'élève à l'ensemble des constructions ne comportant pas d'objets non-spécifiés. En effet, la formulation logique précédente est satisfaite si tout littéral de F peut être déduit de □(S) à partir de TIG et HYPNU. Il suffit donc d'un seul littéral de F non déductible de □(S) pour que la construction ne réponde pas au contrat. Comme TIG ne comporte pas d'axiomes permettant de déduire l'existence d'un objet (voir 3.2.1), le littéral de typage représentant dans F un objet non décrit par la spécification ne peut être déduit de □(S) à partir de TIG et la construction ne peut répondre au contrat.

Évidemment, cette limitation est trop restrictive car il est fréquent que pour réaliser une construction, l'élève s'appuie sur la construction d'objets intermédiaires. L'exemple de la spécification du triangle isocèle montre même qu'il est quelquefois impossible, vu les primitives de construction disponibles, de réaliser une construction répondant à la spécification sans s'appuyer sur des objets non-spécifiés.

La première idée qui vient pour apporter une solution à ce problème est simplement d'étendre la spécification avec les objets non-spécifiés de la construction de l'élève, en ajoutant à cette spécification lesdits objets et les propriétés qui y sont attachées tels qu'on les trouve dans la construction. Ainsi est on assuré que ces nouveaux objets ne sont pas un obstacle à la réalisation du contrat. Malheureusement, cette méthode très simple ne s'accorde pas avec l'exigence d'une construction réalisant seulement les hypothèses de la spécification. En effet, il est possible que les propriétés des objets non-spé-

cifiés induisent des contraintes non spécifiées sur les autres objets de la construction, rendant ainsi la figure particulière.

Exemple :

Soit la spécification suivante :

```
droite(L1), droite(L2)
```

La construction de la figure I.17

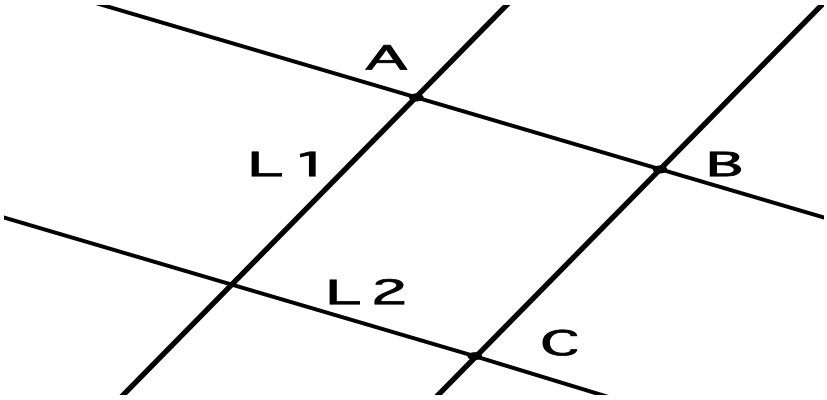


Figure I.17

dont l'énoncé Cabri-géomètre est le suivant :

```
A : point quelconque
B : point quelconque
D#1 : droite passant par A et B
D#2 : droite passant par B et perpendiculaire à D#1
C : point de la droite D#2
L2 : droite passant par C et perpendiculaire à D#2
L1 : droite passant par A et perpendiculaire à D#1
```

ajoute la contrainte que L1 et L2 sont perpendiculaires, non décrite par la spécification, rendant ainsi la construction particulière (en supposant que TIG contient les axiomes définissant les relations usuelles entre parallélisme et perpendicularité).

Il faut donc trouver une façon d'exprimer le contrat didactique qui permette à l'élève d'ajouter des objets dont les propriétés ne font pas de la construction un cas particulier par rapport à la spécification.

Pour définir les contraintes sur la construction de tels objets, supposons que nous disposions d'une construction C répondant au contrat. Il est clair que toute construction d'un objet supplémentaire O dans C n'ajoute pas de propriétés rendant la construction particulière à condition que l'existence de O soit garantie par les propriétés de C.

Exemple :

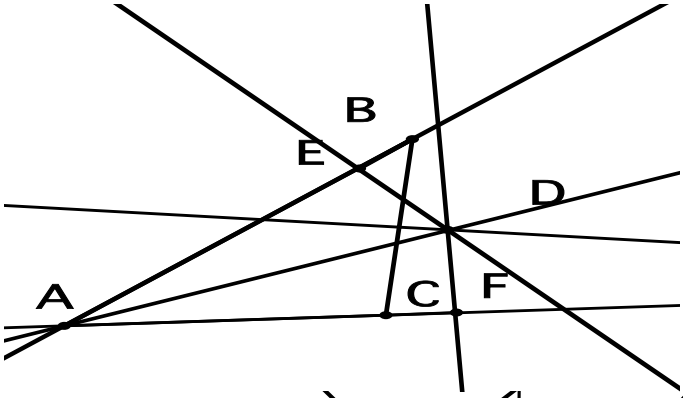
Si la construction C comporte deux droites perpendiculaires, il est clair que la construction de leurs intersections ne rend pas la construction particulière (si TIG comprend un axiome exprimant que deux droites perpendiculaires sont sécantes). Par contre, si C comporte deux droites sans relations entre elles, l'ajout de l'intersection P rend la construction particulière, du fait que l'existence de P n'est pas déductible de C.

Nous proposons d'adopter l'extension forte (cf 2.3.1.2) sur les objets non-spécifiés par rapport aux objets spécifiés. Rappelons que l'exigence de l'extension forte signifie qu'un objet ne peut être construit que si son existence est assurée par la construction. Ici cela signifie que l'existence de tout objet non-spécifié doit être assurée par l'ensemble des objets spécifiés et leurs propriétés.

Remarquons que bien souvent, les figures construites sur papier (ou avec une l'interface graphique d'un ordinateur) ne respectent pas cette exigence. C'est pourtant un point essentiel car les exemples sont nombreux de démonstrations fausses difficiles à mettre en défaut qui s'appuient sur une figure représentant un cas particulier [Greenberg 4] : les propriétés représentées sur la figure donnent l'impression d'être toujours valides, induisant ainsi en erreur.

Exemple :

On peut démontrer que tout triangle est isocèle si l'on raisonne sur la figure II.18. Le raisonnement est fondé sur le fait que |A F| =|A C| + |A C|. La figure II.19 montre l'erreur commise par cette supposition.



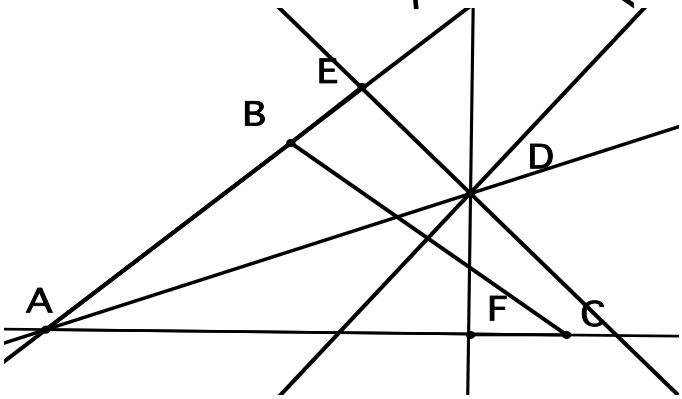


Figure II.18 Figure II.19

L'enseignant est d'ailleurs lui aussi confronté à cette difficulté pédagogique : l'élève a beaucoup de mal à admettre que bien qu'on exhibe sur la figure un exemple où l'objet est effectivement construit, il n'est pas constructible dans le cas général et donc qu'on ne peut s'appuyer sur son existence dans une démonstration. Du point de vue de l'élève, cette exigence d'extension forte sur les objets non-spécifiés de la construction revient en pratique à restreindre a posteriori les conditions d'usage des outils de construction de SCL.

Pour exprimer cette exigence d'extension forte sur les objets non-spécifiés de la construction, nous introduisons un ensemble d'axiomes, nommé TEXT (pour Théorie d'EXTension). Chacun des axiomes de TEXT décrit sous quelles conditions l'élève est autorisé à construire un objet non-spécifié.

Exemple :

L'axiome suivant spécifie que quels que soient deux points différents, la construction de la droite passant par ses deux points est possible :

```
∀p1, p2, point(p1) ∧ point(p2) ∧ p1≠p2 ⇒
∃d droite(d) ∧ appdr(p1,d) ∧ appdr(p2,d)
```

La définition de cet ensemble d'axiome est sous la responsabilité de l'enseignant. En définissant TEXT, il précise les conditions de validité des outils qu'il autorise à l'élève. La définition de TEXT fait donc partie du second degré de liberté que TALC propose à l'enseignant (voir 1.2).

NB : un exemple d'ensemble d'axiomes d'extensions TEXT est donné en annexe B.

Une première formulation permettant d'accepter les d'objets non-spécifiés construits avec la contrainte d'extension forte par rapport à TEXT pourrait être la suivante

(f) TIG, HYPNU , TEXT ⁄ F ⊢ S_0

Elle exprime le fait que la formule F ⊢ S_0 doit être déduite par rapport non seulement aux théories TIG et HYPNU mais aussi à la théorie TEXT. Cela permet d'accepter comme répondant au contrat les objets non-spécifiés correspondant aux conditions décrites par TEXT.

4.1.2.3. Formulation faisant intervenir une fonction d'extension.

Il reste que l'expression (f) présente l'inconvénient majeur de permettre à l'élève de ne pas construire tous les objets spécifiés.

Exemple :
Si la spécification demande à l'élève une droite passant par deux points, alors la construction de deux points seulement répond au contrat.

Cette expression est donc en contradiction avec les choix que nous avons exposés en 4.1.1.2, exigeant que tout objet spécifié soit construit. Intuitivement, le fait que la formule F ⊢ S_0 doive être prouvée par rapport à TEXT permet d'accepter deux sortes de constructions :
- d'une part les constructions comportant des objets non-spécifiés. Dans ce cas, on peut dire que cela signifie que la formule S est "étendue" par l'ajout d'objets à l'aide de TEXT.
- d'autre part les constructions ne comportant pas tous les objets spécifiés, les objets manquants étant déductibles à l'aide de TEXT. Dans ce cas, on peut dire que cela signifie que la formule F est "étendue" par l'ajout des objets manquants à l'aide de TEXT.

C'est cette deuxième sorte de construction que l'on ne doit pas accepter pour que le contrat reste cohérent avec les choix que nous avons faits.

Pour établir une expression du contrat correcte, nous proposons d'introduire une multifonction que nous nommons FE (pour Fonction d'Extension). Cette multifonction a pour domaine le quadruplet <S,F,TIG,TEXT> et donne pour résultat un ensemble de formules S* telles que S ⊢ S*. Intuitivement, elle définit un ensemble d'extensions d'une formule S par ajout de littéraux (de typage ou de propriété) représentant les objets non-spécifiés de F, en fonction de TIG et de TEXT. L'idée est de déclarer le contrat satisfait si la construction satisfait une des extensions.

Nous donnons d'abord une définition logique de la multifonction d'extension. Nous donnons ensuite une définition opérationnelle que nous montrons équivalente à la définition logique.

4.1.2.3.1. Définition de la fonction d'extension.

Définition :
La fonction d'extension FE est telle que S*∈FE(S_0,F,TIG,TEXT), avec S* pourvue des propriétés suivantes :

(1) S* close et V(F) =V(S*) où V(E) est le vocabulaire de la formule E.

Cette contrainte signifie que les objets de S* sont l'union des objets de S et des objets non-spécifiés de F.

Intuitivement, cela permet de ne pas considérer dans FE les extensions qui comportent des objets non présents dans F.

(2) TIG, HYPNU /F □ S* □ S_0

Cette contrainte signifie que les propriétés déductibles de S* doivent aussi être déductibles de F et que chaque propriété déductible de S doit être déductible de S*.

Intuitivement, cela signifie :

- d'une part que S* doit comporter au moins les objets de S_0 (puisque TIG ne permet pas de déduire l'existence d'objets) et permettre de déduire au moins autant de propriétés,
- d'autre part que S* ne doit pas comporter des objets non présents dans F ni permettre de déduire plus de propriétés que F.

On note que cela signifie aussi que les S* ne sont définis que dans le cas où la construction réalise toutes les hypothèses de la spécification.

(3) TEXT, TIG⁄S* □ S_0

Cette contrainte signifie que toute propriété déductible de la spécification d'origine S_0 doit être déductible de la spécification étendue S* compte-tenu de la théorie TEXT (et bien sur de TIG).

Intuitivement, cette dernière contrainte définit que l'existence des objets et propriétés supplémentaires de S* par rapport à S est déduite de TEXT.

A partir de cette définition de la fonction d'extension FE, nous formulons le contrat didactique de la façon suivante :

il existe □tel que TIG, HYPNU ⁄ F □ S*
avec □S* □ FE(□(S),F,TIG,TEXT)

Notons que la formulation précédente ne permettait pas de garantir la décidabilité alors que celle-ci le permet. Cela est dû au fait que les axiomes de TEXT ne sont pas sous la forme de Bernays-Schönfinkel : la précédente formulation demandait des preuves vis-à-vis de TEXT alors que celle-ci non.

4.1.2.3.2. Définition opérationnelle de la fonction d'extension.

Nous donnons ici une méthode opérationnelle de calcul de la fonction FE.

Supposons que la construction réalise toutes les hypothèses de la spécification, soit

il existe □tel que TIG, HYPNU ⁄ F □ □(S)

□ étant donné vérifiant cette expression, la définition opérationnelle de la fonction d'extension s'appuie sur la construction d'une suite S_0, ..., S_n de formules LDL telle que⊡

- S_0 = □(S)

- □ i ≥ 0, $S_{i+1} = S_i$ □ $Propmin_j$

où $Propmin_j$ est un ensemble de propriétés dont l'ajout à S_i est assuré par l'axiome j de la théorie TEXT.

- S_n = S* si V(S*) =V(F)

Chaque axiome de TEXT est de la forme

□$y_1,y_2,\ldots,y_n$, Cond($y_1,y_2,\ldots,y_n$) □
□x, $z_1,z_2,\ldots,z_n$ Propmin(x, $z_1,z_2,\ldots,z_n$, $y_1,y_2,\ldots,y_n$)

A chacun de ces axiomes est associé le schéma de calcul suivant :

```
si
      Si ⊨TIG Cond(y1,y2,…,yn)
 et
      F ⊨TIG Propmin(x, z1,z2,…,zn, y1,y2,…,yn)
 avec
      y1,y2,…,yn ∈ V(Si) et x, z1,z2,…,zn ∈ V(F)\V(Si)
alors
      Si+1 = Si ∪ Propmin(x, z1,z2,…,zn, y1,y2,…,yn)
      et V(Si+1) = V(Si) ∪ {x, z1,z2,…,zn}
```

Ce schéma exprime que si l'on peut déduire que les conditions Cond sont vérifiées dans S_i et que les propriétés Propmin sont vérifiées dans la construction, alors on peut construire S_{i+1} en ajoutant les propriétés Propmin à S_i. Intuitivement, cela signifie

- d'une part qu'il faut que les propriétés que l'on veut ajouter par l'axiome à la spécification soient déjà vérifiées dans la construction.
- d'autre part que les conditions de l'ajout soient vérifiées dans la spécification.

Le calcul d'un élément de la suite S_{i+1} à partir de S_i à l'aide de ce schéma opérationnel des axiomes s'effectue ainsi :

- vérification que $F \vdash_{TIG}$ Propmin(x, $z_1,z_2,\ldots,z_n$, $y_1,y_2,\ldots,y_n$) en prenant x, $z_1,z_2,\ldots,z_n$ dans les objets non-spécifiés de S_i et $y_1,y_2,\ldots,y_n$ dans les objets communs de S_i et F.
- vérification que $S_i \vdash_{TIG}$ Cond($y_1,y_2,\ldots,y_n$).

Calculer l'ensemble FE(Σ(S),F,TIG,TEXT) revient à produire toutes les suites définies précédemment en considérant :

- tous les σ tels que σ ∈ (TIG, HYPNU ⊨F σ ∈ Σ(S))
- toutes les possibilités d'application d'axiomes de TEXT.

Premier exemple d'axiomes exprimés selon ce schéma.

Quelles que soient deux droites différentes et non parallèles, il existe un point intersection de ces droites

```
∀l1,l2, droite(l1) ∧ droite(l2) ∧ non_par(l1, l2) ∧ non_egal_dr(l1,l2)
⇒
∃p∧point(p) ∧ appdr(p,l1) ∧ appdr(p,l2)
```

Le schéma de calcul associé est :

```
si
   Si ⊨TIG droite(l1) ∧ droite(l2) ∧ non_par(l1, l2) ∧
non_egal_dr(l1,l2)
 et
   F ⊨TIG point(p) ∧ appdr(p,l1) ∧ appdr(p,l2)
 avec
      l1,l2 ∈ V(Si) et p ∈ V(F)\V(Si)
alors
      Si+1 = Si ∪ point(p) ∧ appdr(p,l1) ∧ appdr(p,l2)
      et V(Si+1) = V(Si) ∪ {p}
```

Intuitivement, ce schéma signifie que si la construction comporte un point p non-spécifié par rapport à l'extension courante (représentée par S_i) tel que

- on peut déduire de la construction que p appartient à la fois à l1 et l2;
- l1 et l2 sont des droites de l'extension courante telles qu'elles ne soient ni parallèles ni égales dans cette extension

alors l'ajout du point p à l'extension courante est possible en le liant à l1 et l2 par une propriété d'appartenance.

Second exemple.

Quels que soient deux points différents, il existe un point milieu de ces points.

```
∀p1,p2, point(p1) ∧ point(p2) ∧ non_egal_pt(p1,p2) ⇒
∃i, l,d point(i) ∧ droite(l) ∧ appdr(i,l) ∧ distance(d)
∧ distancep(d,p1,i) ∧ distancep(d,i,p2)
```

Le schéma de calcul associé est :

```
si
  Si ⊨TIG  point(p1) ∧ point(p2) ∧ non_egal_pt(p1,p2)
et
   F ⊨TIG point(i) ∧ droite(l) ∧ appdr(i,l) ∧ distance(d)
∧ distancep(d,p1,i) ∧ distancep(d,i,p2)
 avec
      p1,p2 ∈ V(Si) et  i, l,d ∈ V(F)\V(Si)
alors
      Si+1 = Si ∧ point(i) ∧ droite(l) ∧ appdr(i,l) ∧ distance(d)
∧ distancep(d,p1,i) ∧ distancep(d,i,p2)
      et V(Si+1) = V(Si) ∪{i,l,d}
```

Ce second exemple permet de montrer pourquoi on ne peut se restreindre au cas où les propriétés à ajouter à l'extension courante sont exactement contenues dans la construction.

Exemple :

Soit la spécification

```
point(A), point(B)
```

Soit la construction dont l'énoncé est le suivant

```
A : point quelconque
I : point quelconque
L : droite passant par A et I
K : cercle de centre I passant par A
B : intersection de la droite L et du cercle K (A est l'autre point)
```

à laquelle correspond la figure II.20.

On ne trouve pas dans la traduction en LDL de l'énoncé les littéraux `distancep(d,A,I)` et `distancep(d,I,B)`. Pourtant ils sont déductibles de la construction si TIG contient un axiome exprimant que tout point sur un cercle est à une distance r de son centre, avec r le rayon du cercle.

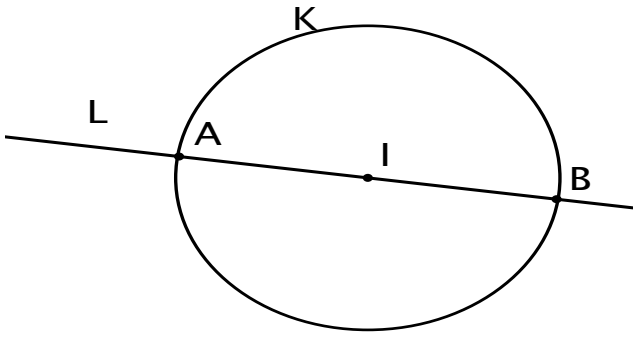


Figure II.20

Notons que chaque axiome de TEXT est censé reproduire une construction à la règle et au compas. On peut imaginer raisonnable d'introduire comme axiome de TEXT ceux qui correspondent aux primitives de SCL (qui sont des constructions à la règle et au compas).

Nous démontrons en annexe que tout S* ainsi construit appartient à FE(□(S),F,TIG,TEXT).

Exemple d'extension.

Soit la spécification suivante, décrivant un triangle isocèle en A :

```
s1 = [A B],
s2 = [B C],
s3 = [C A],
|A B| = |A C|.
```

Soit la construction dont l'énoncé est (voir figure II.21) :

A : point quelconque
B : point quelconque
segment [A B]
C#1 : cercle de centre A passant par B
C : point du cercle C#1
segment [A C]
segment [C B]

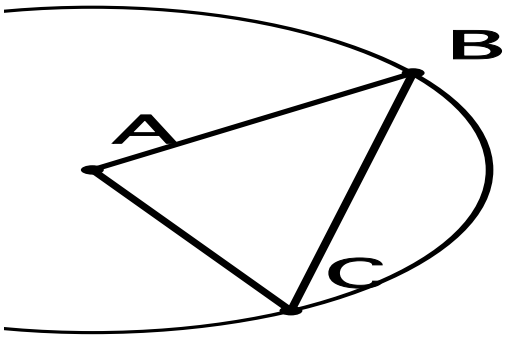


Figure II.21

Les traductions en LDL de la spécification et de l'énoncé sont les suivantes :

Spécification	Énoncé
point(A),point(B), point(C) segment(s1,A, B, l1) segment(s2,B, C, l2) segment(s3,C, A, l3) distance(d) distancep(d,A,B) distancep(d,A,C)	point(A),point(B), point(C) segment(s1,A, B, l1) segment(s2,B, C, l2) segment(s3,C, A, l3) distance(d) cercle(C#1,A,d) appcc(B,C#1) appcc(C,C#1)

Pour que cette construction soit acceptée, il faut :

• que TIG contienne les axiomes

(1) "Tout point d'un cercle est à une distance r de son centre, r étant le rayon du cercle"

```
distancep(r,o,p) □ cercle(c,o,r) appcc(p,r)
```

(2) "Un point appartient à un cercle si il est à une distance r de son centre, r étant le rayon du cercle"

```
appcc(p,r) □ cercle(c,o,r) distancep(r,o,p)
```

• que TEXT contienne l'axiome d'extension

```
p1≠p2 □ cercle(c,p1,r) distance(r) appcc(p2,r)
```

Pour prouver que cette construction satisfait toutes les hypothèses de la spécification, il faut prouver :
`(d) TIG, HYPNU / F ⊢ S`
Comme, par l'axiome (1), on peut prouver
`TIG, HYPNU, F / distancep(d,A,B) distancep(d,A,C).`
alors (d) est prouvée.

Deux extensions sont possibles ajoutant le cercle C#1 :
- l'une fait passer C#1 par B
- l'autre fait passer C#1 par C.
La condition p1≠p2 de l'axiome est assurée par la traduction.

Chacune de ces extensions S* vérifie bien
`TIG, HYPNU / F ⊢ S*`
car l'axiome (2) permet de prouver
`TIG, HYPNU, S / appcc(B,C#1)`
ou (selon l'extension)
`TIG, HYPNU, S / appcc(B,C#1)`

Considérons une construction identique à la précédente mais où le segment `[A B]` est perpendiculaire au segment `[A C]`.
Cette construction ne répond pas au contrat. En effet, elle satisfait toutes les hypothèses de la spécification mais les deux extensions possibles (identiques aux précédentes) sont toutes deux particulières du fait de la perpendicularité des deux segments.

4.2. Cohérence de la spécification et de la construction vis-à-vis de la théorie.

Pour que le contrat ait un sens, la moindre des choses est que la spécification comme la construction soient cohérentes vis-à-vis de la théorie.
Cette cohérence s'exprime logiquement par les deux formules suivantes, où ❒ représente la clause vide :
(1) TIG, S ‘ ❒
(2) TIG, F ‘ ❒

Pour que la formule (2) soit vérifiée, une condition raisonnable est la suivante : tout axiome correspondant à une construction de SCL doit être cohérent avec TIG.
Exemple :
Si l'élève peut construire une perpendiculaire à une droite passant par un point, il faut que TIG ne soit pas contradictoire avec l'axiome définissant que par un point ne passe qu'une droite perpendiculaire à une autre.

La restriction aux clauses de Horn que nous avons adoptée pourrait aussi faire penser que (1) aussi bien que (2) sont toujours satisfaites puisque toute théorie sous forme de clauses de Horn (avec un littéral positif) est toujours cohérente. C'est évidemment le cas d'un point de vue formel.
D'un point de vue pratique, avec la façon dont nous traitons la négation, il faudrait démontrer que l'atome `p(x)` n'implique pas la formule atomique `non_p(x)`.

Exemple :
Supposons que le professeur ait spécifié deux droites à la fois parallèles et perpendiculaires.

Supposons présent dans TIG l'axiome spécifiant que deux droites perpendiculaires ne sont pas parallèles et réciproquement. Il est exprimé sous forme d'une clause de Horn par :

```
non_perp(l1,l2)   non_par(l1,l2)
```

Il est possible de montrer `par(l1,l2)` car c'est un fait. Il est aussi possible de montrer `non_par(l1,l2)` par application de l'axiome avec le fait `perp(l1,l2)`.

Il s'agit d'un problème complexe pour lequel nous n'avons pas pour l'instant de solution pratique et qui est à l'étude (voir la conclusion).

4.3. Existence d'une construction répondant à la spécification.

Il est clair que si la spécification donnée par le professeur ne correspond à aucune construction possible à l'aide des outils disponibles à l'élève, celui-ci est bien en peine pour répondre au contrat.

Montrer qu'il existe ou non une figure constructible à la règle et au compas satisfaisant une spécification donnée est un problème difficile [Carréga 89] mais décidable [Lebesgue 89].

L'exemple typique de spécification non constructible à la règle et au compas est le trisecteur d'un angle de $\pi/3$ qui s'exprime en CDL par

```
O = centre(K), P1 e K, P2 e K,
|O P1| = |O P2|, |O P4|=|P1 P4|,
|P1 M1|=|M1 M2|, |M1 M2|=|M2 P2|.
```

et auquel correspond la figure II.22

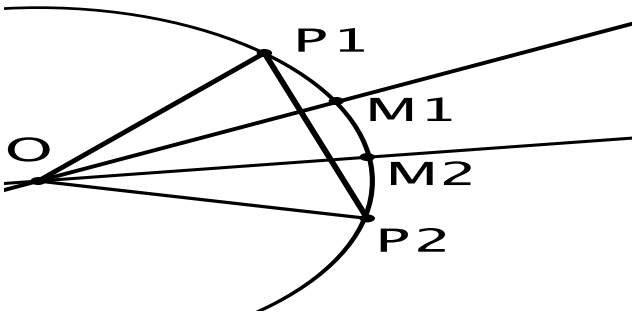


Figure II. 22

Dans le cadre où nous nous plaçons, il paraît simple, plutôt que d'employer des moyens lourds, de demander au professeur d'exhiber une construction correspondant à la spécification qu'il décrit. Si le système peut vérifier que cette construction est correcte, alors non seulement la constructibilté est vérifiée, mais du même coup la cohérence de la spécification l'est aussi. En effet, supposons qu'une formule LDL traduction d'une construction en SCL soit cohérente avec La Géométrie. Cela est réalisé si les outils de construction de SCL sont cohérents avec TIG et que chaque axiome de TIG est cohérent avec la Géométrie. Dans ce cas l'équivalence de la construction et de la spécification entraîne la cohérence de la spécification.

4.4. La spécification doit être satisfiable par une unique construction.

Nous venons de définir le contrat didactique de façon qu'il exprime qu'une construction satisfasse toutes les hypothèses de la spécification et seulement ces hypothèses. Implicitement, nous n'avons fait référence qu'à une seule construction pour répondre au contrat. La question qui se pose pour lever cet implicite est de savoir s'il est possible

que ce soit un ensemble de constructions exclusives qui réponde au contrat, ce que l'on peut formuler par :

TIG, HYPNU , TEXT / (F1 F2 ... Fn) S_0

où {F1, F2, ..., Fn} serait un ensemble de traductions de figures construites par l'élève pour répondre au contrat.

Si l'enseignant propose une spécification disjonctive, il est possible qu'une construction unique ne puisse représenter toutes les hypothèses de la spécification. Dans ce cas, c'est l'ensemble de plusieurs constructions qui répond à la spécification mais aucune construction ne satisfait à elle seule toutes les hypothèses requises.

Notre point de vue est qu'il faut que l'élève puisse répondre au contrat par une seule construction. Nous le justifions par les pratiques des enseignants : dans le type de situation où l'élève doit distinguer plusieurs cas exclusifs dans l'activité qui suit la construction (par exemple une démonstration de théorème), nous considérons qu'il s'agit en fait de plusieurs situations distinctes pour lesquelles il doit fournir à chaque fois une construction différente.

Pour répondre à cette exigence, nous présentons deux restrictions nécessaires : la restriction de CDL à des conjonctions et la restriction de la formulation des axiomes à des clauses de Horn.

4.4.1. Restriction du langage CDL à des conjonctions.

Pour éviter cette situation où l'élève ne peut remplir le contrat par une seule construction, il faut au moins rendre l'expression des disjonctions impossible dans les spécifications. C'est la raison pour laquelle les formules de CDL sont restreintes à des conjonctions de littéraux.

Mais on doit noter que l'absence de l'opérateur de disjonction n'est pas suffisante pour empêcher l'expression de disjonction. En effet la négation d'une conjonction est équivalente à une disjonction. C'est la raison pour laquelle les négations ne peuvent porter que sur des atomes.

Cependant, ces restrictions sur le langage CDL ne sont encore suffisantes pour écarter toutes les spécifications indésirables.

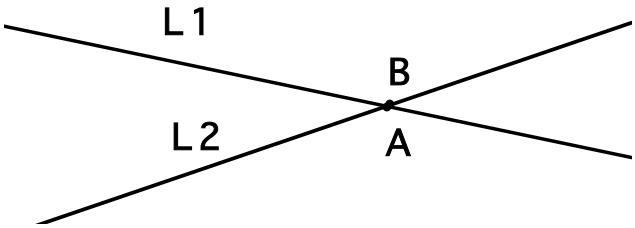
Exemple :

Soit la spécification

A ⇒ L1 ⇒ B ⇒ L1 ⇒A ⇒ L2 ⇒B ⇒ L2.

Elle ne comporte pas de disjonctions. Pourtant les constructions des figures II.23 et II.24 peuvent lui correspondre si l'on considère que TIG comprend l'axiome

a ⇒ l1 ⇒ b ⇒ l1 ⇒ a ⇒ l2 ⇒ b ⇒ l2 ⇒ a = b ⇒ l1 = l2.



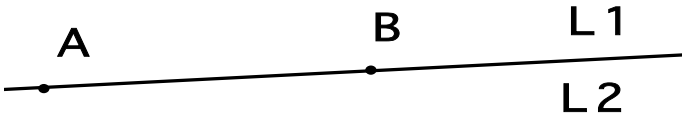


Figure II.23 Figure II.24

Cela est dû au fait que les disjonctions peuvent être "introduites" par les axiomes de TIG. Il faut donc aussi restreindre la forme des axiomes de TIG.

4.4.2. Restriction de la formulation des axiomes de TIG.

Pour caractériser une restriction représentant l' exigence attendue, nous proposons la contrainte suivante portant sur la théorie TIG et se rapportant à la théorie des modèles logiques :
l'ensemble TIG ∪ S doit posséder un unique modèle minimal [Gregoire 90] [McCarthy 80].

Définitions :
On note $|P|_M$ l'extension d'une propriété P dans un modèle M.

Un modèle M d'un ensemble de formule E est appelé "modèle minimal" pour un prédicat P si
∀ a ∈ $|P|_M$, M\{a} n'est pas un modèle de E.
En d'autres termes, si l'on enlève un seul des éléments de l'extension de P, l'ensemble obtenu n'est plus un modèle.

Un modèle M d'un ensemble de formule E est appelé "modèle minimal" pour E si
∀ P prédicat de E, M est un modèle minimal pour P.

Si TIG ∪ S répond à cette exigence, alors
- d'une part il existe au moins une figure géométrique FM réalisant à elle seule toutes les hypothèses de la spécification, celle qui correspond au modèle minimal de TIG ∪ S. Deux sont néanmoins possibles : soit cette figure est constructible avec les outils de SCL disponibles à l'élève, soit elle ne l'est pas (voir 4.3).
- d'autre part, cette figure géométrique FM réalise à elle seule toutes les hypothèses de la spécification et seulement ces hypothèses. En effet, d'une part FM réalise toutes les hypothèses de la spécification car elle correspond à un modèle de TIG ∪ S, d'autre part comme le modèle minimal - s'il est unique - d'un ensemble de formules E est l'intersection de tous les modèles de E, alors FM est la plus générale de toutes les figures représentant un modèle de TIG ∪ S. Cela signifie que toutes les autres figures correspondant à un modèle de TIG ∪ S autre que le modèle minimal possèdent au moins une propriété que ne possède pas FM : cette propriété n'est pas déductible de la spécification du fait de l'existence d'une construction plus générale FM. Notons qu'il est possible

qu'avec les outils qui lui sont proposés l'élève puisse fournir un nombre important de constructions différentes correspondant à cette figure unique FM.

Au contraire, si TIG ⊬ S ne possède pas un unique modèle minimal, alors il est possible qu'il existe deux modèles m1 et m2 de S ⊢ TIG tel que leur intersection ne soit pas un modèle. Chacun de ces deux modèles rend alors compte des deux figures disjointes.

Exemple :
Aux figures II.23 et II.24, on peut attacher respectivement les modèles m1 et m2, dont l'intersection n'est pas un modèle et tels que :

$| egal_pt |_{m1} = \{(A,B)\}$, $| egal_dr |_{m2} = \varnothing$, $| appdr |_{m1} = \{(A,L1), (A,L1)\}$
$| egal_pt |_{m2} = \varnothing$, $| egal_dr |_{m2} = \{(L1,L2)\}$ $| appdr |_{m1} = \{(A,L1), (A,L1)\}$

Une façon de contraindre S ⊢ TIG à posséder un unique modèle minimal consiste à restreindre les axiomes de TIG à des clauses de Horn car on sait que la complétion d'un ensemble de clauses de Horn possède un modèle minimal [Lloyd 87][Kowalski 79].
Des raisons d'aboutir à cette restriction ont déjà été évoquées dont la facilité de rédaction et de lisibilité de TIG (voir 3.2.2), la simplicité de mise en œuvre des démonstrateurs (voir chapitre troisième, 1.1.1) et aussi la qualité des explications possible en cas d'erreur. Néanmoins, on peut aussi admettre que le professeur rédacteur de TIG peut être assez raisonnable pour définir une théorie admettant un modèle minimal de S ⊢ TIG. Nous reprenons ce point en conclusion.

4.5. Synthèse des exigences du contrat didactique.

Nous résumons ici schématiquement l'ensemble des exigences du contrat didactique que nous avons définies.

• Il faut que la construction satisfasse toutes les hypothèses de la spécification.
Pour cela, il faut que chaque hypothèse de la spécification soit déductible de la construction vis à vis de la théorie TIG et de l'hypothèse de nom unique sur les objets que l'enseignant a désigné par un nom (dits "identifiés").
Il faut que tout objet décrit dans la spécification soit construit par l'élève, explicitement ou implicitement.
L'élève peut choisir à son gré les noms des objets (dits "non-nommés") que le professeur a désigné soit par un nom commençant par une minuscule, soit par un terme.

• Il faut que la construction satisfasse seulement les hypothèses de la spécification.
Pour cela, il faut que toute propriété de la construction soit déductible de la spécification vis à vis de la théorie TIG, dans le cas où l'élève n'a pas construit d'objets supplémentaires.
Dans le cas contraire, il faut que toute propriété de la construction soit déductible d'une extension de la spécification vis à vis de la théorie TIG.
Chaque extension de la spécification est construite en ajoutant, à la règle et au compas, via les axiomes de TEXT, chaque objet manquant dans la spécification par rapport à la construction, à condition que son existence soit prouvée vis à vis de la spécification et de TIG.

• La spécification et la construction doivent être cohérentes vis à vis de la théorie.
Cela ne pose pas de problème pour la construction du fait des outils de construction proposés si la théorie est cohérente avec La Géométrie. Pour la spécification, bien que cela soit théoriquement garanti, le professeur doit le vérifier lui-même.

• La spécification doit être satisfiable en une seule construction et non en plusieurs constructions disjointes.

Chapitre troisième.
Mise en œuvre de TALC.

Ce troisième chapitre a pour but de présenter comment la mise en œuvre de TALC correspond à la définition que nous avons donnée au deuxième chapitre.

Dans ce but, nous décrivons tout d'abord les différentes mises en œuvre du même démonstrateur de base que nous avons conçues pour chacune des démonstrations nécessaires à la vérification de la correction et les optimisations qui nous leur avons apportées. Nous présentons ensuite la mise en œuvre de TALC-Enseignant, qui est le système avec lequel l'enseignant définit le contrat qu'il propose à l'élève. La présentation de la mise en œuvre de ce système nécessite principalement de définir l'interface proposée à l'enseignant et la traduction des éléments du contrat en un exercice. Nous présentons la mise en œuvre de TALC-Élève, qui est le système avec lequel l'élève répond au contrat proposé par un enseignant. Pour cela, nous décrivons la mise en œuvre de l'interface proposée à l'élève, puis la mise en œuvre de la communication avec le logiciel Cabri-géomètre et enfin la traduction de la construction de l'élève qui permet d'effectuer le diagnostic. Ensuite, nous présentons la mise en œuvre de la correction, en particulier celle concernant la construction de l'extension. Une fois cette mise en œuvre de la correction décrite, nous pouvons présenter comment sont construits les diagnostics que produit TALC-Élève et les différents types d'explication envisagés. Enfin nous présentons les résultats que nous avons obtenus avec cette mise en œuvre.

1. Mise en œuvre du démonstrateur.

L'objectif de ce chapitre est de décrire le démonstrateur que nous avons mis en œuvre. Ce démonstrateur doit effectuer les démonstrations suivantes :

- démonstration des égalités induites par la traduction d'une spécification en CDL. Cela permet de construire la traduction réduite d'une traduction de base de cette spécification.
- démonstration des égalités induites par la traduction d'une construction en SCL. Cela permet de détecter les constructions introduisant un objet égal à un objet précédemment construit.
- démonstration que la construction satisfait toutes les propriétés de la spécification.
- démonstration que les conditions d'application d'un axiome de TEXT sont bien remplies, dans le but de construire la multifonction d'extension d'une spécification par rapport à une construction.
- démonstration que la construction satisfait seulement les propriétés de la spécification (plus exactement les propriétés d'une des extensions de la spécification vis-à-vis de la construction).

Dans chacune de ces cinq utilisations du démonstrateur, les démonstrations à effectuer sont des implications. De ce fait, grâce au théorème de la déduction, cela revient à effectuer la démonstration de la formule impliquée par rapport à la théorie et à la formule impliquante.

Exemple :

Dans le cas de la démonstration que la construction satisfait toutes les propriétés de la spécification, il s'agit de montrer qu'il existe une substitution telle que :

TIG, HYPNU ⊢ F (S))

ce qui équivaut à montrer que
TIG, HYPNU, F ⊢ (S).

Dans ce chapitre nous présentons en premier lieu le principe général de la mise en œuvre du démonstrateur et ses trois versions, chacune adaptée à un type de démonstrations. Nous proposons ensuite des améliorations de ses versions ayant pour but d'augmenter les performances du démonstrateur. Enfin nous proposons une méthode basée sur l'emploi de contre-modèle à partir d'un démonstrateur travaillant sur la géométrie dans son ensemble, permettant d'améliorer encore les performances.

1.1. Mise en œuvre de base du démonstrateur

Puisque les formules à prouver sont des conjonctions et que la théorie est sous forme de clauses de Horn, l'emploi de Prolog pour la mise en œuvre vient immédiatement à l'esprit. Dans cette optique, notre objectif pour la mise en œuvre du démonstrateur est d'utiliser au maximum les possibilités de Prolog vis-à-vis de l'unification et la gestion du retour-arrière, tout en garantissant la complétude des déductions. Pour cela, l'idée générale consiste à utiliser le mode de démonstration de Prolog (chaînage arrière en profondeur d'abord) en lui ajoutant un mécanisme permettant d'éviter les bouclages.

Nous présentons en premier lieu la façon dont nous représentons les axiomes de TIG et les faits. Nous expliquons en second lieu la question posée par les boucles et quelle mise en œuvre nous avons effectuée pour les détecter. Nous présentons ensuite une seconde mise en œuvre du démonstrateur permettant des preuves sur des ensembles de faits variables, puis une troisième mise en œuvre des preuves sur une théorie variable. Enfin, nous présentons une mise en œuvre générique par le biais d'un interpréteur.

1.1.1. Représentation des axiomes et des faits par des règles Prolog.

Une idée simple à mettre en œuvre consiste à représenter les faits et les axiomes de TIG directement sous la forme de règles Prolog et de lancer comme but la suite des atomes de la conjonction à prouver.

Représenter un fait ou un axiome sous forme d'une règle Prolog consiste à transformer toute clause de Horn, soit :

```
P ∨ Q1 ∨ … ∨ Qn
```

en une règle

```
P -> Q1 … Qn ;
```

Encore faut-il préciser comment sont représentés les atomes de la clause en Prolog. Pour cela, une première idée consiste à utiliser un prédicat Prolog de même nom et de même arité que le prédicat LDL de l'atome à représenter. Comme la syntaxe des identificateurs Prolog que doivent respecter les prédicats est respectée par tous les prédicats LDL, cette méthode est praticable.

Exemple :

L'atome LDL `distancep(ad,aa,ab)` est représenté par `distancep(as,aa,ab)`.

NB : la présence de 'a' préfixant chaque identificateur est expliquée en 2.2.4.3 et en annexe C.

Cette représentation immédiate des atomes, considérant un atome comme un objet, ne permet pas des manipulations "génériques". En effet, soit il faut introduire pour chaque traitement des atomes des règles particulières selon le nombre d'arguments de chaque atome, soit il faut transformer chaque atome avant traitement en une liste et uti-

liser un traitement générique sur les listes puis retransformer la liste une fois traitée en un atome. Comme ces méthodes sont coûteuses respectivement en mémoire et en temps, nous représentons les atomes par un prédicat Prolog de même nom, d'arité 1 et dont l'argument est la liste des arguments du prédicat LDL correspondant.

Exemple :
L'atome LDL `distancep(ad,aa,ab)` est représenté par `distancep([as,aa,ab])`.

Pour les atomes représentant des faits, la représentation est immédiate car toutes les constantes LDL issues de la traduction d'une construction ou d'une spécification suivent la syntaxe des identificateurs Prolog.

Exemple de représentation d'axiomes et de faits :

Les axiomes de TIG suivants :
`(8.2) appdr(p,l)  segment(s,p1,p2,l) appseg(p,s)`
dont l'énoncé en langue naturelle est
`Tout point appartenant à un segment appartient à la droite support de ce segment.`

```
(8.3) appdr(p,l)  segment(s,a,b,l1) appseg(i,s) distancep(d,i,a)
distancep(d,i,b) perp(l,l1) appdr(i,l) distancep(d',p,a)
distancep(d',p,b)
```
dont l'énoncé en langue naturelle est
`Tout point à égale distance de deux points A et B appartient à la médiatrice de [A B].`
sont représentés respectivement par les deux règles Prolog suivantes :

```
appdr([p,l]) :-
      segment([s,p1,p2,l])
      appseg([p,s]) ;
appdr([p,l]) :-
      segment([s,a,b,l1])
      appseg([i,s])
      distancep([d,i,a])
      distancep([d,i,b])
      perp([l,l1])
      appdr([i,l])
      distancep([d',p,a])
      distancep([d',p,b]);
```
Les faits
`segment(as,aa,ab,al) point(aa) point(bb) droite(al)`

sont représentés respectivement par les règles Prolog

```
segment([as,aa,ab,al])  :- ;
point([aa])  :- ;
point([bb])  :- ;
droite([al]) :-;
```

On peut lancer comme but `appdr([aa,al])`.

Cette solution a l'avantage d'utiliser pleinement le mécanisme de résolution de Prolog :
- gestion des retours-arrière.
- unification des variables.
- non-déterminisme.

1.1.2. La question des boucles

La solution précédente présente l'inconvénient majeur de ne pas préserver la complétude des preuves du fait des bouclages possibles dus à la recherche par profondeur d'abord de Prolog et à la forme récursive des axiomes de TIG.

Exemple :
L'axiome définissant la symétrie du parallélisme (`par(l1,l2)  par(l2,l1)`) est sous une forme récursive amenant Prolog à des bouclages.

Il faut donc concevoir un mécanisme supplémentaire permettant d'éviter les boucles tout en garantissant la complétude.

Une solution consiste à ne pas tenter la preuve de tout sous-but dont un ancêtre dans la démonstration est lui-même. De la sorte on garantit la complétude. En effet, comme aucune démonstration du but recherché n'existe ayant pour sous-but lui-même, alors on peut abandonner toute recherche de preuve d'un but si on detecte que la preuve qu'on est en train de construire contient le but initial comme sous-but. De la sorte on prouve que toutes les recherches abandonnées ne pouvaient mener à une preuve. De ce fait, toutes les démonstrations possibles sont bien parcourues tout en évitant les bouclages.
Malheureusement, cette solution n'est pas applicable si tous les sous-buts ne sont pas totalement instanciés, car alors elle n'assure pas la complétude.
Exemple :
Soit à prouver `p(x)` dans la théorie suivante, exprimée en Prolog :

```
p(x) ->
     p(y)
     r(x,y);
p(a) -> ;

r(b,a) -> ;
```

Si l'on utilise l'échec sur un ancêtre non instancié, la trace d'exécution en Prolog est la suivante :

```
p(x)
     p(y) [1ère règle]
          échec sur ancêtre
     x=a [2nde règle] : réussite
```

Cette trace montre que `p(b)` n'est pas démontré par la mise en œuvre alors que cet atome est prouvable dans cette théorie.

Par contre, si la théorie est écrite de telle façon que le démonstrateur garantisse que tout sous-but est instancié, alors la complétude est assurée. Dans notre cas, cette condition est remplie si on impose à TIG :
- que le but initial soit totalement instancié.
- que toute référence à une variable dans une queue de règle soit précédée d'un atome de typage définissant son type.

En effet, comme il n'existe aucun axiome de TIG permettant de déduire un atome de typage (car TIG n'admet aucun axiome "existentiel", voir deuxième chapitre, 3.2.1), les seules règles permettant de prouver un atome de typage sont des faits. Essayer de prouver un but représentant un atome de typage revient donc à essayer de l'unifier avec un fait, et donc de l'instancier. De ce fait si tout atome de typage précède la référence à une variable, son instanciation est garantie par le démonstrateur.

Exemple :
l'axiome (8.3) est dans ce but défini par

```
(8.3) appdr(p,l)  segment(s,a,b,l1) point(i) appseg(i,s) distance(d)
distancep(d,i,a) distancep(d,i,b) perp(l,l1) appdr(i,l) distance(d')
distancep(d',p,a) distancep(d',p,b)
```

où les atomes de typages `point(i)` et `distance(d)` ont été ajoutés respectivement avant la référence à i et à d de façon à garantir l'instanciation de i et de d durant une démonstration.

1.1.3. Mise en œuvre d'un mécanisme de détection des boucles.

Pour éviter ce problème des bouclages, nous présentons une première proposition de mise en œuvre. Au vu des problèmes de complétude qu'elle pose, une seconde version est présentée pour les résoudre.

1.1.3.1. Première proposition de mise en œuvre.

Pour mettre en œuvre le mécanisme de non-bouclage, nous proposons d'encadrer chaque paquet de règles définissant un atome `Q` par les deux règles suivantes :
- en tête du paquet :

```
Q ->
      asserta(Q, [!, fail] )
      fail ;
```

- en queue du paquet :

```
Q ->
      retract(Q, [!, fail])
      fail;
```

Ainsi, lors de la première exécution d'un but `Q` (totalement instancié), une règle

```
Q -> ! fail;
```

est ajouté indiquant que cette instanciation de `Q` est en cours de résolution et garantissant que tout nouvel appel de ce but provoquera un échec.

Quand la dernière règle du paquet est essayée, cela signifie que toutes les autres règles du paquet ont échoué. La dernière règle permet de retirer la règle indiquant que cette instanciation de `Q` est en cours de résolution. Ce retrait signifie que le but `Q` n'est plus un ancêtre de la démonstration en cours. Bien évidemment, cette dernière règle échoue une fois la règle de bouclage enlevée, préservant ainsi le fait qu'il n'y a pas de solution pour `Q`.

Exemple :
Le paquet de règles définissant l'atome `appdr(p,l)` en considérant les deux axiomes (8.2) et (8.3) est alors le suivant :

```
appdr([p,l]) ->
      asserta(appdr([p,l]),[!,fail])
      fail ;
appdr([p,l]) ->    % (8.2)
      segment([s,p1,p2,l])
      appseg([p,s]) ;
appdr([p,l]) ->          % (8.3)
      segment([s,a,b,l'])
      point([i])
      appseg([i,s])
      distance([d])
```

```
        distancep([d,i,a])
        distancep([d,i,b])
        perp([l,l1])
        appdr([i,l])
        distance([d'])
        distancep([d',p,a])
        distancep([d',p,b]);
appdr([p,l]) ->
        retract(appdr([p,l]),[fail,!]) !
        fail;
```

Il faut cependant regarder de plus près une question délicate : celle de l'ajout dynamique de règles Prolog. En effet, ajouter une règle (ou l'enlever) à un paquet en cours d'utilisation, n'est pas une opération dont la sémantique est universellement définie. Cette sémantique dépend du choix de la mise en œuvre de l'appel d'un but par l'interprète Prolog. En ce qui concerne PrologII+, la mise en œuvre permet d'affirmer, dans le cas où il reste des choix dans la pile pour le but p/n (où p est un prédicat et n son arité), que :

- si une règle de tête p/n est ajoutée dynamiquement en tête du paquet alors elle est ignorée en cas de retour-arrière sur les choix restants de la pile.
- si une règle de tête p/n est ajoutée dynamiquement en queue du paquet alors elle est prise en compte en cas de retour-arrière sur les choix restants de la pile.
- si une règle de tête p/n est enlevée dynamiquement en tête du paquet alors aucun retour-arrière n'est effectué si cette règle avait été déjà choisie.
- si une règle de tête p/n est enlevée dynamiquement en queue du paquet alors elle est ignorée en cas de retour-arrière sur les choix restants de la pile.

On peut résumer cette sémantique de façon intuitive en disant qu'on peut modifier le futur, mais que le passé reste le passé. Cela montre bien comme ce type d'ajout/retrait dynamique de règles doit être utilisé avec précautions.

Pour notre mise en œuvre, il s'avère que la sémantique donné par PrologII+ est bien adaptée. D'une part, l'ajout en tête ne modifie pas l'exécution du but courant ou de ses ancêtres mais seulement celle des ses descendants, ce qui est l'objectif recherché. D'autre part, le retrait en tête sur la dernière règle ne modifie pas les recherches sur le but courant ni sur les buts ancêtres. Il ne modifie pas non plus les recherches sur les buts descendants du but courant; en effet, au moment où le retrait est effectué on peut garantir que tous les choix sur les buts descendants ont été éxaminés. Les modifications ne concernent donc que les buts suivant le but courant du point de vue de l'exécution.

Malheureusement cette proposition de mise en œuvre du mécanisme d'échec sur bouclage ne fournit pas une solution correcte. Toute tentative de preuve échoue si ce but a été prouvé préalablement et qu'il restait des choix pour cette preuve. Cela est dû au fait que les règles empêchant le bouclage ne sont enlevées que si toutes les solutions ont été examinées. C'est en quelque sorte comme si on considérait que le but courant est un ancêtre des buts le suivant du point de vue de l'exécution.

Exemple :

Soit la théorie présentée précédemment et les faits
`segment(as,aa,aa,ll) appseg(pp,ss)`
La preuve des deux buts successifs `appdr(pp,ll) appdr(pp,ll)` échoue : la preuve du premier but réussit mais la preuve du second échoue car la règle empêchant les bouclages n'a pas été enlevée.

1.1.3.2. Seconde proposition de mise en œuvre.

Pour résoudre le problème rencontré en cas de réussite de la PREuve d'un but, il faut que la mise en œuvre proposée permette d'enlever les règles ajoutées dans tous les cas, réussite ou échec. Pour cela, nous définissons une nouvelle traduction des axiomes de TIG et nous proposons d'introduire un meta-niveau permettant de gérer correctement le retrait des règles de bouclage. La mise en œuvre proposée est décrite par les trois points suivants, pour un paquet de règles décrivant un prédicat `p` :

1) les faits sont représentés comme précédemment par des règles sans queue.

Exemple :
Les faits `appdr(aa,al1)` et `appdr(ab,al1)` sont représentés respectivement par les règles suivantes :

```
appdr([aa,al1]) ->  ;
appdr([ab,al1]) -> ;
```

2) Par rapport à la solution précédente, chaque axiome de TIG est traduit en une règle obtenue en remplaçant :
- l'atome de tête par un atome comportant un paramètre de plus : la liste d'ancêtres.
- le prédicat de la tête de règle par le prédicat primé correspondant au prédicat d'origine.
- chaque atome de propriété de la queue de règle par un atome comportant un paramètre de plus : la liste d'ancêtres.

Exemple :
Les axiomes (8.2) et (8.3) sont représentés par les règles suivantes, où `l_a` est le paramètre représentant la liste d'ancêtres :

```
appdr'(p,l,l_a) ->
      segment(s,p1,p2,l)
      appseg(p,s,l_a) ;
appdr'(p,l,l_a) ->
      point(i)
      segment(s,a,b,l')
      appseg(i,s,l_a)
      distance(d)
      distancep(d,i,a,l_a)
      distancep(d,i,b,l_a)
      perp(l,l1,l_a)
      appdr(i,l,l_a)
      distance(d')
      distancep(d',p,a,l_a)
      distancep(d',p,b,l_a);
```

3) deux règles sont ajoutées au paquet de clauses définissant le prédicat `p` en queue du paquet, après les règles décrivant les faits (en tête du paquet), de la forme :

```
p(_args) ->
      ajouter_ancêtre(p(_args))
      p'(_args)
      enlever_ancêtre(p(_args));

p(_args) ->
      enlever_ancêtre(p(_args))
      fail;
```

où `_args` est la liste des arguments du prédicat `p`.

Les règles `ajouter_ancêtre` et `enlever_ancêtre` sont définies respectivement par :

```
ajouter_ancêtre(p(_args)) ->
      asserta(p(_args), [!, fail] ) !

enlever_ancêtre(p(_args)) ->
      retract(p(_args), [!, fail]) ! ;
```

Exemple :

En considérant les axiomes (8.2) et (8.3) et les faits `appdr(aa,al1)` et `appdr(ab,al1)`, l'ensemble de règles définissant le prédicat `appdr` est alors le suivant :

```
appdr([aa,al1],_) ->  ! ;
appdr([ab,al1],_) -> ! ;
appdr([p, l]) ->
      ajouter_ancêtre(appdr([p, l]))
      appdr'([p, l])
      enlever_ancêtre(appdr([p, l]));
appdr([p, l]) ->
      enlever_ancêtre(appdr([p, l])) !
      fail;

appdr'([p,l]) □>
      segment([s,p1,p2,l])
      appseg([p,s]) ;
appdr'([p,l],l_a) □>
      point([i])
      segment([s,a,b,l'])
      appseg([i,s])
      distance([d])
      distancep([d,i,a])
      distancep([d,i,b])
      perp([l,l1])
      appdr([i,l])
      distance([d'])
      distancep([d',p,a])
      distancep([d',p,b]);
```

Comme par hypothèse tout but est instancié, on note qu'il est possible d'ajouter des cut (!) aux règles représentant des faits. En effet, si un but instancié est prouvé par un fait, il est inutilement coûteux de chercher par la suite à le prouver d'une façon mettant en jeu l'ensemble des axiomes.

Cette solution répond à notre exigence de complétude.

1.1.4. Mise en œuvre permettant des preuves sur des ensembles de faits variables.

Pour chaque utilisation différente du démonstrateur, l'ensemble des faits est spécifique. De ce fait, il nous faut un moyen de préciser à quel ensemble de faits une démonstration donnée fait référence. Une idée consiste à passer un paramètre indiquant par rapport à quel ensemble de faits la démonstration doit être faite. On ajoute alors à chaque règle représentant un fait ce même paramètre, instancié.

Exemple :

Les faits appdr(aa,al1) et appdr(ab,al1) faisant partie de la traduction d'une spécification donnée par le professeur sont représentés respectivement par les règles suivantes :

```
appdr([aa,al1],prof) ->  ;
appdr([ab,al1],prof) -> ;
```

La construction de la multifonction d'extension pose un autre problème plus difficile. Dans ce cas, il est a priori intéressant de profiter du retour-arrière implicite de Prolog pour prouver qu'une spécification correspond bien à une construction. Pour cela, il est naturel de programmer cette preuve à l'aide de la règle :

```
correction(F,S) ->
      implication(F,S)
      extension(F,S,S*)
      implication(S*,F);
```

où

`implication(E1,E2)` est satisfait si `TIG/E1` ⊢ `E2`

`extension(F,S,S*)` est satisfait si `S*` est un élément de la multifonction d'extension de S par rapport à F dans la théorie TEXT.

De la sorte, si une extension S* ne permet pas de prouver que TIG / S* ⊢ F, le programme effectue un retour-arrière sur les choix restants pour la règle "extension" en vue de trouver une autre extension qui puisse convenir. Trouver une autre extension consiste à revenir sur un choix fait pour construire un élément de la suite définissant S* (voir deuxième chapitre, 4.1.2.3.2), c'est à dire à remettre en cause le calcul d'un élément de la suite S_i à partir l'élément précédent S_{i-1}. Comme ce calcul est effectué par l'ajout d'un ensemble de faits, sa remise en cause correspond au retrait de cet ensemble de faits. Ainsi l'ensemble des faits de la spécification doit pouvoir augmenter (passage de S_i à S_{i+1}) ou diminuer (retour-arrière sur le passage de S_i à S_{i+1}) dynamiquement.

La modification dynamique des faits est aussi intéressante pour les faits de la construction. Dans le cas où la construction ne répond pas au contrat, on peut ainsi émettre des suppositions sur les éléments manquant de la construction : cela revient à ajouter les éléments manquants comme faits de la construction et à vérifier si la nouvelle construction fictive correspond au contrat.

Si une représentation des faits par des règles Prolog est adoptée, l'ajout et le retrait doivent se faire respectivement par les primitives assert et retract, qui modifient la mémoire globale du programme. Pour pouvoir modifier dynamiquement des faits tout en permettant un retour-arrrière avec un contexte préservé, il faut enlever exactement les faits ajoutés par la règle sur laquelle on revient. Gérer ce retrait de règle est donc assez lourd et ne permet pas l'utilisation du retour-arrière fourni par Prolog.

La solution que nous proposons est de représenter l'ensemble des faits sous la forme d'une liste passée en paramètre. Tester si un atome est un fait revient alors à tester s'il appartient à cette liste. De la sorte, la gestion du retrait est inutile, le retour-arrière implicite de Prolog en fait office.

De manière générale, les paquets de clauses définissant un prédicat `p` sont les suivants :

```
p(_args,l_f) ->
      fait(p(_args),l_f)) !  ;
```

```
p(_args,l_f) ->
      ajouter_ancêtre(p(_args,l_f))
      p'(_args,l_f)
      enlever_ancêtre(p(_args,l_f));
p(_args,l_f) ->
      enlever_ancêtre(p(_args,l_f))
      fail;


p'(_args,l_a,l_f) □> % premier axiome
      q(_args1)
      r(_args2)
      … ;

… % autres axiomes

fait(q,l_f) □>
      member(q,l_f) ;
```

Exemple :

L'exemple d'axiomes et de faits précédemment cité est représenté par les règles suivantes, (où `l_f` est la liste de faits).

```
appdr([p,l],l_f) ->
      fait(appdr([p,l]),l_f)) ! ;

appdr([p, l],l_f) ->
      ajouter_ancêtre(appdr([p, l],l_f))
      appdr'([p, l],l_f)
      enlever_ancêtre(appdr([p, l],l_f));

appdr([p, l],l_f) ->
      enlever_ancêtre(appdr([p, l],l_f))
      fail;


appdr'([p,l],l_a,l_f) □>
      segment([s,p1,p2,l])
      appseg([p,s],l_a,l_f) ;
appdr'([p,l],l_a) □>
      point([i])
      segment([s,a,b,l'])
      appseg([i,s],l_a,l_f)
      distance([d])
      distancep([d,i,a],l_a,l_f)
      distancep([d,i,b],l_a,l_f)
      perp([l,l1],l_a,l_f)
      appdr([i,l],l_a,l_f)
      distance([d'])
      distancep([d',p,a],l_a,l_f)
      distancep([d',p,b],l_a,l_f);
```

1.1.5. Mise en œuvre permettant des preuves sur une théorie variable.

Une mise en œuvre permettant des preuves sur une théorie variable est décrite par les points suivants.

1) L'ensemble des axiomes qu'il est possible de considérer est représenté sous forme d'un paquet de règles Prolog, de la forme suivante :

```
axiome(_identificateur,_tête,_queue) -> ;
```

où `_identificateur` est un terme Prolog identifiant de façon unique un axiome de TIG,

`_tête` est l'atome tête de l'axiome;

`_queue` est la liste des atomes de la queue de l'axiome.

Exemple :

L'axiome

```
(8.2) appdr(p,l)  segment(s,p1,p2,l) appseg(p,s)
```

est représenté par

```
axiome(8.2, appdr([p,l]), [segment([s,p1,p2,l]), appseg([p,s])]) -> ;
```

2) L'ensemble des axiomes par rapport auquel une preuve est cherchée (ensemble des axiomes dits "valides") est un sous-ensemble de cet ensemble d'axiomes possibles, représenté de deux manières possibles :

- par la liste des identificateurs des axiomes valides dans le cas où l'on veut utiliser le retour-arrière implicite de Prolog. Dans ce cas, on ajoute la liste des axiomes valides en paramètre à tous les prédicats utilisés par le démonstrateur. Pour un prédicat `p` donné, on remplace les règles primées correspondantes par les règles suivantes :

```
p'(_args,_axiomes_valides) ->
      axiome(_identificateur,p(_args,_axiomes_valides),_queue)
      axiome_valide(_identificateur,_axiomes_valides)
      prouver_queue(_queue,_axiomes_valides);

prouver_queue([],_axiomes_valides) -> ;
prouver_queue([r(_args,_axiomes_valides)|l],_axiomes_valides) ->
      r'(_args,_axiomes_valides)
      prouver_queue(l,_axiomes_valides);

axiome_valide(_identificateur,_axiomes_valides) ->
      member(_identificateur,_axiomes_valides) ;
```

- par un ensemble de faits Prolog dans le cas où l'utilisation du retour-arrière n'est pas indispensable, de la forme :

```
axiome_valide(_identificateur) -> ;
```

Dans ce cas, on remplace les règles primées correspondant à un prédicat donné `p` par les suivantes :

```
p'(_args) ->
      axiome(_identificateur,p(_args),_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue);

prouver_queue([]) -> ;
prouver_queue([_atome|l]) ->
      _atome
      prouver_queue(l);
```

Cette mise en œuvre permet de ne pas avoir à effectuer des ajouts et des retraits sur les règles représentant les axiomes mais seulement des faits, ce qui est bien plus efficace. Dans le cas où l'on veut utiliser un algorithme basé sur le retour-arrière de Prolog et modifiant la théorie, la représentation des axiomes valides par une liste est utilisée.

1.1.6. Mise en œuvre générique à l'aide d'un interpréteur.

Au vu des trois versions de mise en œuvre différentes que nous avons proposées, il s'avère

- d'une part que la conservation de la forme des axiomes et des faits est peu respectée par certaines des versions du fait de la nécessité d'ajouter des paramètres aux prédicats d'origine.
- d'autre part qu'il n'est pas toujours possible de représenter directement les axiomes et les faits par des règles Prolog.

Du point de vue du génie logiciel, il nous apparaît donc préférable de reconsidérer chacune de ces mises en œuvre en réalisant un interpréteur des faits et des axiomes. L'avantage de cette solution est de permettre d'exprimer chaque démonstrateur de façon générique, c'est à dire sans devoir définir un paquet de règles par prédicat LDL. De cette façon les trois versions sont assez similaires, permettant des modifications conjointes faciles. Cette solution permet aussi de conserver leur forme aux faits et aux axiomes; il n'est donc plus nécessaire d'ajouter des paramètres aux prédicats LDL. Par contre, les faits et les règles ne sont plus représentés directement par des règles Prolog.

Nous reprenons dans ce sous-chapitre chacune des versions du démonstrateur précédemment présentées pour en donner une mise en œuvre générique à l'aide d'un interpréteur.

1.1.6.1. Démonstrateur générique sans modification dynamique des faits et de la théorie.

Cet interpréteur est nécessaire pour la mise en œuvre du premier démonstrateur, en prenant en compte plusieurs ensembles de faits. Il est réalisé par les règles suivantes :

```
prouver_atome(a,_type_faits) ->
      ancêtre(a) !
      fail;

prouver_atome(a,_type_faits) ->
      fait(a,_type_faits) !;

prouver_atome(a,_type_faits) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      enlever_ancêtre(a) ;

prouver_atome(a,_type_faits) ->
      enlever_ancêtre(a)
      fail;

prouver_queue([],_type_faits) -> ;
prouver_queue([r|l],_type_faits) ->
      prouver_atome(r,_type_faits)
      prouver_queue(l,_type_faits);
```

Les faits sont représentés par des règles de la forme :

```
fait(a,_type_faits) -> ;
```

où `a` est l'un des faits de référence

`_type_faits` est l'ensemble de faits auxquels la démonstration fait référence.

Exemple :
Les faits `appdr(aa,al1)` et `appdr(ab,al1)` sont représentés par les règles

```
fait(appdr([aa,al1]),prof) -> ;
fait(appdr([ab,al1]),prof) -> ;
```

Les ancêtres sont ajoutés et retirés par les règles :

```
ajouter_ancêtre(a) ->
      asserta(ancêtre(a)) ;
enlever_ancêtre(a) ->
      retract(ancêtre(a)) ;
```

On note que du fait qu'on ne mélange plus les faits de référence et les règles permettant la détection d'ancêtres, l'utilisation d'assert et de retract ne pose pas de problème. En effet, il est certain qu'au moment de l'insertion ou de la suppression d'une règle ancêtre il ne reste aucun choix sur une règle ancêtre pour la première règle de "prouver_atome", car si on est passé à une des règles suivantes pour "prouver_atome", c'est que :
- soit tous les ancêtres possibles ont été envisagés et il ne reste aucun choix possible.
- soit un ancêtre a été trouvé et tous les choix restants ont été enlevés par le cut.

L'ensemble des axiomes possibles est représenté sous forme d'un paquet de règles Prolog, de la forme suivante :
`axiome(_identificateur,_tête,_queue) -> ;`
où `_identificateur` est un terme prolog identifiant de façon unique un axiome de TIG,
`_tête` est l'atome tête de l'axiome;
`_queue` est la liste des atomes de la queue de l'axiome.

L'ensemble des axiomes valides est représenté sous forme d'un paquet de faits Prolog, de la forme suivante :
`axiome_valide(_identificateur) -> ;`

1.1.6.2. Démonstrateur générique avec modification dynamique des faits.

Le démonstrateur prenant en compte des modifications dynamiques des faits est défini par des règles similaires au précédentes, au paramètre près `l_f`, représentant la liste des faits, près:

```
prouver_atome(a,_type_faits,l_f) ->
      ancêtre(a) !
      fail;

prouver_atome(a,_type_faits,l_f) ->
      fait(a,_type_faits,l_f) !;

prouver_atome(a,_type_faits,l_f) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits,l_f)
      enlever_ancêtre(a) ;

prouver_atome(a,_type_faits,l_f) ->
      enlever_ancêtre(a) !
      fail;
```

```
prouver_queue([],_type_faits,l_f) -> ;
prouver_queue([r|l],_type_faits,l_f) ->
      prouver_atome(r,_type_faits,l_f)
      prouver_queue(l,_type_faits,l_f);
```

La règle `fait` est définie par :

```
fait(a,_type_faits,l_f) ->
      member(a,l_f) ;
```

1.1.6.3. Démonstrateur avec modification dynamique des faits et de la théorie.

Le démonstrateur prenant en compte des modifications dynamiques de la théorie est défini par des règles similaires aux précédentes, au paramètre `l_av`, représentant la liste des axiomes valides, près :

```
prouver_atome(a,_type_faits,l_f,l_av) ->
      ancêtre(a) !
      fail;

prouver_atome(a,_type_faits,l_f,l_av) ->
      fait(a,_type_faits,l_f) !;

prouver_atome(a,_type_faits,l_f,l_av) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur,l_av)
      prouver_queue(_queue,_type_faits,l_f,l_av)
      enlever_ancêtre(a) ;

prouver_atome(a,_type_faits,l_f,l_av) ->
      enlever_ancêtre(a) !
      fail;

prouver_queue([],_type_faits,l_f,l_av) -> ;
prouver_queue([r|l],_type_faits,l_f,l_av) ->
      prouver_atome(r,_type_faits,l_f,l_av)
      prouver_queue(l,_type_faits,l_f,l_av);
```

La règle `axiome_valide` est définie par :

```
axiome_valide (_idf,l_av) ->
      member(_idf,l_av) ;
```

1.1.6.4. Prise en compte de l'hypothèse de nom unique.

Jusqu'ici nous avons présenté la mise en œuvre du démonstrateur sans prendre en compte l'hypothèse de nom unique sur les objets identifiés de la spécification. Prendre en compte cette hypothèse, c'est intégrer au démonstrateur les axiomes de HYPNU représentant cette hypothèse sur les identificateurs. Ces axiomes expriment que deux constantes sont toujours non_égales.

L'hypothèse de nom unique n'étant pas sur le même plan que les atomes de TIG, sa prise en compte est réalisée directement dans une règle prolog, en utilisant le prédicat prédéfini prolog *dif*. La règle suivante est insérée après la première règle permettant d'éviter les boucles :

```
prouver_atome(non_egal(o1,o2),_type_faits) ->
```

```
identifie(o1,_type_faits))
identifie(o2,_type_faits))
dif(o1,o2) ! ;
```

Cette règle suppose que la recherche d'une preuve sur un atome d'inégalité n'est jamais effectuée sur des objets de types différents, ce qui assuré:

- pour les preuves via un axiome d'une théorie. En effet, il suffit que la théorie soit cohérente du point de vue des types. Comme les théories TIG et TEXT sont cohérentes par rapport à La Géométrie, elles le sont aussi du point de vue des types.
- pour les preuves d'un atome d'une formule traduction d'une spécification ou d'une construction, par la définition des traductions.

La première règle est la traduction directe d'un axiome de HYPNU. La seconde règle est une optimisation triviale permettant de ne pas rechercher une preuve introuvable via les axiomes de la théorie.

1.2. Améliorations des performances:

Pour pouvoir disposer d'un système qui soit réellement efficace, les trois versions du démonstrateur que nous avons proposées doivent être améliorées. Nous présentons dans ce sous-chapitre quatre améliorations :

- La première vise à améliorer l'efficacité de la détection des bouclages, par recherche rapide sur les ancêtres.
- La deuxième a pour objectif d'accélérer la recherche des faits pour la version utilisant la modification dynamique des faits.
- la troisième vise à améliorer l'efficacité dans le cas où la preuve recherchée existe.
- La quatrième, la plus importante par le gain qu'elle apporte, est basée sur l'utilisation de lemmes et de propriétés particulières de notre système pour éviter de chercher plusieurs fois la démonstration d'un même atome.

1.2.1. Amélioration du mécanisme de détection de bouclages.

PrologII+ nous fournit depuis peu la possibilité de disposer d'un accès très performant (car indexé) à des faits, que l'on peut modifier dynamiquement et rapidement via les primitives `fassert/fretract` [Prologia 92]. Le programmeur peut définir lui-même des index sur les arguments de ces faits et fixer la taille de la table de hasch-code accélérant l'accès.

Avec la mise en œuvre via un interpréteur que nous proposons, les ancêtres sont mémorisés par un fait de la forme `ancêtre(a)`. Nous pouvons donc utiliser les primitives `fasserta` et `fretract` pour définir les règles `ajouter_ancêtre` et `enlever_ancêtre` à la place des primitives `asserta` et `retract` et ainsi accélérer considérablement notre mise en œuvre.

1.2.2. Amélioration de la version "faits variables" par recherche indexe sur les prédicats LDL pour les faits

La mise en œuvre que nous avons proposée pour le cas où on veut manipuler un ensemble de faits dynamiques peut s'avérer fort coûteuse en temps d'exécution. En effet, dans le cas le pire, c'est à dire quand le but à prouver n'est unifiable avec aucun fait, l'instanciation amène à effectuer sur chaque nœud de l'arbre de démonstration le parcours de la liste complète des faits.

Pour remédier à ce problème, nous proposons de segmenter l'accès à la liste des faits. L'idée est de donner un accès indexé sur le prédicat LDL de l'atome recherché.

Pour cela, on construit à partir de la liste des faits une liste de sous-liste. Chaque sous-liste ne contient que des atomes de prédicat identique. La liste contient autant de sous-listes que de prédicats LDL, dans un ordre prédéterminé.

La recherche d'un atome dans une telle liste s'effectue grâce à la fonction `arg2` de Prolog, donnant un accès direct au nième élément d'une liste, par la règle `member2` suivante :

```
member2(_atome,_liste) ->
      prédicat(_atome,_pred)
      ordre(_pred,n)
      arg2(n,_liste,_sous_liste)
      member(_atome,_sous_liste);
```

où

`prédicat(_atome,_pred)` est satisfait si `_pred` est le prédicat de l'atome `_atome`

`ordre(_pred,n)` est satisfait si n est l'ordre prédéfini pour le prédicat `_pred`.

La règle `fait` est alors définie par :

```
fait(a,_type_faits,l_f) ->
      member2(a,l_f) ;
```

Comme il est clair que le coût d'insertion n'est pas beaucoup plus élevé que précédemment et que les insertions sont beaucoup moins fréquentes que les recherches, le gain réalisé par cette amélioration est appréciable.

1.2.3. Amélioration par retardement de l'instanciation.

Pour les trois versions du démonstrateur, l'instanciation préalable systématique des variables, rendue nécessaire pour une détection des bouclages préservant la complétude, a des effets combinatoires. En effet, le démonstrateur essaie tous les objets existants du type donné.

Exemple :

Avec l'axiome (2.2) ["par deux points distincts il ne passe qu'une droite parallèle à une autre"] suivant :

```
(2.2) egal_dr(l1,l2) ⇐  point(p) appdr(p,l1) appdr(p,l2) par(l1,l)
par(l2,l)
```

et les faits

```
point(aa), point(ab), point(ac),
appdr(ac,al1), appdr(ac,al2)
par(al1,AL) par(al2,AL)
```

la preuve du but `egal_dr(al1,al2)` choisit successivement les points `aa`, `ab` et `ac` pour instancier le point `p` de l'axiome alors que seule l'instanciation de `p` avec la constante `ac` permet de trouver une preuve.

Bien qu'on ne puisse éviter de parcourir combinatoirement tous les cas possibles dans le cas où un but n'est pas démontrable, il est intéressant d'éviter ce parcours dans le cas contraire, c'est à dire de trouver plus rapidement une démonstration existante.

Pour y arriver, l'idée est d'instancier les buts le plus tard possible, c'est à dire juste avant de tester si le but a un ancêtre. Pour cela, le principe consiste à ne pas instancier a

priori les buts et d'essayer en premier lieu d'unifier le but sur un des faits avant d'essayer une preuve à l'aide de la théorie. Dans ce cas, l'instanciation doit être effectuée explicitement et la forme de la théorie ne nécessite plus d'atomes instanciant toute variable comme précédemment.

Exemple :
L'axiome (2.2) est maintenant sous la forme suivante :

```
(2.2) egal_dr(l1,l2)  appdr(p,l1) appdr(p,l2) par(l1,l) par(l2,l)
```

En considérant les faits
`point(aa), point(ab), point(ac),`
`appdr(ac,al1), appdr(ac,al2)`
`par(al1,AL) par(al2,AL)`
la preuve du but `egal_dr(al1,al2)` choisit en premier le point `ac` pour instancier le point `p` de l'axiome, au lieu d'instancier `p` successivement à `aa` et `ab` comme précédemment.

Pour mettre cette idée en œuvre, il faut tout d'abord définir l'instanciation d'un atome. L'instanciation est définie par rapport au profil du prédicat de l'atome à instancier, par des règles comme :

```
instanciation(perp([l1,l2])) ->
      fait(droite([l1]))
      fait(droite([l2]));

instanciation(appseg([p,s])) ->
      fait(point([p]))
      fait(segment([s,p1,p2,l]));
```

A partir de cette instanciation, on redéfinit la mise en œuvre du démonstrateur de la manière suivante :

```
prouver_atome(a,_type_faits) ->
      instanciation(a)
      prouver_atome2(a,_type_faits);

prouver_atome2(a,_type_faits) ->
      ancêtre(a) !
      fail;

prouver_atome2(a,_type_faits) ->
      fait(a,_type_faits,l_f) !;
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      enlever_ancêtre(a) ;

prouver_atome2(a,_type_faits) ->
      enlever_ancêtre(a) !
      fail;
```

Toutefois cette solution n'est pas complète. En effet, dans le cas où l'atome à prouver n'est pas totalement instancié, le fait de couper la recherche dans le cas où il s'instancie avec un fait (deuxième règle) supprime les possibilités restantes d'instanciation. Enlever le cut de cette règle est une solution. Il est pourtant dommage de ne pas arrêter la recherche dans le cas où le but est totalement instancié.

Pour ne pas perdre cette possibilité tout en restant complet, la solution est de distinguer deux cas : le cas où tous les paramètres sont instanciés, pour lequel on peut laisser le cut, et le cas contraire, où il faut l'enlever. Cela revient a scinder la deuxième règle en deux, de la façon suivante :

```
prouver_atome(a,_type_faits) ->
      bound2(_args)
      fait(a,_type_faits) !;

prouver_atome(a,_type_faits) ->
      not(bound2(_args))
      fait(a,_type_faits) ;
```

où `bound2(x)` est satisfait si tous les arguments de x sont instanciés.

Pour éviter un double calcul de `bound2`, on propose de le pré-calculer et de passer sa valeur en paramètre. En effet on ne peut mettre un cut après le `bound2` de la première règle et enlever le `bound2` de la deuxième règle, car d'autres règles suivent ces deux règles.

On crée ainsi un niveau supplémentaire comme suit :

```
prouver_atome(a,_type_faits) ->
      valeur(bound2(a),v)
      prouver_atome3(a,_type_faits,v) ;

prouver_atome3(a,_type_faits,vrai) ->
      fait(a,_type_faits) !;

prouver_atome3(a,_type_faits,faux) ->
      fait(a,_type_faits) ;

prouver_atome3(a,_type_faits,_) ->
      instanciation(a)
      prouver_atome2(a,_type_faits);

prouver_atome2(a,_type_faits) ->
      ancêtre(a) !
      fail;

prouver_atome2(a,_type_faits) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      enlever_ancêtre(a) ;

prouver_atome2(a,_type_faits) ->
      enlever_ancêtre(a) !
      fail;

valeur(q,vrai) ->
      q !;
valeur(q,faux) -> ;
```

1.2.4. Amélioration par utilisation de lemmes

Au vu des expérimentations, il est apparu que de nombreuses démonstrations d'un sous-but étaient essayées un grand nombre de fois, en particulier en cas d'échec total de la démonstration du but principal.

Exemple :
En considérant l'axiome de transitivité du parallélisme suivant

```
(2.2) par(l1,l2) ⇐  par(l1,l3) par(l3,l2) non_egal_dr(l1,l3)
non_egal_dr(l1,3)
```

et les faits

```
droite(ll1)
droite(ll2)
droite(ll3)
```

le but par(ll1,ll2) nécessite l'examen répété de chacun de ses sous-buts.

Pour remédier à cela, nous avons étudié la possibilité d'utiliser des "lemmes" et des "anti-lemmes".

Définition :
Un lemme est un atome totalement instancié pour lequel le démonstrateur a pu fournir une preuve, étant donnés une théorie et un ensemble de faits.
Un anti-lemme est un atome totalement instancié pour lequel le démonstrateur n'a pu fournir aucune preuve, étant donnés une théorie et un ensemble de faits.

1.2.4.1. Intérêt de l'utilisation de lemmes.

Utiliser les lemmes et les anti-lemmes permet de ne pas effectuer plusieurs fois les mêmes recherches. Pour que l'utilisation des lemmes présente un intérêt, il faut qu'il y ait un gain de performances significatif. Cela signifie qu'il faut que le coût d'utilisation des lemmes ne dépasse pas le gain réalisé par leur utilisation.

Pour que le gain soit suffisant, il est préférable que l'accès aux lemmes soit rapide. Comme les lemmes sont des atomes totalement instanciés, on peut les représenter par des faits prolog. L'utilisation de lemmes semble donc intéressante a priori grâce à l'utilisation des primitives `fassert` et `fretract`.

L'intérêt des lemmes est d'autant plus grand dans les démonstrations de plusieurs formules (conjonctives) successives. En effet si l'ensemble de faits et la théorie sont constants, une recherche de preuve peut hériter des lemmes et des anti-lemmes de la recherche précédente. C'est le cas par exemple, pour la démonstration du fait que la construction réalise toutes les propriétés de la spécification : dans ce cas, la preuve de chaque atome de la traduction de la spécification est requise en considérant un ensemble de faits fixes (celui représentant la traduction de la construction).

Si l'ensemble de faits et la théorie ne sont pas constants, la validité des lemmes ou des anti-lemmes n'est pas toujours assurée.
Quand la théorie augmente ou quand des faits sont ajoutés, l'ensemble des lemmes reste valide alors que les anti-lemmes ne le sont plus. Cela est dû à la monotonie de la logique du premier ordre. C'est le cas pour la traduction des phrases d'énoncé d'une construction, où les faits représentant la traduction de la phrase sont ajoutés à la précédente traduction. C'est le cas aussi dans les démonstrations successives d'égalité sur les objets d'une spécification ou d'une construction : l'algorithme se ramène à l'introduction des axiomes d'égalité à la théorie. Même si concrètement l'ensemble des faits diminue par substitution, du point de vue logique le nouvel ensemble est équivalent à l'ancien ensemble augmenté d'axiomes d'égalité.
Quand la théorie ou l'ensemble de faits diminue, comme c'est le cas lors des retours-arrière pour la construction de toutes les suites définissant la multifonction d'extension,

au contraire les lemmes ne sont plus a priori valides alors que les anti-lemmes le sont pour la même raison.

C'est pour cette raison que nous nous sommes servi de cette possibilité de conserver les lemmes après modification des faits pour la traduction incrémentale de la construction et pour les démonstrations d'égalités sur la traduction d'une spécification en CDL et sur la traduction d'une construction en SCL.

Par contre, dans le cas des retours-arrière pour la construction de toutes les suites définissant la multifonctions d'extension, l'intérêt de l'utilisation des lemmes et anti-lemmes est moins évident. Cela est dû d'une part au fait qu'on ne peut représenter les faits (et donc les lemmes et les anti-lemmes) par des règles si l'on veut utiliser correctement le retour-arrière implicite de prolog. D'autre part cela provient du fait qu'après un retour-arrière qui enlève des faits on ajoute immédiatement de nouveaux faits. Les anti-lemmes qui auraient pu être gardés lors du retour-arrière doivent être supprimés dés que des objets peuvent être ajoutés à la spécification. Comme les lemmes sont supprimés par le retour-arrière, lemmes et anti-lemmes n'ont ici qu'un intérêt restreint. Ces deux raisons nous ont amenés à choisir de ne pas utiliser les lemmes dans ce cas.

L'intérêt de l'utilisation des lemmes et anti-lemmes est important dans le cas où un but ne peut être prouvé. En effet, pour affirmer qu'un but ne peut être prouvé, le démonstrateur doit essayer toute la combinatoire, ce qui entraîne une forte redondance des calculs. Dans ce cas, c'est surtout les anti-lemmes qui améliorent les performances. En effet, si aucune preuve ne peut être trouvée pour un but, c'est aussi probablement qu'aucune preuve ne peut être donnée pour au moins un des sous-buts de chaque règle. Il y a donc de fortes chances pour que la preuve du même sous-but soit essayée plusieurs fois sans succès, dans la mesure où il s'agit de buts instanciés.

Les hypothèses que nous avons retenues sur la théorie nous permettent d'utiliser plus intensivement encore les lemmes et les anti-lemmes.

En effet, la théorie TIG doit être cohérente avec La Géométrie. De ce fait, bien qu'il n'y ait pas a priori de lien logique entre un atome `p(_args)` et l'atome représentant sa "négation" `non_p(_args)` (entre par exemple `perp(l1,l2)` et `non_perp(l1,l2)`), on peut toutefois affirmer que :

- si l'atome `p(_args)` a pu être prouvé dans TIG, la recherche d'une preuve pour `non_p(_args)` est inutile.
- si l'atome `non_p(_args)` a pu être prouvé dans TIG, la recherche d'une preuve pour `p(_args)` est inutile.

En effet, comme TIG est cohérent avec La Géométrie, toute formule qui est prouvée par rapport à TIG est prouvée par rapport à La Géométrie. Si l'atome `p(_args)` a pu être prouvé dans TIG, cela signifie que `p(_args)` peut être prouvé dans La Géométrie. Comme dans La Géométrie, si `p(_arg)` a pu être prouvé alors `¬p(_args)` ne peut être prouvé si `non_p(_args)` était prouvé dans TIG, cela signifierait que TIG n'est pas cohérent avec la géométrie. Donc si `p(_args)` a pu être prouvé dans TIG, `non_p(_args)` ne peut être prouvé dans TIG. La démonstration est symétrique pour la seconde affirmation.

1.2.4.2. Mise en œuvre utilisant les lemmes.

Pour utiliser des lemmes et des anti-lemmes dans le cas où l'ensemble de faits est fixe, nous proposons la mise en œuvre suivante :

```
prouver_atome(a,_type_faits) ->
      valeur(bound2,v)
      prouver_atome3(a,_type_faits,v) ;
```

```
prouver_atome3(a,_type_faits,vrai) ->
      fait(a,_type_faits) !;

prouver_atome3(a,_type_faits,vrai) ->
      lemme(a,_type_faits) !;

prouver_atome3(a,_type_faits,vrai) ->
      anti_lemme(a,_type_faits) !
      fail ;

prouver_atome3(a,_type_faits,vrai) ->
      atome_opposé(a,n_a)
      lemme(n_a,_type_faits) !
      fail;

prouver_atome3(a,_type_faits,faux) ->
      fait(a,_type_faits) ;

prouver_atome3(a,_type_faits,faux) ->
      lemme(a,_type_faits) ;

prouver_atome3(a,_type_faits,vrai) ->
      prouver_atome21(a,_type_faits);

prouver_atome3(a,_type_faits,faux) ->
      instanciation(a)
      prouver_atome22(a,_type_faits);

prouver_atome22(a,_type_faits) ->
      atome_opposé(a,n_a)
      lemme(n_a,_type_faits) !
      fail;

prouver_atome22(a,_type_faits) ->
      atome_opposé(a,n_a)
      lemme(n_a,_type_faits) !
      fail;

prouver_atome22(a,_type_faits) ->
      prouver_atome21(a,_type_faits) ;

prouver_atome21(a,_type_faits) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      ajouter_lemme(a)
      enlever_ancêtre(a) ;

prouver_atome21(a,_type_faits) ->
      ajouter_anti_lemme(a)
      enlever_ancêtre(a) !
      fail;
```

La séparation de la régle `prouver_atome2` en deux régles `prouver_atome21` et `prouver_atome22` a pour unique but d'optimiser le cas où l'atome est déjà instancié, en évitant de tester à nouveau si cet atome est un anti-lemme ou si son opposé a été prouvé.

Les lemmes et anti-lemmes sont ajoutés par les règles :

```
ajouter_lemme(a) ->
      fasserta(lemme(a)) ;
ajouter_anti_lemme(a) ->
      fasserta(anti_lemme(a)) ;
```

1.2.4.3. Problèmes posés par cette mise en œuvre

Cette amélioration importante présente néanmoins le défaut de ne pas préserver la complétude. Cela est dû à l'admission de deux types d'échec dans la recherche d'une démonstration : des échecs sur ancêtre et des échecs sur anti-lemme.

Nous avons montré que l'échec sur ancêtre utilisé seul préserve la complétude. Il en va de même de façon évidente de l'échec sur anti-lemme utilisé seul. Si l'on admet les deux types d'échec, on doit vérifier que les anti-lemmes introduits sont correctement définis, c'est à dire qu'aucune démonstration n'est possible étant donnés les faits et la théorie.

La mise en œuvre que nous venons de proposer ne garantit pas que les anti-lemmes soit correctement définis.

Exemple :
Étant donné la théorie suivante

```
P  Q
P  R
Q  P
R  vrai
```

la preuve de l'atome `P` amène à introduire un anti-lemme sur `Q`, car `Q` n'est pas prouvable avec `P` pour ancêtre alors que `Q` est prouvable sans ancêtre. De la sorte, cette mise en œuvre ne peut prouver `Q` si `P` a été préalablement essayé bien que `Q` soit prouvable.

De façon générale, le problème vient du fait que la mise en œuvre proposée ne différencie pas les échec dus aux boucles et les echecs dus au fait qu'il n'existe plus de regles à appliquer. On peut introduire un anti-lemme si l'on n'est sur qu'un echec vient du fait que toutes les régles ont été essayées sans succès. Or ce n'est pas le cas avec le mécanisme empêchant les boucle, où un échec dû à une boucle empêche le démonstrateur d'essayer toutes les règles.

1.2.4.4. Caractérisation de l'introduction d'anti-lemmes.

D'une façon générale, on peut caractériser une introduction correcte d'anti-lemmes conjointement avec un mécanisme d'échec sur ancêtre de la façon suivante : l'introduction d'un anti-lemme pour un atome A est correcte si l'échec de la démonstration de A n'est pas lié à un échec sur ancêtre sur un atome ancêtre de A.

Exemples :
Considérons les théories suivantes :

T1	T2
P ← Q	P ← Q
P ← R	P ← R
Q ← P	Q ← P
R ← S	Q ← vrai
	R ← S
	S ← Q

• La recherche de preuve de Q dans T1 (voir figure III.1) permet d'introduire les anti-lemmes :

S car l'échec de la preuve de R est dû à l'hypothèse du monde clos.

R car l'échec de la preuve de R n'est pas dû à un échec sur un ancêtre de Q car il n'est lié à aucun échec sur ancêtre.

Q car l'échec de la preuve de Q n'est pas dû à un échec sur un ancêtre de Q (Q n'a pas d'ancêtre).

La recherche de preuve de Q dans T1 ne permet pas d'introduire l'anti-lemme P, car l'échec de la preuve de P est lié à l'échec sur ancêtre de P (Q en l'occurrence). Il est possible que la preuve de P existe, ce qui n'est pas le cas dans T1.

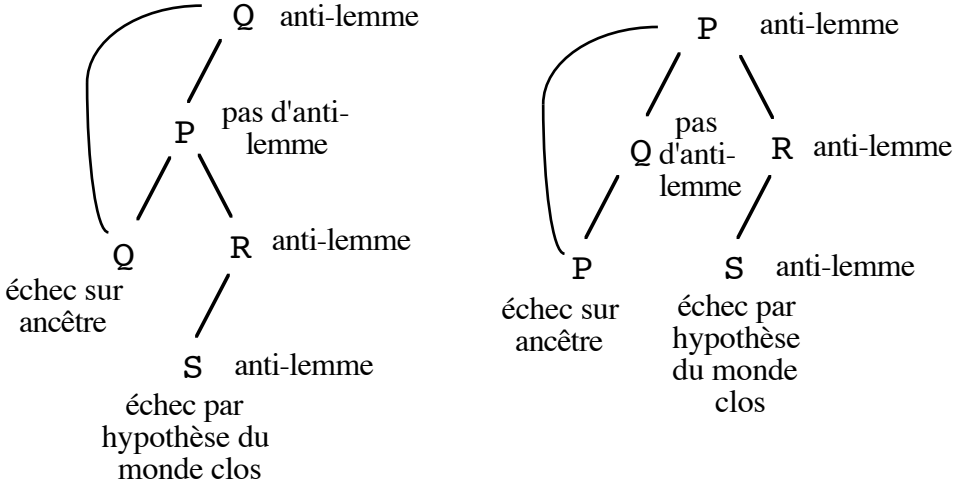


Figure III.1

• La recherche de preuve de `S` dans `T2` (voir figure III.2) ne permet d'introduire aucun anti-lemme.

La recherche de preuve de `Q` dans `T2` ne permet pas d'introduire les anti-lemme :

R car l'échec de la preuve `R` est lié à un échec sur un ancêtre de `R` (en l'occurrence `S`).

`P` car l'échec de la preuve `P` est lié à un échec sur un ancêtre de P (en l'occurrence `Q` et `S`). Il est possible que la preuve de `P` existe, ce qui est le cas dans `T2`.

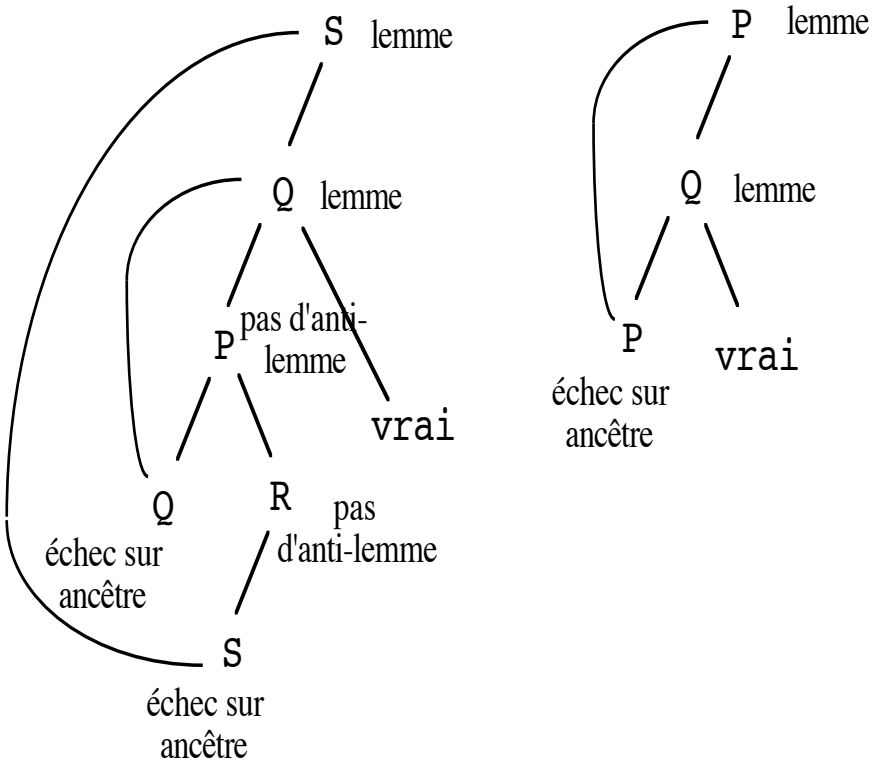


Figure III.2

1.2.4.5. Complétude d'un algorithme basé sur cette caractérisation de l'ajout d'anti-lemmes.

À la détection de l'échec de la preuve d'un but Q, trois cas se présentent :

1) soit la recherche d'une preuve de Q n'a pas été interrompue par un échec sur ancêtre□dans ce cas, l'échec est définitif et l'on peut ajouter un anti-lemme. Notre caractérisation autorise cet ajout.

2) soit la recherche de preuve a été interrompue par au moins un échec sur ancêtre et tous ces échecs sont liés à un ancêtre de Q. Dans ce cas, comme nous l'avons vu dans les exemples précédents, on ne peut rien conclure quand à la validité d'un ajout d'anti-lemme. Notre caractérisation n'autorise pas l'ajout. Dans le cas où Q n'est jamais prouvable, cela garantit la complétude. Dans le cas contraire, cela ne change rien par rapport à l'algorithme sans anti-lemmes, qui est complet.

3) soit la recherche de preuve a été interrompue par au moins un échec sur ancêtre et aucun de ces échecs n'est lié à un ancêtre de Q. Deux cas se présentent :

3.1) soit tous les échecs sont des échecs sur ancêtre vis-à-vis de Q. Dans ce cas, il est certain que toutes les démonstrations possibles ont été parcourues car il n'existe pas de démonstration ayant comme sous-but Q pour Q. Notre caractérisation autorise l'ajout d'anti-lemme et ne compromet pas la complétude.

3.2) soit il existe au moins un échec qui ne soit pas un échec sur ancêtre vis-à-vis de Q. Pour chacun de ces échecs sur un atome A, toutes les démonstrations possibles pour □ ont été essayées. En effet, puisque A n'est pas un ancêtre de Q et qu'il y a eu échec sur ancêtre pour A, il faut que A soit sous-but de Q. Si A est sous-but de Q sans être un de ses ancêtres alors toutes les démonstrations possibles pour A ont été essayées (on est ramené au cas 3.1 pour A). Puisque les autres échecs relèvent du cas 3.1, alors le fait que notre caractérisation autorise l'ajout d'anti-lemme ne compromet pas la complétude.

1.2.4.6. Une mise en œuvre utilisant cette caractérisation.

L'idée la plus simple pour mettre en œuvre cette caractérisation à partir de la mise en œuvre précédente consiste à :

- mémoriser les échecs sur ancêtre liés à la preuve d'un atome.
- tester si l'un de ces échecs fait partie de la liste d'ancêtre de l'atome :
  - si ce n'est pas le cas, ajouter un anti-lemme.
  - si c'est le cas, ne pas ajouter d'anti-lemme.

La difficulté est que la mise en œuvre précédente est basée sur la gestion implicite de l'échec et du retour-arrière par Prolog. Mais il semble difficile de conserver cette caractéristique de notre mise en œuvre car :

- soit on utilise une méthode consistant à mémoriser les échecs dans la mémoire globale (par des assert ou des assign) et cette méthode résiste a priori aux retours arrières.
- soit on désire mémoriser les échecs dans une variable locale pour conserver les possibilités de retours arrière. Malheureusement, cela est impossible car l'interpréteur Prolog ne mémorise pas les échecs. Pour y arriver, il faudrait ajouter une règle gérant ces échecs et on ne se servirait plus alors du mécanisme de gestion des échecs et du retour-arrière fourni par Prolog.

De plus, en gérant par programme les échecs et les retours arrières, on perd l'efficacité fournie par l'interprète Prolog et donc l'avantage d'utiliser le langage.

Pour que l'ajout d'anti-lemmes soit intéressant, il faut donc un mécanisme permettant de mémoriser les échecs de façon rapide et globale tout en permettant l'utilisation du mécanisme d'échecs et de retours arrière de Prolog. Un tel mécanisme est possible, basé sur une propriété d'ordre sur les anti-lemmes.

Cet ordre est basé sur l'ordre de parcours de l'arbre de preuve d'une démonstration. On ordonne tout noeud de cet arbre de la façon suivante : un nœud n1 de l'arbre de parcours est plus petit qu'un nœud n2 si la preuve de n1 a été essayée avant celle de n2.

Supposons que l'on ajoute un fait `échec(a,t)` au moment `t` où un lemme sur ancêtre se produit pour l'atome `a`. L'ensemble des échecs est alors ordonné selon l'ordre de parcours de l'arbre de preuve.

En transformant la régle prouver_atome21 par

```
prouver_atome21(a,_type_faits) ->
      temps(t1)
      prouver_atome23(a,_type_faits,t1) ;

prouver_atome23(a,_type_faits,_) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      ajouter_lemme(a)
      enlever_ancêtre(a) ;

prouver_atome23(a,_type_faits,t1) ->
      temps(t2)
      ajouter_anti_lemme(a,t1,t2)
      enlever_ancêtre(a) !
      fail;
```

La définition de l'ajout de l'anti-lemme pour un but `a` donné est simple :

Supposons que la recherche d'une preuve pour a ait commencé au temps `t1` et qu'il ne reste plus aucun choix possible au temps `t2`.

- s'il existe une règle `échec(b,t)` telle que `t1` ≤ `t` ≤ `t2` et `b` est un ancêtre de `a`, alors l'ajout n'est pas possible. En effet, cela signifie qu'un échec sur ancêtre s'est produit pendant la recherche de preuve pour `a` car la date `t` de cet échec est comprise entre la date de début et de fin de recherche. Pour tester si `b` est un ancêtre de `a`, comme tous les ancêtres courant sont représentés par des règles `ancêtre(x)`, il suffit de montrer `ancêtre(b)`.
- s'il n'existe pas de régle `échec(b,t)` telle que `t1` ≤ `t` ≤ `t2` et `b` est un ancêtre de `a`, alors l'ajout est possible.

On peut optimiser la définition de l'ajout de l'anti-lemme en évitant le test vis-à-vis des ancêtres. En effet, on s'aperçoit que dès que la recherche d'une preuve pour un atome `b` est terminée (par un échec ou une réussite) tous les échecs concernant l'atome `b` ne sont plus examinés car l'ancêtre `b` est enlevé. L'idée est de se servir de ce lien existant entre les ancêtres courants et les échecs examinés. Elle consiste à enlever toutes les règles `échec` au moment où l'on sait qu'elles sont inutiles. Cela correspond à enlever la règle `échec` correspondant à l'atome (si il en a une) en même temps qu'on ôte l'ancêtre correspondant.

Il semblerait alors qu'il suffise qu'il reste une seule règle `échec` pour que l'ajout d'un anti-lemme soit impossible. C'est le cas si l'on raisonne seulement sur un arbre de démonstration OU, c'est à dire en considérant tous les axiomes comme une implication entre deux atomes. Comme en général l'arbre de démonstration est un arbre ET/OU, cette caractérisation très simple de la possibilité d'ajout d'un anti-lemme n'est pas correcte. En effet, dans ce cas la recherche de preuve d'un atome bi faisant partie d'un axiome de la forme a <= b1 b2 … bn hérite d'une partie des règles `échec` occasionnées par la recherche des bj, j<i. L'ajout d'anti-lemme pour un atome `a` est donc possible s'il reste une seule règle `échec` telle qu'elle ait été ajoutée entre le début et la fin de la recherche d'une preuve pour `a`.

Le gain réalisé vient de deux améliorations :

- la recherche sur les ancêtres des échecs est devenue inutile.
- la recherche sur le paquet de règles `échec` est accélérée car le nombre de règles du paquet est considérablement restreint.

Par contre, cette solution nécessite des retraits de règles inexistants auparavant. Mais comme chacun de ces retraits est a priori aussi coûteux qu'une unification sur le paquet de règles et que chacun des retraits fait diminuer le nombre de règles, le temps d'accès aux échecs n'est plus strictement croissant et le coût des retraits est en tout état de cause inférieur au gain.

Les règles obtenues sont les suivantes :

```
prouver_atome21(a,_type_faits) ->
      temps(t1)
      prouver_atome23(a,_type_faits,t1) ;

prouver_atome23(a,_type_faits,_) ->
      ajouter_ancêtre(a)
      axiome(_identificateur,a,_queue)
      axiome_valide(_identificateur)
      prouver_queue(_queue,_type_faits)
      ajouter_lemme(a)
      enlever_ancêtre(a)
      enlever_echec(a)  ;
```

```
prouver_atome23(a,_type_faits,t1) ->
      enlever_ancêtre(a)
      enlever_echec(a)
      temps(t2)
      ajouter_anti_lemme(a,t1,t2)!
      fail;

ajouter_anti_lemme(a,t1,t2) ->
      échec(a,t)
      t1 <= t <= t2  !
      fassert(anti_lemme(a);

ajouter_anti_lemme(a,t1,t2) -> ;
```

1.3. Utilisation d'un contre-modèle.

Pour certains exemples, le temps de réponse pour la réduction de la traduction de la figure de l'élève reste cependant difficilement acceptable. Dans le cas de littéral improuvable, le démonstrateur explore toute la combinatoire des possibilités. Une idée d'amélioration consiste, pour un littéral donné, à trouver un contre-exemple dans une figure (correcte) déjà construite. Si c'est le cas, on dispose d'un modèle dans lequel le littéral est faux et il est inutile de chercher à le prouver. Cette idée était à la base des travaux de Gelertner [Gelernter et al. 63] sur la démonstration automatique en géométrie.

Pour trouver ce contre-exemple, nous proposons quatre possibilités :

- la première est d'effectuer un calcul formel dans la théorie générale de la géométrie. Si on trouve un contre-exemple dans cette théorie, alors c'en est un aussi pour toute sous-théorie.

Exemple :

GEOSPECIF (voir chapitre premier, 2.1.4) sait décider si la figure en cours de construction est un contre-exemple pour certaines constructions.

- la deuxième est d'utiliser un modèle de construction, fourni auparavant par le professeur, et sur lequel le système aura pré-calculé toutes les propriétés possibles (en nombre fini).

- la troisième est d'utiliser l'oracle de CABRI. Si il trouve un contre-exemple, avec une précision des calculs suffisamment fine et une marge pour l'égalité suffisamment grande, alors on peut estimer la propriété fausse en général.

C'est cette troisième possibilité que nous avons commencé à mettre en œuvre.

- une quatrième enfin consiste à utiliser la programmation logique avec contraintes sur intervalles [Older et al. 93] qui peut rapidemment dans certains cas donner des résultats approchés mais exacts suffisants pour prouver des contre-propriétés.

2. Mise en œuvre de TALC-Enseignant.

Dans ce sous-chapitre, nous présentons la mise en œuvre du système dont dispose l'enseignant, dénommé TALC-Enseignant.

Nous présentons en premier lieu les fonctionnalités offertes par cet outil à l'enseignant pour lui permettre d'exprimer le contrat didactique qu'il propose à l'élève. Nous présentons ensuite la mise en œuvre de la traduction simple d'une formule de CDL en une formule de LDL. Enfin, nous présentons la phase de réduction qui permet à partir d'une traduction simple et de la théorie TIG, d'obtenir une traduction complète.

2.1. Mise en œuvre de l'interface de TALC-Enseignant.

Pour définir un exercice, l'enseignant doit définir trois éléments :
- la spécification de l'exercice en CDL.
- la théorie TIG qui lui est associée.
- la théorie TEXT décrivant quels objets non-spécifiés l'élève peut construire.

L'interface TALC-Enseignant permet de définir ces trois éléments du contrat.

2.1.1. Edition des éléments du contrat

L'interface propose à l'enseignant un éditeur pour créer, modifier et sauvegarder les éléments du contrat sous la forme d'un fichiers texte

2.1.1.1. Édition de TIG

Pour définir les axiomes de TIG, l'enseignant a à sa disposition un éditeur de texte. TALC-enseignant utilise l'éditeur fournit par PrologII+.

Pour être validés, les axiomes de TIG doivent respecter la syntaxe suivante :

`n1.n2.n3;  T`
lt -> lq_1 lq_2 … lq_n;

`n1` est un entier identifiant le prédicat du littéral impliqué et son "opposé". (par exemple, 9 pour appdr et non_appdr).
`n2` est un entier numérotant entre eux les axiomes d'un prédicat.
n3 est égal à zéro si le prédicat du litéral de tête est "positif", un sinon.

Les littéraux des clauses suivent la syntaxe de LDL.

Exemple :

6.3.0; "Deux distances |A I| et |I B| telles que |A I|= 1/2|A B| sont égales si I e [A□B]"
egal_dis(r1,r2) ->
  segment(s,p1,p2,l)
  appseg(i,s)
  distancep(r1,p1,i)
  distancep(r2,p2,i)
  distancep(r3,p1,p2)
  demidist(r1,r3);

6.1.1; "Deux distances sont différentes si l'une est inférieure à l'autre"
non_egal_dis(d1,d2) ->
```
 infdist(d1,d2);
```

2.1.1.2. Édition de la spécification de la figure.

Pour définir une spécification, l'enseignant utilise l'éditeur de texte de TALC-enseignant à sa disposition, en respectant la syntaxe de CDL.

2.1.1.3. Édition des outils de construction.

Pour définir les outils de construction dont dispose l'élève pour répondre au contrat, l'enseignant doit disposer d'une interface lui permettant d'invalider certains des menus de construction de TALC-Éleve. Cette interface est en cours de réalisation.

2.1.1.4. Édition de TEXT.

Pour définir TEXT, l'enseignant utilise l'éditeur de texte de TALC-Enseignant à sa disposition.

Un axiome de TEXT, de la forme
```
∀y1,y2,…,yn, Cond(y1,y2,…,yn) ⇒
∃x, z1,z2,…,zn Propmin(x, z1,z2,…,zn, y1,y2,…,yn)
```

est représenté en prolog par une règle

```
axiome_text(x,type_x,n,
                  Cond(y1,y2,…,yn),
                  Propmin(x, z1,z2,…,zn, y1,y2,…,yn)) -> ;
```

où `type_x` est l'atome de typage de l'objet `x` dans `Propmin(x, z1,z2,…,zn, y1,y2,…,yn)`
où `n` représente le degré de liberté de l'objet `x` construit
et où les formules Cond(...) et Propmin(...) sont représentées par des listes d'atomes Prolog.

Exemple :
l'axiome
```
∀l1,l2, droite(l1) ∧ droite(l2) ∧ non_par(l1, l2) ⇒ non_egal_dr(l1,l2)
∧ ∃p∈point(p) ∧ appdr(p,l1) ∧ appdr(p,l2)
```
exprimant que :
"Quelles que soient deux droites différentes et non parallèles, il existe un point intersection de ces droites"

est représenté par
```
 axiome_text(p,point(p),0,
      [droite(l1), droite(l2), non_par(l1, l2), non_egal_dr(l1,l2) ],
      [point(p), appdr(p,l1), appdr(p,l2)]) -> ;
```

2.1.2. Création d'un exercice

Pour créer un exercice, l'enseignant doit avoir précédemment sélectionné les théories TIG et TEXT concernées. Pour cela, deux solutions sont possibles :
- soit il peut définir des théories par défaut, chargées automatiquement au démarrage du système.
- soit il peut charger provisoirement d'autres théories à partir d'un fichier.

Pour définir TIG, le système lui propose une facilité supplémentaire, qui lui permet de désigner une sélection d'axiomes de la théorie chargée. Cette sélection s'opère par l'intermédiaire d'une boîte de dialogue lui permettant de designer les axiomes valides parmi la liste d'axiomes.

Il faut aussi qu'il ait désigné les outils de construction qu'il met à la disposition de l'élève.

Une fois ces deux théories et les outils de construction définis, il lui suffit de demander au système d'effectuer la traduction d'une spécification. Cette spécification est soit celle de la fenêtre active soit celle contenue dans un fichier quelconque. Le système produit alors un fichier utilisable par l'application TALC-Élève, qui contient :
- la traduction de la spécification.
- les théories TIG et TEXT du contrat.
- la description des outils disponibles à l'élève.

NB : La traduction tient compte de la théorie TIG pour construire l'exercice. Elle détecte les éventuelles erreurs de la spécification. Les erreurs détectées sont les suivantes :
- le type d'un identificateur n'est pas connu.
- un identificateur représente deux objets de types différents.
- un identificateur représente deux objets différents de même type.
- une égalité a été posée entre termes non identifiables.
- un objet auquel appartient un autre n'est pas un ensemble de points.
- un objet parallèle (ou perpendiculaire) à un autre n'est pas un ensemble de points alignés.

Si une de ces erreurs est détectée, la traduction ne produit pas d'exercice. De plus, le système met en garde l'utilisateur contre les propriétés définies plusieurs fois, en supposant que cela peut signifier de sa part une erreur de frappe.

2.2. Mise en œuvre de la traduction simple de CDL.

La traduction simple de CDL s'effectue en deux phases principales. La première phase, appelée analyse des identificateurs, a pour objet de construire l'ensemble des atomes de typage pour chaque identificateur (appelé environ) ainsi qu'un dictionnaire des termes identifiés. La seconde phase construit la traduction simple à partir de ces ensembles et de la spécification, sous la forme d'une liste de littéraux LDL.

Ces deux phases sont précédées d'une phase d'analyse syntaxique. Elle a pour objet de transformer le texte source de la spécification en une suite de lexèmes. Chacune des deux phases principales effectue une seule passe sur cette suite de lexèmes.

Ces trois phases sont présentées en annexe F.

2.3. Réduction de la traduction simple de CDL pour obtenir une traduction complète.

Comme nous l'avons déjà défini, une traduction complète est une traduction dans laquelle tout objet de la spécification traduite est différent des autres objets au regard de la théorie TIG. Une traduction complète d'une spécification en CDL est obtenue par "réduction" de la traduction simple de cette spécification. La réduction d'une traduction simple est un processus qui détecte les couples d'objets égaux vis-à-vis de TIG et substitue l'un des deux par l'autre. Ce processus est défini par les règles Prolog suivantes :

`Reduction(F,F2)` est satisfait si
`F` la formule LDL traduction simple d'une spécification
`F2` est la formule LDL réduite en tenant compte de la théorie TIG

```
reduction(F,F2) ->
      objet(o1,F)
      objet(o2,OF)
      dif(o1,o2)
      test_egal(o1,o2,F,TIG)
      ordonner(o1,o2,o1',o2')
      substitution(o1',o2',F,F1) !
      reduction(F1,F2) ;
reduction(F,F) -> ;

test_egal(o1,o2,TIG,F) ->
      meme_typep(o1,o2)
      or(non_identifié(o1) , non_identifié(o2) )
      predicat_egalite_type(o1,p)
      prouver_atome(p(o1,o2),F);
```

`ordonner(o1,o2,o1',o2')` est satisfait
si `o1` << `o2` et `o1`=`o1'`, `o2`=`o2'`
si `o2` << `o1` et `o1`=`o2'`, `o2`=`o1'`

`substitution(o1,o2,F,F1)` est satisfait si `F1` est la formule `F` dans laquelle toute occurrence de `o1` a été substituée par `o2` et ne contenant pas de doublons.

`meme_type(o1,o2)` est satisfait si `o1` et `o2` sont de même type.

`predicat_egalite_type(o,p)` est satisfait si `p` est le prédicat d'égalité correspondant au type de `o`

prouver_atome(a,F) est satisfait s'il existe une preuve du littéral Q dans la théorie TIG en utilisant les faits `F`.
Cette règle est celle définie en 1.2.4.6. Cela signifie que l'on utilise ici le démonstrateur pour lesquels les faits ne sont pas passés en paramètre et utilisant les lemmes et les anti-lemmes. En effet, puisque l'algorithme réessaie toutes les égalités possibles après substitution, il n'est pas nécessaire de passer les faits en paramètre. Et comme nous l'avons expliqué en 1.2.4.1, l'utilisation des lemmes et anti-lemmes est possible à condition d'enlever tous les anti-lemmes à chaque substitution opérée.

La principale caractéristique de ce processus est qu'une fois une substitution effectuée pour un couple d'objets égaux, l'égalité est à nouveau examinée pour chacun des couples d'objets restants. En effet, il est possible que les substitutions effectuées permettent de trouver une preuve d'une égalité qu'on ne pouvait trouver avec la formule précédente. Cela est dû au fait que TIG ne contient pas les axiomes d'égalité. Le processus présenté prend en compte ces axiomes :
- l'axiome de substitution par l'appel à `substitution`.
- l'axiome de réflexivité est assurée par l'égalité syntaxique des identificateurs de Prolog.
- l'axiome de symétrie est assuré par le fait que l'on essaie de prouver les égalités sur tout couple d'objets de la spécification, aussi bien le couple (a,b) que le couple (b,a).
- l'axiome de transitivité par les substitutions successives effectuées.

Notons que le processus n'examine pas les égalités entre objets identifiés, du fait de l'hypothèse de nom unique que nous avons retenue pour ces objets.

3. Mise en œuvre de TALC-Élève

Dans ce sous-chapitre, nous présentons la mise en œuvre de TALC-Élève. TALC-Élève est l'outil de construction de figures dont dispose l'élève pour répondre au contrat défini préalablement par l'enseignant dans TALC-Enseignant.

Nous présentons en premier lieu la mise en œuvre de l'interface et les fonctionnalités offertes par cet outil à l'élève. Nous présentons ensuite la mise en œuvre de la communication entre TALC-Élève et Cabri-géomètre. Une fois présentée la mise en œuvre de la traduction simple d'un énoncé de Cabri-géomètre en une formule de LDL, nous pouvons expliquer la mise en œuvre de la phase de réduction qui permet à partir de la traduction courante de la construction et de la traduction simple de l'énoncé produit par le dernier pas de construction d'obtenir une traduction complète.

3.1. Mise en œuvre de l'interface de TALC-Élève.

TALC-Élève dispose d'une interface de construction de figures géométriques dont les fonctionnalités principales sont celles de Cabri-géomètre. De ce fait, l'utilisation de TALC-Élève a été conçue de façon à ne pas perturber les élèves familiers de Cabri-géomètre. C'est pourquoi un objectif de la mise en œuvre de l'interface est que la plupart des fonctionnalités de TALC-Élève utilisant Cabri-géomètre soit actionnées d'une manière identique à celle de Cabri-géomètre.

Dans ce sous-chapitre, nous présentons tout d'abord les fonctionnalités offertes explicitement par TALC-Élève à l'élève pour ensuite présenter les fonctionnalités transparentes de son point de vue et qui sont nécessaires.

3.1.1. Fonctionnalités explicites.

Pour construire une figure répondant au contrat défini dans un exercice, l'élève dispose des fonctionnalités suivantes :

1) Choix de l'exercice à réaliser.

L'élève choisit un exercice parmi ceux élaborés par l'enseignant. A un moment donné, un seul exercice est chargé.

L'élève peut changer d'exercice en chargeant un autre exercice.

L'exercice chargé contient les quatre éléments définis en 2.1.1. L'élève prend conscience des outils de construction disponibles en examinant quels menus de construction lui sont proposés. Pour les autres éléments du contrat, il lui est possible de consulter□

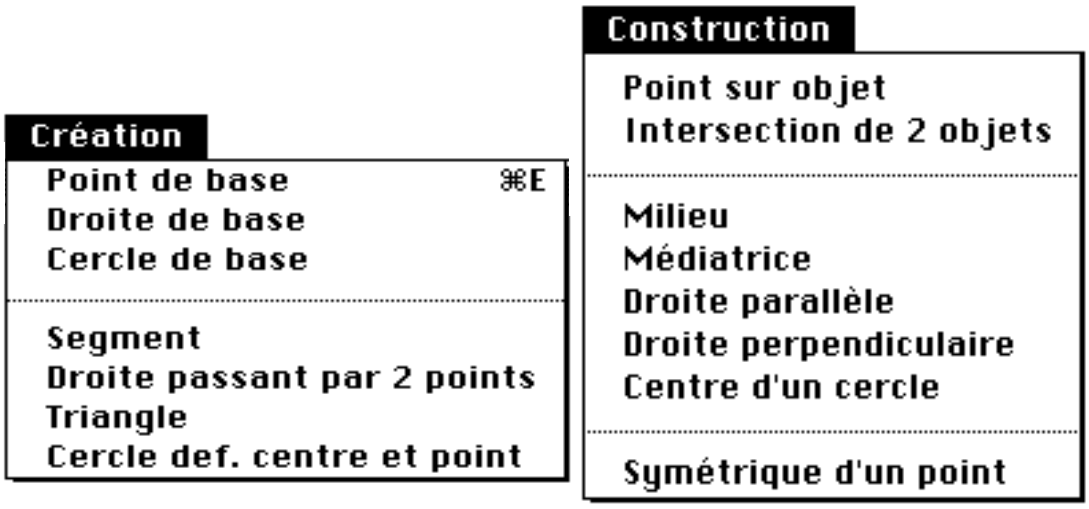
- l'énoncé en langue naturelle que le professeur a traduit en une spécification en CDL.
- l'ensemble des axiomes de TIG valides.
- l'ensemble des axiomes de TEXT valides.

2) Création d'une construction géométrique ou chargement d'une figure déjà construite dans une fenêtre.

Comme dans Cabri-géomètre, à chaque fenêtre (donc à chaque construction) est attaché un énoncé de la figure dans le langage d'énoncés de Cabri-géomètre (voir chapitre deuxième, 2.3.2 et annexe D). Cette énoncé est visualisé à la demande dans une fenêtre et rafraîchi automatiquement à chaque modification de la construction correspondante.

3) Enregistrement d'une construction.

4) Construction d'objets dans la fenêtre courante. Ces constructions sont effectuées à travers une sous partie de l'ensemble des menus potentiellement disponibles suivants :

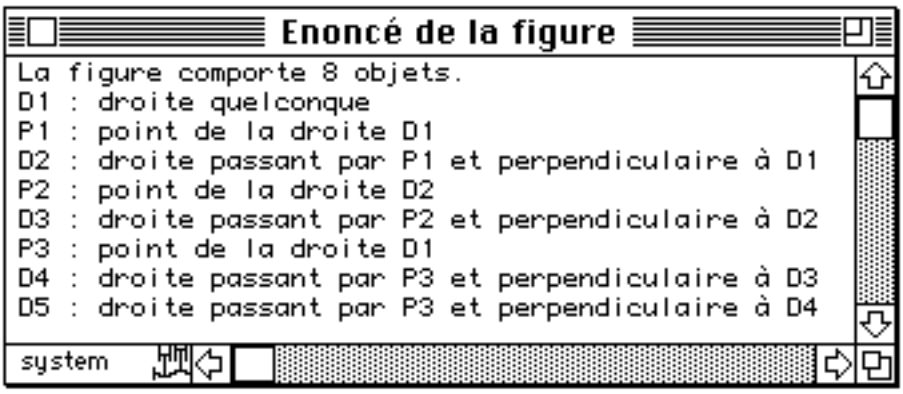


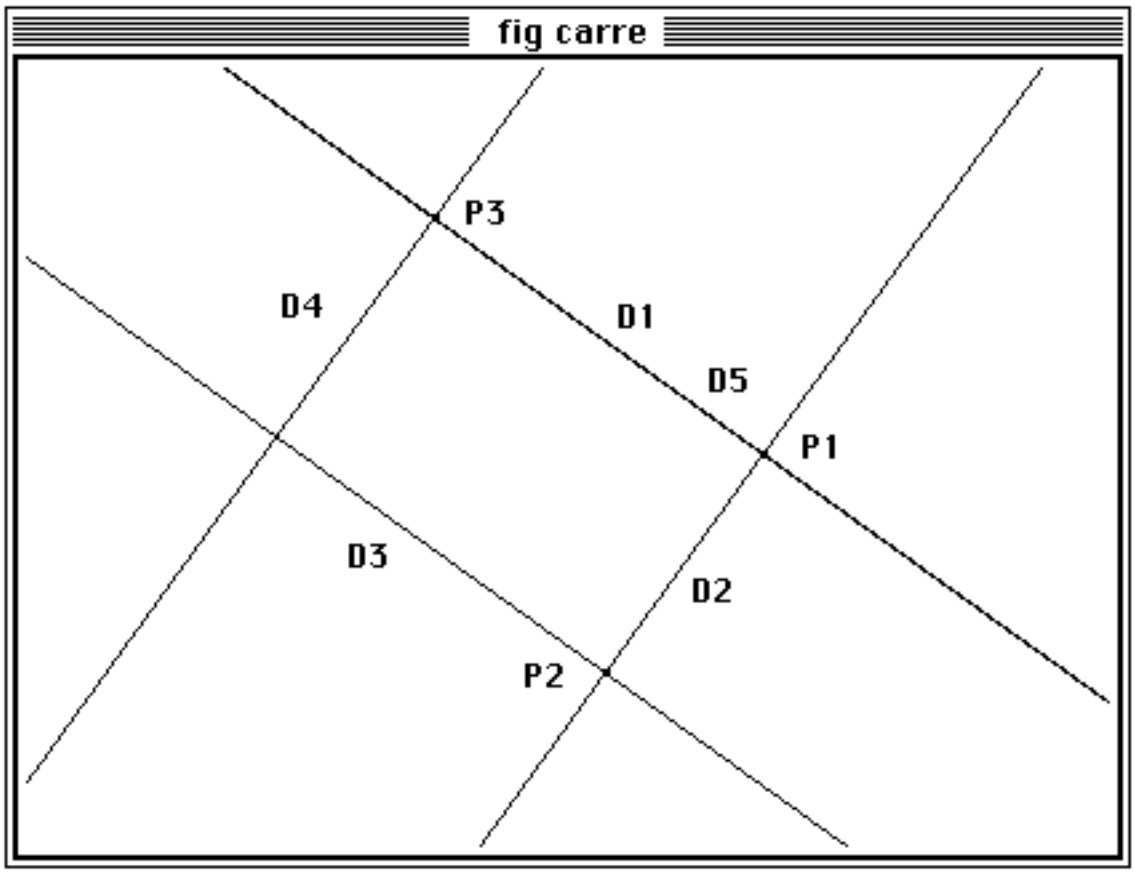
Comme dans Cabri-géomètre, la dernière commande effectuée est actionnable par le raccourci clavier ⌘E. De même les interactions de Cabri-géomètre (messages d'erreur, aide, abandon, …) sont reproduites par TALC-Élève.

Par contre, à la différence de Cabri-géomètre, un message est affiché si le dernier objet construit est égal à un de ceux précédemment construits, en prenant en compte la théorie TIG définie. L'objet est alors supprimé par TALC-Élève de la représentation graphique.

Exemple :

Si l'élève construit la figure ci-dessous comme correspondant à la spécification décrite dans la fenêtre d'énoncé suivante :





alors TALC fournit le diagnostic suivant :



et l'objet D1 est supprimé de la représentation graphique.

5) Nommer des objets : fonctionnalité identique à celle Cabri-géomètre. Cette fonctionnalité permet de donner des noms aux objets construits. Elle est essentielle pour que l'élève puisse répondre au contrat.

6) Supprimer un objet : fonctionnalité identique à celle Cabri-géomètre. Supprime un objet et ceux qui en dépendent s'il y en a.

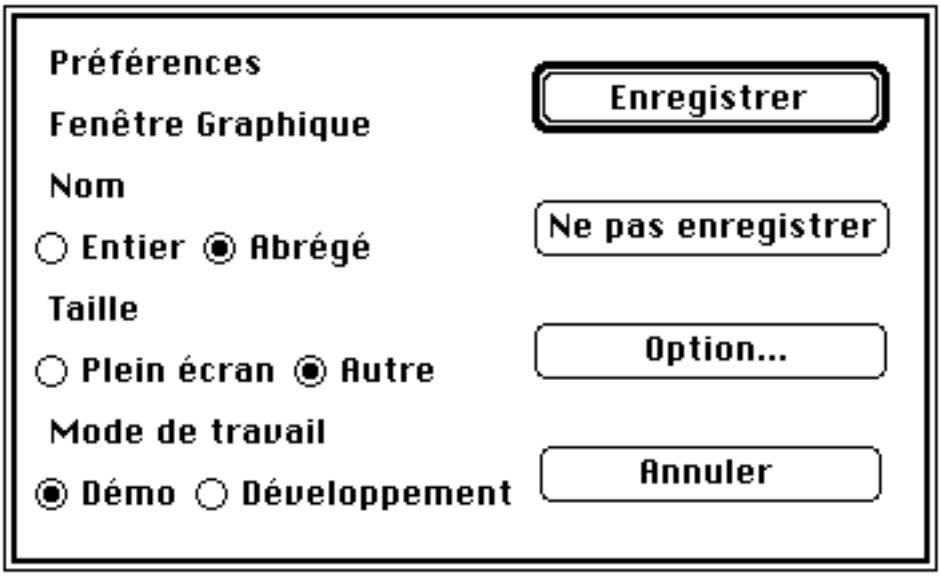
7) Bouger : cette fonctionnalité est transparente (comme dans Cabri-géomètre). Elle permet à l'utilisateur de modifier à tout moment la position d'un objet. Cette modification entraîne la modification des positions de tous les objets dépendants de cet objet, en respectant les relations géométriques de la construction.

8) Vérifier une construction. C'est la fonctionnalité principale de TALC-Élève. Quand l'élève estime que sa construction correspond à la spécification du professeur, il demande à TALC-Élève de diagnostiquer la correction.

Quatre cas principaux se présentent :
- la construction est correcte.
- il manque un ou plusieurs objets à la construction. Cela peut venir d'un manque réel ou d'un nommage inadéquat.
- il manque une ou plusieurs propriétés à la construction.
- des propriétés rendent la figure particulière.

Pour chacun de ces cas, le diagnostique est affiché et le cas échéant de plus amples détails peuvent être demandés.

9) Définir des préférences. Ces préférences sont définies par l'intermédiaire de la boîte de dialogue suivante :



Elles concernent actuellement principalement le positionnement de la fenêtre de construction.

3.1.2. Fonctionnalités transparentes nécessaires.

La mise en œuvre de TALC nécessite aussi de résoudre les problèmes suivants :

1) rafraîchissement de la fenêtre de construction.
PrologII+ propose un mécanisme simple pour le rafraîchissement des fenêtre graphiques. Dès qu'une modification de fenêtrage est intervenue, PrologII+ appelle une règle. Cette règle peut être définie par le programmeur. Dans notre cas, il suffit de demander à Cabri-géomètre de redessiner la construction par une requête Rafraichir.

2) Récupération d'erreurs.
La communication avec Cabri-géomètre entraîne des erreurs (par exemple si Cabri-géomètre n'est a inopinément terminé). Ces erreurs sont récupérées et l'utilisateur est informé du problème.

3) Activation des menus.

Suivant l'état de la construction courante, certaines opérations n'ont pas de sens.
Par exemple si aucun exercice n'a été choisi, la visualisation de la théorie TIG courante n'a pas de sens. Ou encore la construction d'une perpendiculaire n'a pas de sens si aucune droite n'a encore été construite.

En conséquence, les menus correspondant à ces opérations dénués de sens ont été mis hors fonction (concrètement, l'utilisateur les voit grisés et leur utilisation n'a aucun effet).

De même, lorsque TALC est en train d'effectuer des calculs (recherche d'égalités après une construction, substitution après un renommage, vérification de correction,...), aucune action de l'utilisateur ne doit se produire avant la fin du calcul. Toutes les actions disponibles sont alors verrouillées et un curseur (représentant une montre dont l'aiguille tourne) est affiché signifiant à l'utilisateur d'attendre.

3.2. Mise en œuvre de la communication avec Cabri-géomètre.

Pour que TALC-Élève puisse utiliser les fonctionnalités de Cabri-géomètre en tant que client de cette application, il faut disposer d'un moyen de communication entre les deux applications.
La mise en œuvre de cette communication est présentée dans l'annexe G.

3.3. Mise en œuvre de la traduction de SCL.

Afin de présenter la mise en œuvre de la traduction de SCL, nous commençons par expliquer la façon dont le programme représente et manipule les faits. Ensuite nous présentons la mise en œuvre de la traduction simple d'une phrase d'énoncé de Cabri-géomètre pour aboutir à la présentation de la mise en œuvre de la traduction complète d'un énoncé. Enfin, nous présentons la façon dont nous avons mis en œuvre les deux fonctionnalités entrainant une modification des faits (en dehors des primitives de construction) : les primitives Nommer et Supprimer (voir annexe G).

3.3.1. Représentation et manipulation des faits

Du fait de l'interactivité de SCL, les commandes du langage sont asynchrones. Nous utilisons les capacités de PrologII+ pour construire les menus activant ces commandes : PrologII+ associe à chaque item de menu une règle prolog. De ce fait, notre programme ne contrôle pas lui-même l'enchaînement des commandes et il n'est pas possible de passer en paramètre des commandes la traduction courante de la construction.
La seule possibilité est de stocker la traduction courante globalement. La solution que nous avons retenue est de représenter la traduction par des règles Prolog de la forme :

```
fait(_litteral,eleve) -> ;
```

Chacun de ces faits représente un des littéraux de la formule LDL traduction de la construction. L'utilisation des primitives `fassert` et `fretract` permet des ajouts et des retraits rapides de ces faits.

3.3.2. Mise en œuvre de la traduction simple d'une phrase d'énoncé.

Nous avons défini la traduction d'une phrase d'un énoncé de Cabri-géomètre en traduisant chaque phrase d'énoncé en CDL (voir chapitre second, 2.3.2.2). Nous avons

noté qu'une construction peut être vue comme un cas particulier de spécification. En effet, les propriétés suivantes sont garanties sur une phrase d'énoncé :

- tout objet possède un nom, sauf les segments.
- le type d'un identificateur est toujours explicite.
- tout objet, différent de l'objet construit par la primitive produisant une phrase d'énoncé, a été construit précédemment et est donc défini par une phrase précédente de l'énoncé.

Du fait de ces propriétés, la mise en œuvre des fonctions de composition utilisées pour définir la sémantique de CDL peut être simplifiée par rapport à la mise en œuvre de la traduction de CDL.

La mise en œuvre de la fonction NOM est immédiate par traduction des identificateurs de Cabri-géomètre en identificateurs LDL, sauf pour les segments. Pour les segments, il suffit de retrouver dans la traduction courante l'atome de déclaration dont les extrémités s'unifient avec les extrémités du segment recherché.

Comme le type d'un identificateur est toujours connu dans la phrase d'énoncé, la fonction TYPE n'a pas d'utilité.

La fonction TYPAGE est mise en œuvre par une recherche sur les atomes de typage de la traduction courante : il suffit que l'identificateur de l'objet défini par l'atome de typage s'unifie avec l'identificateur recherché. Le même principe est utilisé pour retrouver le support d'un segment, sa première et sa seconde extrémité, le centre et le rayon d'un cercle.

Notons que du fait que SCL ne manipule pas de demi-droites, la fonction ORIGINE est inutile. De même les fonctions PREDAPP (le type de l'objet sur lequel porte l'appartenance est toujours connu dans la phrase d'énoncé) et NOT (il n'existe pas de construction négative).

La mise en œuvre de la traduction d'une phrase d'un énoncé de Cabri-géomètre est alors réalisée par la traduction de la spécification CDL qui la traduit en utilisant la mise en œuvre des fonctions de composition que nous venons de définir.

Exemple :
Reprenons une ligne du tableau donné en 2.3.2.2.

Phrase du langage d'énoncés	traduction en CDL
d : droite passant par p1 et parallèle au segment [p2 p3]	d !- [p2 p3], p1 e d.

Pour traduire cette phrase, il nous faut retrouver la droite support du segment [p2 p3] par l'intermédiaire de la fonction SUPPORT, le prédicat de typage des points p1, p2 et p3 et du segment [p1 p2] et de cette droite support. La traduction obtenue est la suivante, en supposant que le segment [p2 p3] soit représenté par l'identificateur var1 et sa droite support par var2 :

```
droite(ad), segment(var1,ap1,ap2,var2) point(ap2) point(ap2)
perp(ad,var2) point(ap1) appdr(ap1,ad).
```

3.3.3. Mise en œuvre de la traduction complète d'un énoncé.

La traduction complète d'un énoncé est obtenue incrémentalement à chaque pas de construction de l'élève. Chaque commande de dessin de l'élève est réalisée par l'appel d'une requête à Cabri-géomètre de type C3 (voir annexe G).

Pour chaque requête de dessin, un énoncé est fourni. Il représente les objets construits par cette requête. Cet énoncé comporte un ensemble de phrases d'énoncé. Chacune de ces phrases est traduite de la façon suivante :

• une traduction simple de la phrase est obtenue comme défini plus haut.

• la traduction de la phrase est ajoutée à la traduction courante en ajoutant chacun des littéraux de la traduction s'il n'est pas déjà présent dans la traduction courante.

• pour chacun des objets ajoutés par cette traduction, des égalités sont recherchées avec un des objets précédemment construits et de même type, par rapport à la théorie TIG. Cette recherche d'égalité fait appel au même démonstrateur que pour la réduction des égalités de CDL, à ceci près qu'il n'intègre pas l'hypothèse de noms uniques, pour des raisons que nous avons développées au chapitre deuxième, 2.3.2.3. Si une égalité est prouvée, la traduction est modifiée de la façon suivante:

- si l'objet ajouté est implicite, l'objet auquel il est égal lui est substitué et les littéraux doublons d'un autre littéral sont supprimés.

- si l'objet ajouté est explicite, cela signifie que l'élève vient de construire un objet égal à un autre objet déjà construit. Tous les objets de l'énoncé rendu sont supprimés et l'élève est prévenu que la construction n'a pas été effectuée car un objet identique existe déjà (voir plus haut).

3.3.4. Fonctionnalités modifiant la traduction autres que les primitives de construction .

La traduction est modifiée par les deux primitives suivantes :

• Nommer des objets.

Cette requête à Cabri-géomètre ne rend aucune information (si ce n'est un code d'erreur). Les changements de noms sont seulement visibles sur la fenêtre de construction. Il faut donc à la suite de chaque requête Nommer faire une requête Énoncé pour pouvoir effectuer les substitutions adéquates dans la traduction.

Un problème surgit du fait de la non-atomicité du Nommer (plusieurs objets peuvent être nommés en utilisant une seule requête). Cette non-atomicité permet à l'utilisateur un renommage circulaire.

Exemple :

L'interaction suivante avec Cabri-géomètre :

donner le nom x à C

donner le nom C à B

donner le nom B à A

donner le nom A à x.

produit du point de vue de TALC le renommage A -> B, B-> C, C -> A.

Pour résoudre ce problème, deux solutions sont possibles :

- disposer d'un Nommer atomique. Cela est du ressort des developpeurs de Cabri-géomètre.

- effectuer le renommage en deux temps, en utilisant des noms intermédiaires.

Par exemple, A -> A', B -> B', C -> C' puis A' -> B, B' -> C, C' -> A.

C'est la solution adoptée pour l'instant en attentant de disposer d'un Nommer atomique.

• Supprimer un objet

Cette requête à Cabri-géomètre rend le nom de l'objet supprimé sur la fenêtre de construction et indique si des objets dépendants ont été supprimés.

Pour rendre la traduction cohérente avec la construction courante, on procède ainsi :

- l'ensemble des objets supprimés dépendants de l'objet supprimé est reconstitué par comparaison entre l'ancien et le nouvel énoncé.

- chacun des objets de cet ensemble est supprimé de la traduction. Cela signifie que tous les littéraux de la traduction faisant référence à un objet supprimé sont enlevés.

Le point délicat est la suppression des objets implicites (créés précédemment par la traduction). En effet, on ne peut effectuer leur suppression pure et simple comme pour les objets explicites, car il se peut qu'ils interviennent dans la traduction d'un objet non supprimé.

Exemple :

Si l'élève a construit un segment [A B], puis un point C sur ce segment et enfin le segment [C B], la traduction est la suivante

```
point(AA)
point(AB)
point(AC)
segment(VarS1,AA,AB,var1)
segment(VarS2,AB,AC,var1)
droite(var1)
as(AC,VarS1).
```

Dans cette traduction, la droite support de [A B] et de [C B] est la même, car TIG permet de déduire cette égalité : la substitution a été effectuée par la traduction. La supprimer en supprimant [B C] rendrait la traduction non conforme à sa spécification.

Il faut donc vérifier que l'objet implicite à supprimer n'est pas en relation avec un objet restant de la construction avant de le supprimer. Concrètement, cela signifie qu'il faut vérifier que l'identificateur de l'objet n'est pas le paramètre d'un atome de la traduction différent de l'atome de typage le définissant.

Sur notre exemple, la suppression du point C amène la suppression du segment [C B], mais pas de la droite var1, ce qui donne la traduction suivante après suppression

```
point(AA)
point(AB)
segment(VarS1,AA,AB,var1)
droite(var1).
```

4. Mise en œuvre de la vérification de la correction.

Disposant de la traduction en LDL d'une spécification, des théories TIG et TEXT qui y sont attachées, TALC fournit un diagnostic de la correction de la construction de l'élève à sa demande, quand il estime remplir le contrat, à partir de la traduction de la construction courante.

Rappelons que le contrat est vérifié si il existe une substitution σ et un élément S* de la multifonctions FE(σ(S),F,TIG,TEXT) tel que

TIG, HYPNU / F σ S*

Si ce n'est pas le cas, TALC-Élève doit fournir un diagnostic le plus précis des causes de non-respect du contrat, pour éventuellement proposer des remèdes ou des tâches permettant à l'élève de rectifier son point de vue.

La vérification de la correction de la construction est mise en œuvre en trois étapes que nous présentons dans ce chapitre:

• une première étape consiste à construire l'ensemble des substitutions σ portant sur les objets non-dénomés de S telles que

TIG |- F σ σ(S)

avec F la traduction de la construction

S la traduction de la spécification.

Si cet ensemble est non vide, cela signifie qu'il existe une correspondance entre les noms des objets de la construction et ceux de la spécification telle que la construction respecte au moins toutes les propriétés de la spécification.

• une seconde étape consiste à construire la multifonctions FE(σ(S), F,TIG,TEXT), c'est à dire l'ensemble des extensions de la spécification par rapport à l'ensemble d'axiomes TEXT. Chacune de ces extensions doit ajouter à σ(S) chacun des objets non-spécifiés de F sans lui ajouter de propriétés supplémentaires.

• une dernière étape consiste à vérifier s'il existe un couple <σ,S*> avec σ une substitution de l'ensemble précédemment construit et S*=FE(σ(S),F,TIG,TEXT) une des extensions associées à σ tel que

TIG |- F σ σ(S*)

Si c'est le cas, la construction correspond bien au contrat, sinon deux principaux cas se présentent :

- il n'existe pas de substitution σ telle que

TIG |- F σ σ(S)

Dans ce cas :

+ soit l'élève a donné aux objets des noms différents de ceux demandés
+ soit il n'a pas construit tous les objets demandés,
+ soit sa construction ne permet pas de déduire toutes les propriétés requises.

Un diagnostic lui est fourni précisant de quel cas relève sa construction (voir § 5).

- la construction présente des propriétés particulières. Cela signifie qu'aucun couple <σ,S*> ne satisfait TIG |- F σ σ(S*) car une propriété de F ne peut être prouvée dans S*, soit qu'elle n'ait pu être ajoutée par aucune extension S* à partir de TEXT, soit simplement qu'elle constitue une propriété particulière sur des objets spécifiés.

Pour favoriser le cas où la correction est obtenue, la mise en œuvre ne produit pas séquentiellement toutes les substitutions possibles puis toutes les extensions possibles pour finalement chercher un couple <subsitution,extension> qui vérifie la correction. La mise en œuvre utilise l'indéterminisme de Prolog pour produire une substitution, trouver une extension correspondante puis vérifier que ce couple <subsitution,extension> vérifie la correction. Si le couple ne vérifie pas l'extension, le retour arrière permet d'essayer le couple suivant.

Dans le cas le plus défavorable cette méthode est aussi coûteuse que celle produisant séquentiellement les substitutions et les extensions possibles pour trouver un couple correct. Elle n'a donc que des avantages.

La vérification est mise en œuvre par la règle Prolog suivante :

```
verifier_correction(S) ->
      suffisance(S)
      une_extension(S,S*)
      necessite(S*) ! ;
```

où implicitement :

- les preuves sont faites dans la théorie TIG de l'exercice et par rapport à la traduction F de la construction, toutes deux accessibles par des règles.
- une extension est construite à l'aide de la théorie TEXT de l'exercice.

La représentation explicite de la substitution est inutile comme nous l'expliquons ci-dessous.

Nous présentons dans ce sous-chapître chacun des paquets de règles définissant ces trois prédicats.

4.1. Vérification que la construction satisfait toutes les hypothèses de la spécification.

Vérifier que la construction satisfait toutes les hypothèses de la spécification consiste à trouver une substitution sur les objets non-dénommés de S telle que `TIG “ F □ □(S)`.

Elle est réalisée par la règle prolog suivante :

```
suffisance(S) ->
      non_nommés_vers_variables(S,S1)
      liste_objets(S1,L)
      tous_dif(L) !
      preuve(F,S1);
```

non_nommés_vers_variables(S,S1) est satisfait si S1 est la formule S où chaque identificateur non-nommé est remplacé par une variable prolog différente.

liste_objets(S,L) est satisfait si L est la liste d'objets de la formule S

tous_dif(L) est satisfait si la contrainte dif(x,y) a été posée sur tout couple <x,y> de la liste L

preuve(S) est satisfait si chacun des littéraux de S est prouvable à partir des faits de F et de la théorie TIG. Les règles définissant preuve sont les suivantes :

```
preuve([]) -> ;
preuve([_atome|S]) ->
      prouver_atome(_atome, élève)
      preuve(S);
```

La règle prouver_atome est le point d'entrée du démonstrateur. La version du démonstrateur utilisée est celle pour laquelle les faits et les axiomes sont fixes. De ce fait, elle permet de ne jamais remettre en cause ni lemmes ni d'anti-lemmes entre chaque démonstration d'un fait de S, ce qui lui donne une très bonne efficacité.

La substitution recherchée n'est pas représentée explicitement. Elle est représentée implicitement par les instanciations effectuées sur les variables de la formule S par le démonstrateur. Comme le démonstrateur instancie toutes les variables, alors si chaque atome de la formule S est vérifié cela signifie qu'il est aussi instancié. La mise en œuvre assure ainsi qu'une substitution est définie pour tout objet non-identifié de la spécification.

Il faut noter enfin que puisque la traduction de la spécification ne contient pas d'objets égaux selon la théorie, la substitution trouvée ne doit pas introduire d'égalité entre les objets non-identifiés. C'est la raison pour laquelle il est nécessaire de poser la contrainte que les variables représentant des objets non-identifiés soient différentes deux à deux. Cette contrainte est posée par la règle tous_dif.

4.2. Mise en œuvre de la fonction d'extension.

Pour construire une extension S*=FE(□(S),F,TIG,TEXT), la mise en œuvre utilise le paquet de règles suivant :

```
une_extension(S*,S*) ->
      liste_d_objets(S*,[]) ! ;

une_extension(S,S*) ->
      liste_d_objets(S,L)
      un_pas_d_extension(S,S1,L)
      une_extension(S1,S*) ;
```

où liste_d_objets(S,L) est satisfait si L est la liste des objets de F non présents dans S (F est disponible globalement).

La première règle permet d'arrêter la construction d'une extension si aucun objet de F ne manque à S*.

La règle un_pas_d_extension est ainsi définie :

```
un_pas_d_extension(S,S1,L) ->
      choix_d_objet(L,o)
      propriétés_liéés(S,o,r)
      choix_d_axiome(o,r,a)
      prouver_cond(S,a)
      ajout_propmin(a,S,S1);
```

où choix_d_objet(L,o) est satisfait si o est un des objets choisi dans la liste L;

propriétés_liéés(S, o,r) est satisfait si r est l'ensemble composé de l'atome de typage de o dans F et des atomes de F dont o est un paramètre et tous les autres paramètres sont des objets de S.

choix_d_axiome(o,r,a) est satisfait si a est un axiome choisi en fonction de o et de r;

prouver_cond(a,S) est satisfait si la condition de l'axiome a est vérifiée par rapport à TIG et S;

ajout_propmin(a,S,S1) est satisfait si S1 = S @ Propmin, où Propmin est l'ensemble des atomes ajoutés par l'axiome a.

Expliquons plus en détail certaines de ces règles.

• choix_d_objet(L,o).
Le choix d'un objet est effectué en utilisant la primitive member. D'autres solutions sont envisageables, comme par exemple de choisir un objet en fonction de son ordre d'introduction par l'élève.

• choix_d_axiome(o,r,a).
Pour rendre ce choix efficace, la mise en œuvre filtre les axiomes candidats au choix de deux façons :
- en ne sélectionnant que des axiomes construisant un objet du type de l'objet o.
- parmi ces derniers, en ne sélectionnant que des axiomes ajoutant un ensemble de propriétés Propmin déductibles de F. En effet, comme il faut que toute propriété ajoutée soit déductible de F (du fait de la condition (2), voir deuxième chapitre, 4.1.2.3.1), alors seuls ces axiomes sont potentiellement applicables.
Mais, pour des raisons d'efficacité a priori, notre mise en œuvre ne correspond pas exactement à la définition de la fonction d'extension. Au lieu de vérifier si Propmin est déductible de F, nous avons choisi dans un premier temps de seulement vérifier que tous les atomes de Propmin appartiennent à F. De ce fait, le filtrage des axiomes ne nécessite pas de tenir compte de F en entier, mais de la partie de F dans laquelle Propmin peut être incluse. C'est cette partie de F que définit r. En effet, comme l'axiome définit la construction de l'objet o, inévitablement tout atome de Propmin a o comme paramètre. De ce fait, il n'est pas nécessaire de chercher si un atome de Propmin est contenu dans l'ensemble des atomes de F sans r.

De plus, pour trouver plus rapidement une extension satisfaisante, la mise en œuvre choisit en premier les axiomes construisant des objets de degré de liberté nul puis des objets de degré de liberté 1 et enfin des objets de degré de liberté 2. Cette méthode favorise la recherche d'une extension adéquate dans le cas où les objets non-spécifiés sont en relation avec les objets spécifiés.

• prouver_cond(a,S) utilise la version du démonstrateur permettant la modification dynamique des faits, permettant ainsi des retours-arrière corrects sur cette règle.

4.3. Vérification de la nécessité d'une construction.

Une fois les deux étapes précédentes franchies avec succès, s'assurer que la construction ne satisfait que les hypothèse de la spécification consiste à vérifier que, avec la substitution et l'extension courante, chacun des littéraux de la formule F est bien déductible de (S*). Cette vérification est effectuée par la règle `necessite`(S*).

Si ce n'est pas le cas, Prolog effectue des retours-arrière sur les choix restant possible pour les règles `suffisance` et `une_extension` pour trouver un couple <substitution, extension> qui vérifie cette exigence.
Le démonstrateur utilisé est pour cette raison ici aussi celui permettant la modification dynamique des faits.

5. Diagnostics en vue d'une explication.

Le but ultime de TALC est de pouvoir proposer à l'élève des explications sur la construction qu'il a produite. Ces explications doivent lui permettre de remédier à ses erreurs ou de réviser certaines de ses conceptions.

Notre point de vue est que la définition de telles explications n'est pas du ressort de l'informaticien mais du didacticien. Par contre, le travail de l'informaticien est de fournir au didacticien les moyens de produire de telles explications.

A partir de ce point de vue, nous reprenons la classification de Chandrasekaran [Chandrasekaran et al. 89]. Il considère deux façons d'expliquer une décision vis-à-vis de connaissances :

- par "introspection". Cela consiste à retrouver comment la décision a été prise et les connaissances utilisées à cet effet.
- par "concoction". Cela consiste à produire une justification indépendante de la façon dont la décision a été prise, en utilisant des capacités extérieures à celles utilisées pour prendre la décision.

Nous appelons l'explication produite par introspection le diagnostic. Cela signifie que nous prenons le terme diagnostic dans le sens d'un objet et non d'une action.

Dans ce chapitre, nous présentons en premier lieu les diagnostics fournis par TALC et la manière dont ils sont construits. Nous définissons ensuite des propositions d'explications "concoctées" dont la mise en œuvre n'a pas été réalisée.

5.1. Diagnostics.

Dans le chapitre précédent, nous avons dégagé les trois situations possibles détectées par TALC :

a) La construction ne satisfait pas toutes les hypothèses de la spécification. En d'autres termes, il manque des éléments à la construction vis-à-vis du contrat.

b) La construction satisfait toutes les hypothèses de la spécification et uniquement ces hypothèses. Dans ce cas le contrat est rempli.

c) La construction satisfait toutes les hypothèses de la spécification mais certaines propriétés de la construction ne sont pas déductibles de la spécification. En d'autres termes, la construction représente un cas particulier de la spécification.

Nous présentons dans un premier temps la façon dont est construit le diagnostic qui indique à quelle situation correspond la construction de l'élève. Dans un deuxième temps, nous présentons un exemple de spécification et des constructions correspondant à chacun des diagnostics possibles. Dans un troisième temps, nous définissons un diagnostic plus fin pour les cas où la construction ne remplit pas le contrat et nous présentons sa mise en œuvre.

5.1.1. Construction du diagnostic.

Rappelons que la vérification est mise en œuvre par la règle Prolog suivante :

```
verifier_correction(S) ->
      suffisance(S)
      une_extension(S,S*)
      necessite(S*) ! ;
```

La situation b) correspond au cas où la règle `verifier_correction` est satisfaite.

La situation c) correspond au cas où la règle `verifier_correction` n'est pas satisfaite mais la règle `suffisance` est satisfaite au moins une fois, c'est à dire qu'il existe une substitution □ telle que `TIG “ F □ □(S)`.

La situation a) correspond au cas où la règle `suffisance` n'est jamais satisfaite. En effet, la règle `une_extension` est toujours satisfaite car la multifonction d'extension n'est jamais un ensemble vide (il suffit de définir tous les objets non-spécifiés avec une construction de degré de liberté 2).

Pour déterminer à quelle situation correspond la construction de l'élève, le programme procède de la façon suivante :

- dans le cas de la situation b), la détermination ne pose aucun problème.
- la mise en œuvre de la règle `verifier_correction` présentée ne permet pas de distinguer les cas a) et c), tous les deux se caractérisant par un échec de cette règle, du fait de l'utilisation du retour-arrière de Prolog.

Une première possibilité pour lever l'indétermination consiste à essayer à nouveau la règle `suffisance`. Si elle réussit, c'est que l'échec provient de la règle `nécessite` (situation c)) , sinon la construction correspond à la situation a).

Une seconde possibilité consiste à modifier la mémoire globale par l'ajout d'une règle de marquage dès que la règle `nécessite` réussit une fois. Si la règle `verifier_correction` échoue, il suffit alors de tester si cette règle existe pour lever l'indétermination. Cette solution est bien plus efficace. Bien entendu, la règle éventuellement ajoutée doit être enlevée avant tout nouvelle vérification.

La vérification est ainsi mise en œuvre de la façon suivante :

```
verifier_correction(S) ->
      suffisance(S)
      marquage_suffisance
      une_extension(S,S*)
      necessite(S*) !
      diagnostic("CORRECT");

verifier_correction(S) ->
      marque_suffisance !
      retract(marque_suffisance,[])
      diagnostic("CONSTRUCTION PARTICULIERE");

verifier_correction(S) ->
      diagnostic("CONSTRUCTION INSUFFISANTE");

marquage_suffisance ->
      marque_suffisance !;
marquage_suffisance ->
      assert(marque_suffisance,[]) ;
```

5.1.2. Exemple illustrant les diagnostics possibles.

Soit la spécification suivante, définissant un triangle (A B C) isocèle en A :

```
s1=[A B], s2=[B C],  s3=[C□A], |A□B|=|A□C|.
```

On considère que TIG contient les axiomes suivants :

```
12.1.0;  "Un point appartient à un cercle de centre o et de rayon r\
 s'il est à une distance r de o"
appcc(p,c) ->
   cercle(c,o,r)
   distancep(r,o,p) ;
```

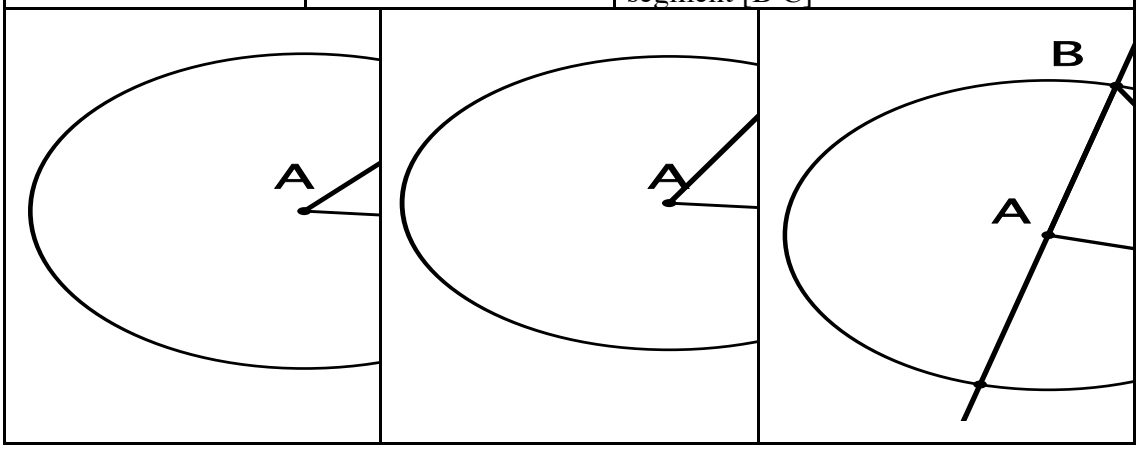
```
17.1.0;   "La propriété distancep est symétrique"
distancep(r,o,p) ->
   distancep(r,p,o);

17.2.0;   "Un point est à une distance r d'un point o\
 s'il appartient au cercle de rayon r et de centre o"
distancep(r,o,p) ->
   cercle(c,o,r)
   appcc(p,c) ;
```

L'ensemble d'axiomes de TEXT est celui présenté en Annexe B. En particulier, il permet d'étendre une spécification avec un cercle de centre et de rayon existants et avec une droite perpendiculaire à une droite existante passant par un point existant.

Le tableau ci-dessous contient trois énoncé de constructions correspondant à chacun des trois diagnostics possibles. Les figures correspondantes sont représentées sous chacun des énoncés.

Construction 1.	**Construction 2**	**Construction 3**
La figure comporte 7 objets. A : point quelconque C : point quelconque segment [A C] B : point quelconque C#1 : cercle de centre A passant par B segment [A B] segment [B C]	La figure comporte 7 objets. B : point quelconque A : point quelconque segment [B A] C#1 : cercle de centre A passant par B C : point du cercle C#1 segment [B C] segment [C A]	La figure comporte 9 objets. A : point quelconque C : point quelconque segment [A C] D#1 : droite passant par A et perpendiculaire au segment [A C] C#1 : cercle de centre A passant par C B : intersection de la droite D#1 et du cercle C#1 P#4 : intersection de la droite D#1 et du cercle C#1 (B est l'autre point) segment [B A] segment [B C]

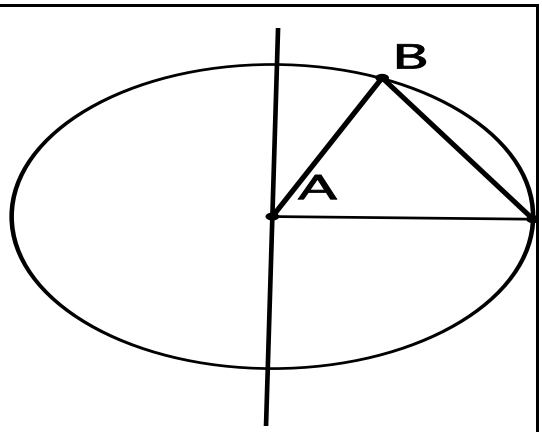


La construction 1 ne satisfait pas toutes les hypothèses de la spécification. En effet, on ne peut prouver que |A B|=|A C|.

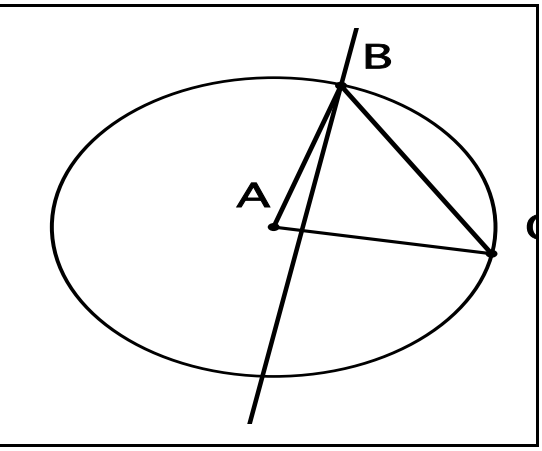
La construction 2 est correcte. Une seule extension de la spécification existe vis-à-vis de cette construction. Elle consiste à lui ajouter le cercle de centre A et de rayon r, où r=|A B|=|A C|. Il faut alors prouver que le fait que B et C appartiennent au cercle de la construction est déductible de cette spécification étendue, ce qui est le cas vu les axiomes de TIG.

La construction 3 comporte une propriété particulière. Trois extensions sont possibles pour ajouter à la spécification la droite passant par A et B et perpendiculaire à [A C] dans la construction, avec un degré de liberté nul :

- celle construisant la droite passant par A et B. Dans ce cas la propriété particulière est la perpendicularité de (A B) et [A C].
- celle construisant la droite perpendiculaire à [A C] et passant par A. Dans ce cas, la propriété particulière est l'appartenance de B à cette perpendiculaire. Une construction correcte serait la suivante :

La figure comporte 8 objets. A : point quelconque C : point quelconque segment [A C] D#1 : droite passant par A et perpendiculaire au segment [A C] C#1 : cercle de centre A passant par C B : point du cercle C#1 segment [A B] segment [C B]	

- celle construisant la droite perpendiculaire à [A C] et passant par B. Dans ce cas, la propriété particulière est l'appartenance de A à cette perpendiculaire. Une construction correcte serait la suivante :

La figure comporte 8 objets. A : point quelconque C : point quelconque segment [A C] C#1 : cercle de centre A passant par C B : point du cercle C#1 D#1 : droite passant par B et perpendiculaire au segment [A C] segment [A B] segment [B C]	

NB : nous n'avons pas considéré les extensions utilisant des constructions ayant un degré de liberté non nul, qui ne permettent évidemment pas non plus de montrer que la construction est correcte.

5.1.3. Affinement du diagnostic.

Pour affiner le diagnostic dans les situations où la construction ne répond pas au contrat (situations a) et c)), TALC propose un complément d'information basé sur des informations recueillies lors de la détermination du diagnostic. Le principe général en est simple : ce complément d'information précise à l'élève une propriété qui rend la construction non conforme à la spécification.

5.1.3.1. Diagnostic dans le cas où des propriétés manquent.

Dans le cas où la construction ne satisfait pas toutes les propriétés de la spécification, la propriété désignée est le premier littéral de la spécification (dans l'ordre arbitraire de la traduction) qui n'a pu être prouvée quelles que soient les substitutions envisagées.

Dans l'exemple de la construction 1, la propriété ainsi désignée est l'égalité entre les distances |A B| et |A C|.

Dans la pratique, la mise en œuvre actuelle ne donne pas exactement ce diagnostic. En effet, TALC permet de préciser le premier littéral de la traduction de la spécification qui n'a jamais pu être prouvé et non la partie de la spécification en cause. Dans notre exemple, il s'agit du fait que la distance r=|A B|=|A C| de la spécification n'est pas celle reliant les deux points A et B (ou A et C suivant l'ordre des littéraux de la traduction) dans la construction, traduit respectivement par les littéraux `distancep(r,A,B)` et `distancep(r,A,C)`. Pour l'instant un diagnostic reprenant les termes exacts de la spécification n'est donc pas fourni. Le construire à partir du littéral détecté constitue un problème en soi. On note que le type de diagnostic le plus difficilement compréhensible est celui où la propriété met en cause des objets implicites de la construction.

Exemple :
Si la propriété en cause est la perpendicularité de deux segments, le diagnostic porte sur les droites supports de ces segments et non sur les segments eux-mêmes. Cela est dû au fait que la perpendicularité n'est définie qu'entre deux droites.

Pour remédier à ce problème, on pourrait éviter de mentionner des objets implicites et fournir un diagnostic en fonction des objets de la spécification, ce qui ne semble pas devoir poser de grosses difficultés.

Ce diagnostic pose aussi un problème vis-à-vis des identificateurs de objets. Puisque les identificateurs de la spécification ne sont pas nécessairement ceux de la construction, il faut non seulement exhiber la propriété en cause mais aussi les substitutions pour lesquelles sa preuve a échoué. De cette façon, l'élève peut examiner avec précision toutes les hypothèses de correspondance de identificateurs envisagées par le démonstrateur.

Exemple :
Si la spécification requiert deux droites perpendiculaires et que la construction de l'élève comporte trois droites quelconques, le diagnostic précise que la perpendicularité manquant peut porter sur un des trois couples de droites possibles, la droite hors du couple étant considérée comme inutile.

La construction de ce diagnostic est réalisée en numérotant les littéraux de la traduction de la spécification par ordre croissant. A chaque preuve d'un littéral donné, le programme met à jour un entier représenté en mémoire globale (par la primitive `assign`) de façon à ce qu'il représente le numéro du littéral prouvé de rang le plus élevé dans la liste. Conjointement, il enregistre les substitutions correspondant à ce numéro.

Après l'échec final de la preuve de suffisance, le littéral exhibé par le diagnostic est le littéral suivant celui enregistré.

5.1.3.2. Diagnostic dans le cas où la construction contient des propriétés particulières.

Le principe de la construction du diagnostic affiné dans ce cas est le même que précédemment. La propriété désignée est le premier littéral de la construction (dans l'ordre

arbitraire de la traduction) qui n'a pu être prouvée quelles que soient les extensions envisagées.

Comme il n'y a pas de problèmes de correspondance d'identificateurs, le diagnostic consiste à seulement exhiber la propriété en cause.

Notons que la construction de ces deux diagnostics présente certaines similitudes avec la détection d'erreur produite par un compilateur. Il est difficile pour un compilateur de délivrer plus d'une erreur à la fois, car le risque est grand que les erreurs détectées après la première erreur soient une conséquence de celle-ci. Dans ce cas les erreurs signalées risquent de jeter le trouble dans l'esprit de l'utilisateur. C'est la raison pour laquelle de nombreux compilateurs s'arrêtent sur la première erreur. Notre choix actuel tient d'une part d'un raisonnement analogue et d'autre part de l'importante difficulté qu'il y aurait à chercher un diagnostic portant aussi sur les autres propriétés.

5.2. Proposition d'explications.

La mise en œuvre de la production d'explications définies dans ce chapitre n'a pas encore été envisagée. De ce fait, ce que nous proposons ici constitue avant tout une ébauche des principes qui nous semblent pouvoir aider l'élève dans sa tâche. Ce qui nous semble important, c'est que la sémantique précise donnée au contrat didactique nous permet une définition aisée des explications possibles par rapport aux termes du contrat [Desmoulins 93].

En premier lieu, il nous semble que si l'élève propose une construction c'est qu'il estime a priori qu'elle répond au contrat. Si la construction ne correspond pas au contrat, un premier niveau d'explication consiste alors à convaincre l'élève. Il nous semble en effet que le diagnostic n'est pas suffisant pour convaincre l'élève. Le diagnostic informe l'élève du problème que le système a détecté, il ne le convainc pas forcément.

Exemple :
En définissant la construction 1, il se peut que graphiquement l'élève construise le point C sur le cercle. Dans ce cas il ne comprend pas le diagnostic proposé.

Pour convaincre l'élève, nous proposons deux méthodes :

• lui suggérer un contre-exemple pour lequel la propriété incriminée est visiblement absente, dans le cas où sa construction n'est pas suffisante.

Dans l'exemple précédent, il serait possible de modifier automatiquement les coordonnées du point C de façon à ce qu'il n'apparaisse pas sur le cercle. Cette possibilité est déjà une capacité de Cabri-géomètre pour certaines propriétés, ce qui nous permettrait une mise en œuvre rapide.

• lui montrer un exemple de construction répondant au contrat mais que la construction qu'il propose ne recouvre pas. Dans ce cas, le système peut lui demander d'essayer de déplacer les objets de sa construction pour la superposer sur celle proposée. Cette activité lui permettra de prouver par lui-même, sinon que sa construction ne répond pas au contrat, du moins qu'il ne peut donner la preuve qu'elle est suffisamment générale.

En second lieu, nous estimons que l'explication doit permettre à l'élève de donner une nouvelle construction correspondant mieux au contrat que la précédente. Dans ce but, nous proposons deux formes possibles d'explications :

(1) une explication ayant la forme :

"A condition de supposer que ..., la construction proposée est correcte".

Cette explication est basée sur l'hypothèse que les productions de l'élève sont cohérentes au regard de ses conceptions sur la matière enseignée. En lui désignant parmi ces conceptions celles qui sont erronées par rapport au contrat, le système lui permet d'es-

sayer de trouver une autre construction ne s'appuyant pas sur cette conception. Cette méthode rejoint celle de DEBUGGY [Burton 82].

(2) une explication montrant une construction correcte la plus proche possible de celle de l'élève. De cette façon, le système lui fournit une représentation concrète d'une solution sans nécessairement lui indiquer la manière de la produire. En effet, il est reconnu que proposer systématiquement une solution toute faite à l'élève ne lui permet pas toujours de progresser : il fournit une solution presque au hasard et attend que le système le guide. Notons que ce dernier point illustre clairement que si l'on peut réclamer à l'informaticien un algorithme pour pouvoir exhiber une telle construction, il n'est pas certain que du point de vue du didacticien elle soit toujours intéressante.

Par rapport à la formulation du contrat didactique que nous avons donnée, la première forme d'explication requiert la détermination d'une théorie TIG' représentant les conceptions de l'élève et telle que sa construction satisfasse la spécification. Dans ce cas évidemment cette théorie TIG' n'est pas cohérente avec la géométrie.

Exemple :

Pour que la construction 1 soit satisfaite, TIG' peut comporter l'"axiome" suivant :

`pres(p,c)` □ `appcc(p,c)`

où `pres(p,c)` signifie que le point `p` est situé "près" du cercle `c` avec une précision donnée.

Pour construire une explication de la forme (2), nous proposons de chercher à renforcer (dans le cas où la construction n'est pas suffisante) ou a relâcher (dans le cas où la construction comporte une propriété particulière) les relations géométriques liant les objets de la construction de l'élève de façon à obtenir une construction correcte. Autrement dit, il s'agit de sur-contraindre (resp. sous-contraindre) certains objets en modifiant le degré de liberté qu'ils possèdent.

Exemples :

Pour la construction 1, sur-contraindre le point C consiste à redéfinir le point sur le cercle plutôt que sur le plan.

Pour la construction 3, sous-contraindre consiste à redéfinir le point A (ou le point B) comme appartenant seulement au cercle et non à la perpendiculaire.

Concrètement, notre idée pour déterminer une telle construction la plus proche de celle de l'élève et répondant au contrat est d'utiliser les axiomes de TEXT :

- dans le cas où il faut sur-contraindre un objet de la construction, il s'agirait de trouver un axiome de TEXT de degré de liberté plus petit que la primitive utilisée pour la construction ce cet objet.
- dans le cas où il faut sur-contraindre un objet de la construction, il s'agirait de trouver une extension de la figure par rapport à la spécification. Formellement, cela signifie qu'il faut trouver une extension F* telle que

`TIG, HYPNU` ⊢ `F*` □ `S*`

avec □ `F*` □ `FE(F,S*,TIG,TEXT)` où `FE` est la multifonction d'extension.

6. Résultats.

Après avoir présenté la façon dont nous avons mis en œuvre TALC, nous présentons dans ce sous-chapitre les résultats de cette mise en œuvre.

Dans un premier temps, nous présentons quantitativement l'importance de cette mise en œuvre et nous analysons les avantages et les inconvénients rencontrés dans l'utilisation de PrologII+ comme outil de génie logiciel. Dans un second temps, nous présentons une catégorisation des problèmes effectivement traités par le système et des exemples de ces problèmes. Dans un troisième temps, nous présentons les performances du système sur les exemples sur lesquels nous avons effectué une expérimentation.

6.1. Évaluation de la mise en œuvre.

La mise en œuvre que nous avons présentée dans les précédentes parties de ce chapitre utilise deux langages de programmation. La majeure partie utilise PrologII+ [Prologia 92]. La partie permettant la communication client/serveur avec Cabri-géomètre est une extension de PrologII+ que nous avons écrite en THINK C [Borenstein et al. 91], grâce à la collaboration de Sylvie Tessier qui avait déjà réalisé un programme de communication à partir d'une pile Hypercard et qui surtout a réalisé la version serveur de Cabri-géomètre [Tessier et al. 94].

Nous présentons en premier lieu la décomposition modulaire de chacune des applications TALC-Élève et TALC -Enseignant et la taille respective du code produit. En second lieu, nous analysons les avantages et les inconvénients que nous avons rencontrés dans l'utilisation de Prolog comme outil de génie logiciel.

6.1.1. Importance de la mise en œuvre.

La mise en œuvre de TALC-Élève et TALC-Enseignant est décomposée en modules, au sens de PrologII+.

Le tableau suivant précise la taille en nombre de règles de chaque module et leur contenu :

Nom du module	Contenu du module.	Taille en nombre de règles
Théories	Les théories initiales TIG_0 et TEXT.	88
Utilitaires	Les programmes communs à TALC-Élève et à TALC-Enseignant, en particulier les démonstrateurs.	265
Traduction CDL	Le traducteur de CDL en LDL et l'analyse des erreurs.	355
Interface enseignant	L'interface de TALC-Enseignant.	95
Traduction SCL	Le traducteur de SCL en LDL.	126
Interface élève	L'interface de TALC-Elève.	250
Diagnostic correction	Le programme établissant le diagnostic de la correction et construisant la multifonction d'extension.	124

Somme de la taille des modules		1303

Deux des modules sont partagés par les deux applications, les modules "Théories" et "Utilitaires", les autres sont spécifiques à chacune des applications.

Ce qui fait que les deux applications sont décomposées en modules de la façon suivante :

Application	Nom du module	Taille en nombre de règles
TALC-Enseignant.	Théories	88
	Utilitaires	265
	Traduction CDL	355
	Interface enseignant	95
Taille Totale		803
TALC-Élève	Théories	88
	Utilitaires	265
	Traduction SCL	126
	Interface élève	250
	Diagnostic correction	124
Taille Totale		853

Aux 853 règles PrologII+, il faut ajouter 3200 lignes de code en THINK C réalisant le lien entre Prolog et Cabri-géomètre par l'utilisation des Apple-events [Apple 1993].

Les chiffres que nous donnons ici sur les tailles respectives de TALC-Enseignant et TALC-Élève montrent qu'il s'agit de mise en œuvre d'envergure importante.

L'importance de cette mise en œuvre est due en premier lieu à la nécessité de disposer de véritables applications. En effet, comme nous l'avons fait remarquer au premier chapitre, 5.3., la construction d'un EIAO ne peut être validée sur des prototypes trop peu élaborés, du fait d'un biais trop important introduit dans la situation didactique par le manque de finition, qui masque à l'élève les véritables enjeux. La proportion dans la mise en œuvre des questions d'interfaces est à cet égard éloquente : 345/1303 soit un quart de l'application, non compte-tenu de la partie du programme en THINK C.

En second lieu, la mise en œuvre des démonstrateurs et de la construction de la multifonction d'extension qui représente le noyau dur du système, constitue un tiers de la mise en œuvre.

Le reste de l'application est composé des théories et des modules de traduction des langages CDL et SCL en LDL. Il faut noter à ce sujet, qu'un part importante du module de traduction de CDL est consacrée à l'analyse des erreurs dans l'écriture des spécifications, ce qui peut être assimilé à des questions d'interface.

6.1.2. Utilisation de Prolog comme outil de génie logiciel.

Notre choix de PrologII+ comme langage de programmation est basé à l'origine sur les possibilités de mise en œuvre de démonstrateurs en logique du premier ordre qu'il

offre. Notre restriction à des clauses de Horn pour les axiomes de TIG n'a fait que confirmer ce choix. En plus de ces deux avantages de Prolog vis-à-vis de notre problématique, PrologII+ s'est avéré un outil de génie logiciel très adapté et performant.

En premier lieu, une des premières qualités de Prolog est d'être un langage de haut niveau où les programmes sont proches de leur expression naturelle. Cela permet d'exprimer rapidement des prototypes et de ne s'intéresser aux problèmes d'efficacité que dans un second temps. Cela a ainsi permis des mises en œuvre rapides de versions successives de TALC et des comparaisons fructueuses en terme d'efficacité entre ces versions. Prolog permet aussi une mise en œuvre facile d'un programme à plusieurs metaniveaux, dont l'avantage est d'une part de ne pas mélanger les connaissances en géométrie et le contrôle par le démonstrateur et d'autre part de représenter les diverses versions des démonstrateurs d'une façon unifiée (via un interpréteur), ce qui facilite les comparaisons.

En second lieu, PrologII+ est un Prolog compilé, possédant des caractéristiques évoluées telles le ramasse-miettes automatique et l'optimisation de l'appel récursif terminal. Ces caractéristiques permettent d'envisager des mises en œuvre performantes.

En troisième lieu, la notion de module proposée par PrologII+ est bien adaptée à l'écriture de programme de taille conséquente, en permettant de profiter du fait que les programmes sont compilés pour réaliser des chargements/déchargements rapides d'ensemble de règles. Cela a permis d'une part un gain de temps important dans la mise en œuvre : alors que l'insertion des règles du programme prend plusieurs minutes, l'insertion d'un module compilé prend de l'ordre de la seconde. Ce gain a été un argument important pour la représentation d'un exercice (contenant la traduction de la spécification et les théories associées) sous la forme d'un module : le chargement de l'exercice parait instantané à l'élève. D'autre part, il a permis la compilation séparée des différents modules.

En quatrième lieu, le débogeur proposé par PrologII+ est un outil très utile : bien que Prolog soit un langage de haut niveau, il reste toujours des problèmes dus aux erreurs typographiques. Le débogueur permet de les résoudre rapidement.

Enfin, une des dernières fonctionnalités proposée par PrologII+ en 1992 a été très utile sur le plan des performances. Il s'agit des ajouts et retraits rapides de faits via les règles fassert et fretract. Ces règles ont permis d'augmenter les performances du démonstrateur de façon très significative, par une utilisation adéquate de lemmes et surtout d'anti-lemmes (lemmes non prouvables).

6.2. Les problèmes traités par TALC.

L'étendue des problèmes que peut traiter TALC est difficile à définir a priori. Ce qu'il est plus aisé de définir, ce sont les objectifs que nous avons déterminés à ce sujet.

Étant donné la combinatoire engendrée par l'objectif de complétude, nous estimions a priori que les problèmes comportant un nombre d'objets limités constituaient déjà un objectif intéressant. C'est sur cette base que nous nous sommes fixé comme but que TALC puisse traiter les énoncés des problèmes de la classe de 4ème, car c'est dans cette classe que la géométrie est abordée le plus largement dans une vision axiomatique.

Avant de donner des exemples de problèmes typiques traités par TALC, il nous faut faire une distinction importante. L'objectif de TALC est avant tout d'offrir un système permettant de s'assurer que la construction produite par un élève correspond bien à la figures sur laquelle s'appuie la situation d'apprentissage ou de validation des connaissances. L'objectif de TALC n'est pas en soi de proposer un problème de construction à l'élève, même si cela est possible. Cela signifie que l'on peut séparer les problèmes traités par TALC en deux catégories :

- d'une part les spécifications entrant dans le cadre d'un problème plus vaste, et où la production d'une construction ne pose pas de problème difficile. TALC a été conçu pour traiter ce genre de spécification.
- d'autre part les spécifications exprimant un problème non trivial de construction. Dans ce cas, on utilise TALC dans un nouveau but, en faisant de la construction une fin en soi. Dans le programme de 4ème, par exemple, cela ne constitue qu'une faible partie des exercices proposés [Julien et al. 88].

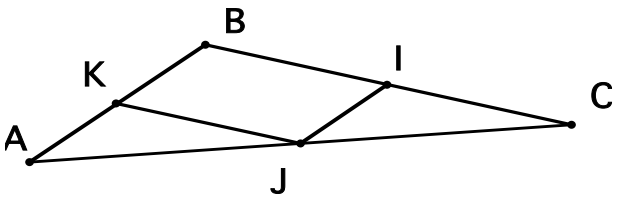
Nous estimons que produire une construction est difficile quand :
- il faut déterminer un ordre de construction adéquat sur des objets décrits par la spécification.
- la construction requiert l'ajout d'objets non-spécifiés.
- le problème de construction fixe des objets de base (TALC ne prend d'ailleurs pas en compte cet espect du contrat).

Bien entendu la frontière du facile et du difficile change au cours de l'apprentissage, du fait de l'acquisition de procédures de construction de certaines figures (par exemple la construction d'une médiatrice au compas).

Nous présentons dans cette sous-partie pour chacune de ces deux catégories des exemples traités par TALC et un exemple posant des problèmes de spécification. Chaque exemple est donné dans le langage des manuels scolaires, en indiquant en particulier le but général de l'exercice, quand il y en a un. Nous donnons ensuite une spécification en CDL exprimant les hypothèses de cet exercice. Nous présentons enfin un exemple de construction avec TALC répondant à la spécification.

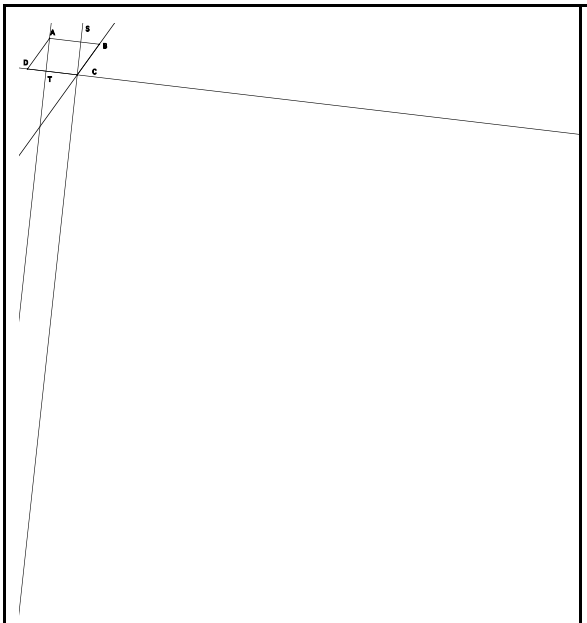
6.2.1. Des spécifications requérant une construction simple.

Exemple 6.2.1.1.

<table>
<tr><td colspan="2">Soit (A B C) un triangle.
Soient I,J,K les milieux respectifs de [B C], [A C] et [A B].
Montrer que (B I J K) est un parallélogramme</td><td>I e [B C], |B I| = |I C|,
J e [A C], |A J| = |J C|,
K e [A B], |A K| = |K B|.</td></tr>
<tr><td colspan="2">
</td><td>La figure comporte 9 objets.
B : point quelconque
A : point quelconque
C : point quelconque
triangle B A C
K : milieu des 2 points B et A
I : milieu des 2 points C et B
J : milieu des 2 points A et C
segment [K J]
segment [J I]</td></tr>
</table>

Exemple 6.2.1.2.

Soit (A B C D) un parallélogramme. Soit T la droite perpendiculaire à (A B) et passant par A. Soit S la droite perpendiculaire à (D C) et passant par C. Montrer que S est parallèle à T.	[A B] // [D C], [A D] // [B C], [A B] !- T, droite(T), [D C] !- S, droite(S).



La figure comporte 12 objets.

A : point quelconque

B : point quelconque

segment [A B]

D : point quelconque

segment [A D]

D#1 : droite passant par D et parallèle au segment [A B]

D#2 : droite passant par B et parallèle au segment [A D]

C : intersection des 2 droites D#2 et D#1

segment [D C]

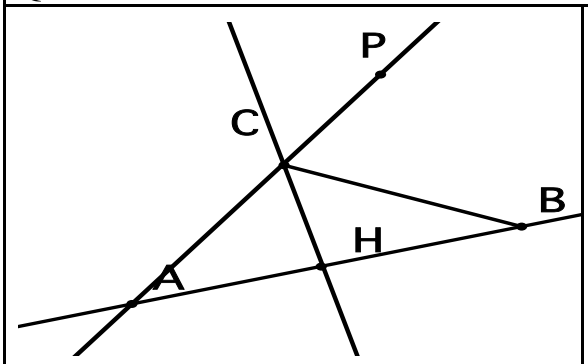
segment [C B]

T : droite passant par A et perpendiculaire au segment [A B]

S : droite passant par C et perpendiculaire au segment [D C

Exemple 6.2.1.3.

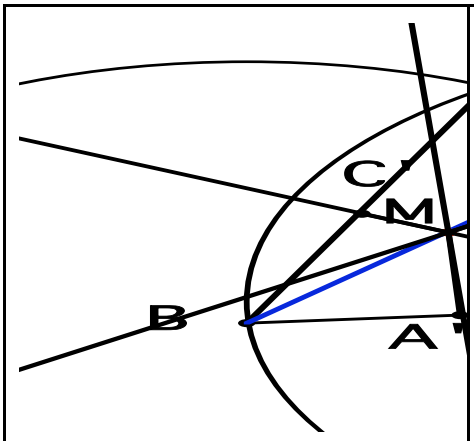
Soit (A P) une droite. Soit C un point sur cette droite. Soit B un point. Soit H le pied de la hauteur issue de C dans le triangle (A B C). Mesure l'angle Δ = <B A C Pour plusieurs angles Δ obtenus en modifiant la position de P, choisi plusieurs positions pour C et mesure les longueurs |A H| et |A C|. consigne-les dans un tableau et calcule |A H|/|A C. Quelle conclusion tires-tu?	C e (A P), H e [A B], [C H] !- [A B].



La figure comporte 9 objets.
A : point quelconque
P : point quelconque
D#1 : droite passant par A et P
C : point de la droite D#1
B : point quelconque
D#2 : droite passant par A et B
segment [C B]
D#3 : droite passant par C et perpendiculaire à D#2
H : intersection des 2 droites D#3 et D#2

Exemple 6.2.1.4.

Soit (A B C) un triangle équilatéral, Soit M un point à l'intérieur du triangle, Soit A' le projeté orthogonal de M' sur [B C], Soit B' le projeté orthogonal de M' sur [A C],. Soit C' le projeté orthogonal de M' sur [A B], Soit H le pied de la hauteur issue de C dans le triangle (A B C). Montrer que |M A'| + |M B'|+ |M C'| = |A H|.	|A B|= |B C|, |B C |= |C A|, m e [A B], M e [m C], [M A'] !- [B C], A' e [B C], [M B'] !- [A C], B' e [A C], [M C'] !- [A B], C' e [A B], [A H] !- [B C], H e [B C].



La figure comporte 21 objets.
B : point quelconque
C : point quelconque
segment [B C]
C#1 : cercle de centre C passant par B
C#2 : cercle de centre B passant par C
A : intersection du cercle C#1 et du cercle C#2
P#4 : intersection du cercle C#1 et du cercle C#2 (A est l'autre point)
segment [B A]
segment [C A]
m : point du segment [C A]
segment [m B]
M point du segment [m B]
M : point quelconque
D#1 : droite passant par M et perpendiculaire au segment [B C]
D#2 : droite passant par M et perpendiculaire au segment [B A]
D#3 : droite passant par M et perpendiculaire au segment [C A]
D#4 : droite passant par A et perpendiculaire au segment [B C]
H : intersection du segment [B C] et de la droite D#4
A' : intersection du segment [B C] et de la droite D#1
B' : intersection du segment [C A] et de la droite D#3
C' : intersection du segment [B A] et de la droite D#2
segment [C' M]
segment [A' M]
segment [M B']

Les exemples 6.2.1.1 et 6.2.1.2 sont typiques d'une situation où l'objectif principal est la démonstration d'une propriété. Le premier est tiré d'un manuel de 4ème [Jullien & al. 88], le second est un exemple traité par MENTONIEZH [Py 90].

L'exemple 6.2.1.3. est typique d'une situation expérimentale. Il a pour objectif la découverte du cosinus. Cet exemple est actuellement utilisé au collège de Moirans avec Cabri-géomètre.

Le dernier exemple montre un exercice tiré d'un manuel qui ne peut être spécifié dans TALC qu'en introduisant des objets sans rapport avec l'énoncé (ici le point `m` et le segment `[B m]`). Cela est dû à l'absence de l'objet demi-plan dans les langages CDL, SCL et LDL.

6.2.2. Des spécifications exprimant un problème de construction.

Les exemples de spécifications exprimant un problème de construction que nous donnons ici ont tous la caractéristique de nécessiter la construction d'objets non spécifiés.

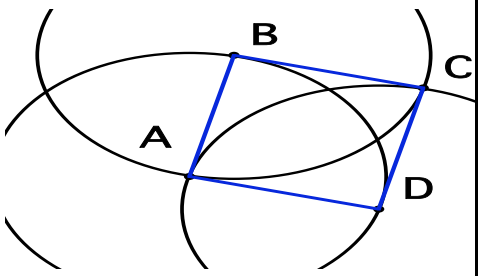
On peut dégager deux types d'exercices parmi ces exemples :

- des exercices où les objets de base sont donnés. Cela signifie, soit qu'une construction comportant ces objets est effectivement donnée à l'élève et qu'il doit la compléter, soit qu'on l'oblige à construire certains objets en premier. Notons que TALC ne peut

actuellement contrôler ce genre de condition. Il faut soit faire confiance à l'élève, soit que l'enseignant contrôle lui-même ce point.

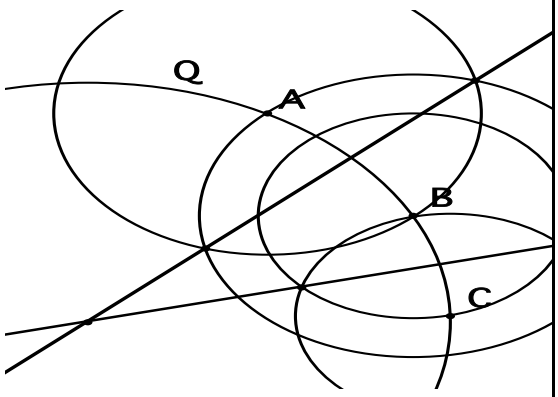
- des exercices où aucun ordre n'est imposé.

6.2.2.1. Exemples où aucun objet n'est donné.

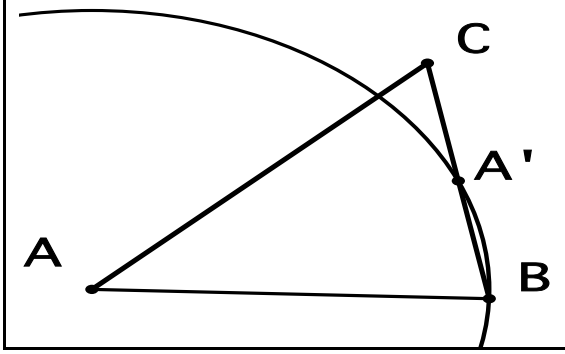
Soit (A B C D) un losange	|A B| = |B C|, |B C| = |C A|.
	A : point quelconque D : point quelconque segment [A B] C#1 : cercle de centre A passant par D C#2 : cercle de centre D passant par A B : point du cercle C#1 C#3 : cercle de centre B passant par A C : intersection du cercle C#2 et du cercle C#3 (A est l'autre point) segment [B C] segment [C D] segment [D A]

6.2.2.2. Exemples où les objets de base sont donnés.

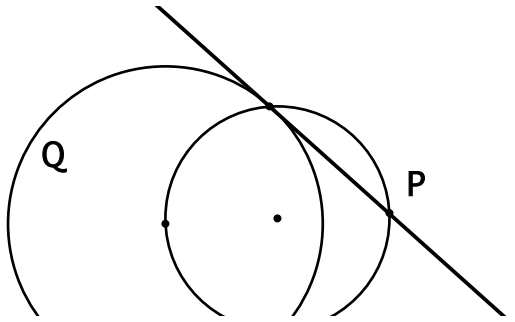
Exemple 6.2.2.2.1.

Soit trois points A, B et C donnés. Construire le cercle Q passant par ces trois points.	cercle(Q), A e Q, B e Q, C e Q.
	La figure comporte 15 objets. A : point quelconque B : point quelconque C : point quelconque C#1 : cercle de centre B passant par A C#2 : cercle de centre A passant par B C#3 : cercle de centre B passant par C C#4 : cercle de centre C passant par B P#4 : intersection du cercle C#2 et du cercle C#1 P#5 : intersection du cercle C#2 et du cercle C#1 (P#4 est l'autre point) P#6 : intersection du cercle C#3 et du cercle C#4 P#7 : intersection du cercle C#3 et du cercle C#4 (P#6 est l'autre point) D#1 : droite passant par P#5 et P#4 D#2 : droite passant par P#7 et P#6 P#8 : intersection des 2 droites D#1 et D#2 Q : cercle de centre P#8 passant par C

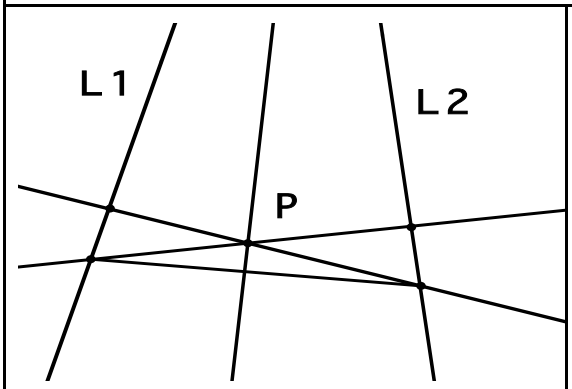
Exemple 6.2.2.2.2.

Soit deux points A et B donnés. Construire les points C et A' tels que [A A'] soit une médiane du triangle (A B C) et |A A'| = |A B|. Quel est l'ensemble de points A' possibles?	A' e [B C], |B A'| = |A' C|, cercle(Q), A e Q, B e Q, C e Q.
	La figure comporte 8 objets. A : point quelconque B : point quelconque segment [A B] C#1 : cercle de centre A passant par B A' : point du cercle C#1 C : symétrique du point B par rapport au point A' segment [C B] segment [C A]

Exemple 6.2.2.2.3.

Soit donnés un cercle Q et un point P. Construire la tangente à Q passant par P.	o = centre(Q), m e Q, m e L, L !- (o m)
	La figure comporte 8 objets. P : point quelconque Q : cercle quelconque P#2 : centre du cercle Q P#3 : milieu des 2 points P#2 et P C#2 : cercle de centre P#3 passant par P#2 P#4 : intersection du cercle C#2 et du cercle Q P#5 : intersection du cercle C#2 et du cercle Q (P#4 est l'autre point) D#1 : droite passant par P#4 et P

Exemple 6.2.2.2.4.

<table>
<tr><td colspan="2">Soit données deux droites L1 et L2 non parallèles.
Soit donné un point P.
Construire la droite passant par P et l'intersection de L1 et L2 sans construire cette intersection.</td><td>Pas de spécification possible.</td></tr>
<tr><td>
</td><td colspan="2">La figure comporte 11 objets.
L1 : droite quelconque
L2 : droite quelconque
P : point quelconque
D#3 : droite passant par P et perpendiculaire à L2
D#4 : droite passant par P et perpendiculaire à L1
P#2 : intersection des 2 droites L1 et D#4
P#3 : intersection des 2 droites D#3 et L1
P#4 : intersection des 2 droites L2 et D#4
P#5 : intersection des 2 droites D#3 et L2
segment [P#3 P#4]
D#5 : droite passant par P et perpendiculaire au segment [P#3 P#4]</td></tr>
</table>

Les exemples 6.2.2.2.1 et 6.2.2.2.3 sont typiques d'une situation où l'objectif principal est la construction d'une figure.

L'exemple 6.2.2.2.2. est typique d'une situation de recherche d'un lieu géométrique.

Le dernier exemple montre un cas où le contrat didactique défini dans TALC ne permet pas d'exprimer l'énoncé. En effet, le contrat stipule que tous les objets de la spécification doivent être construits. Pour vérifier que la droite attendue possède bien les propriétés requises, il faut les exprimer dans la spécification, ce qui n'est possible qu'en mentionnant le point d'intersection, qui doit alors être construit.

Ces quatre exemples sont tirés du même manuel de 4ème que précédemment [Jullien & al. 88].

6.3. Performances.

Les performances que nous évaluons portent en premier lieu sur les temps de réponses du système.

Concernant TALC-Enseignant, il s'agit des temps de traduction de la spécification. La réduction des égalités requiert la majeure partie du temps de traduction. Les temps de réponses attendus de TALC-Enseignant ne sont pas très exigeants car l'enseignant peut se permettre d'attendre une réponse.

Concernant TALC-Élève, deux temps de réponses sont à considérer. Il s'agit d'une part du temps de réponse après l'utilisation d'une primitive de construction, notamment du aux vérifications d'égalité entre les nouveaux objets construits et les précédents. D'autre part, il s'agit du temps de construction du diagnostic.

D'après les observations faites sur les élève utilisant Cabri-géomètre, on peut dire approximativement que le temps écoulé entre chaque construction est en moyenne de trente secondes. Nous avons donc pris cette durée pour définir en première approximation la limite supérieure du temps de réponse du système après chaque construction d'objet que l'élève peut accepter. Concernant le temps de construction du diagnostic, nous estimons qu'il doit être du même ordre de grandeur. La limite supérieure acceptable à l'élève dans ce cas peut tout de même être supérieure, de l'ordre de la minute

Les expérimentations que nous avons menées n'ont pu se dérouler en situation réelle, c'est à dire dans les classes. Cela est dû principalement à la disponibilité tardive de TALC-Élève. Cette disponibilité tardive vient d'une part du fait que la version communiquante de Cabri-géomètre est très récente. Seule une utilisation réelle dans les classes a été faite par les concepteurs de cette version, à travers HYPERCARD [Tahri 93]. Elle vient d'autre part du fait que de telles expérimentations supposent de disposer d'un logiciel dans un état de robustesse très important, sous peine de voir les résultats faussés par des problèmes annexes.

Les expérimentations menées ont consisté à nous mettre successivement à la place d'un enseignant construisant un exercice et à la place d'un élève produisant une construction.

Sur le plan quantitatif, les objectifs que nous nous étions fixés sont atteints dans la majorité des cas. Nous avons évalué pour les exemple cités précédemment, sur un Macintosh Quadra 700□

• que le temps de traduction d'une spécification ne dépasse pas 20 secondes. Ce temps est généralement d'environ une seconde.

•□que le temps de traduction de la construction d'un objet possède les mêmes caractéristiques.

• que le temps de construction du diagnostic ne dépasse pas 30 secondes dans la majorité des cas.

Quelques cas particuliers dépassent largement cette limite et posent aussi des problèmes de place mémoire. L'analyse de ces cas difficiles a montré :

- qu'il s'agit toujours de problèmes de construction en soi. Il est bon de rappeler que l'objectif de TALC n'est pas de traiter ce type de problèmes a priori.

- que les temps de réponse importants sont dus à des constructions demandant l'extension de la spécification par de nombreux objets et pour lesquelles aucune extension ne permet de vérifier que la construction satisfait la spécification. Ces temps de réponse sont dus la combinatoire nécessaire à la complétude dans ces cas.

Pour faire face correctement à cette catégorie de problèmes, nous estimons que des gains importants peuvent être obtenus par l'utilisation de modèles dans La Géométrie. L'idée est que si une propriété est fausse dans La Géométrie, alors il est inutile d'en rechercher une preuve dans TIG qui en est un sous-ensemble. Cette idée était à la base des travaux de Gelertner [Gelernter et al. 63] sur la démonstration automatique en géométrie.

Plusieurs possibilités s'offrent à nous, reposant sur des modèles analytiques d'une figure géométrique. Une première possibilité consiste à utiliser la programmation logique avec contrainte sur les intervalles [Older et al. 93]. Une deuxième possibilité consiste à utiliser la méthode de triangularisation de Wu [Wu 86] et les bases de Gröbner [Kutz 90]. Une troisième possibilité consiste à l'utiliser de l'oracle de Cabri-géomètre. Cette dernière possibilité peut être envisagée à très court terme car la version communiquante de Cabri-géomètre propose cette fonctionnalité depuis le début de l'année. La restriction de cette fonctionnalité aux objets nommés n'est pas très gênante et n'est que provisoire.

Sur le plan qualitatif, les expérimentations que nous avons menées ont montré la valeur de TALC autant sur le fond que sur l'interface. Ainsi, nous avons longtemps considéré comme une erreur de TALC son refus de la construction d'un losange. Mais bien que cet exemple ait permis par ailleurs de découvrir un certain nombre d'erreurs, nous avons fini par être convaincus que le système ne se trompait pas : une fois ces erreurs corrigées, le système a pu nous fournir le diagnostic précis nous permettant de comprendre notre erreur.

Le système a aussi montré ses qualités par le fait qu'une démonstration du logiciel a été effectuée sans problème majeur alors que l'orateur n'avait eu que de brèves explications sur la manipulation de l'interface. Sur ce plan, une erreur introduite dans les axiomes d'extension dans TALC-Enseignant a pu être corrigée en quelques minutes, par une simple édition de la théorie TEXT incriminée. Les expérimentations menées ont aussi montré l'adéquation des messages d'erreur du système concernant la spécification. Concernant TALC-Élève enfin, la construction de figures ne semble pas poser de problèmes aux utilisateurs habitués à Cabri-géomètre, ce qui était notre objectif.

Conclusion

Dans ce document, nous avons présenté la définition et la mise en œuvre de TALC, un système tuteur pour la construction de figures géométriques logiquement correctes. Vérifier automatiquement qu'une construction satisfait une spécification peut paraître de prime abord une question assez simple. Le travail exposé dans ce document montre qu'il n'en est rien et que la question est plus difficile qu'elle ne le parait.

On peut comparer cette question à celle de la vérification qu'un programme satisfait une spécification. En effet, on peut considérer une construction géométrique comme un programme dont les données sont les objets géométriques et le résultat la représentation graphique de cette construction. Par rapport à la vérification de programme, ce qui diffère éventuellement pour la vérification de la correction d'une construction est le contexte d'apprentissage dans lequel nous sommes placés. D'une part, alors qu'un programme est considéré comme satisfaisant s'il implique la spécification donnée, une construction n'est satisfaisante que si elle est "équivalente" à la spécification, ceci afin de rejeter les constructions particulières. D'autre part, alors que les connaissances de références pour la vérification d'un programme peuvent être considérés comme fixées une fois pour toutes, les connaissances nécessaires à la vérification d'une construction évoluent en même temps que les connaissances de l'apprenant.

La définition et la réalisation de ce système ont posé des problèmes informatiques nouveaux et difficiles à résoudre concernant les méthodes de spécification logique, les techniques de l'intelligence artificielle et l'utilisation de la programmation logique comme outil de génie logiciel.

Dans cette conclusion, nous présentons, concernant chacun de ces trois volets, d'une part le résumé du travail présenté dans ce document et les avancées qu'il constitue, et d'autre part les points délicats qui sont encore à améliorer. Cela nous permet ensuite de définir les perspectives de prolongement de ce travail, à court terme aussi bien qu'à long terme.

Les problèmes informatiques posés.

Méthodologie de spécification logique.

Le propos que nous soutenons dans ce document est qu'une définition en terme de logique ce que nous appelons le contrat didactique proposé à l'élève par l'enseignant est nécessaire pour un système utilisant des capacités de raisonnement. Nous avons divisé la définition de ce contrat en deux parties, une partie dite disponible et une partie dite prédéfinie. Le travail que nous avons réalisé montre pour la définition de chacune de ces deux parties que les problèmes soulevés demandent une définition explicite et claire si l'on veut fournir à l'enseignant un système répondant à son attente et auquel il puisse se fier .

La partie disponible du contrat représente les moyens langagiers mis à la disposition de l'enseignant. Nous avons ainsi donné la sémantique des langages d'interface dont dispose d'une part l'enseignant pour décrire une spécification (le langage CDL) et le contexte (la théorie TIG) qui y est attaché, d'autre part l'élève pour réaliser sa construction (le langage SCL). La définition de ces sémantiques par rapport à un troisième langage logique (LDL) a permis de placer la vérification de la correction dans un cadre logique bien délimité.

Le langage LDL a été défini comme un langage de la logique du premier ordre, permettant ainsi de profiter de la précision et la clarté de la logique du premier ordre pour la sémantique de CDL et de SCL.

CDL a été défini de façon à être proche des langages de spécification généralement utilisés dans les manuels scolaires en France. De ce fait c'est un langage déclaratif acceptant des typages implicites des identificateurs. Sa sémantique définie par rapport à LDL permet d'expliciter à l'enseignant l'exigence exacte qu'il formule dans sa spécification. Cette sémantique exprime par exemple que si la spécification CDL introduit un segment, la droite support est aussi introduite, à la différence de la distance entre les extrémités qui ne l'est pas.

Les primitives de construction de SCL ont été choisies comme un sous-ensemble de Cabri-géomètre. Cela a permis à TALC de profiter des services proposés par la version serveur de ce remarquable logiciel de construction géométrique et en particulier de proposer une interface d'une grande qualité. La sémantique de SCL permet de préciser à l'élève la définition des constructions les plus élaborées en fonction des objets et relations primitifs de LDL. Par exemple, la médiatrice de deux points A et B est définie comme la droite perpendiculaire à la droite (A B) passant par le milieu de A et B. Cela signifie pour l'élève que même si la droite (A B) n'est pas représentée graphiquement à l'écran , elle fait partie de la construction qu'il a produite.

L'enseignant spécifie les connaissances présupposées de l'élève pour la réalisation du contrat à l'aide d'un ensemble d'axiomes géométriques nommé TIG. L'ensemble des connaissances requises pour répondre au contrat est celui engendré par l'application des règles de déduction de la logique du premier ordre à cet ensemble d'axiomes. Nous avons justifié le choix du langage LDL pour exprimer ces axiomes par le fait que ce langage est plus adapté à l'évolution des connaissances de l'élève que les langages utilisés habituellement par les mathématiciens pour définir des axiomatiques de la géométrie. L'adoption de formules dites de Bernays-Schönfinkel [Ackerman 84] pour les axiomes de TIG répond au souci logique indispensable de décidabilité. Cette restriction impose aux formules de ne contenir ni termes fonctionnels, ni quantification existentielle. Nous avons donné aussi une première justification pour la restriction de la forme des axiomes à des clauses de Horn par la facilité qu'elles permettent dans la définition de la théorie par le professeur et par la clarté des explications des raisonnements qu'elles permettent. Enfin nous proposons une méthodologie de définition des théories TIG ainsi qu'un premier exemple nommé TIG_0 visant à aider l'enseignant dans sa tâche de définition du contrat. Chacune des restrictions apportées reflète l'exigence de l'attitude logique que nous avons choisie et la fécondité de cette approche pour l'obtention d'une définition claire des déductions du système et de la qualité des explications qu'il permet.

La définition de la partie prédéfinie du contrat est déterminée par les choix de conception que nous avons effectués pour TALC et que l'enseignant ne peut modifier. Ces choix ont été effectués en vue de répondre aux quatre exigences suivantes :

- Une exigence par rapport à la spécification donnée par l'enseignant : il faut d'une part que la construction satisfasse toute les hypothèses de la spécification et d'autre part qu'elle ne satisfasse que ces hypothèses. Cela signifie qu'il faut d'une part que la construction implique la spécification et d'autre part que la spécification implique la construction, vis-à-vis de la théorie TIG, ce qui s'exprime par

$TIG, HYPNU \ / \ F\ \ \ (S)$

où est une substitution faisant correspondre les identificateurs de l'élève à ceux de l'enseignant et HYPNU représente l'hypothèse de nom unique que nous avons adoptée pour les objets représentés par un identificateur. Cette hypothèse modélise le fait que l'enseignant désire la construction de tous les objets qu'il a cités.

Du fait du choix des formules de Bernays-Schönfinkel pour les axiomes de TIG, nous avons montré que cette équivalence ne permet pas à l'élève de proposer des constructions

faisant intervenir des objets non décrits dans la spécification. Ceci est trop restrictif par rapport aux exigences habituelles de l'enseignant. Nous avons été amenés à définir logiquement une multifonction dite d'extension. L'idée principale de cette définition est de se guider sur la construction proposée par l'élève pour augmenter la spécification avec les objets supplémentaires de la construction, sans ajouter de relation géométrique superflue entre les objets précédemment présents. Cet ajout est basé sur une théorie axiomatique existentielle nommée TEXT où chaque axiome définit un ajout possible d'objet à la règle et au compas.

Le contrat est alors réalisé s'il existe □ tel que

TIG, HYPNU / F □ S*

avec □ S* □ FE(□(S),F,TIG,TEXT) où FE est la multifonction d'extension.

-□ne exigence par rapport à la théorie TIG : il faut que la spécification (et la construction) soit cohérente de vis-à-vis de cette théorie. Nous avons montré que le problème était résolu pour la construction en prenant pour acquis que les outils de construction sont conformes à la géométrie dans son ensemble. Concernant la spécification, l'exigence est garantie en théorie du fait de l'utilisation des clauses de Horn pour TIG et de la façon dont nous traitons la négation, en pratique il apparaît raisonnable de l'assurer.

-□ne exigence par rapport aux outils de construction disponibles : il faut qu'une construction répondant à la spécification donnée par l'enseignant soit possible avec les outils qu'il propose. Cette vérification nécessitant l'emploi de moyens lourds, nous avons préféré pour l'instant demander au professeur d'exhiber une construction correspondant à la spécification qu'il décrit, montrant de cette manière l'existence d'au moins une construction répondant au contrat.

-□ne exigence liée à la construction : il faut que la spécification puisse être satisfaite par une construction à elle seule. Cette exigence nous a amené d'une part à restreindre les langages CDL et LDL à des conjonctions de littéraux. D'autre part, nous avons caractérisé les spécifications répondant à cette exigence dans la théorie des modèles logiques par l'existence d'un unique modèle minimal pour l'ensemble composé de la spécification et la théorie. Cette caractérisation est vérifiée si la théorie est restreinte à des clauses de Horn, ce qui constitue la seconde raison de cette restriction de la forme des axiomes de TIG. Comme il est nécessaire de pouvoir disposer néanmoins d'une expression de la négation dans les axiomes de TIG, nous avons adopté un traitement particulier pour les littéraux négatifs : chaque littéral négatif non(p(x1,…,xn)) a été transformé en un littéral positif non_p(x1,…,xn).

Cette formulation logique de la partie prédéfinie du contrat exprime sans ambiguïté possible à l'enseignant comme à l'élève le contrat qui les lie à travers le système. Elle explicite un ensemble d'exigences importantes qui, bien que pouvant paraître évidentes dans le cadre d'un enseignement classique d'enseignant à élève, ne peuvent être implicites dans le cadre d'un enseignement où un système informatique représente l'enseignant face à l'élève. Elle permet aussi une définition aisée des explications possibles par rapport aux termes du contrat. Ces deux caractéristiques justifient le propos que nous soutenons que l'approche logique pour l'expression du contrat didactique est exigente et féconde, notamment concernant la question délicate de l'extension, dont une définition paralogique a pu être donnée.

Les points délicats restant à améliorer quant à l'expression du contrat didactique concernent principalement trois questions□□adéquation entre les outils proposés à l'élève et les théories (TIG et TEXT), l'extension des langages CDL et LDL et l'expression de la négation.

Concernant les outils de construction proposés à l'élève, il faudrait disposer d'un moyen automatique d'assurer que les théories TIG et TEXT concordent avec les propriétés induites par ces outils de construction. Par exemple, si la construction d'une droite perpendiculaire à une autre droite et passant par un point est disponible, il faudrait que TEXT contienne un axiome permettant de construire une telle perpendiculaire et que TIG permette de déduire l'unicité d'une perpendiculaire passant par un point, puisque cette construction possède un degré de liberté nul.

Concernant le langage CDL, il s'avère que certaines spécifications de figures constructibles à la règle et au compas ne sont possibles dans ce langage qu'en utilisant des objets sans rapport avec l'énoncé.

Exemple :

La figure 1 ci-dessous, où l'on désire que les points A et D soient "du même côté" par rapport au segment [B C] est de ce type. En effet, une spécification en CDL de cette figure n'introduisant pas d'objets "sans rapport" pourrait aussi correspondre la figure 2, où A et D ne sont pas du même côté par rapport à [B C].



Figure 1



Figure 2

Pour pouvoir spécifier cette figure, il faut introduire par exemple le symétrique D' de D par rapport à C et spécifier que l'intersection de [A D'] et [B C] existe, ce qui n'est pas très naturel.

Pour pouvoir spécifier de telles figures de façon plus naturelle, il faudrait pouvoir faire référence à des demi-plans en CDL (et en LDL par conséquent).

Un autre problème du langage CDL concerne la sémantique que nous lui avons donnée, où tout terme représente un seul objet. Cette sémantique ne permet pas l'expression de figures dans certaines géométries non-euclidiennes, alors qu'il est possible par contre que TIG représente une théorie non-euclidienne de la géométrie.

Le choix de la restriction de la forme des axiomes de TIG à des clauses de Horn, qui d'une part permet d'éviter les spécifications demandant la construction de plusieurs figures exclusives et d'autre part permet de générer des explications faciles à comprendre, a des effets négatifs sur la sémantique donnée à la négation. En effet, le traitement de la négation que nous avons adopté n'est pas conforme avec le sens usuel de la négation logique et il est de ce fait difficile d'expliquer sa sémantique à l'enseignant.

Sur le plan des techniques de l'intelligence artificielle, notre travail a consisté d'une part à définir la mise en œuvre de plusieurs démonstrateurs, d'autre part à donner une définition opérationnelle de la construction de la multifonction d'extension.

Les démonstrateurs que nous avons mis en œuvre possèdent comme caractéristique commune le traitement des boucles dans la démonstration. Du fait que les théories géométriques définissent le plus souvent les prédicats de manière récursive, nous avons introduit un mécanisme de détection des boucles garantissant la terminaison des démonstrations. À partir d'un démonstrateur de base, nous avons défini des démonstrateurs adaptés aux besoins spécifiques, soit des vérifications d'égalité dans les traductions, soit de la construction de la multifonction d'extension, soit de la vérification de la correction. Nous avons déterminé parmi ces cas ceux où l'utilisation de propriétés de la théorie peut améliorer l'efficacité des preuves. Par exemple, on peut profiter du fait que les preuves d'une propriété et de son "contraire", au sens de notre négation ou au sens de l'hypothèse du monde clos, ne peuvent exister conjointement à condition que chaque axiome de TIG soit correct vis-à-vis de la géométrie en général.

La définition opérationnelle que nous avons donnée de la multifonction d'extension, dont nous avons prouvé qu'elle correspond bien à la définition logique, nous a permis de montrer que l'utilisation de la logique du premier ordre pour exprimer le contrat didactique non seulement permet de disposer d'une expression claire mais aussi est valide sur le plan opérationnel. Nous pensons d'ailleurs que la méthode que nous avons définie devrait pouvoir être appliquée dans d'autres contextes de vérification de la correction d'une solution dans lesquels il est nécessaire de pouvoir prendre en compte des objets supplémentaires. Comme nous l'avons vu dans le chapitre premier, les systèmes qui peuvent admettre des solutions comportant de tels objets sont quasi-inexistants. Cela tient sans aucun doute à la difficulté que cela apporte à la définition du système, difficulté accrue quand le système doit aussi garantir la complétude de ses raisonnements et accepter des théories modifiables.

Certaines questions concernant l'extension ne sont pourtant pas encore bien résolues.

Dans le cas où une extension existe, il serait bon que le système la trouve le plus rapidement possible. Pour cela, il peut paraître intéressant d'essayer de construire une extension en suivant l'ordre de construction de l'élève. Sans doute cela serait-il appréciable dans bien des cas. Pourtant, si les objets supplémentaires de l'élève sont les premiers qu'il a construits, la question reste difficile. Mieux encore, on sait que dans la géométrie prise en son entier, si une extension est correcte alors toute autre extension l'est aussi car elles sont équivalentes. De façon duale, cela signifie que si une extension incorrecte existe alors aucune extension correcte ne peut être trouvée. Si la même propriété pouvait être prouvée sur une théorie TIG, alors le gain en efficacité serait vraiment très appréciable.

La façon dont nous avons défini l'extension peut aussi avoir un effet pervers, souvent mentionné dans les systèmes informatiques pour l'enseignement : comme le système accepte des objets supplémentaires en considérant la solution la plus favorable à l'élève, celui-ci peut adopter comme stratégie de résolution de construire un nombre important d'objets supplémentaires, en espérant que l'un d'eux satisfasse le contrat. Pour résoudre ce problème, nous envisageons deux méthodes. Une première méthode consiste à faire en sorte que le professeur exige un nom pour tous les objets qu'il décrit, empêchant alors que ce soit le système qui détermine la correspondance la plus favorable à l'élève. Une seconde méthode consiste à détecter si des objets n'ont aucune d'utilité dans la preuve de la correction et à demander à l'élève qu'il supprime de lui-même ces objets superflus.

La programmation logique comme outil de génie logiciel.

L'utilisation de la programmation logique a comporté pour la mise en œuvre des avantages importants.

PrologII+ a permis des mises en œuvre rapides de versions successives de TALC et des comparaisons fructueuses en terme d'efficacité entre ces versions. PrologII+ est un outil de haut niveau où les programmes sont proches de leur expression naturelle. Les difficultés de programmation sont principalement liées à des erreurs typographiques que le débogueur puissant permet de résoudre rapidement. Prolog permet aussi une mise en œuvre facile d'un programme à plusieurs meta-niveaux, dont l'avantage est d'une part de ne pas mélanger les connaissances en géométrie et le contrôle par le démonstrateur et d'autre part de représenter les diverses versions des démonstrateurs d'une façon unifiée (via un interpréteur), ce qui facilite les comparaisons.

La notion de module compilé de PrologII+ s'est avérée cruciale vu la taille importante de TALC. En effet, l'utilisation des modules compilés a permis un gain important sur le plan du génie logiciel : le chargement d'un module compilé est quasi-immédiat sur un Macintosh Quadra700 alors que le chargement de l'ensemble des règles composant TALC est de l'ordre de la dizaine de minutes. Ces modules ont l'avantage classique de permettre la compilation séparée. Ils permettent aussi d'obtenir un chargement très rapide de TALC tout en maintenant les facilités d'un interprète. Nous avons ainsi utilisé un module compilé pour représenter un exercice. Il contient les règles représentant la traduction de la spécification, les théories TIG et TEXT, et leur chargement semble immédiat.

Une des dernières fonctionnalités proposée par PrologII+ en 1992 a été utilisée à fond□il s'agit des ajouts et retraits rapides de faits via les règles fassert et fretract. Nous avons trouvé une méthodologie d'utilisation de ces règles qui a permis d'augmenter les performances du démonstrateur très sensiblement, grâce à un algorithme utilisant de façon adéquate des lemmes et surtout des anti-lemmes (lemmes non prouvables) et basé sur des propriétés induites par l'hypothèse du monde clos.

Du point de vue du génie logiciel, PrologII+ s'est avéré un outil permettant une intégration facile de fonctionnalités nouvelles définies dans un autre langage : l'interfaçage entre PrologII+ et le langage C est très simple et a permis l'utilisation des Apple-events proposés par le système d'exploitation 7 du Macintosh. L'utilisation de ces Apple-events fait de TALC un des premiers EIAO défini comme le client d'un autre EIAO, ici Cabri-géomètre.

TALC-Enseignant, le logiciel dont dispose l'enseignant pour définir le contrat et TALC-Élève, le logiciel dont dispose l'élève pour réaliser ce contrat constituent des réalisations importantes : TALC-Enseignant est constitué d'environ 800 règles Prolog; TALC-Élève est constitué d'à peu près autant de règles auxquelles il faut ajouter 3200 lignes de code C pour réaliser le lien entre Prolog et Cabri-géomètre. Par sa taille et son ambition, TALC constitue aujourd'hui une des mises en œuvre importante à base de PrologII+. Sa réalisation a été l'occasion de nombreux échanges avec la société qui commercialise PrologII+, contribuant ainsi à améliorer ce produit.

Enfin les interfaces proposées par TALC-Enseignant et TALC-Élève ont montré leur qualité lors des expérimentations menées. Par exemple, concernant TALC-Enseignant, une erreur introduite dans les axiomes d'extension a pu être corrigée en quelques minutes, par une simple édition de la théorie TEXT incriminée. Les expérimentations qui ont été menées à ce jour ont aussi montré l'adéquation des messages d'erreur du système concernant la spécification. Concernant TALC-Élève, une démonstration du logiciel a été effectuée récemment par exemple sans problème majeur alors que le démonstrateur n'avait eu que de brèves explications sur la manipulation de l'interface.

Cependant les réalisations effectuées ne sont pas suffisamment satisfaisantes sur certains points.

L'interface de TALC-Enseignant ne permet pas une manipulation structurée des axiomes. Il faudrait que l'interface puisse distinguer les axiomes en catégories, par

exemple les axiomes de définition, les théorèmes, les axiomes indispensables à un niveau donné.

Du point de vue de la communication avec Cabri-géomètre, l'interface de TALC-Éleve ne reproduit pas encore suffisamment fidèlement les possibilités de Cabri-géomètre , du point de vue par exemple de la disponibilité des menus ou de la modification dynamique de la forme du pointeur.

Du point de vue des performances du système, les résultats obtenus ne sont pas toujours suffisants pour un système interactif, en particulier dans les cas où la spécification pose un problème de construction nécessitant de nombreux objets supplémentaires.

Perspectives.

A partir du travail présenté dans ce document, de nombreuses perspectives sont ouvertes, à court terme comme à moyen terme.

A court terme, TALC-Élève peut à peu de frais être intégré dans des logiciels existants où la vérification de la correction d'une figure est importante. C'est le cas de HYPERCABRI [Capponi 91] ou du tuteur hybride de Tahri [Tahri 93], pour lequel la complétude apportée par TALC-Élève du point de vue géométrique compléterait judicieusement l'analyse didactique existante : dans le premier cas il s'agit de vérifier que la construction représente bien un carré, dans le second le système requiert de l'élève la construction du symétrique d'un segment par rapport à une droite.

Il est aussi possible d'intégrer rapidement à Cabri-géomètre lui-même les capacités de détection des égalités que comporte TALC.

Ces trois utilisations de TALC par d'autres EIAO pourraient se faire aisément en utilisant les Apple-events [Apple 93], en considérant cette fois TALC comme le serveur et Cabri-géomètre, HYPERCABRI ou le tuteur hybride de Tahri comme le client.

Nous étudions aussi en ce moment la coopération qui pourrait s'effectuer entre TALC et MENTONIEZH. l'intégration des ces deux environnements permettrait d'obtenir le système d'enseignement de la géométrie défini initialement dans le projet Mentoniezh et dont sont issues ces deux réalisations. En effet, comme MENTONIEZH comporte maintenant une phase d'exploration des propriétés de la construction et que TALC intègre les capacités de Cabri-géomètre, l'objectif du projet serait pleinement atteint. Cette intégration devrait pouvoir être rapidement menée du fait de la réécriture en PrologII+ de MENTONIEZH. Nous envisageons aussi à plus long terme une intégration sur PC, du fait que PrologII+ est disponible sur ces machines, que MENTONIEZH est déjà implanté sur PC et que les concepteurs de Cabri-géomètre travaillent aujourd'hui à permettre à la version PC du logiciel de communiquer comme le fait celle sur Macintosh.

Des améliorations de TALC lui-même peuvent aussi être rapidement menées. Nous pouvons à brefs délais réaliser une interface permettant à l'enseignant de définir les outils de construction qu'il donne à l'élève. Cela implique de définir pour chaque outil les axiomes correspondant de TIG et de TEXT et de les introduire automatiquement dans le contrat en fonction des outils définis par l'enseignant. Pour améliorer les performances des démonstrateurs, nous prévoyons à très court terme l'utilisation de l'oracle de Cabri-géomètre pour éviter la démonstration de propriétés manifestement fausses, soit dans la construction de l'élève, soit dans une construction modèle fournie par l'enseignant.

A plus long terme, les questions suivantes doivent être étudiées :

- l'extension du langage CDL, en ajoutant la notion de demi-plan et permettant la définition de macro-définitions. Mieux encore, il faudrait pouvoir autoriser l'enseignant à définir lui-même le langage de spécification CDL, le langage LDL et la traduction de l'un

vers l'autre. Cela lui permettrait d'une part de pouvoir spécifier toutes les figures désirées et d'autre part de se placer dans des géométries non-euclidiennes.

- Le traitement de la négation doit être amélioré. Nous songeons à utiliser des formalismes "presque-Horn" comme ceux étudiés dans [Ostier 92] qui permettent de ne pas perdre la qualité des explications due aux clauses de Horn tout en permettant une sémantique de la négation plus facile à comprendre.

- Les performances du système pour les constructions comportant de nombreux objets non décrits par la spécification doivent être améliorées. Cette amélioration peut s'effectuer par deux voies non exclusives. En premier lieu, les démonstrateurs doivent pouvoir s'appuyer sur des résultats fournis en considérant la géométrie dans son ensemble. En effet, si une propriété est fausse dans la géométrie dans son ensemble, alors il est clair qu'elle l'est aussi dans la théorie TIG et que la recherche d'une démonstration pour cette propriété est inutile. Nous songeons dans cette optique à utiliser des démonstrateurs travaillant sur les coordonnées algébriques des objets géométriques utilisant la programmation logique par contraintes sur les intervalles [Older et al. 93] ou utilisant la méthode de triangularisation de Wu [Wu 85] et les bases de Gröbner [Kutz 90]. En second lieu, nous espérons pouvoir caractériser des théories TIG pour lesquelles la recherche combinatoire de toutes les extensions possibles n'est pas nécessaire, comme c'est le cas pour la géométrie dans son ensemble.

Enfin un travail important à effectuer consiste à élaborer autour de TALC un environnement permettant de définir des situations didactiques évoluées. Dans ce but, doivent être étudiés :

- des ensembles d'exercices correspondant à des niveaux donnés, représentés par différentes théories TIG. Ces ensembles pourraient être la base de séquences d'enseignement pré-établies proposées aux enseignants. Ce travail a été déjà commencé par un chercheur en visite dans notre laboratoire, le professeur Nguyen Than Thuy, de l'Institut National Polytechnique de Hanoï.

- des capacités de déduction automatique d'un modèle de l'élève, par observation de variables didactiques adéquates puisées dans l'historique des constructions effectuées par l'élève. La construction de ce modèle nécessite l'utilisation de méthodes d'apprentissage automatique telles que celles introduites dans [Shapiro 81].

- un langage d'expression des explications générées par le système en fonction de la construction proposée par l'élève et du modèle de l'élève. Ce langage serait mis à la disposition du professeur et constituerait un degré de liberté supplémentaire offert par TALC. Il lui permettrait de préciser la réaction du système à certaines actions de l'élève. Cela consisterait par exemple à déterminer la réaction face à la construction de deux objets égaux, ou encore à définir si l'explication doit être fournie en terme de contre-exemple ou à partir d'une activité permettant de mettre le doigt sur le problème.

Dans ce but, nous envisageons une campagne d'expérimentation auprès des enseignants et une collaboration étroite avec des didacticiens pour la définition de ce langage et des situations à proposer.

Bibliographie.

[Ackerman 84] Ackermwan, W., *Solvable cases of the Decision Problem*, North Holland, 1984.

[Aida 83] Aida, H., Tanoka, H., Moto-Oka, T., *A Prolog Extension for Handling Negative Knowledege*, New Generation Computing, n°1, Springer Verlag, 1983.

[Allard et al. 86] Allard, J.C., Pascal, C., *Euclide, un langage pour la géométrie plane*, logiciel et manuel, 1986, Cedic-Nathan.

[Allen et al. 93] Allen, R., Idt, J., Trilling, L., *Constraint Based Automatic Construction and Manipulation of Geometric Figures*, Proceedings of the 13th IJCAI Conference, Chambery, Morgan Kaufmann Publishers, Los Altos, 1993.

[Anderson et al. 85] Anderson, J.R., Boyle, C.F., Yost, G., *The Geometry Tutor*, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, Los Altos, 1985.

[Anderson et al. 87] Anderson, J.R., Boyle, C.F., Farell, R., Reiser, B.J., *Cognitive principles in the design of computers tutors*, Modelling Cognition, P. Morris Ed, John Wiley & Sons Ldt, 1987.

[Apple 93] *Inside Macintosh, Interapplication Communication*, Apple Technical Library, Addison-Wesley Publishing Company, New York, 1993.

[Apt & al. 88] Apt, K. R., Blair, H., Walker, A., Towards a theory of declarative knowledge, Foudations of Deductive Databases and Logic Programming, ed. J. Minker, Morgan Kaufmann, 1988.

[Artigue 90] Artigue, M., *Analyse de processus d'enseignement en environnement informatique*, Actes de l'Université d'été Informatique et Enseignement de la Géométrie, IREM de Toulouse, 1990.

[Balacheff 90] Balacheff, N., *Exigences épistémologiques des recherches en EIAO*, Journées EIAO du PRC-GDR Intelligence Artificielle, Cachan, 1990, pp 317-339.

[Balacheff 93] Balacheff, N., *Didactique et Intelligence Artificielle : problèmes et perspectives,* in Gras, R., Ed, Fascicule de didactique des mathématiques, IRMAR, Rennes, 1993, PP 35-40.

[Balacheff 94] Balacheff, N., *Artificial Intelligence and Real Teaching*, in Keitel C., Ruthven K. (eds.) Learning through computers: Mathematics and Educational Technology, Berlin, Springer Verlag, 1994.

[Balbiani et al 92] Balbiani, P., Farinãs del Cerro, L., *A tableaux-based engine for geometrical reasoning*, Semantics of Time, Space, Movement and Spacio-temporal Reasoning, 4th TMS Workshop, Aurnague, M., Borillo, A., Borillo, B., Bras, M, editors, Toulouse, 1992.

[Baron et al. 93] Baron, M., Gras, R., Nicaud, J.F., *Environnements Interactifs d'Apprentissage avec Ordinateur*, Eyrolles, Paris, 1993.

[Baulac 90] Baulac, Y., *Un micromonde de géométrie, Cabri-géomètre*, Thèse, Université Joseph Fourier - Grenoble, 1990.

[Baulac et al. 91] Baulac, Y., Giorgiutti, I., *Interaction micro-monde/tuteur, le cas de Cabri-géomètre et DEFI*, Actes des deuxièmes journées EIAO de Cachan, Editions de l'Ecole Normale Supérieure de Cachan, 1991, pp 11-18.

[Bazin 93] Bazin, J.M., *GEOMUS, un resolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé*, These de l'Université de Paris VI, 1993.

[Bellemain 92] Bellemain, F., *Conception, réalisation et expérimentation d'un logiciel d'aide à l'enseignement de la géométrie, Cabri-géomètre*, Thèse de l'Université Joseph Fourier, Grenoble, 1992.

[Bernat 93] Bernat, P., *Pour une aide au raisonnement non linéaire basé sur la prégnance*, Environnements Interactifs d'Apprentissage avec Ordinateur, Eyrolles, Paris, 1993.

[Bonnaire et al. 1991] Bonnaire, R., Perrin, L., *Développement de simulations pédagogiques dans un environnement orienté objets*, Actes des deuxièmes journées EIAO de Cachan, Editions de l'Ecole Normale Supérieure de Cachan, 1991, pp 211-220.

[Borenstein et al. 91] Borentstein, P., Mattson, J., *Think C, User Manual*, Symantec Corporation, Cupertino, Californie, 1991.

[Bournaud et al, 1993] Bournaud, I., Mathieu, J., Zucker, J. D., *COOPERE, Un formalisme de représentation des COnnaissances Organisées Pour l'Explication, le Résolution et l'Enseignement*, Environnements Interactifs d'Apprentissage avec Ordinateur, Eyrolles, Paris, 1993.

[Bruillard 90] Bruillard, E., *ARRIA : Un système ouvert de gestion de fragments pour l'apprentissage de raisonnements et de leur communication*, Actes de l'Université d'été Informatique et Enseignement de la Géométrie, IREM de Toulouse, 1990.

[Brousseau 86] Brousseau, G., *Fondements et méthodes de la didactique des mathématiques*, Recherches en didactique des mathématiques, 7.2, 1986, pp 33-115.

[Brousseau 90] Brousseau, G., *Le contrat didactique : le milieu*, Recherches en didactique des mathématiques, 9.3, 1990, pp 309-336.

[Burton 82] Burton, R., *Diagnosing bugs in a simple procedural skill*, in Intelligent Tutoring Systems, Sleeman, D., Brown, J.S, (Eds.), Academic Press, New-York, 1982.

[Capponi 91] Capponi, B., *Hypercarré, Problèmes d'un système tutoriel en géométrie*, Actes des deuxièmes journées EIAO de Cachan, Editions de l'Ecole Normale Supérieure de Cachan, 1991, pp 77-88.

[Carrega 89] Carrega, J.C., *Théorie des Corps, la règle et le compas*, réédit., Herman, Paris, 1989.

[Chandrasekaran et al. 89] Chandrasekaran, B., Tanner, M. & Josephson, J.R., *Explaining Control Strategies in Problem Solving*, IEEE Expert, spring, 9-20, 1989.

[Chouraqui et al. 90] Chouraqui, E., Inghilterra, C., *Le raisonnement analogique et l'apprentissage symbolique en géométrie : le cas du tutoriel Archimède*, Actes de l'Université d'été Informatique et Enseignement de la Géométrie, IREM de Toulouse, 1990.

[Cuppens 90] Cuppens, R., *Intelligence artificielle et enseignement de la géométrie*, Actes de l'Université d'été Informatique et Enseignement de la Géométrie, IREM de Toulouse, 1990.

[Delozanne et al. 89] Delozanne, E., Carrière, E., *Niveaux de connaissance dans un tuteur intelligent*, Actes des journées EIAO, PRC-IA, Cachan, Décembre 1989.

[Desmoulins 90] Desmoulins, C. *Construction d'une formule logique associée à la spécification d'une figure géométrique*, DEA, Université Joseph Fourier, Grenoble, 1990.

[Desmoulins et al 91] Desmoulins, C., Trilling, L., *Translation of a figure specification into a logical formula in a system for teaching geometry*, Proceedings of the Sixth International PEG Conference, Rapallo, Italy, 1991, pp.292-303.

[Desmoulins 93] Desmoulins, C., *On Relationship Between Semantics and Explanation in an ITS for Teaching Geometry*, Proceedings of AI-ED 93, World Conference on Artificial Intelligence in Education, Edinburgh, Scotland, 1993.

[Dillenbourg 93] Dillenbourg, P., *De la généralisablité d'un environnement d'apprentissage*, Environnements Interactifs d'Apprentissage avec Ordinateur, Eyrolles, Paris, 1993.

[Dreyfus 81] Dreyfus, H.L., *From Micro-Worlds to Knowledge Representation: Ai at Impasse*, Mind Design, J. Haugeland editor, Cambridge, Massachusset, MIT Press, pp 161-204, 1981.

[Elsom-Cook 90] Elsom-Cook, M., *Guided Discovery Tutoring*, Paul Chapman Publishing, London, 1990.

[Fauvergue 88] Fauvergue, P., *Mathématiques 4ème*, Istra, Editions Casteilla , Paris, 1988.

[Gavignet 90] Gavignet, E., *Une approche nouvelle pour la construction de systèmes d'enseignement assistés par ordinateurs*, Journées EIAO du PRC-GDR Intelligence Artificielle, Cachan, 1990, pp 155-166.

[Gelernter et al. 63] Gelernter, H., Hansen, J.R., Loveland, D.W., *Empirical exploration of the Geometry Theorem Proving Machine*, in Computer et Thought, edited by E. Feigenbaum and J.Feldman, McGraw-Hill, New York, pp. 134-152, 1963.

[Gras 88] Gras, R., *Aide logicielle aux problèmes de démonstration géométriques dans l'enseignement secondaire*, 17, Petit x, IREM, Grenoble, 1988.

[Greenberg 74] Greenberg, M. J., *Euclidean and Non-Euclidean Geometries*, Freeman, New York, 1974.

[Gregoire 90] Grégoire, E., *Logiques non monotones et intelligence artificielle*, Hermès, Paris, 1990.

[Hilbert 71] Hilbert, D., *Les fondements de la géométrie* édition critique préparée par Paul Rossier, Dunod, Paris, 1971.

[Jackiw 89] Jackwick, N., *Geometer Sketchpad*, Swarthmore College, Software by Key Curriculum Press, Inc, Berkeley, 1989.

[Jullien & al. 88] Julien, V., Bras, A., Le Goff, A., Penninckx, J., *Mathématiques 4ème*, Magnard, Paris, 1988.

[Kutz 90] Kutzler, B., *Deciding a Class of euclidean Geometry Theorems with Buchberger's algorithm*, Revue d'Intelligence Artificielle, vol. 4 n° 3, 1990, pp 81-97.

[Kautz 87] Kautz, H. A., *A formal theory of plan recognition*, PhD Thesis, Department of computer science, University of Rochester, 1987.

[Kowalski 79] Kowalski, R., *Logic for Problem Solving*, North-Holland, New York, 1979.

[Laborde 86] Laborde, J. M., *Projet d'un ométrie*, Présentation de projet LSDD-IMAG, Grenoble, 1989.

[Laborde et al 89] Laborde, J.-M. Trilling, L., *Conception et réalisation d'un système intelligent d'apprentissage de la géométrie*, Présentation de projet LSDD-IMAG, Grenoble, 1989.

[Lebesgue 89] Lebesgue, H., *Leçons sur les constructions géométriques*, rééd., Gauthier-Villard, Paris, 1989.

[Lelong-Ferrand 85] Lelong-Ferrand, J., *Les fondements de la géométrie*, Presses Universitaires de France, Paris, 1985.

[Lloyd 87] Lloyd, J. W., *Foudations of Logic Programming*, second extended version, Springer Verlag, 1987.

[Luengo 93], Luengo, V., *Contraintes informatiques et apprentissage de la démonstration, à propos de trois logiciels*, DEA, Université Joseph Fourier, Grenoble, 1993.

[Major 93] Major, N., *Reconstructiong Teaching Strategies with Coca*, Proceedings of the World Conference on Artificial Intelligence in Education, Edinburgh, 1993.

[Matiussi 90] Matiussi, C., *L'extension Euclide+*, Actes de l'Université d'été Informatique et Enseignement de la Géométrie, IREM de Toulouse, 1990.

[McCarthy 80] McCarthy, J., *Circumscription: a form of non-monotonic reasoning*, Artificial Intelligence, 13, 1980, pp. 27-39.

[Minsky et al. 70] Minsky, M., Papert, S., *Draft on a proposal to ARPÄ for research on artificial intelligence at MIT*, 1970.

[Nanard 90] Nanard, J., *La manipulation directe en interface homme-machine*, Thèse, Université des sciences et techniques du Languedoc, Montpellier, 1990.

[Nicaud et al. 88] Nicaud, J.F., Vivet, M., *Les tuteurs intelligents : réalisations et tendances de recherches*, Techniques et Science Informatiques, vol 7, n°1, 1988, pp 21-45

[Nicaud 89] Nicaud, J.F., *APLUSIX, un système expert pédagogique et un environnement d'apprentissage dans le domaine algèbrique*, Technique et Science Informatique, vol 8, 1989, pp 145-155.

[Nicaud et al. 90] Nicaud, J.F., Aubertin, C., Nguyen Xuan, A., Saidi, M., Wach, P., *APLUSIX, un environnement d'apprentissage à plusieurs niveaux dans le domaine du raisonnement algébrique, Journées EIAO du PRC-GDR Intelligence Artificielle*, Cachan, 1990, pp 297-316.

[Nicolas 89] Nicolas, P., *Construction et vérification de figures géométriques dans le système MENTONIEZH*, Thèse, Université de Rennes, 1989.

[Older et al. 93] Older B., Vellino A., *Constraint Arithmetic on Real Intervals*, Constraint Logic Programming, F. Benhamou & A. Colmerauer, editors, The MIT Press, Cambridge, MA, 1993.

[Ostier 92] Ostier, P., *Vérification de la correction de figures géométriques à l'aide de démonstrateurs pour des clauses "presque de Horn"*, DEA, Université Joseph Fourier, 1992.

[Papert 80] Papert, S., *Jaillissement de l'esprit, ordinateurs et apprentissage*, Flammarion, Paris, 1980.

[Pintado 91] Pintado, M., *Une approche pour un tuteur informatique d'entraînement à la résolution d'exercices de géométrie élementaire*, Actes des deuxièmes journées EIAO de Cachan, Editions de l'Ecole Normale Supérieure de Cachan, 1991, pp 45-60.

[Prologia 92] *PROLOG II+*, version 2.4, Manuel de référence et d'utilisation, Prologia, Marseille, 1992.

[Py 90a] Py, D., *Reconnaissance de plan pour l'aide à la démonstration dans un tuteur intelligent de la géométrie*, Thèse de l'Université de Rennes 1, Rennes, 1990.

[Py 90b] Py, D., *MENTONIEZH, a geometry I.T.S. for figure drawing and prof setting*, Journal of Artificial Intelligence in Education, Vol. 1, N°3, 1990, pp 41-56.

[Qaife 89] Quaife, A., *Automated development of Tarski's geometry*, Journal of Automated Reasoning, vol. 5, 1989, pp-97-118.

[Ruthven 94] Ruthven, K., *Technology and rationalisation of teaching*, in Keitel C., Ruthven K. (eds.) Learning through computers: Mathematics and Educational Technology, Berlin, Springer Verlag, 1994.

[Schreck 92] Schreck, P., *Automatisation des constructions géométriques à la règle et au compas*, Thèse de l'Université Louis Pasteur, Strasbourg, 1992.

[Senet 93] Senet, C., *Réalisation d'un protocole de communication entre un tuteur de construction de figures géométriques, TALC, et un micro-monde de géométrie, Cabri-géomètre*, Rapport de stage de maîtrise, Université Joseph Fourier, 1993.

[Shapiro 81] Shapiro E., *A general incremental algorithm that infers theories from facts.*, in Seventh IJCAI, pp 446-451, 1981.

[Tahri 93] Tahri, S., *Modélisation de l'interaction didactique, un tuteur hybride sur Cabri-géomètre pour l'analyse de décisions didactiques*, Thèse de l'Université Joseph Fourier, Grenoble, 1993.

[Terracher & al. 88] Terracher, P., Delord, R., Vinrich, G., Privat, B., *Mathématiques 4ème*, Hachette, Paris, 1988.

[Tessier et al. 94] Tessier, S., Laborde, J. M., *Description des événements Apple acceptés par Cabri-géomètre*, Rapport technique LSD2 n° 105, Grenoble, 1994.

[Thaysse et al. 89] Thaysse, A. et al., *Approche logique de l'intelligence artificielle*, Dunod, 1989.

[Vivet 90] Vivet, M., *Usage des tuteurs intelligents : prise en compte du contexte, rôle du maître*, Journées EIAO du PRC-GDR Intelligence Artificielle, Cachan, 1990, pp 239-246.

[Wenger 87] Wenger, E., *Artificial Intelligence and Tutoring Systems*, Morgan Kaufmann Pub. Inc., Palo Alto, 1987.

[Wu 86] Wu, W. T. *Basic Principles of Mechanical Theorem Proving in Elementary Geometry*, Journal of Automated Reasoning, 1986, pp 221-252

Annexe A.
TIG0 exemple d'une théorie de la géométrie.

Cette annexe présente un exemple de théorie, TIG_0 exprimée sous la forme disponible à l'enseignant dans TALC-Enseignant, telle que définie au troisième chapitre, 2.1.1.

A. Axiomes portant sur des égalités d'objets.

1. Egalité de points.

```
1.1.0;  "Deux droites distinctes se coupent en un seul point"
egal_pt(p1,p2) ->
   appdr(p1,l1)
   non_egal_dr(l1,l2)
   appdr(p1,l2)
   appdr(p2,l1)
   appdr(p2,l2) ;

1.2.0;    "Deux points sont égaux s'ils sont les centres d'un même cercle"
egal_pt(o1,o2) ->
   cercle(c,o1,r)
   cercle(c,o2,r) ;

1.3.0;    "Au maximum deux points sont à la fois à une distance donnée \
          d'un point et sur une droite donnée"
egal_pt(p1,p2) ->
   appdr(p1,l)
   appdr(p2,l)
   appdr(p3,l)
   distancep(r,o,p1)
   distancep(r,o,p2)
   non_egal_pt(p2,p3)
   distancep(r,o,p3)
   non_egal_pt(p1,p3) ;

1.4.0;   "Si d est la distance d'un point p à une droite l,
un seul point est à la fois à une distance d de p et sur l"
egal_pt(p1,p2) ->
   appdr(p1,l)
   appdr(p2,l)
   appdr(p,l')
   appdr(p2,l')
   perp(l,l')
   distancep(d,p,p1)
   distancep(d,p,p2) ;
```

```
1.5.0;   "L'intersection de deux cercles de centres différents
est composée de deux points au maximum"
egal_pt(p1,p2) ->
  cercle(c1,o1,d1)
  cercle(c2,o2,d2)
  appcc(p1,c1)
  appcc(p2,c1)
  appcc(p3,c1)
  appcc(p1,c2)
  appcc(p2,c2)
  appcc(p3,c2)
  non_egal_pt(o1,o2)
  non_egal_pt(p1,p3)
  non_egal_pt(p2,p3);

1.6.0;   "L'intersection de deux cercles de centres diférents \
est unique si la distance des centres égale la somme des rayons"
egal_pt(p1,p2) ->
  distancep(d1,o1,p1)
  distancep(d1,o1,p2)
  distancep(d2,o2,p1)
  distancep(d2,o2,p2)
  distancep(d,o1,o2)
  somdist(d1,d2,d)
  non_egal_pt(o1,o2);

1.7.0 ; "L'intersection de deux cercles de centres diférents \
est unique si la distance des centres égale la différence des rayons"
egal_pt(p1,p2) ->
  distancep(d1,o1,p1)
  distancep(d1,o1,p2)
  distancep(d2,o2,p1)
  distancep(d2,o2,p2)
  distancep(d,o1,o2)
  somdist(d1,d,d2)
  non_egal_pt(o1,o2);

1.8.0;   "Un seul point est situé sur une demi-droite\
 à une distance donnée de l'origine"
egal_pt(p1,p2) ->
  demi_droite(h,o,l)
  appdd(p1,h)
  appdd(p2,h)
  distance(d)
  distancep(d,o,p1)
  distancep(d,o,p2) ;

1.1.1;  "Deux points sont différents s'ils sont première et seconde extrémité d'un
segment"
non_egal_pt(p1,p2) ->
  segment(s,p1,p2,l);
```

```
1.2.1;   "Deux points sont différents si il existe une distance entre eux"
non_egal_pt(p1,p2) ->
  distancep(d,p1,p2);
```

2. Egalité de droites.

```
2.1.0;   "Deux points distincts définissent une seule droite"
egal_dr(l1,l2) ->
  appdr(p1,l1)
  appdr(p1,l2)
  appdr(p2,l1)
  appdr(p2,l2)
  non_egal_pt(p1,p2);

2.2.0;   "Par un point n'appartenant pas à une droite l ne passe qu'une droite parallèle
à l"
egal_dr(l1,l2) ->
  appdr(p,l1)
  appdr(p,l2)
  par(l1,l)
  par(l2,l) ;

2.3.0;   1 "Par un point ne passe qu'une perpendiculaire à une droite"
egal_dr(l1,l2) ->
  appdr(p,l1)
  appdr(p,l2)
  perp(l1,l)
  perp(l2,l) ;

2.4.0;   1 "Deux droites perpendiculaires sont différentes"
non_egal_dr(l1,l2) ->
  perp(l1,l2) ;

2.5.0;   1 "Deux droites parallèles sont différentes"
non_egal_dr(l1,l2) ->
  par(l1,l2) ;
```

3. Egalité de demi-droites.

```
3.1.0;   "Deux demi-droites sont égales si elles ont même origine et même sens"
egal_dd(h1,h2) ->
 memesens(h1,h2) ;
```

4. Egalité de segments.

```
4.1.0;    "Deux segments sont égaux s'ils ont mêmes premières et secondes
extrémités"
egal_seg(s,s') ->
  segment(s,p1,p2,l)
  segment(s',p1,p2,l');

4.2.0;   "Deux segments sont égaux si la première extrémité de l'un est égale à la
seconde de l'autre et vice-versa."
egal_seg (s,s') ->
```

```
  segment(s,p1,p2,l)
  segment(s',p2,p1,l');
```

5. Egalité de cercles.

```
5.1.0;   "Deux cercles sont égaux s'ils ont même centre et même rayon"
egal_cc(c1,c2) ->
  cercle(c1,o,r)
  cercle(c2,o,r) ;
```

6. Egalité de distances.

```
6.1.0;   "Deux distances entre deux points sont égales si elles ont même extrémités"
egal_dis(r1,r2) ->
  distancep(r1,p1,p2)
  distancep(r2,p1,p2) ;

6.2.0;  "Deux distances |a p| et |b p| sont égales si p est sur la médiatrice de [a b]"
egal_dis(r1,r2) ->
  distancep(r1,a,p)
  distancep(r2,b,p)
  segment(s,a,b,l1)
  appseg(i,s)
  distancep(r,i,a)
  distancep(r,i,b)
  appdr(i,l)
  perp(l,l1);

6.3.0;   "Deux distances |A I| et |I B| tellles que |A I|= 1/2|A B| sont égales si I e
[A B]"
egal_dis(r1,r2) ->
  segment(s,p1,p2,l)
  appseg(i,s)
  distancep(r1,p1,i)
  distancep(r2,p2,i)
  distancep(r3,p1,p2)
  demidist(r1,r3);

6.1.1; "Deux distances sont différentes si l'une est inférieure à l'autre"
non_egal_dis(d1,d2) ->
  infdist(d1,d2);
```

B. Axiomes portant sur les propriétés de sens.

7. Sens égaux.

```
7.1.0;   "Sur une droite exactemant deux sens sont définis"
memesens(h,h1) ->
  demi_droite(h2,o2,l)
  non_memesens(h1,h2)
```

```
   non_memesens(h,h2) ;

7.2.0;   "Réfexivité de mêmesens"
memesens(h,h) -> ;

7.3.0;   "Symétrie de mêmesens"
memesens(h1,h2) ->
   memesens(h2,h1) ;

7.4.0;   "Transitivité de mêmesens"
memesens(h1,h3) ->
   demi_droite(h2,o2,l)
   memesens(h1,h2)
   memesens(h2,h3) ;

7.5.0;   Deux demi-droites de même origine ont même sens
 si un second point de l'une appartient à l'autre"
memesens(h1,h2) ->
   non_egal_pt(p1,o)
   non_egal_pt(p2,o)
   appdd(p1,h1)
   appdd(p2,h2)
   appdd(p1,h2) ;

7.6.0;   "Deux demi-droites ayant un point commun \
et des origines différentes ont même sens \
si l'origine de l'une appartient à l'autre \
et l'origine de l'autre n'appartient pas à l'une"
memesens(h1,h2) ->
   demi_droite(h1,o1,l1)
   demi_droite(h2,o2,l2)
   non_egal_pt(o1,o2)
   appdd(p,h1)
   appdd(p,h2)
   appdd(o1,h2)
   non_appdd(o2,h1) ;

7.1.1;   "Deux demi-droites de même support ne sont pas de même sens\
 si elles sont de sens inverse"
non_memesens(h1,h2) ->
   demi_droite(h1,o,l)
   demi_droite(h2,o,l)
   invsens(h1,h2);
```

8. Sens inverses.

```
8.1.0;   2 "Deux demi-droites de même support sont de sens inverse\
 si elles ne sont pas de même sens"
invsens(h1,h2) ->
   demi_droite(h1,o,l)
   demi_droite(h2,o,l)
   non_memesens(h1,h2) ;
```

C. Axiomes portant sur les propriétés d'appartenance.

9. Appartenance à une droite.

9.1.0; "Tout point appartenant à une demi-droite appartient à son support"
```
appdr(p,l) ->
  demi_droite(h,o,l)
  appdd(p,h) ;
```

9.2.0; "Tout point appartenant à un segment appartient à son support"
```
appdr(p,l) ->
  segment(s,p1,p2,l)
  appseg(p,s) ;
```

9.3.0; "Tout point à égale distance de A et B appartient à la médiatrice de [A B]"
```
appdr(p,l) ->
  appseg(i,s)
  non_egal_pt(i,p) % sinon axiome inutile i e l => i e l
  segment(s,a,b,l1)
  distancep(r,i,a)
  distancep(r,i,b)
  distancep(r1,p,a)
  distancep(r1,p,b)
  appdr(i,l)
  perp(l,l1);
```

9.4.0; "si p appartient à une demi-droite l alors il appartient sa droite support"
```
appdr(p,l) ->
  demi_droite(h1,o,l)
  appdd(p,h1) ;
```

10. Appartenance à une demi-droite

10.1.0; "L'origine d'une demi-droite lui appartient"
```
appdd(o,h) ->
  demi_droite(h,o,l) ;
```

10.2.0; "un point p appartient à une demi-droite si il appartient a si p appartient sa droite support et qu'il n'appartient pas a l'autre demi-droite de meme origine et de meme support"
```
appdd(p,h1) ->
  demi_droite(h1,o,l1)
  demi_droite(h2,o,l1)
   non_egal_dd(h1,h2)
   appdr(p,l)
   non_appdd(p,h2) ;
```

10.3.0; "Tout point appartient à une demi-droite h s'il appartient au segment dont la première extremité o1 est l'origine h et la seconde appartient a h et s'il appartient aussi à

la demi-droite dont l'origine est la seconde extremite du segment et passant par la premiere"

```
appdd(p,h1) ->
  demi_droite(h1,p1,l)
  segment(s,p1,p2,l)
  appseg(p,s)
  demi_droite(h2,p2,l)
  appdd(p2,h1)
  appdd(p1,h2) ;
```

11 . Appartenance à un segment.

```
11.1.0;  "Tout point appartient à un segment s'il appartient aux deux demi-droites\
 définies à partir d'une extrémité et passant par l'autre"
appseg(p,s) ->
  segment(s,p1,p2,l)
  point(p)
  demi_droite(h1,p1,l)
  demi_droite(h2,p2,l)
  appdd(p2,h1)
  appdd(p1,h2)
  appdd(p,h1)
  appdd(p,h2) ;


11.2.0;  "Chaque extrémité d'un segment lui appartient"
appseg(p1,s) ->
  segment(s,p1,p2,l);

11.3.0;  "Un point appartient à un segment [A B] s'il appartient à sa droite support\
 et est situé à égale distance de A et de B"
appseg(i,s) ->
  segment(s,p1,p2,l)
  appdr(i,l)
  distance(d)
  distancep(d,i,p1)
  distancep(d,i,p2);
```

12. Appartenance à un cercle.

```
12.1.0;  "Un point appartient à un cercle de centre o et de rayon r\
 s'il est à une distance r de o"
appcc(p,c) ->
  cercle(c,o,r)
  distancep(r,o,p) ;
```

D. Axiomes portant sur les droites.

13. Droites parallèles.

```
13.1.0;  "La relation parallèle est non reflexive"
non_par(l,l) -> ;
```

```
13.2.0;   "La relation parallèle est symétrique"
par(l1,l2) ->
   non_egal_dr(l1,l2)
   par(l2,l1) ;

13.3.0;   "La relation parallèle est transitive"
par(l1,l3) ->
   non_egal_dr(l1,l3)
   par(l1,l2)
   par(l2,l3);

13.4.0;   "Deux droites perpendiculaires à une même droite sont parallèles"
par(d1,d2) ->
       perp(d1,d')
       perp(d',d2)
       non_egal_dr(d1,d2);
13.1.1;   "Deux droites perpendiculaires ne sont pas parallèles"
non_par(l1,l2) ->
   perp(l1,l2) ;
```

14. Droites perpendiculaires.

```
14.1.0;   "La relation perpendiculaire est symétrique"
perp(l1,l2) ->
   perp(l2,l1) ;

14.2.0;   "La médiatrice de [A B] lui est perpendiculaire"
perp(l1,l) ->
   segment(s,a,b,l)
   appdr(p1,l1)
   distancep(r,a,p1)
   distancep(r,b,p1)
   appdr(p2,l)
   non_egal_pt(p1,p2)
   distancep(r,p2,a)
   distancep(r,p2,b) ;

14.3.0;   "Si l1 est perpendiculaire à l2, l2 à l3 et l3 à l4\
 alors l1 est prerpendiculaire à l4"
perp(l1,l4) ->
       perp(l1,l2)
       perp(l2,l3)
       non_egal_dr(l1,l3)
       perp(l3,l4)
       non_egal_dr(l2,l4);

14.4.0;    "Deux droites parallèles sont perpendiculaires à la même droite"
perp(d1,d2) ->
       par(d1,d')
       perp(d',d2)
       non_egal_dr(d1,d');
```

```
14.1.1;   "La relation perpendiculaire est non réflexive"
non_perp(l,l) -> ;

14.2.1;   "Deux droites parallèles ne sont pas perpendiculaires"
non_perp(l1,l2) ->
   par(l1,l2) ;
```

E. Axiomes portant sur les distances.

15. Demi-distance.

```
15.1.0;   "La somme de deux moitiés de la même distance est cette distance"
demidist(d,D) ->
   distancep(D,p1,p2)
   somdist(d,d,D) ;

15.2.0;   "Si un point I est sur [A B] à égale distance de A et de B,\
 alors |A I| = 1/2 |A B|"
demidist(r,r2) ->
   distancep(r,p1,i)
   distancep(r,p2,i)
   distance(r2)
   distancep(r2,p1,p2)
   segment(s,p1,p2,l);
```

16. Distances inférieures.

```
16.1.0;   "La relation d'ordre < est totale"
infdist(d1,d2) ->
   distance(d1)
   distance(d2)
   non_infdist(d2,d1)
   non_egal_dis(d1,d2) ;

16.2.0;   "La relation d'ordre < est transitive"
infdist(d1,d3) ->
   non_egal_dis(d1,d3)
   infdist(d1,d2)
   infdist(d2,d3) ;

16.3.0;   "La somme de deux distances est supérieure à chacune d'elles"
infdist(d1,D) ->
   distance(d2)
   somdist(d1,d2,D ) ;

16.4.0;   "Inégalité du triangle"
infdist(d,D) ->
   distancep(d,p1,p2)
   distancep(d1,p,p1)
   distancep(d2,p,p2)
   appdr(p1,l)
```

```
   appdr(p2,l)
   non_appdr(p,l)
   somdist(d1,d2,D) ;

16.5.0;   "La distance entre un point p d'une droite l \
et un point en dehors de l est inférieure\
 à celle entre p et son projeté sur l"
infdist(d1,d2) ->
   appdr(p,l)
   appdr(p1,l)
   appdr(p1,l2)
   appdr(p2,l2)
   non_egal_pt(p1,p2)
   perp(l,l2)
   distancep(d1,p,p1)
   distancep(d2,p,p2) ;

16.1.1;   "La relation d'ordre < est stricte"
non_infdist(d,d) ->  ;
```

17. Distance entre deux points.

```
17.1.0;   "La propriété distancep est symétrique"
distancep(r,o,p) ->
   distancep(r,p,o);

17.2.0;   "Un point est à une distance r d'un point o\
 s'il appartient au cercle de rayon r et de centre o"
distancep(r,o,p) ->
   cercle(c,o,r)
   appcc(p,c) ;

17.3.0;  "Un point I est à une distance d de A si d =|B I| et I milieu de [A B]"
distancep(d,i,a) ->
   segment(s,a,b,l)
   appseg(i,s)
   distancep(d1,a,b)
   distancep(d,i,b)
   demi_dist(d,d1);


17.4.0;  "Un point P est à une distance d de A si d=|B P|\
 et P sur la médiatrice de [A B]"
distancep(r,p,a) ->
   segment(s,a,b,l1)
   appseg(i,s)
   non_egal_pt(i,p)
   distancep(r2,i,a)
   distancep(r2,i,b)
   perp(l,l1)
   appdr(i,l)
   appdr(p,l)
   distancep(r,p,b) ;
```

18. Somme de distances.

Pour éviter de multiplier le nombre d'axiomes du fait des deux cas possibles sur les sommes de distance, nous avons défini un prédicat auxiliaire somdist(d1,d2,d) signifiant que d est la somme des distances d1 et d2.

```
18.1.0; "Somdist est définie sur un segment"
somdist(d1,d2,d) ->
  segment(s,a,b,l)
  appseg(p,s)
  non_egal_pt(a,p)
  non_egal_pt(b,p)
  distancep(d,a,b)
  distancep(d1,a,p)
  distancep(d2,p,b);

18.2.0; "La somme de deux distances moitiés d'une distance est cette même distance"
somdist(d,d,D) ->
  distancep(D,p1,p2)
  demidist(d,D) ;
```

Annexe B
Exemples d'axiomes de TEXT

Constructions sans degré de liberté.

Deux droites non parallèles se coupent en un point

□p □ F, □l1, l2□ S,
[not(par(l1, l2) □ droite(l1) □ droite(l2) □ non(egal_dr(l1,l2))] □
□p □ S*,
[point(p) □ appdr(p,l1) □ appdr(p,l2)]

Milieu d'un segment [A B] défini par |A I|=|I B|

□i □ F, □s,p1,p2 □ S,
[segment(s,p1,p2,l)] □
□i □ S*,
[point(i) □ appseg(i,s) □ distance(r) □ distancep(r,i,p1) □ distancep(r,i,p2)]

Milieu d'un segment [A B] défini par |A I|=1/2 |A B|

□i □ F, □s,p1,p2 □ S,
[segment(s,p1,p2,l)] □
□i □ S*,
[point(i) □ appseg(i,s) □
distance(r1) □ distance(r2) □ distancep(r1,i,p1) □ distancep(r2,p1,p2) □ demidis
t(r1,r2)]

Deux cercles de rayons r1 et r2 et de centres p1 et p2 s'intersectent si il existe un triangle (A B C) tel que |A B|=|p1 p2|, |A C|=r1 et |B C|=r2

□ p □ F, □ c1, c2, p3, p4, p5 □ S,
[cercle(c1,p1,r1) □ cercle(c2,p2,r2) □ point(p3)□ point(p4) □
point(p5) □ distancep(r,p1,p2) □
distancep(r,p3,p4) □ distancep(r1,p3,p5) □ distancep(r2,p4,p5)] □
□p □ S*,
[point(p) □ appcc(p,c1) □ appcc(p,c2)]

Constructions avec un degré de liberté.

Si une droite passe par le centre d'un cercle, elle intercepte 2 fois ce cercle.

□ p □ F, □ d, c □ S,
[droite(d) □ cercle(c,p1,r) □ appdr(p1,d)] □
□p □ S*,
[point(p) □ appdr(p,d) □ appcc(p,c)]

Point sur droite

□ p □ F, □ l □ S,
[droite(l)] □
□p □ S*,
[point(p) □ appdr(p,l)]

Point sur segment

□ p □ F, □ s □ S,
[segment(s,p1,p2,l)] □
□p □ S*,
[point(p) □ appseg(p,s)]

Point sur cercle

□ p □ F, □ s □ S,
[cercle(c,p,r)] □
□p □ S*,
[point(p) □ appcc(p,l)]

Constructions avec deux degrés de liberté.

Point sur plan

□ p □ F,
□p □ S*,
[point(p)]

Droite sur plan

□ d □ F,
□d □ S*,
[droite(d)]

Cercle sur plan

□ c □ F,
□c □ S*,
[cercle(c,p,r) □ point(p) □ distance(d)]

Annexe C
Sémantique complète de CDL.

1. Définition de la traduction, règle par règle.

La définition de la traduction que nous donnons ici s'appuie sur un ensemble de fonctions décrites dans le chapitre qui suit. Elle suit le principe de composition . Chaque règle se présente de la façon suivante :

<NON_TERMINAL> ::= … <NON_TERMINAL$_1$> … <NON_TERMINAL$_N$> …
Attribut1 =… , Attribut2 =…, … **Attribut1$_1$ Attribut2$_1$ … … Attribut1$_N$ Attribut2$_N$**

Pour chaque non-terminal de la partie droite de la règle figurent les attributs (hérités) qui lui sont associés.

Sous la partie gauche figure la valeur de chacun des attributs du non-terminal en fonction des attributs des non-terminaux de la partie droite et de la "spécification simplifiée" de la spécification source CDL (notée Spec).

Définition :

Une spécification abstraite Spec d'une spécification CDL Specif est telle que

• chaque littéral de Spec correspond à un littéral de Specif.

• le littéral Ls de Spec correspondant à un littéral L de Specif est le littéral L dans lequel

- tout atome est remplacé par un atome simplifié

Term = i est remplacé par i = Term

- tout terme représentant un objet est remplacé par un terme simplifié

(Term1, Term2) est remplacé par (Term1 Term2)

[Term1, Term2] est remplacé par [Term1 Term2]

|Term1, Term2| est remplacé par |Term1 Term2|

[Term1, Term2) est remplacé par [Term1 Term2)

(Term1, Term2] est remplacé par [Term1 Term2)

(Term1 Term2] est remplacé par [Term1 Term2)

- tout identificateur i est remplacé par l'identificateur LDL correspondant CORR_LDL(i)

avec CORR_LDL(i) = (si MAJUSCULE(PREMIERCAR(i)) alors "A" sinon "a") & i

où & est l'opérateur de concaténation

PREMIERCAR(i) est le premier caractère de i

MAJUSCULE(c) est vrai si le caractère c est une majuscule

Un exemple de spécification simplifiée est donné dans le chapitre suivant.

L'expression de la valeur des attributs de la partie gauche utilise des fonctions décrites dans le chapitre suivant.

Pour améliorer la lisibilité, plusieurs règles sont regroupées quand cela est possible.

Les fonctions sont notées en petites capitales (FONCTION). Les symboles entièrement en minuscule sont des symboles de LDL (droite, memesens). Les attributs sont dénotés par un identificateur commençant par un majuscule (Attribut).

Exemple :

<DROITE_TERME> ::= (<POINT> <POINT>)
::= (<POINT> , <POINT>)
Trad1 Idf1 Term1 Trad2 Idf2 Term2
Trad = droite(Idf) appdr(Idf1,Idf) appdr(Idf2,Idf) not(egalpt(Idf1,Idf12)) @ Trad1 @ Trad2 ,
Idf = NOM(Term,Spec), Term = droite(Term1,Term2)

Dans cet exemple, les deux non-terminaux <POINT> ont respectivement les trois attributs Trad1, Idf1 et Term1 d'une part et Trad2, Idf2 et Term2 d'autre part.

.Le non-terminal <droite_terme> a trois attributs Trad, Idf et Term dont la valeur est fonction de Trad1, Trad2, Idf1, Idf2, Term1 et Term2 exprimées à l'aide des fonctions NOM ET @.

2. Sémantique de CDL, règle par règle

<CDL> ::= <LISTE_LITTERAUX>
Trad =Trad1 **Trad1**

<LISTE_LITTERAUX> ::= <LITTERAL> , <LISTE_LITTERAUX>
Trad =Trad1 @ Trad2 **Trad1** **Trad2**

<LISTE_LITTERAUX> ::= <LITTERAL> .
Trad = Trad1 **Trad1**

<LITTERAL> ::= <ATOME>
Trad = Trad1 **Trad1**
::= ¬ (<ATOME_PROPRIETE>)
Trad = NOT (Trad1 ,Prop) **Trad1 Prop**

<ATOME> ::= <ATOME_PROPRIETE>
::= <ATOME_TYPAGE >
::= <ATOME_EGALITE>
Trad = Trad1 **Trad1**

<ATOME_TYPAGE > ::= point (<identificateur>)
Trad = point(Idf) **Idf**
::= droite (<identificateur>)
Trad = droite(Idf) **Idf**
::= demi_droite (<identificateur>)
Idf
Trad = demi_droite(Idf , ORIGINE(Idf,Spec), SUPPORT(Idf,Spec)) point(ORIGINE(Idf,Spec)), droite(SUPPORT(Idf,Spec))

::= segment (<identificateur>)
Idf
Trad = segment(Idf , PREMIERE_EXTREMITE(Idf,Spec), SECONDE_EXTREMITE(Idf,Spec),SUPPORT(Idf,Spec)) point(PREMIERE_EXTREMITE(Idf,Spec)), point(SECONDE_EXTREMITE(Idf,Spec)), droite(SUPPORT(Idf,Spec))
::= cercle (<identificateur>)
Idf

Trad =
cercle(Idf , CENTRE(Idf,Spec),RAYON(Idf,Spec)), point(CENTRE(Idf,Spec)), distance(RAYO N(Idf,Spec))

::= distance (<identificateur>)
Trad = distance(Idf) **Idf**

<ATOME_EGALITE> ::= <identificateur> = <OBJET_TERME>
Trad = Trad1 **Idf1** **Trad1 Idf2**

::= <OBJET_TERME> = <identificateur>
Trad = Trad1 **Trad1 Idf2** **Idf1**

::= <OBJET_TERME> = <OBJET_TERME>
Trad1 Idf1 **Trad2 Idf2**
Trad = Trad1 @ Trad2

<ATOME_PROPRIETE> ::= <POINT> e <ENS_POINTS>
Trad1 Idf2 **Trad2 Idf2**
Trad = app(Idf1,Idf2) @ Trad11 @ Trad2 avec app= PREDAPP(Idf2,Spec)
Prop = app(Idf1,Idf2)

::= <ENS_PTS_ALIGNES> // <ENS_PTS_ALIGNES>
Trad1 Idf2 **Trad2 Idf2**
Trad = par(SUPPORT(Idf1),SUPPORT(Idf2)) @ Trad1 @ Trad2
Prop = par(SUPPORT(Idf1),SUPPORT(Idf2))

::= <ENS_PTS_ALIGNES> !- <ENS_PTS_ALIGNES>
Trad1 Idf2 **Trad2 Idf2**
Trad = perp(SUPPORT(Idf1),SUPPORT(Idf2)) @ Trad1 @ Trad2
Prop = perp(SUPPORT(Idf1),SUPPORT(Idf2))

::= <DISTANCE> '<' <DISTANCE>
Trad1 Idf2 **Trad2 Idf2**
Trad = infdist(Idf1,Idf2)) @ Trad1 @ Trad2
Prop = infdist(Idf1,Idf2))

::= memesens (<DEMI-DROITE>,<DEMI-DROITE>)
Trad1 Idf2 **Trad2 Idf2**
Trad = memesens(Idf1,Idf2)) @ Trad1 @ Trad2
Prop = memesens(Idf1,Idf2))

::= invsens (<DEMI-DROITE>,<DEMI-DROITE>)
Trad1 Idf2 **Trad2 Idf2**
Trad = invsens(Idf1,Idf2)) @ Trad1 @ Trad2
Prop = invsens(Idf1,Idf2))

<OBJET_TERME> ::= <POINT_TERME>
::= <ENSPT_TERME2>
::= <DISTANCE_TERME>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENS_POINTS> ::= <identificateur>
Trad = TYPAGE(Idf1), Idf = Idf1 **Idf1**

::= <ENS_POINTS_TERME>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENS_POINTS_TERME> ::= <ENS_PTS_ALIGNES>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENSPT_TERME2> ::= <ENS_PTS_ALIGNES_TERME2>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENS_PTS_ALIGNES> ::= <identificateur>
Trad = TYPAGE(Idf1,Spec), Idf = Idf1 **Idf1**
::= <ENS_PTS_ALIGNES_TERME>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENS_PTS_ALIGNES_TERME> ::= <DROITE_TERME>
::= <DEMI-DROITE_TERME>
::= <SEGMENT_TERME>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<ENS_PTS_ALIGNES_TERME> ::= <DROITE_TERME>
::= <DEMI-DROITE_TERME>
::= <SEGMENT_TERME2>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<POINT> ::= <identificateur>
Trad = point(Idf1), Idf = Idf1, Term=Idf1 **Idf1**
::= <POINT_TERME>
Trad = Trad1, Idf = Idf1, Term=Term1 **Trad1 Idf1 Term1**

<POINT_TERME> ::= centre (<CERCLE>)
Trad1 Idf1 Term1
Trad = point(Idf) @ Trad1, Idf= CENTRE(Idf1,Spec), Term = centre(Term1)

<DROITE_TERME> ::= (<POINT> <POINT>)
::= (<POINT> , <POINT>)
Trad1 Idf1 Term1 **Trad2 Idf2 Term2**
Trad = droite(Idf) appdr(Idf1,Idf) appdr(Idf2,Idf) not(egalpt(Idf1,Idf12)) @ Trad1 @ Trad2 ,
Idf = NOM(Term,Spec), Term = (Term1 Term2)

<DEMI-DROITE> ::= <identificateur>
Idf1
Trad = demi_droite(Idf , ORIGINE(Idf,Spec), SUPPORT(Idf,Spec)) point(ORIGINE(Idf,Spec)),
droite(SUPPORT(Idf,Spec))
Idf = Idf1
::= <DEMI-DROITE_TERME>
Trad = Trad1, Idf = Idf1 **Trad1 Idf1**

<DEMI-DROITE_TERME> ::= [<POINT> <POINT>)
::= [<POINT> , <POINT>)
Trad1 Idf1 Term1 **Trad2 Idf2 Term2**
::= (<POINT> <POINT>]
::= (<POINT> , <POINT>]
Trad1 Idf1 Term1 **Trad2 Idf2 Term2**
Trad = demi_droite(Idf,ORIGINE(Idf,Spec), SUPPORT(Idf,Spec)) appdd(Idf2,Idf)
droite(SUPPORT(Idf,Spec)) @ Trad1 @ Trad2 ,
Idf = NOM(Term,Spec), Term = [Term1 Term2)

<SEGMENT_TERME> ::= [<POINT> <POINT>]
::= [<POINT> , <POINT>]

<SEGMENT_TERME2> ::= [<identificateur> <identificateur>]
::= [<identificateur> , <identificateur>]
Trad1 Idf1 Term1 Trad2 Idf2 Term2
Trad =
segment(Idf , PREMIERE_EXTREMITE(Idf,Spec), SECONDE_EXTREMITE(Idf,Spec), SUPPORT(Idf,Spec)) droite(SUPPORT(Idf,Spec)) @ Trad1 @ Trad2 ,
Idf = NOM(Term,Spec), Term = [Term1 Term2]

<CERCLE> ::= <identificateur>
Idf1
Trad =
cercle(Idf , CENTRE(Idf,Spec),RAYON(Idf,Spec)), point(CENTRE(Idf,Spec)), distance(RAYON(Idf,Spec))
Idf = Idf1, Term = Idf1

<DISTANCE> ::= <identificateur>
Trad = distance(Idf), Idf = Idf1, Term = Idf1 Idf1

::= <DISTANCE_TERME>
Trad = Trad1, Idf = Idf1, Term = Term1 Trad1 Idf1 Term

<DISTANCE_TERME> ::= 2 <DISTANCE>
Trad1 Idf1 Term1
Trad = distance(Idf) demi_dist(Idf1,Idf) @ Trad1, Idf= NOM(Term,Spec), Term = 2 Term1

::= 1/2 <DISTANCE>
Trad1 Idf1 Term1
Trad = distance(Idf) demi_dist(Idf,Idf1) @ Trad1, Idf= NOM(Term,Spec), Term = 1/2 Term1
::= | <POINT> <POINT> |
::= | <POINT> , <POINT> |
Trad1 Idf1 Term1 Trad2 Idf2 Term2
Trad = distance(Idf) distancep(Idf,Idf1,Idf2) @ Trad1,
Idf= NOM(Term,Spec), Term = |Term1 Term2|
::= rayon (<CERCLE>)
Trad1 Idf1
Trad = distance(Idf) @ Trad1, Idf= CENTRE(Idf1,Spec), Term = rayon(Term)

3. Définition des fonctions de composition.

Nous définissons ici un ensemble de fonctions de composition. Quand cela est nécessaire, nous les illustrons sur la spécification Specif suivante :

L = (A B),
droite(L1),
L1 !- L,
p e (A, B),
p e [C D),
h = (p , centre(Q1)],
[centre(Q1) p) = [C D),
[E F) = [E G),
q = centre(Q),
r = rayon(Q),

h // [N M],
memesens(h,h2),
segment(s),
s2 = [N, M],
cercle(Q2),
r = 1/2 d.

La specification simplifiée Spec de cette spécification est la suivante :
AL = (AA AB),
droite(AL1),
AL1 !- AL,
ap e (AA AB),
ap e [AC AD),
ah = [centre(AQ1) ap),
[AC ap) = [centre(AQ1) AD),
[AE AF) = [AE AG),
aq = centre(AQ),
ar = rayon(AQ),
ah // [AN AM],
memesens(ah,ah2),
segment(as),
as2 = [AN AM],
cercle(AQ2),
ar = 1/2 ad.

• @ est l'opération de composition de traductions permettant de composer deux traductions en une seule.
L'opération t1 @ t2 vérifie la propriété suivante⊡
Si l1 est un littéral de t1 et l2 un littéral de t2 alors
si l1=l2 alors
t1 @ t2 contient uniquement l2.
sinon
t1 @ t2 contient l1 et l2.

Exemple :
point(AO) @ (cercle(AQ,AO,var1) point(AO) distance(var1)) =
cercle(AQ,AO,var1) point(AO) distance(var1)

• NOM(Term,Spec) est l'identificateur LDL de l'objet représenté par le terme Term dans la spécification Spec, défini par :

si Term est un identificateur alors
NOM(Term,Spec) = Term
si Term n'est pas un identificateur LDL et Term appartient à un atome de Spec de la forme Term = Term' (ou Term' = Term) tel que NOM(Term',Spec) = Idf est un identificateur de Spec
NOM(Term,Spec) = Idf
si Term n'est pas un identificateur LDL et Term n'appartient à aucun atome de Spec de la forme Term = Term' (ou Term' = Term) tel que NOM(Term',Spec) = Idf n'est pas un identificateur de Spec
NOM(Term,Spec) est un identificateur unique attribué par la fonction CREATION

Exemples :
NOM(AL,Spec) = AL
NOM((AA AB)), Spec) = AL
NOM([AC AD),Spec) = ah car NOM([AC ap),Spec) = ah
NOM((AC AD)),Spec) = var1

• TYPE(Idf,Spec) est le type de l'identificateur Idf, déduit de la spécification Spec, défini par :

TYPE(Idf,Spec) = point si
- Idf dérive d'un non-terminal <POINT>
- ou Idf dérive d'un des non-terminaux <identificateur> des règles

<ATOME_TYPAGE > ::= point (<identificateur>)
<SEGMENT_TERME2> ::= [<identificateur> <identificateur>]
::= [<identificateur> , <identificateur>]

- ou Idf dérive du non-terminal <identificateur> des règles suivantes et <OBJET_TERME> dérive sur <POINT_TERME>

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>

Exemples :
TYPE(aq,Spec) = point (premier cas)
TYPE(AM,Spec) = point (deuxième cas, deuxième règle)
TYPE(ap,Spec) = point (troisième cas, première règle)

TYPE(Idf,Spec) = droite si
- ou Idf dérive du non-terminal <identificateur> de la règle

<ATOME_TYPAGE > ::= droite (<identificateur>)

- ou Idf dérive du non-terminal <identificateur> des règles suivantes et <OBJET_TERME> dérive sur <DROITE_TERME>

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>

Exemples :
TYPE(AL1,Spec) = droite (premier cas)
TYPE(AL,Spec) = droite (deuxième cas, première règle)

TYPE(Idf,Spec) = demi_droite si
- Idf dérive d'un non-terminal <DEMI_DROITE>
- ou Idf dérive du de non-terminal <identificateur> de la règle

<ATOME_TYPAGE > ::= demi_droite (<identificateur>)

- ou Idf dérive du non-terminal <identificateur> des règles suivantes et <OBJET_TERME> dérive sur <DEMI_DROITE_TERME>

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>

Exemples :
TYPE(ah2,Spec) = demi_droite (premier cas)
TYPE(ah,Spec) = demi_droite (troisième cas, première règle)

TYPE(Idf,Spec) = segment si
- ou Idf dérive du de non-terminal <identificateur> de la règle

<ATOME_TYPAGE > ::= segment (<identificateur>)

- ou Idf dérive du non-terminal <identificateur> des règles suivantes et <OBJET_TERME> dérive sur <SEGMENT_TERME2>

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>

TYPE(Idf,Spec) = cercle si
- Idf dérive d'un non-terminal <CERCLE>
- ou Idf dérive du de non-terminal <identificateur> de la règle

<ATOME_TYPAGE > ::= cercle (<identificateur>)

Exemples :
TYPE(AQ,Spec) = cercle (premier cas)
TYPE(AQ2,Spec) = cercle (second cas)

TYPE(Idf,Spec) = distance si
- Idf dérive d'un non-terminal <DISTANCE>
- ou Idf dérive du de non-terminal <identificateur> de la règle

<ATOME_TYPAGE > ::= distance (<identificateur>)

- ou Idf dérive du non-terminal <identificateur> des règles suivantes et <OBJET_TERME> dérive sur <DISTANCE_TERME>

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>

Exemples :
TYPE(ad,Spec) = distance (premier cas)
TYPE(ar,Spec) = distance (troisième cas, première régle)

• TYPAGE(Idf,Spec) est l'atome de typage de l'identificateur Idf déduit de la spécification Spec, défini par (avec*type* = TYPE(Idf,Spec)

TYPAGE(Idf,Spec) = point(Idf) si *type* = point
TYPAGE(Idf,Spec) = droite(Idf) si *type* = droite
TYPAGE(Idf,Spec) = demi_droite(Idf,ORIGINE(Idf,Spec),SUPPORT(Idf,Spec))
si *type* = demi_droite
TYPAGE(Idf,Spec) =
segment(Idf,PREMIERE_EXTREMITE(Idf,Spec), SECONDE_EXTREMITE(Idf,Spec), SUPPORT(Idf,Spec))
si *type* = segment
TYPAGE(Idf,Spec) = cercle(Idf,CENTRE(Idf,Spec),RAYON(Idf,Spec))
si *type* = cercle
TYPAGE(Idf,Spec) = distance(Idf) si *type* = distance

Exemples :
TYPAGE(ap,Spec) = point(ap)
TYPAGE(AL1,Spec) = droite(AL1)
TYPAGE(ah,Spec) = demi_droite(ah,AC,var17)
TYPAGE(as2,Spec) = segment(as2,AM,AM,var5)
TYPAGE(as,Spec) = segment(as,var58,var23,var15)
TYPAGE(AQ,Spec) = cercle(AQ,aq,ar)
TYPAGE(AQ1,Spec) = cercle(AQ1,var45,var2,var63)
TYPAGE(ad,Spec) = distance(ad)

• ORIGINE(Idf,Spec) est l'identificateur de l'origine de la demi-droite identifiée par Idf dans la spécification Spec, défini par
s'il existe un atome de Spec de la forme [T1 T2) = Term' ou Term' = [T1 T2) tel que NOM(Term',Spec)= Idf' est un identificateur de Spec et T1 et T2 sont des termes.

ORIGINE(Idf,Spec) = Idf'
sinon
ORIGINE(Idf,Spec) = est un identificateur unique attribué par la fonction CREATION

• SUPPORT(Idf,Spec) est l'identificateur du support de l'objet identifié par Idf dans la spécification Spec, défini par
SUPPORT(Idf,Spec) est un identificateur unique attribué par la fonction CREATION

• PREMIERE_EXTREMITE(Idf,Spec) et SECONDE_EXTREMITE(Idf,Spec) sont respectivement la première et la seconde extrémité du segment représenté par Idf dans Spec, et sont définies par
Soit T = [T1 T2] un terme de Spec tel que NOM(T,Spec) = Idf (il y a au plus deux tels termes différents : [T1 T2] et [T2 T1]) et T1 et T2 sont des termes.
PREMIERE_EXTREMITE(Idf,Spec) = PREMIER(NOM(T1,Spec),NOM(T2,Spec))
SECONDE_EXTREMITE(Idf,Spec) = SECOND(NOM(T1,Spec),NOM(T2,Spec))

Exemple :
PREMIERE_EXTREMITE(as2,Spec) = AM
SECONDE_EXTREMITE(as2,Spec) = AN

• PREMIER (Idf1,Idf2) et SECOND(Idf1,Idf2) sont respectivement le premier et le second de Idf1 et Idf2 en considérant l'ordre >> sur les identificateurs défini par
Idf1 >> Idf2 est vérifié si
- CREE(Idf1) et CREE(Idf2) et Idf1 précéde Idf2 dans l'ordre alphanumérique usuel.
- ¬ CREE(Idf1) et ¬CREE(Idf2) et Idf1 précéde Idf2 dans l'ordre alphanumérique usuel.
- CREE(Idf1) et ¬CREE(Idf2).

Exemples :
PREMIER (AN,AM) = AM
SECOND(AN,AM) = AM
PREMIER (AN,var15) = AN
SECOND(var15,var2) = var15

• RAYON(Idf,Spec) est l'identificateur du centre du cercle Identifié par Idf dans la spécification Spec, défini par
s'il existe un atome de Spec de la forme rayon(Idf) = Term' ou Term' = rayon(Idf) et NOM(Term',Spec)= Idf' est un identificateur de Spec
RAYON(Idf,Spec) = Idf'
sinon
RAYON(Idf,Spec) est un identificateur unique attribué par la fonction CREATION

Exemples :
RAYON(AQ,Spec) = ar
RAYON(AQ1,Spec) = var44

• CENTRE(Idf,Spec) est l'identificateur du centre du cercle identifié par Idf dans la spécification Spec, défini par

s'il existe un atome de Spec de la forme centre(Idf) = Term' ou Term' = centre(Idf) et NOM(Term',Spec)= Idf' est un identificateur de Spec
CENTRE(Idf,Spec) = Idf'
sinon
CENTRE(Idf,Spec) est un identificateur unique attribué par la fonction CREATION

Exemples :
CENTRE(AQ,Spec) = aq
CENTRE(AQ1,Spec) = var74

• PREDAPP(Idf,Spec) est le prédicat d'appartenance correspondant au type de l'objet représenté par Idf dans Spec, défini par
soit *type* = TYPE(Idf,Spec)
PREDAPP(Idf,Spec) = appdr si *type*=droite
PREDAPP(Idf,Spec) = appdd si *type*=demi_droite
PREDAPP(Idf,Spec) = appseg si *type*=segment
PREDAPP(Idf,Spec) = appcc si *type*=cercle

Exemple :
PREDAPP(ah,Spec) = appdd

• NOT(T,A) est la fonction tranformant la traduction LDL d'un atome de propriété en sa négation, définie par :
NOT(T,A) = not(A) @ MOINS(T,A)
où MOINS(T,A) est tel que T = A@ MOINS(T,A) et T ≠ MOINS(T,A)

Exemple :
NOT(<droite(var15) appdr(ap,var15) point(AA) point(AB) appdr(AA,var15) appdr(BB,var15)>, appdr(ap,var15)) =
<droite(var15) not(appdr(ap,var15) point(AA) point(AB) appdr(AA,var15) appdr(BB,var15)>

• CREE(Idf) est une fonction booléenne vérifiée sur Idf a été crée par la fonction CREATION.

4. Mots réservés de CDL.

La liste des mots réservés du langage comprend les mots suivants ⊡
point
droite
demi_droite
segment
cercle
distance
memesens
invsens
centre
rayon
e

e représente l'appartenance plutôt que□ pour rendre la saisie d'énoncés possible quelle que soient les polices disponibles (et l'habilité de l'enseignant à les manipuler!).

A ces mots il convient d'ajouter l'ensemble des identificateurs définis par la traduction pour les termes non identifiés. Ils suivent la syntaxe suivante :

<identificateur_créé> ::= var<entier>

5. Syntaxe abstraite de CDL

La syntaxe abstraite des termes de CDL est une syntaxe simplifiée par rapport à la syntaxe des termes de CDL.

La syntaxe de CDL est la suivante, exprimée sous forme BNF.

```
<OBJET_TERME>                ::= <POINT_TERME>
                             ::= <ENSPT_TERME2>
                             ::= <DISTANCE_TERME>

<ENS_POINTS>                 ::= <identificateur>
                             ::= <ENS_POINTS_TERME>
<ENS_POINTS_TERME>           ::= <ENS_PTS_ALIGNES>

<ENSPT_TERME2>               ::= <ENS_PTS_ALIGNES_TERME2>

<ENS_PTS_ALIGNES>            ::= <identificateur>
                             ::= <ENS_PTS_ALIGNES_TERME>
<ENS_PTS_ALIGNES_TERME>      ::= <DROITE_TERME>
                             ::= <DEMI-DROITE_TERME>
                             ::= <SEGMENT_TERME>

<ENS_PTS_ALIGNES_TERME2>     ::= <DROITE_TERME>
                             ::= <DEMI-DROITE_TERME>
                             ::= <SEGMENT_TERME2>

<POINT>                      ::= <identificateur>
                             ::= <POINT_TERME>
<POINT_TERME>                ::= centre ( <CERCLE> )

<DROITE_TERME>               ::= ( <POINT>  <POINT> )

<DEMI-DROITE>                ::= <identificateur>
                             ::= <DEMI-DROITE_TERME>
<DEMI-DROITE_TERME>          ::= [ <POINT>  <POINT> )

<SEGMENT_TERME>              ::= [ <POINT>  <POINT> ]

<SEGMENT_TERME2>             ::= [ <identificateur>  <identificateur> ]
                             ::= [ <identificateur> , <identificateur> ]

<CERCLE>                     ::= <identificateur>

<DISTANCE>                   ::= <identificateur>
                             ::= <DISTANCE_TERME>
<DISTANCE_TERME>             ::= 2 <DISTANCE>
                             ::= 1/2 <DISTANCE>
                             ::= | <POINT>  <POINT> |
                             ::= rayon ( <CERCLE> )
```

Annexe D.
Le langage d'énoncés de Cabri-géomètre.

Cette annexe présente le langage d'énoncé de Cabri-géomètre. Sa grammaire est extrêmement simple, car elle ne comporte qu'un seul niveau. On pourrait donc la représenter avec un seul non-terminal, le non-terminal décrivant une phrase du langage. Le langage est défini comme une suite de telles phrases. Seuls restent à définir les identificateurs. Leur définition est simple à l'extrême : Cabri-géomètre accepte n'importe quelle suite de caractères.

Nous donnons ici chaque phrase du langage. Les # représentent un identificateur.

NB : la version du langage que nous présentons ici est celle ne décrivant pas les coordonnées des objets. Comme nous ne manipulons pas ces coordonnées, l'autre version ne nous est pas utile.

Le type des identificateurs est celui habituellement attendu en géométrie euclidienne.

: droite passant par # et
: cercle de centre # passant par
: milieu des 2 points # et
: médiatrice de # et
: intersection des 2 droites # et
: droite passant par # et parallèle à
: droite passant par # et perpendiculaire à
: centre du cercle
: intersection de la droite # et du cercle
: intersection de la droite # et du cercle # (# est l'autre point)
: intersection du cercle # et du cercle
: intersection du cercle # et du cercle # (# est l'autre point)
: droite passant par # et parallèle au segment [# #]
: droite passant par # et perpendiculaire au segment [# #]
: point de la droite
: point du cercle
: point du segment [# #]
: intersection du segment [# #] et du segment [# #]
: intersection du segment [# #] et de la droite
: intersection du segment [# #] et du cercle
: intersection du segment [# #] et du cercle # (# est l'autre point)
: symétrique du point # par rapport au point
: symétrique du point # par rapport à la droite
: bissectrice de l'angle formé par # #
angle formé par # # #
: point quelconque
: droite quelconque
: cercle quelconque
segment [# #]
triangle # # #

Annexe E.
Preuve de la satisfaction de la définition logique de la multifonction d'extension par sa définition opérationnelle.

Nous présentons dans cette annexe la preuve du théorème assurant que tout S* construit de façon opérationnelle appartient à la multifonction d'extension. Pour cela, il faut prouver que la fonction d'extension FE calculée est telle que

S* ∈ FE(S0,F,TIG,TEXT)

avec S* pourvue des propriétés suivantes :

(1) S* close et V(F) =V(S*) où V(E) est le vocabulaire de la formule E.
(2) TIG, HYPNU ⊢F ⇒ S* ⇒ S0
(3) TEXT, TIG⊢S* ⇒ S0

1. Preuve de (1)

(1) est vérifiée si

• S* est close.

Puique S0= S est close par définition de la traduction d'une spécification de CDL en une formule LDL, il suffit pour montrer que S* est close de montrer que tout objet ajouté à partir d'un axiome de TEXT au vocabulaire de Si pour construire Si+1 est une constante.

Puisque V(Si+1) = V(Si) ∪ {x, z1,z2,…,zn} et x, z1,z2,…,zn ∈ V(F)\V(Si) alors les objets ajoutés appartiennent à V(F). Puisque F est close par définition de la traduction d'une construction de SCL en une formule de LDL, alors tout objet ajouté à partie d'un axiome de TEXT est une constante.

• V(S*) = V(F).

Cette condition est vérifiée par la définition de la suite S0, ..., Sn.
Sn = S* si V(S*) = V(F).

2. Preuve de (2)

(2) est vérifiée si

Comme on suppose préalablement que l'expression
∀Φ (TIG, HYPNU ⊢F ⇒ Φ(S))
est vérifiée, alors ils suffit de montrer que, étant donné un Φ,

• TIG ⊢S* ⇒ Φ(S)

La suite est construite par ajouts successifs de littéraux à Φ(S). Autrement dit, S* = Φ(S) ∪ Propsupp, où Propsupp est l'ensemble des litéraux ajoutés à Φ(S) pour obtenir S*. On a donc

Propsupp = $\bigcup_{i=1}^{n}$ Propmin$_i$.

Il suffit alors de montrer que l'ajout de Propmin$_j$ à une étape de construction de S* n'introduit pas d'incohérence, pour montrer que S* implique Φ(S).

C'est le cas si TEXT n'est pas incohérente avec TIG. Comme TIG et TEXT sont par définition cohérente avec La Géométrie, alors TIG et TEXT ne sont pas incohérentes entre elles.

• TIG $\not\vdash_F \Box S^*$

Comme $F \Box \Box(S)$, il suffit de montrer que $F \not\vdash_{TIG}$ Propsupp. Comme pour tout axiome on vérifie

$F \not\vdash_{TIG}$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

alors $F \not\vdash_{TIG}$ Propsupp.

3. Preuve de (3)

(3) TEXT, TIG $\not\vdash S^* \Box S_0$

TEXT est composé d'axiomes de la forme

$\Box y_1,y_2,\ldots,y_n$, Cond($y_1,y_2,\ldots,y_n$) $\Box$

$\Box x, z_1,z_2,\ldots,z_n$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

Cela signifie que

(a) TEXT $\not\vdash_{TIG} \Box y_1,y_2,\ldots,y_n$, Cond($y_1,y_2,\ldots,y_n$) $\Box$

$\Box x, z_1,z_2,\ldots,z_n$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

La forme opératoire des axiomes de TEXT, soit

si

$S_i \not\vdash_{TIG}$ Cond($y_1,y_2,\ldots,y_n$)

et

$F \not\vdash_{TIG}$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

avec

$y_1,y_2,\ldots,y_n \Box V(S_i)$ et $x, z_1,z_2,\ldots,z_n \Box V(F)\backslash V(S_i)$

alors

$S_{i+1} = S_i \Box$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

et $V(S_{i+1}) = V(S_i) \Box \{x, z_1,z_2,\ldots,z_n\}$

permet d'assurer que tout S_{i+1} est construit si

(b) $S_i \not\vdash_{TIG} \Box y_1,y_2,\ldots,y_n$, Cond($y_1,y_2,\ldots,y_n$)

Par modus ponens sur (a) et (b), on tire

S_i, TEXT $\not\vdash_{TIG} \Box x, z_1,z_2,\ldots,z_n$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

ce qui donne par skolémisation

S_i, TEXT $\not\vdash_{TIG}$ Propmin($cx, cz_1,cz_2,\ldots,cz_n, y_1,y_2,\ldots,y_n$) où $cx, cz_1,cz_2,\ldots,cz_n$, sont des constantes de $V(F)\backslash V(S_i)$

Comme $S_{i+1} = S_i \Box$ Propmin($x, z_1,z_2,\ldots,z_n, y_1,y_2,\ldots,y_n$)

alors

S_i, TEXT $\not\vdash_{TIG} S_{i+1}$

ce qui équivaut à

TIG, TEXT $\not\vdash S_i \Box S_{i+1}$

d'où du fait que $S_n = S^*$ et par transitivité de l'implication

TIG, TEXT $\not\vdash \Box(S) \Box S^*$

Annexe F.
Mise en œuvre de la traduction simple de CDL.

1. Analyse lexicale.

L'analyse lexicale construit la suite des lexèmes d'un texte source CDL.

Elle reconnaît les mots réservés (point, droite, demi_droite, segment, cercle, distance, memesens, invsens, centre, rayon et e), les séparateurs ("," "(" ")" "[" "]" "|") , les symboles de prédicats ("//" "=" "!-" "<" "1/2" "2") et les identificateurs CDL. Chaque identificateur CDL est transformé en identificateur LDL par la fonction ajouter_a définie par

ajouter_a :

Identificateurs CDL -> Identificateurs LDL

i -> (Conc(si M(PC(t)) alors "A" sinon "a", i)

où Conc, M et PC sont des fonctions définies en annexe C.

L'analyse lexicale est basée sur l'utilisation de la primitive read_unit de PrologII+. Elle permet de reconnaître des unités lexicales répondant à la syntaxe de PrologII+ [Prologia 92] [Desmoulins 90]. Une transformation simple permet d'obtenir les lexèmes voulus.

A la fin de l'analyse lexicale, le texte intermédiaire produit se trouve sous la forme d'une liste Prolog des lexèmes du texte source.

2. Analyse des identificateurs.

L'analyse des identificateurs a pour objet de construire deux ensembles :

- d'une part un "environ", ensemble des atomes de typage associés à un identificateur et déduits de la spécification.
- d'autre part un dictionnaire des termes identifiés. Pour chaque terme t tel qu'il existe un littéral de la forme t = i ou i = t, avec i identificateur, le dictionnaire contient le doublet <t,i>.

L'analyse des identificateurs est faite par synthèse, suivant la syntaxe exposée au deuxieme chapitre, 2.2.2. De façon classique, à chaque non-terminal de la grammaire correspond un paquet de règles Prolog.

Les erreurs de syntaxe sont récupérées en ajoutant une règle pour le non-terminal <LITTERAL>. Cette règle permet de reprendre l'analyse sur la virgule suivante. La suite de lexème non reconnue jusqu'à cette virgule n'est pas prise en compte dans la traduction et un message d'avertissement prévient l'utilisateur de cette non prise en compte.

2.1. Construction de l'environ.

L'environ est constitué d'une liste d'atome LDL de déclaration. Cet environ est hérité au cours de l'analyse syntaxique. Les règles syntaxiques sur lesquelles un atome de déclaration est ajouté à l'environ courant sont les suivantes :

<ATOME_TYPAGE > ::= point (<identificateur>)
::= droite (<identificateur>)
::= demi_droite (<identificateur>)
::= segment (<identificateur>)
::= cercle (<identificateur>)
::= distance (<identificateur>)
<ATOME_PROPRIETE> ::= <OBJET> = <OBJET_TERME>
::= <OBJET_TERME> = <OBJET>

ainsi que les règles dérivant sur <identificateur> (sauf <OBJET>, <ENS_POINTS> et <ENS_PTS_ALIGNES> pour lesquelles le type de l'identificateur est ambigu). Dans ce dernier cas, le type de l'objet est hérité ainsi que la composition des arguments de typage si possible.

Exemples d'atomes de typage ajoutés suivant les règles de la grammaire et le texte sources CDL :

Règles dérivant sur le non-terminal pour lequel l'atome de typage est ajouté	texte source	atome ajouté
<ATOME_TYPAGE > ::= demi_droite (<identificateur>)	demi_droite(H)	demi_droite(AH, p, l)
<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>	S = [A centre(K)]	segment(AS,AA,p,l)
<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME> <OBJET_TERME>::=<POINT_TERME> <POINT_TERME>::= centre (<CERCLE>) <CERCLE> ::= <identificateur>	P = centre(K)	cercle(AK,AP,r)
<POINT> ::= <identificateur>	P e L	point(AP)

La construction de l'environ est basée sur l'utilisation des variables Prolog et de l'unification. En effet, tous les arguments d'un atome de typage ne sont pas connus à un moment donné de l'analyse. Chacun des arguments inconnus d'un atome de typage est représenté par une variable Prolog.

L'ajout d'un atome de typage a à un environ e est réalisé de la façon suivante :

• si un atome de typage a' de e s'unifie avec a, a' est remplacé par l'unification de a et a' dans l'environ e.

Par exemple, à partir de chacun des deux littéraux CDL
S = [A centre(K1)]
S = [centre(K2) B]
on déduit les littéraux de typage
segment(AS, AA, p1,l1)
segment(AS, p2, AB, l2)
qui s'unifient en un littéral unique
segment(AS, AA, AB,l1).

• si le prédicat et l'identificateur d'un atome de typage de e s'unifient avec le prédicat et l'identificateur de a, mais pas une partie des autres arguments, alors l'objet i est défini de deux façons différentes. Une erreur est enregistrée.

• *si l'identificateur d'un atome de typage de e s'unifie avec le prédicat de a, mais pas leurs prédicats, alors il existe deux objets de types différents référant au même identificateur. Une erreur est enregistrée.*

• si l'identificateur de a ne s'unifie avec aucun identificateur de e, l'atome a est ajouté à e.

Un traitement spécial est appliqué au segment. En effet, le prédicat de typage d'un segment doit répondre à une contrainte d'ordre sur les deux arguments représentant les extrémités du segment (voir deuxième chapitre, 2.1.1). Mais cet ordre n'est pas défini sur les variables. Il faut alors étendre l'unification Prolog pour le cas des segments.

Soit `unification(s1,s2)` cette fonction d'unification étendue.

unification(segment(i,p1,p2,l1),segment(i,p3,p4,l2)) = segment(i,p5,p6,l3) tel que

- si p1 et p2 ou p3 et p4 sont des variables, p5 est l'unification Prolog de p1 et p3; p6 est l'unification Prolog de p2 et p4.
Exemple : unification([x,y],[z,t]) = [x,y]

- si p1, p2, p3 et p4 sont des constantes et que p1 = p3 et p2 = p4 ou p1=p4 et p2=p3,
p5 = Premier(p1,p2)
p6 = Second(p1,p2)

Exemples :

si un seul au plus de p1 et p2 et un au plus de p3 et p4 sont des variables
si p1 et p3 sont unifiables et p2 et p4 sont unifiables ou
p1 et p4 sont unifiables et p2 et p3 sont unifiables
p5 = Premier(p1,p2)
p6 = Second(p1,p2)
Exemples :
unification([A B],[A B]) = [A B]
unification([B A],[A B]) = [A B]
unification([B A],[B A]) = [A B]
unification([A x],[A B]) = [A B]
unification([B x],[A B]) = [A B]
unification([A x],[B y]) = [A B]

- dans les autres cas l'unification n'est pas définie. Cela indique une erreur sur les extrémités des segments.
Exemples :
[A B] et [A C]
[A B] et [C A]
[A B] et [C D]
[A B] et [C x]

Dans la construction de l'environ, l'utilisation des variables Prolog et de l'unification permettent de propager instantanément les modifications de l'environ. Cela constitue un des intérêts de l'utilisation de Prolog pour notre mise en œuvre.
Une fois l'ensemble de la liste de lexèmes parcourue, l'environ peut contenir des arguments d'atome de typage non encore instanciés. Chacun de ces arguments

correspond à un objet implicite (créé par la traduction). Avant d'effectuer la traduction naïve, un identificateur est créé pour instancier chacune de ces variables.

2.2. Construction de l'ensemble des termes identifiés.

La construction de cet ensemble est effectuée parallèlement à la construction de l'environ. Elle s'effectue uniquement sur les règles de grammaire suivantes :

<ATOME_PROPRIETE> ::= <identificateur> = <OBJET_TERME>
::= <OBJET_TERME> = <identificateur>
::= <OBJET_TERME> = <OBJET_TERME>

Notons ET l'ensemble courant des termes identifiés.

Pour chaque non-terminal <OBJET_TERME> la suite SL de lexèmes CDL analysée est synthétisée.

Sur la troisième règle, notons SL1 le terme synthétisé sur le premier non-terminal <OBJET_TERME> et SL2 le terme synthétisé sur le second non-terminal <OBJET_TERME>, l'ensemble de termes est ainsi construit :

• s'il n'existe ni doublet <SL1,i1> ni doublet <SL2,i2> dans l'ensemble de termes courant, le doublet <SL1,SL2> est ajouté à ET.

• s'il existe un doublet <SL1,i1> dans l'ensemble de termes courant, le doublet <SL2,i1> est ajouté à l'ensemble courant par l'opération Ajout_doublet ainsi définie:

-s'il existe un doublet <SL2,SL3> ou <SL3,SL2>,

Ajout_doublet(<SL2,i1>,ET)⇒ Ajout_doublet(<SL3,i1>,ET-{<SL2,SL3>}) ∪ {<SL2,i1>}

sinon Ajout_doublet(<SL2,i1>,ET)⇒ ET ∪ {<SL2,i1>}

Une erreur est enregistrée s'il existe un doublet <SL2,i2> avec i1≠i2.

• symétriquement, s'il existe un doublet <SL2,i2> dans l'ensemble de termes courant, le doublet <SL1,i2> est ajouté à l'ensemble par l'opération Ajout_doublet .

• s'il existe un doublet <SL1,i1> et un doublet <SL2,i2> dans l'ensemble de termes courant, deux cas sont possibles :

- i1≠i2 : une erreur est enregistrée.

- i1 = i2 : aucun doublet n'est ajouté, le littéral est alors redondant. Une mise en garde est donnée à l'utilisateur.

Sur les deux premières règles, le doublet <SL,i> est ajouté à l'ensemble (i est l'identificateur Prolog traduction du non-terminal <identificateur>). Si un doublet <SL,i'> avec i≠i' est déjà présent, une erreur est enregistrée.

A la fin de l'analyse, les doublets de la forme <SL1,SL2>, où SL2 n'est pas un identificateurs, sont supprimés de ET.

3. Traduction simple à partir de l'environ et de l'ensemble des termes identifiés.

Une fois l'environ et l'ensemble des termes identifiés construits, la traduction simple s'effectue par un nouveau parcours de la liste de lexèmes, en suivant la même démarche que pour l'analyse des identificateurs. La récupération d'erreurs s'effectue de la même façon, sans toutefois produire de messages d'avertissement pour l'utilisateur (car les messages d'erreurs de syntaxe ont déjà été signalés).

Cette traduction suit la sémantique que nous avons donnée en annexe C . Comme pour l'analyse des identificateurs, à chaque non-terminal de la grammaire correspond un paquet de règles Prolog du programme de traduction.

Les fonctions définies en annexe C sont mises en œuvre de la façon suivante :

- la fonction NOM est réalisée par des recherches dans l'ensemble de termes identifiés. Si la recherche échoue, un identificateur interne nouveau est créé.

- la fonction TYPAGE est réalisée par des recherches dans l'environ. Si la recherche échoue, une erreur est enregistrée (type de l'identificateur inconnu. Par exemple, O dans P e O sans autre référence à O).

- les fonctions ORIGINE, SUPPORT, PREMIERE_EXTREMITE, SECONDE_EXTREMITE, CENTRE et RAYON sont réalisées en utilisant les deux ensembles. Si un terme n'est pas dans ET, un atome de typage est créé pour ce terme et chacune de ces composantes si besoin (ainsi que les identificateurs nécessaires).

Annexe G.
Mise en œuvre de la communication avec Cabri-géomètre.

Dans cette annexe, nous présentons tout d'abord les principe de la première mise en œuvre d'une communication entre TALC-Élève et Cabri-géomètre et ses défauts. Nous présentons ensuite les principes de la seconde mise en œuvre permettant d'éviter ces problèmes grâce à l'utilisation de la dernière version du système d'exploitation du Macintosh. Enfin nous présentons les fonctions de communications offertes par la version spéciale de Cabri-géomètre lui permettant d'être serveur d'une autre application.

1. Première mise en œuvre.

Une première mise en œuvre a été réalisée sur le système d'exploitation 6.xx de Macintosh. Elle permettait d'appeler TALC-Élève depuis Cabri-géomètre. Concrètement, cela signifiait que l'élève se plaçait de lui même dans Cabri-géomètre puis actionnait une commande qui donnait la main à TALC pour la vérification. Ce protocole de communication avait de nombreux inconvénients :

- TALC ne pouvait vérifier qu'un objet construit était égal à un autre qu'une fois l'ensemble de la figure construite (quand l'élève demandait la vérification) et non incrémentalement. Il fallait alors demander à l'élève de revenir dans Cabri-géomètre pour supprimer l'objet en question, alors que cela pouvait être un des premiers objets construits, remettant dans ce cas en cause toute la suite de la construction.
- l'ensemble du protocole de communication était basé sur les manipulations par l'élève du système. Il n'était donc pas très sur et occasionnait un "bruit" important par rapport à la tâche didactique demandée.
- la communication se faisait par l'intermédiaire d'un fichier, entraînant des problèmes d'accès partagés.

2. Seconde mise en œuvre.

Avec l'introduction du système 7.xx, une véritable communication entre application est possible, via l'utilisation des "Apple-events" [Apple 93].

Le principe en est le suivant :

- l'application active émet un message (apple event) en direction d'une application en attente.
- l'application réceptrice devient l'application active et celle qui a envoyé le message se suspend.
- l'application réceptrice traite alors le message.

Les Apple-events introduisent ainsi une communication par coopération entre applications : l'application qui est active le reste tant qu'elle le veut.

Sur ce principe, nous avons défini une relation client-serveur entre TALC-Élève et Cabri-Géomètre, où TALC-Élève est le client et Cabri-Géomètre le serveur. Pour la mettre en œuvre, nous avons réalisé une interface de communication en deux parties :

• La réalisation du protocole de communication entre PrologII+ et Cabri-géomètre est mise en oeuvre en C et utilise les ressources de messageries du système 7.xx de Macintosh.

• L'utilisation des commandes de communication se fait en Prolog au sein du module gérant l'interface utilisateur de TALC-Élève.

Ces deux parties ont été réalisées à l'occasion d'un stage de fin d'étude [Senet 93].

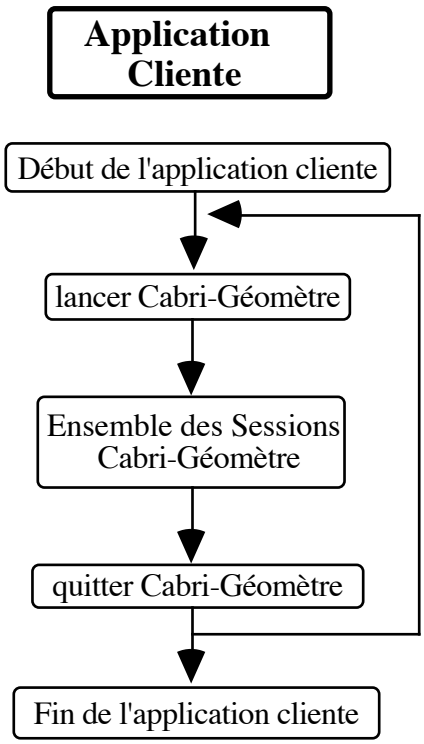
Pour que Cabri-géomètre puisse être serveur d'une autre application, les developpeurs de Cabri-géomètre ont mis au point une version spéciale permettant la communication via les Apple-events de même qu'un premier prototype à partir d'Hypercard. C'est cette version de Cabri-géomètre que nous utilisons [Tessier 94]. La réalisation s'est inspirée de celle réalisée sur Hypercard.

3. La version de Cabri-géomètre serveur d'une autre application.

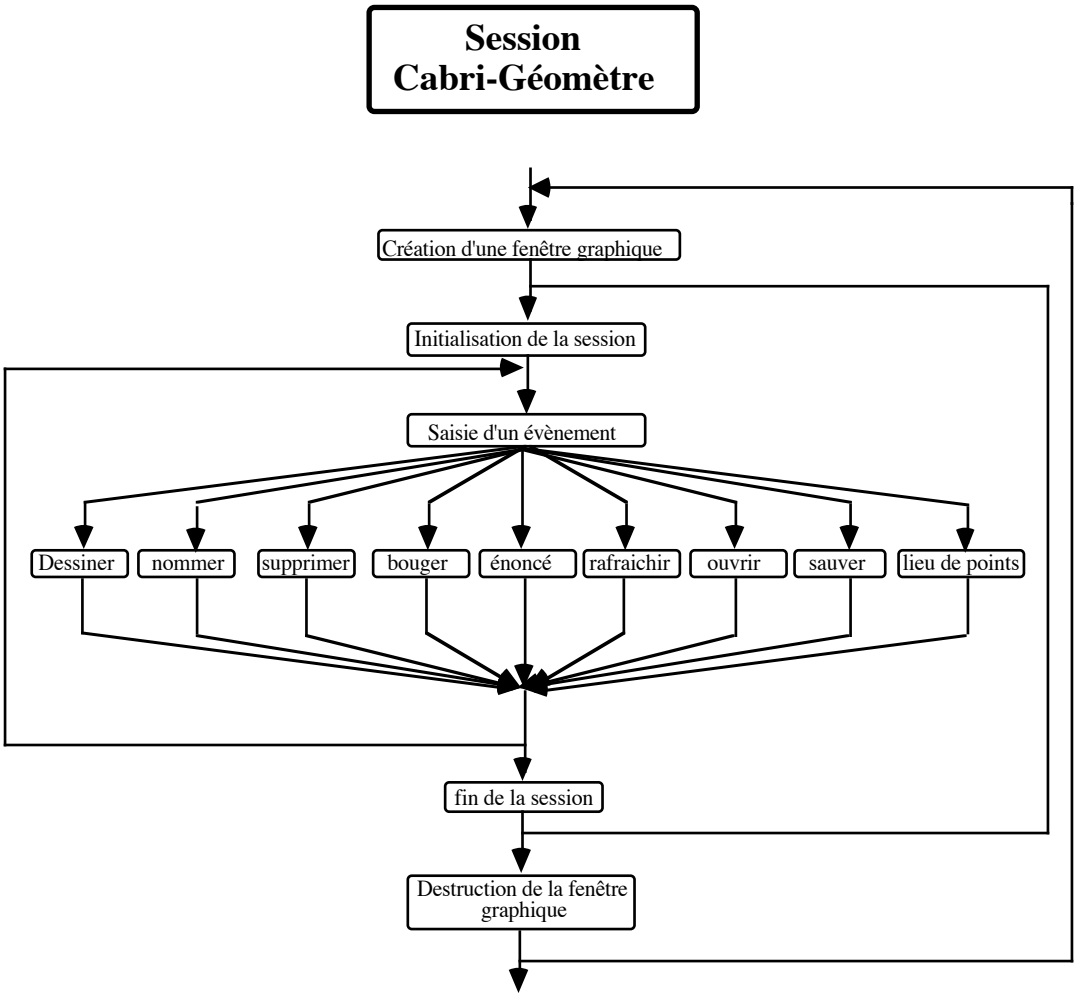
La version spéciale de Cabri-géomètre (en cours de développement) lui permettant d'être serveur d'une autre application propose actuellement les commandes ci-dessous :

C1 : initialisation d'une session Cabri-Géomètre.
C2 : clôture d'une session Cabri-Géomètre.
C3 : dessiner un objet.
C4 : nommer un objet.
C5 : supprimer un objet.
C6 : bouger un objet.
C7 : énoncé de la figure.
C8 : rafraîchir la figure.
C9 : ouvrir une figure.
C10 : sauver une figure.
C11: dessiner le lieu des points.

La communication entre Cabri-géomètre et une autre application suit le schéma suivant, du point de vue de l'application cliente :



Une session est une suite de requêtes (identifiées par C3, ..., C10) du client à Cabri-géomètre commençant par l'initialisation (C1) et se terminant par la clôture de la session (C2). Comme dans toute application interactive, chacune des requêtes est commandée de façon asynchrone par l'utilisation d'un menu ou une manipulation de la souris. Une session suit alors le schéma suivant :



Pour chacune des commandes C1 à C10, nous donnons brièvement l'effet réalisé (pour plus de détails, voir [SENET 93]).

C1 : initialisation d'une session Cabri-Géomètre.

L'intialisation consiste à fournir à Cabri-Géomètre la référence de la fenêtre graphique sur laquelle il doit effectuer des interactions. Cette fenêtre appartient à l'application cliente qui doit la gérer.

Cette commande nous oblige à ne permettre pour l'instant à l'élève de ne disposer que d'une fenêtre de construction à la fois.

C2 : clôture d'une session Cabri-Géomètre.

La clôture d'une session signifie que Cabri-géomètre ne prend plus la fenêtre courante comme fenêtre de travail. Elle est utile dans deux cas :

- soit l'utilisateur désire ouvrir une autre fenêtre.
- soit il demande de quitter TALC-Élève.

C3 : dessiner un objet.

Cette commande permet de créer et de construire un objet.

Elle permet à l'utilisateur de disposer de la même interface que dans Cabri-Géomètre dans son utilisation classique, c'est à dire que l'on dispose de l'aide, de la possibilité d'abandonner l'action en cours, ainsi que des messages d'erreurs fournis par Cabri-Géomètre .

Les créations disponibles sont :

Point de base.
Droite de base.
Cercle de base.
Segment.
Droite passant par deux points.
Triangle.
Cercle défini par centre et point.

Les constructions disponibles sont :

Point sur objet.
Intersection de deux objets.
Milieu.
Médiatrice.
Droite parallèle.
Droite perpendiculaire.
Centre d'un cercle.
Symétrique d'un point.

Elles correspondent évidemment chacune à une primitive de construction de SCL.

Le résultat de la création ou de la construction est l'énoncé de l'objet construit. La description de cet objet est la même que celle que l'on peut retrouver dans l'énoncé complet de la figure. C'est cet énoncé qui est traduit en LDL et ajouté à la traduction courante.

La version actuelle de Cabri-Géomètre dont nous disposons autorise toute commande de construction, même si la construction correpondante est impossible (par exemple le centre d'un cercle si aucun cercle n'a été construit) alors que ce n'est pas le cas dans la version commerciale de Cabri-Géomètre.

C4 : nommer un objet.

Cette commande nous permet de nommer les objets construits. Contrairement aux autres, elle n'est pas atomique, c'est à dire que plusieurs objets peuvent être nommés en utilisant une seule commande. Ce mode de travail n'est pas entièrement satisfaisant car nous ne pouvons avoir un contrôle immédiat sur ce que l'utilisateur vient de faire. Nous aimerions pouvoir vérifier si le nom que l'utilisateur vient de donner à un objet est conforme à la syntaxe que l'on attend.

De plus il serait intéressant que cette commande nous renvoie systématiquement le couple <ancien nom, nouveau nom>. La possibilité d'avoir le couple ancien nom, nouveau nom pouvant nous permettre d'éviter des problèmes comme le renommage circulaire d'objets (l'objet de nom A prend le nom B et vice-versa) dans le cas où cette commande ne serait pas atomique.

C5 : supprimer un objet.

Cette commande permet la suppression d'un objet et des objets dépendants s'il y en a.

Dans sa version actuelle cette commande retourne l'énoncé de l'objet supprimé et signale s'il y a eu ou non d'autres objets supprimés, mais ne donne pas l'énoncé de ces derniers s'il y en a.

C6 : bouger un objet.

Dans Cabri-géomètre, l'utilisateur peut à tout moment saisir un objet et le déplacer, modifiant ainsi la figure tout entière. Cette possibilité de bouger n'est pas disponible pour tout objet (l'objet doit avoir un degré de liberté non nul). De ce fait l'interface affiche un pointeur spécial (une petite main) quand la souris se trouve au dessus d'un objet déplaçable.

Dans TALC-Élève, la requête Bouger est envoyée à Cabri-géomètre quand un clic se produit sur la fenêtre. Si le curseur se trouve sur un objet déplaçable, l'interaction est enclenchée, sinon rien ne se passe. C'est une différence du point de vue de l'utilisateur entre TALC-Élèveet Cabri-géomètre. Elle est due au fait que la gestion du curseur dépend de TALC-Élève. Une solution consisterait à constamment demander à Cabri-géomètre de redessiner le curseur. Elle n'a pas été retenue car elle entraîne des incessants va-et-vient entre les deux applications qui doivent prendre la main tour à tour pour coopérer.

C7 : énoncé de la figure.

Cette commande permet d'obtenir l'énoncé complet de la figure construite sous la forme d'une chaîne de caractères.

C8 : rafraîchir la figure.

Cette commande permet de redessiner la figure.

En effet comme la gestion de la fenêtre étant laissée à l'application cliente, il est nécessaire de pouvoir faire redessiner la figure, par exemple si la fenêtre graphique a été occultée par une autre fenêtre

C9 : ouvrir une figure.

Cette commande permet d'ouvrir une figure.

La figure que l'on ouvre est de type Cabri-Géomètre. Cette figure a été, soit enregistrée par Cabri-Géomètre dans son utilisation classique, soit sauvegardée à l'aide de la commande ci dessous.

C10 : sauver une figure.

Cette commande permet de sauvegarder la figure en cours.

La figure sauvegardée est de type Cabri-Géomètre et elle pourra être rouverte soit par la commande ci dessus, soit par Cabri-Géomètre dans son utilisation classique.

Comme PrologII+ n'offre pas de primitives permettant de communiquer avec d'autres applications via les apple events, l'appel à chacune des requêtes C1 à C10 est réalisé par l'intermédiaire de ressources de code écrites en C. PrologII+ sur Macintosh permet en effet d'étendre les primitives existantes par l'ajout de ressources de code au langage. Une fois ces ressources de codes ajoutées (statiquement ou dynamiquement), chaque primitive nouvelle peut être appelée comme n'importe quelle autre primitive. Un paramètre de la primitive permet si besoin de retourner un code d'erreur si la requête n'a pu être réalisée correctement.