Conception et mise en oeuvre d'un système déclaratif de géométrie dynamique

Stéphane Channac

Thèse

présentée par

Stéphane CHANNAC

pour obtenir le titre de Docteur
de l'Université Joseph Fourier - Grenoble 1
(arrêtés ministériels du 5 juillet 1984 et du 30 Mars 1992)
Spécialité : **Informatique**

Conception et mise en œuvre d'un système déclaratif de géométrie dynamique

---

Date de soutenance : 7 juin 1999

Composition du jury :

Président :	Paul Jacquet
Rapporteurs :	Richard Allen
	Michel Rueher
Examinateur :	Jean-Marie Laborde
Directeur :	Laurent Trilling

Thèse préparée au sein du
Laboratoire Logiciels, Systèmes, Réseaux – IMAG

à ma famille,

à la mémoire de Stéphane Monroe.

Remerciements

Je tiens à remercier les membres du jury :

M. Richard Allen, Professeur au Saint Olaf College dans le Minnesota (USA), d'avoir bien voulu apporter son jugement sur cette thèse. Je tiens à lui exprimer toute ma reconnaissance pour l'intérêt qu'il porte à mon travail depuis notre première rencontre et pour ses encouragements sans cesse renouvelés aux cours des nombreuses discussions passionnantes que nous avons eues.

M. Michel Rueher, Professeur à l'Université de Nice, de m'avoir fait l'honneur d'apporter son jugement sur ce travail. Je tiens à lui exprimer toute mon gratitude pour sa grande franchise et sa grande disponibilité. Je le remercie aussi pour l'agréable accueil qui m'a été fait lors de ma visite à Nice et les commentaires constructifs et très profitables qui m'ont été faits à cette occasion.

M. Paul Jacquet, Professeur à l'Institut National Polytechnique de Grenoble et directeur du laboratoire LSR, pour l'honneur qu'il me fait de présider ce jury. Je tiens à lui exprimer toute ma reconnaissance pour l'intérêt qu'il porte à mon travail, pour m'avoir accueilli au sein de son laboratoire et pour toute la confiance qu'il m'a témoignée au cours de cette entreprise.

M. Jean-Marie Laborde, Directeur de recherche au Laboratoire IMAG - Liebniz et responsable du projet Cabri-Géomètre, pour le grand honneur qu'il me fait de participer à ce jury. Je tiens à lui exprimer toute mon admiration pour le remarquable travail qu'il a accompli, conduisant à ce que je considère comme le modèle de logiciel d'enseignement de la géomètrie : Cabri. Et je le remercie également pour les conseils et les encouragements qu'il a marqués à ma égard.

et surtout M. Laurent Trilling, Professeur à l'Université Joseph Fourier, pour l'enthousiasme, la disponibilité, les conseils, l'aide et la gentillesse dont il a fait preuve tout au long de ces quatre années. J'espère que les résultats présentés dans ce manuscrit lui permettront de faire de beaux Rev.

Je tiens aussi à remercier :

Les membres de l'équipe PLIAGE et plus particulièrement Pierre qui m'a poussé à approfondir mes connaissances des fonctionnalités d'Emacs et de LaTeXpar ses innombrables questions et surtout pour nos inoubliables débats. Je n'oublie pas les autres membres de l'équipe : Denis, Nicolas, Sébastien, Pascal, Paul, Gérard, Jeanne et Michel qui ont su entretenir un climat de bonne humeur.

L'équipe administrative du laboratoire LSR composée de Liliane, Martine, Solange et Rachèle et l'équipe technique de l'ENSIMAG composée de François et Bernard pour la disponibilité et l'efficacité dont ils font preuve tous les jours.

Mes amis ... ils se reconnaîtrons ... Damien pour m'avoir si souvent sorti des méandres de l'informatique préhistorique dans laquelle j'évoluais, Arnaud sans qui cette thèse n'aurait probablement pas vue le jour, Jérôme pour son initiation à ce sport merveilleux qu'est l'escalade et pour mes 15 tendinites qui en ont résulté, Armelle pour m'avoir si bien fait découvrir ce merveilleux pays qu'est le Canada, et surtout Angélique, Emmanuel et Carol.

Et surtout la famille Marié au grand complet en commençant par Delphine, Florence, Malorie, Audrey, Claude et Danny pour sa gentillesse et son soutien ... sans oublier Alain et Patrice.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le travail *qui est présenté ici s'inscrit originellement dans le contexte des environnements informatisés d'apprentissage avec ordinateur. Plus spécifiquement, il s'intéresse à l'enseignement de la géométrie plane dans les collèges et les lycées.*

L'avènement dans les années 90 de logiciels tels que Cabri-Géomètre [Lab95] a conduit à la définition d'une nouvelle sorte de géométrie, la géométrie dynamique *[CG96]. Cette notion a pris son ampleur en particulier grâce à la prise en compte des concepts de manipulation directe [Nan90] par ces logiciels. Ceux-ci autorisent les utilisateurs à construire des figures et à les animer à l'aide de la souris tout en respectant continuellement les propriétés définies.*

La construction de telles figures dynamiques *est devenue une nouvelle activité pédagogique [Lab93, SG94, Cup96]. Elle demande aux élèves de construire des figures comportant un certain nombre de propriétés géométriques à partir de certains ensembles d'objets de base en usant des instruments classiques de la géométrie comme la règle et le compas.*

L'exemple ci-dessous s'inspire d'une situation présentée en classe de $5^{ième}$/$4^{ième}$ [Cab94].

Exemple Introduction à la géométrie dynamique

A partir d'un triangle quelconque ABC, construire un point P tel que les triangles ABP et ACP soient des triangles isocèles. La construction doit permettre d'obtenir des dessins si on déplace A, B ou C.

Une manière de résoudre cet exercice de construction est de s'appuyer sur la notion de médiatrice. En construisant le point P à la fois sur la médiatrice du segment [A, B] et sur la médiatrice du segment [A, C], il s'ensuit que le point P reste d'une part à égale distance des points A et B, et d'autre part à égale distance des points A et C ; et ce quelles que soient les déformations subies par le triangle ABC.

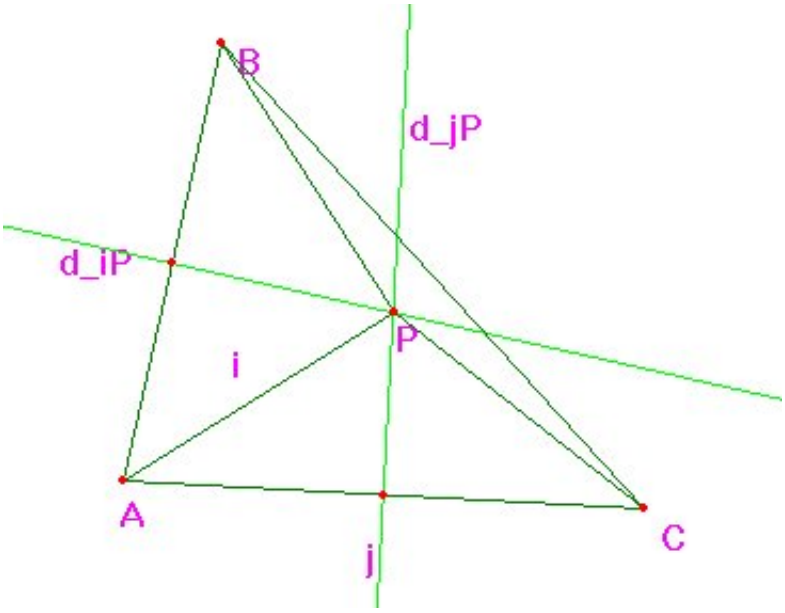

FIG. *0.1* – Introduction à la géométrie dynamique.

Une construction de cette figure suit les étapes suivantes :

1. Créer les points A, B et C.
2. Construire à partir de ces points les segments [A, B], [A, C] et [B, C].
3. Construire les points i et j respectivement milieu des segments [A, B] et [A, C].
4. Construire la droite d_{iP} perpendiculaire à au segment [A, B] et passant par le point i.
5. Construire la droite d_{jP} perpendiculaire à au segment [A, C] et passant par le point i.
6. Construire le point P à l'intersection des droites d_{iP} et d_{jP}.
7. Construire les segments [A, P], [B, P] et [C, P].

Si A, B ou C est déplacé, un système comme Cabri-Géomètre, produit effectivement un dessin correct car la construction est effectuée à partir des points A, B et C arbitraires.

Pour faciliter la construction de telles figures, une méthodologie [ACT98] consiste à étudier les figures dynamiques ayant la même *spécification géométrique*, c'est-à-dire

possédant les mêmes éléments reliés entre eux par les mêmes propriétés, mais n'ayant pas la même *spécification dynamique*, c'est-à-dire construites à partir d'un autre ensemble d'objets de base. L'objectif de cette démarche est par exemple la recherche de lieux ou d'invariants conduisant à la construction désirée, et d'une manière générale l'"exploration" d'une figure dynamique.

Problématique

De notre point de vue, une manière de faciliter l'exploration de figures dynamiques consiste à dissocier les aspects géométriques des aspects dynamiques. La spécification d'une figure dynamique est composée d'une part des éléments qui constituent la figure et des propriétés qui les lient entre eux, et d'autre part d'un ensemble d'objets de base à partir desquels un système doit construire la figure. Dans cette optique, explorer une figure dynamique en étudiant les figures ayant la même spécification géométrique consiste à modifier sa spécification dynamique, c'est-à-dire à modifier l'ensemble des objets de base à partir desquels elle est construite. La problématique à laquelle nous faisons face consiste à définir un système de géométrie dynamique que l'on peut qualifier de déclaratif pour manipuler ainsi ces figures directement. Il se décline en trois volets.

- La définition d'un langage logique géométrique pour décrire des figures dynamiques. Sur ce point, notre objectif est d'étendre significativement les possibilités offertes par des langages comme celui du système GéoSpécif [Bou97] pour qu'il supporte des spécifications modulaires et récursives.

- L'intégration des concepts de manipulation directe dans la définition du système. Cette intégration concerne la définition de fonctionnalités liées à l'animation de la figure elle-même. Elle implique aussi la définition de fonctionnalités liées à la manipulation directe des spécifications géométriques et dynamiques de la figure.

- La conception de méthodes de résolution de contraintes géométriques. Il s'agit ici d'être capable de construire les figures spécifiées. Deux aspects sont à considérer. Le premier aspect concerne la puissance des méthodes de résolution proposées et leur complétude (définition des classes de figures complètement résolues). Le deuxième aspect concerne la rapidité de ces méthodes. C'est-à-dire qu'il faut être en mesure de (re)construire rapidement les figures dans la mesure où elle peuvent être déformées par l'utilisateur.

La conjugaison des solutions que nous proposons à ces problèmes doit conduire à la définition et à la mise un œuvre d'un micromonde déclaratif de manipulation directe permettant aux utilisateurs d'explorer librement, tant au niveau géométrique, qu'au niveau dynamique, des figures géométriques planes.

Démarche

La démarche que nous avons poursuivie commence par la définition du langage logique géométrique. Une solution à ce problème existe déjà dans un cadre déclaratif similaire, celui de la programmation logique avec contraintes. Elle consiste à autoriser la spécification de buts combinés sous la forme de clauses, c'est-à-dire qu'elle consiste à introduire l'implication dans les spécifications. Pour permettre la définition de spécifications modulaires et récursives, notre objectif est d'introduire une forme d'implication permettant de la même manière de définir des figures à l'aide de clauses. Pour interpréter de telles spécifications, un interpréteur "à la prolog" nous paraît être le meilleur choix pour une mise en œuvre efficace, facile à comprendre et simple.

Sur le plan de l'intégration des concepts de manipulation directe, notre démarche s'appuie sur ce qui nous semble être le meilleur exemple actuel et dont l'intérêt pédagogique n'est plus à prouver, c'est-à-dire Cabri-Géomètre [Lab95]. A la différence de ce dernier, nous manipulons non seulement un dessin animable qui représente d'une certaine façon une abstraction de la figure dynamique considérée, mais aussi d'une part la spécification géométrique de cette figure sous la forme d'une formule logique et d'autre part la spécification dynamique de cette figure exprimée par un ensemble d'objets de base. Deux objectifs nous apparaissent essentiels d'un point de vue interface. Le premier est d'assurer comme invariant la cohérence entre dessin et spécification géométrique. Le deuxième est de fournir des capacités équivalentes aux langages manipulant les deux représentations d'une figure dynamique (spécification et dessin).

En ce qui concerne le problème de la définition de méthodes de résolution de contraintes géométriques, l'étude d'approches existantes nous a conduit à la conclusion qu'elles sont complémentaires. C'est pourquoi, notre objectif est d'intégrer dans un environnement différents résolveurs, c'est-à-dire de définir une approche coopérative de la construction automatique de figures géométriques. On peut noter que cette approche s'inscrit dans un courant récent dit de "collaboration de solveurs". Pour ce faire, nous proposons ici une coopération entre différents résolveurs modélisés comme

des agents [1] dans une architecture concurrente [Sar93]. Un premier agent, appelé "agent linéaire", est basé sur l'algorithme d'élimination de Gauss. Un deuxième agent, appelé "agent quadratique", est basé sur un algorithme de résolution d'équations du second degré. Un troisième agent, appelé "agent intervalle", est basé sur un résolveur sur intervalles [Cle87] associé a des heuristiques d'énumération d'intervalles et des méthodes de contrôle de résolution. Issus de l'analyse du domaine d'application, trois autres agents sont proposés. Le premier de ces agents, appelé "agent de complétion de propriétés", a pour charge de rechercher et d'ajouter à la spécification de la figure des propriétés redondantes, c'est-à-dire de rechercher des propriétés non explicitement données, mais déductibles de la spécification. Il est bâti à l'aide d'heuristiques pour la recherche de constructions "linéaires" de spécifications, d'un algorithme d'élimination de contre-propriétés et enfin d'un démonstrateur automatique [Ost97a] s'appuyant sur une théorie de la géométrie. Le deuxième de ces agents est appelé "agent de complétion d'objets". Il a pour charge d'ajouter à la spécification de la figure des objets intermédiaires facilitant la détermination d'une construction de cette dernière. Il est constitué d'un algorithme basé sur la méthode des lieux [Car88]. Le troisième de ces agents est appelé "agent règle et compas" et a pour charge d'élaborer un plan de construction à la règle et au compas de la figure. Il se compose d'un algorithme de construction symbolique s'appuyant sur une théorie de la géométrie.

Nos préférences en matière de construction penchent en premier lieu vers les approches symboliques qui fournissent des solutions plus propres à l'animation, puis numériques exactes. Néanmoins, notre choix d'un agent basé sur un résolveur sur intervalles est symptomatique du fait que nous nous contentons éventuellement d'obtenir une construction même approchée, du moment que le dessin correspondant soit acceptable.

L'affirmation que ces trois objectifs peuvent être effectivement atteints constitue notre thèse. Nous la soutenons en proposant et réalisant un système coopératif de géométrie dynamique déclarative baptisé GDRev pour Géométrie Dynamique Réversible.

1. Ce concept d'agent est à prendre au sens de "processus" autonome réactif moins général que celui entendu dans le domaine de l'IA [WJ95].

Plan du manuscrit

Le plan adopté dans ce manuscrit pour décrire la conception et la mise en œuvre de GDRev se décompose en quatre parties.

- La première partie présente un état de l'art en matière de résolution de contraintes géométriques et de coopération de résolveurs.
  - Le chapitre 1 est consacré à la problématique de la résolution de contraintes géométriques. Nous présentons pour commencer le concept de géométrie dynamique et les principaux systèmes informatiques existants. Puis, les différentes méthodes de résolution de contraintes géométriques proposées jusqu'à présent sont étudiées, afin de situer GDRev dans ce domaine.
  - Le chapitre 2 présente les principales approches de la coopération de résolveurs proposées au sein de la communauté de la programmation logique et par contraintes.
- La deuxième partie porte sur la définition du système GDRev.
  - Le chapitre 3 présente informellement le système GDRev. L'architecture globale est présentée, ainsi qu'un exemple de scénario d'utilisation.
  - Le chapitre 4 décrit les langages logiques géométriques utilisés. Nous présentons le langage de base, puis les extensions permettant la définition de figures modulaires et récursives.
  - Le chapitre 5 propose des langages d'interface de manipulation directe déclaratifs. Suite à l'examen des principes des interfaces de manipulation directe, des propositions portant sur des fonctionnalités d'animation, de construction et d'autres, comme par exemple la vérification de propriétés, sont données.
  - Le chapitre 6 décrit la mise en œuvre des langages d'interface présentés dans le chapitre précédent. Nous décrivons le modèle de l'architecture de notre système, ainsi que les contraintes de mise en œuvre auxquelles nous avons dues nous plier. Nous explicitons les principales structures de données ainsi que les principaux algorithmes les utilisant. Pour finir, les aspects quantitatifs de cette mise en œuvre sont abordés.

- La troisième partie porte sur la définition de méthodes coopératives de résolution de contraintes géométriques.
  - Le chapitre 7 présente les différents agent définis. Ceux-ci sont appelés linéaire, quadratique, intervalle, complétion d'objets, complétion de propriétés, et règle et compas. Nous détaillons les objectifs de chacun d'eux, ainsi que leurs limites au travers d'exemples.
  - Le chapitre 8 décrit la manière dont les différents agents présentés précédemment coopèrent pour résoudre un système de contraintes géométriques.
  - Le chapitre 9 décrit la mise en œuvre des différents agents et de leurs moyens de coopération.
  - Le chapitre 10 présente les résultats obtenus à l'aide de GDRev pour la résolution d'un ensemble de 40 exercices de construction extraits de [But75], et pour la recherche d'un ensemble de 193 exercices de construction "linéaire" extraits de [Cho88].
- La quatrième partie porte sur une autre application de la résolution de contraintes géométriques dans le cadre de la validation de figures dynamiques produites par des élèves vis à vis d'un énoncé donné par un professeur.
  - Le chapitre 11 présente un prototype visant à valider des figures dynamiques et à construire automatiquement des exemples et contre-exemples géométriques dans le cas de figures erronées.

Enfin, le chapitre 12 conclut ce manuscrit en rappelant les contributions théoriques et expérimentales de ce travail, et propose des perspectives.

Ce manuscrit est complété des annexes suivantes :

- L'annexe A comporte l'ensemble des théorèmes utilisés par le démonstrateur automatique mis en œuvre dans le système pour les fonctionnalités de validation de propriétés.
- L'annexe B comporte les détails des exercices de construction de figures dynamiques proposés au système. Pour chacun, nous détaillons l'énoncé, les agents intervenant dans la résolution et le temps nécessaire à la construction.
- L'annexe C comporte les détails des exercices de construction automatiques de figures à partir d'un ensemble quelconque d'objets de base.

Avant d'entamer la lecture de ce manuscrit, il convient de signaler que les figures qu'il contient ont été produites à l'aide du système GDRev, sauf si le contraire est indiqué.

Première partie

Contraintes géométriques et coopération de résolveurs

Chapitre 1

Résolution de contraintes géométriques

De tout temps, l'homme s'est intéressé au problème de la construction de figures géométriques. Les grecs furent parmi les premiers à s'attaquer à ces problèmes à l'aide de la règle et du compas. Ils se heurtèrent à des constructions insolubles comme les célèbres problèmes de la quadrature du cercle (construire un carré d'aire égale de celle d'un cercle donné), de la duplication du cube (construire un cube de volume double de celui d'un cube donné) et de la trisection d'un angle (partager un angle donné en trois parties égales) [Car88]. Aujourd'hui, ce problème de la construction de figures géométriques se trouve au cœur de nombreux domaines. Dans le domaine de l'informatique, on le trouve par exemple en infographie lorsqu'il s'agit de modéliser une scène, en CAO (Conception Assistée par Ordinateur) lorsqu'il s'agit de modéliser l'objet de l'étude, ou enfin en EIAO (Environnements Interactifs d'Apprentissage avec Ordinateur) lorsqu'il s'agit pour les élèves de construire une figure répondant à une spécification donnée par un professeur. Dans cette dernière situation, l'apparition depuis quelques années de logiciels introduisant le concept de géométrie dynamique a d'une part modifié son approche et d'autre part ouvert de nouvelles perspectives.

Nous décrivons dans ce chapitre le domaine d'application de notre travail, à savoir les environnements de programmation géométrique qui regroupent les logiciels dédiés à la construction et à l'animation de figures. Nous présentons dans la section 1.1 la notion de géométrie dynamique. Nous décrivons dans la section 1.2 les principaux systèmes de géométrie dynamique. Nous explicitons dans la section 1.3 les différentes approches proposées pour la résolution de contraintes géométriques. Enfin, la section 1.4 conclue ce chapitre.

1.1 Géométrie dynamique

L'usage répandu de logiciels éducatifs d'apprentissage de la géométrie tels que Cabri-Géomètre [Bau90, Bel92, Lab95] et Geometer's Sketchpad [Jac95] a conduit à l'avènement de la notion de *géométrie dynamique* [CG96, Tri96a], consacrée par les deux sessions qui lui ont été dédiées au Congrès des sociétés savantes mathématiciennes américaines en janvier 1995 (MAA-AMS Joint Meeting, San-Francisco). En effet, de tels logiciels dépassent la simple transposition informatique du papier/crayon et des outils classiques de la géométrie comme la règle, l'équerre et le compas, et permettent à l'élève non seulement de construire une figure, mais aussi de l'animer par manipulation directe[1] tout en respectant continuellement les propriétés définies. Dans cette optique, une figure géométrique dynamique peut être considérée comme composée par :

- Une spécification géométrique, une figure géométrique au sens usuel, c'est-à-dire l'objet mathématique sous la forme d'une formule logique sous-jacente au dessin.
- Une spécification dynamique, exprimée par un ensemble d'objets animables, c'est-à-dire un ensemble d'objets de base.

La géométrie dynamique s'intéresse aux animations d'une figure, puisqu'il est attendu que, comme l'énonce Laborde C :

> *"Le tracé à l'écran d'un dessin attaché à un objet géométrique doit garder au cours du déplacement ses propriétés spatiales rendant compte des propriétés géométriques de cet objet, [...]"*
> [Lab93] (page 91)

Il s'ensuit une nouvelle notion de figure géométrique dans laquelle si des objets sont reliés entre eux par une propriété, alors cette propriété est conservée au cours de toute déformation de la figure.

Bien que comme le fait remarquer Capponi B :

> *"Cet aspect [...] change la nature du travail proposé aux élèves. Ce n'est plus un simple dessin qui doit être réalisé [...]"*
> [Cap93] (page 49)

1. Nous précisons ce concept de manipulation directe au chapitre 5.

force est de constater que ces changements sont bénéfiques. En effet, ils conduisent à une meilleure appropriation de la figure. Keaton M, professeur de mathématiques au S^t Mark's School of Texas, le confirme dans le bilan qu'il dresse après 3 années d'enseignement de la géométrie à l'aide de systèmes de géométrie dynamique.

> *"[...], with interactive geometry software and its ability to alter a figure dynamically to produce virtually hundreds of examples quickly, the students were able to make significant discoveries [...]"*
> [Kea95] (page 64)

Cette capacité d'animer certains objets de la figure a conduit les utilisateurs de systèmes de géométrie dynamique à s'intéresser à la notion de lieux [Pet90, SG95]. Le lieu d'un objet est l'ensemble des positions prises par cet objet au cours d'une animation. L'exemple 1.1 illustre cette notion de lieu. Ce qui est virtuellement difficile pour bon nombre de personnes, c'est-à-dire d'imaginer les positions d'un objet au cours d'une animation, devient facile à appréhender grâce à ces systèmes puisqu'il suffit de déplacer un objet de la figure et d'observer les positions prises par l'objet visé. Certains logiciels comme par exemple Cabri-Géomètre, permettent même de tracer le lieu d'un objet. L'étude des lieux d'un objet d'une figure n'a pas qu'un but éducatif. Il permet aussi de résoudre des problèmes d'optimisation et d'explorer des modèles physique [CG95].

Exemple 1.1 Limaçon de Pascal

Le limaçon de Pascal est le lieu du point B, projection orthogonale du point U sur la droite L tangente au cercle (C) en A, lorsque le point A décrit le cercle (C). Il est dessiné sur la figure 1.1.

C'est un *programme géométrique* qui produit de telles figures dynamiques. Des exemples de tels programmes géométriques sont les constructions réalisées en Cabri-Géomètre.

On peut distinguer deux types de programmation géométrique :

– La programmation géométrique impérative. Dans l'approche impérative, la construction d'une figure respectant des spécifications initiales particulières est définie par une succession d'étapes ordonnées. Les premiers objets construits, plus exactement ceux dont la position a un caractère arbitraire, sont les objets de base qui servent à construire d'autres objets. Chaque étape utilise des objets déjà

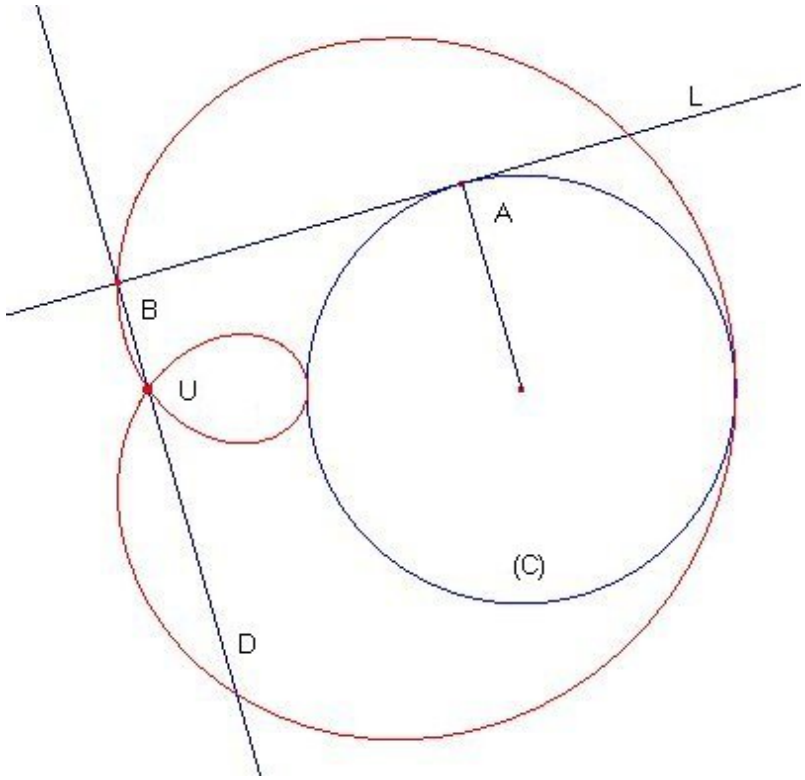

FIG. *1.1* – Limaçon de Pascal (produit à l'aide de Cabri-Géomètre II).

construits pour en produire un autre. Les objets animables sont ici les objets de base. Une figure géométrique dynamique peut être représentée dans cette approche par un graphe orienté s'appuyant sur la relation de dépendance suivante où un objet A est dit directement dépendant des objets $B_1, B_2, \ldots, B_n$ si et seulement si la construction de ce dernier s'est basée sur les objets $B_1, B_2, \ldots, B_n$ [2].

– La programmation géométrique déclarative. En programmation géométrique déclarative, la construction d'une figure géométrique est issue d'une spécification logique de celle-ci et de la donnée d'un ensemble d'objets de base. Dans un langage prédicatif, on peut considérer de telles spécifications logiques sous la forme d'une conjonction de formules atomiques. Dans une vision idéale, les objets animables devraient être ceux qui peuvent être potentiellement des objets de base.

2. Nous revenons sur cette notion de dépendance dans le paragraphe 111.

1.2 Systèmes de géométrie dynamique

Nous présentons dans les sections suivantes les principaux systèmes de programmation géométrique impérative puis déclarative. Enfin, nous situons notre prototype GDRev dans le cadre de la géométrie dynamique.

1.2.1 Systèmes de géométrie impérative

Il existe de nombreux systèmes de programmation géométrique impérative. Dans le domaine de l'enseignement de la géométrie, les plus utilisés sont Cabri-Géomètre et Geometer's Sketchpad. Nous les décrivons dans les paragraphes qui suivent. Nous référençons ensuite d'autres systèmes de programmation géométrique impérative trouvés dans la littérature et sur l'internet.

Cabri-Géomètre

Cabri-Géomètre est l'acronyme de CAhier de BRouillon Interactif (Informatique ou Intelligent) pour la géométrie [Bau90, Bel92, Lab95]. Ce logiciel de géométrie est développé dans le cadre du projet IMAG-Cabri-Géomètre sous la direction de Laborde J-M, et regroupe les laboratoires Leibniz et LSR (Logiciels, Systèmes et Réseaux) de l'institut IMAG (Informatique et Mathématiques Appliqués de Grenoble). Il fonctionne sur Macintosh et sur PC sous DOS et sous Windows, et des versions de démonstration sont accessibles à l'URL : *ftp://ftp.imag.fr/pub/CABRI/*. Il est aussi intégré dans sa deuxième version dans la calculatrice TI-92 de Texas Instrument.

Bien que Cabri-Géomètre [Bau90, Bel92] puis Cabri II [Lab95] soient fondamentalement orientés pour la géométrie dans le plan, ils peuvent simuler la géométrie dans l'espace grâce à leur gestion de l'infini. Néanmoins, une interface 3D pour Cabri-Géomètre a fait l'objet d'une récente étude [Qas97] qui a montré sa faisabilité.

Toutes les versions de Cabri sont basées sur les principes de la manipulation directe [Shn83, Nan90, FvDFH90] comme la continuité, la réversibilité, la temps-réel et le feed-back (retour d'informations), que nous détaillons ultérieurement dans le chapitre 5.

Les objets manipulables dans Cabri II sont nombreux. On trouve les objets de type point, droite, demi-droite, segment, vecteur, triangle, polygone, polygone régulier, cercle, arc et conique. Les principales constructions réalisables à partir de ces objets

sont celles de la perpendiculaire/parallèle à une droite passant par un point, du milieu, de la médiatrice, de la bissectrice, du lieu d'un point, du symétrique axiale et du symétrique central.

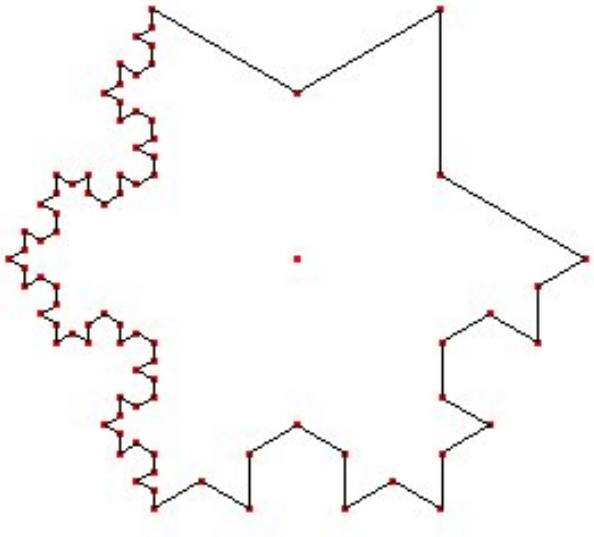

FIG. 1.2 – *Flocon de Koch (produit à l'aide de Cabri-Géomètre II).*

A côté de ces constructions primaires, Cabri-Géomètre offre la possibilité de définir des *macros constructions*. Ces macros constructions sont définies à posteriori sur une figure. Pour définir une macro construction, Cabri-Géomètre a besoin que l'utilisateur désigne des objets initiaux, et des objets finaux tels que ces derniers aient été construits à partir des premiers ou de leurs descendants. C'est-à-dire que Cabri-Géomètre extrait du graphe définissant la figure, le sous-graphe connexe liant les objets initiaux aux objets finaux. Ces macros constructions peuvent être sauvegardées, et sont accessibles dans la barre d'outils comme tout autre outil. Elles facilitent grandement la construction de figures telles que celle illustrée par la figure 1.2.

Geometer's Sketchpad

Geometer's Sketchpad est un micro-monde de géométrie plane développé par Jackiw N dans le cadre du Visual Geometry Project, sous la direction de Klotz E au Swarthmore College [Jac95]. Il est distribué par Key Curriculum Press qui met à disposition des versions de démonstration à l'URL : *http://www.keypress.com/sketchpad/*. Il fonctionne sur Macintosh et sur PC sous Windows.

A l'image de Cabri-Géomètre, Geometer's Sketchpad est basé sur les principes de la manipulation directe. Cependant, on peut noter que l'aspect feed-back (retour d'informations) est bien moins pris en compte dans ce dernier. Il se manifeste par exemple sous la forme de "bassins d'attraction" lors de la construction d'un segment entre deux points donnés ou construits, c'est-à-dire qu'à proximité d'un point, le curseur est comme aimanté. Cependant, lors de la construction d'un point sur objet, cette "attraction" n'est pas présente et aucune indication visuelle est retournée à l'utilisateur indiquant si le système va considérer le point comme libre ou comme appartenant à l'objet pointé.

Les objets manipulables dans Geometer's Sketchpad sont semblables à ceux que l'on trouve dans Cabri-Géomètre, à l'exception notable toutefois des coniques qui en sont absentes.

Geometer's Sketchpad offre la possibilité de définir des *scripts*. Ces scripts sont semblables aux macros constructions de Cabri. Cependant leurs modes de définition diffèrent. En effet, un script est une séquence de constructions enregistrée à l'intar des "macros" de logiciels tels que Microsoft Word[3]. Ces scripts sont cependant plus riches que les macros constructions de Cabri puisqu'ils peuvent être récursifs. Lors de l'utilisation de tels scripts récursifs, une boîte de dialogue s'ouvre et invite l'utilisateur à spécifier le niveau de récurrence souhaité.

Bien que Cabri-Géomètre et Geometer's Sketchpad paraissent semblable[4], ils sont en fait conceptuellement très différents. Cette différence porte sur le mode de fonctionnement des outils. Dans Cabri, l'utilisateur sélectionne un outil, puis désigne ensuite les éléments nécessaire à la construction[5]. A l'inverse, l'utilisateur de Geometer's Sketchpad doit d'abord sélectionner les éléments de sa construction pour ensuite choisir l'outil qu'il souhaite leur appliquer.

Autres systèmes impératifs

A côté des deux principaux systèmes de géométrie dynamique présentés précédemment, il existe de nombreux systèmes de géométrie impérative.

Dans le cadre de logiciels plus particulièrement dédiés à l'apprentissage de la géo-

3. L'auteur se défend ici de faire toute publicité pour ce logiciel.

4. On note que cette similitude va jusqu'à leur appellation puisque Geometer's Sketchpad signifie en français : cahier de brouillon pour la géométrie

5. Il est aussi possible de construire les points nécessaires à toute construction à la volée, ce qui est une facilité fort appréciable.

métrie, on trouve par exemple les systèmes soutenus par le ministère français de l'éducation nationale comme Calques Géométriques [Ber94], ou encore GeoPlanW dont une version de démonstration fonctionnant sur PC sous Windows peut être téléchargée à l'URL : *http://www2.cnam.fr/creem/*.

Dans le même cadre, on peut aussi considérer le système Geometry Inventor distribué par Logal Software Inc et dont des versions de démonstration limitées à 30 jours sont accessibles à l'URL : *http://www.logal.com/*.

Enfin, dans un cadre plus orienté vers la démonstration automatique de théorèmes géométriques, on trouve le système Gex (Geometry Expert) [CGZ97, GC98a, GC98b] qui est accessible à l'URL : *ftp://carl.cs.twsu.edu/pub/gex/*.

1.2.2 Systèmes de géométrie déclarative

Il existe aussi plusieurs systèmes de programmation géométrique déclarative. Nous décrivons dans les paragraphes qui suivent quelques un de ces systèmes par rapport auxquels nous nous situons. Il convient aussi de citer pour les lecteurs désireux d'en connaître davantage le système de Chou S-C, Gao X-S et Zhang J-Z détaillé dans la littérature [CGZ97, GC98a, GC98b].

Le premier paragraphe est consacré à la description du système GéoSpécif [AIT93, Bou95]. Puis nous décrivons les systèmes Progé [Sch93] et YAMS [Mat97].

GéoSpécif

GéoSpécif est un prototype développé au sein de l'équipe PLIAGE (Programmation Logique, Intelligence Artificielle, Génie Educatif) du laboratoire LSR (Logiciels, Systèmes et Réseaux) de l'institut IMAG (Informatique et Mathématiques Appliqués de Grenoble) sous la direction de Trilling L [AIT93, Bou95]. Il est mis en œuvre en PrologIII [Col90], et il fonctionne sur Macintosh et sur station Sun.

Ce logiciel de géométrie plane permet de construire des figures à base de points, droites, segments et cercles, reliés entre eux par des propriétés de perpendicularité, de parallélisme, de milieu et d'appartenance. Il offre les services suivants :

- L'acquisition d'une spécification logique géométrique. Cette acquisition se fait sous une forme textuelle ou au travers des fonctions mises à disposition dans les menus [Gar95].

- L'animation d'une figure géométrique. A l'instar de Cabri-Géomètre, GéoSpécif [Bou95] intègre le concept de manipulation directe d'une figure. Cette animation de la figure se fait au travers des points de base ayant servi à la construction de la figure.
- L'aide à la construction d'une figure par enrichissement de la spécification. Par l'ajout de propriétés redondantes, c'est-à-dire de propriétés non énoncées dans la spécification de la figure mais issues de la construction, GéoSpécif facilite la construction de cette dernière pour laquelle il n'arrive dans une première étape qu'à une construction partielle. Cet enrichissement est cependant manuel, et nécessite une intervention de l'utilisateur.

Un exemple des problèmes résolus par GéoSpécif est le problème 1.3.

Exemple 1.3 PROBLÈME RÉSOLU PAR GÉOSPÉCIF

Construire un triangle ABC et ses médiatrices tel que son sommet A soit sur une droite D_{mn}. La spécification de ce problème dans un langage logique géométrique est :

$point(A) \wedge\ point(B) \wedge\ point(C) \wedge\ point(O) \wedge$
$point(I) \wedge\ point(J) \wedge\ point(K) \wedge$
$point(m) \wedge\ point(n) \wedge\ droite(D_{mn}) \wedge$
$point(P) \wedge\ point(Q) \wedge\ point(R) \wedge$
$droite(D_{AB}) \wedge\ droite(D_{AC}) \wedge\ droite(D_{BC}) \wedge$
$droite(D_{OI}) \wedge\ droite(D_{OJ}) \wedge\ droite(D_{OK}) \wedge$
$appDr(A, D_{AB}) \wedge\ appDr(A, D_{AC}) \wedge\ appDr(A, D_{mn}) \wedge$
$appDr(B, D_{AB}) \wedge\ appDr(B, D_{BC}) \wedge$
$appDr(C, D_{AC}) \wedge\ appDr(C, D_{BC}) \wedge$
$appDr(O, D_{OI}) \wedge\ appDr(I, D_{OI}) \wedge\ appDr(P, D_{OI}) \wedge$
$appDr(O, D_{OJ}) \wedge\ appDr(J, D_{OJ}) \wedge\ appDr(Q, D_{OJ}) \wedge$
$appDr(O, D_{OK}) \wedge\ appDr(K, D_{OK}) \wedge\ appDr(R, D_{OK}) \wedge$
$perp(D_{BC}, D_{OI}) \wedge\ perp(D_{AB}, D_{OJ}) \wedge\ perp(D_{AC}, D_{OK}) \wedge$
$milieu(I, B, C) \wedge\ milieu(J, A, B) \wedge\ milieu(K, A, C).$

L'exécution de ce programme avec la donnée des points de base m, n, O, P, Q et R conduit au dessin rapporté par la figure 1.3.

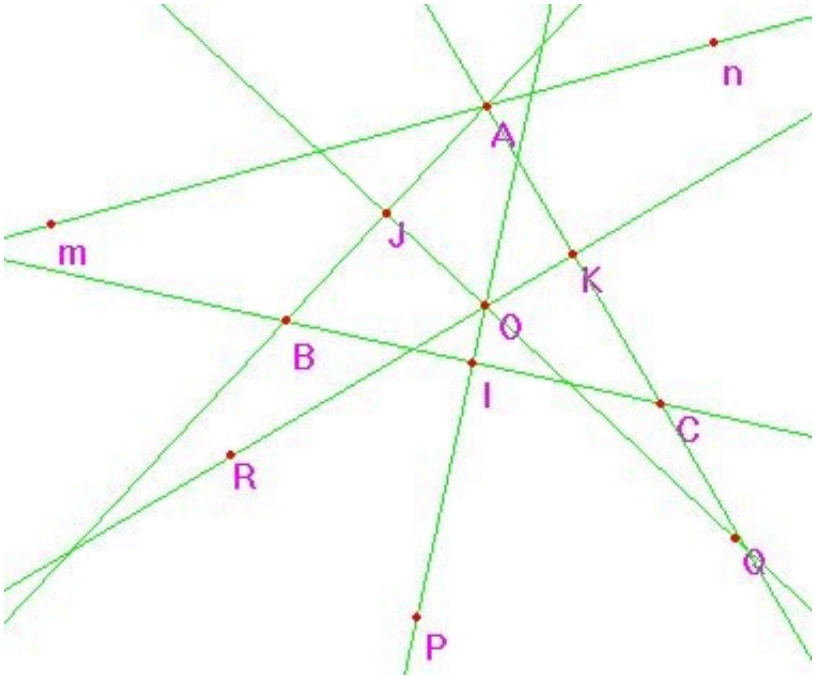

FIG. *1.3* – Triangle ABC construit à partir de ses médiatrices et d'une droite passant par A.

Progé

Progé est un logiciel de construction automatique de figures mis en œuvre en Delphia Prolog sur Station Sun [Sch93]. Cet environnement permet de construire des figures géométriques mettant en jeu des objets de type point, droite, cercle et distance, reliés entre eux par des propriétés de type perpendicularité, parallélisme, milieu, tangence. La saisie de la spécification de la figure se fait sous forme textuelle. Une fois la spécification saisie, l'utilisateur positionne à l'aide de la souris les éléments de base de sa figure et peut visualiser successivement les figures répondant au problème de construction.

L'atout majeur de cet environnement est que son univers géométrique peut être simplement enrichi avec de nouveaux objets et/ou de nouvelles propriétés.

Sa principale faiblesse est que la figure construire est affichée dans une fenêtre, mais qu'il n'est pas possible d'agir directement sur elle par manipulation directe.

YAMS

YAMS (Yet Another Meta Solveur) est un système de programmation géométrique déclarative issu de la CAO qui a été développé par Mathis P [DMS96, Mat97]. Mis en œuvre dans le langage C à l'aide des bibliothèques graphiques OpenGL, il fonctionne sur Silicon Graphics.

La résolution d'un problème de construction dans YAMS comporte 3 phases. Dans une première étape, il s'agit de produire une esquisse de la figure. Cette esquisse peut comporter des objets de type point, segment, cercle, arc de cercle, "polygone ouvert", et polygone. Une fois cette esquisse produite, on spécifie les contraintes liant les éléments de cette dernière. Ces contraintes sont du type angle, rayon, incidence, distance, distance horizontale, tangence et égalité. Enfin, l'utilisateur peut demander la résolution de son problème, et modifier dynamiquement les paramètres de sa figure.

1.2.3 Où placer GDRev dans la géométrie dynamique

Pour un utilisateur, les atouts de la programmation géométrique déclarative vis-à-vis de la programmation géométrique impérative réside d'une part dans la simplicité des descriptions des figures dynamiques, et d'autre part dans la facilité de l'exploration de ces dernières. En effet rappelons nous que, contrairement à la programmation géométrique impérative qui consiste à décrire étape par étape la manière de construire une figure, la programmation géométrique déclarative consiste simplement à spécifier dans un ordre quelconque les éléments d'une figure et les propriétés qui les lient entre eux et à donner l'ensemble des objets de base à partir desquels le système doit construire la figure. C'est pourquoi ces atouts nous ont conduit à nous intéresser ici à ce cadre de la programmation géométrique.

Les raisons de la définition d'un nouveau système de géométrie dynamique déclarative trouvent leurs origines dans l'absence d'une réelle adéquation entre ces derniers et le domaine spécifique de l'enseignement de géométrie dynamique dans les collèges et les lycées. Plus précisément, bien que GéoSpécif et Progé soient des systèmes déclaratifs dédiés à l'enseignement de la géométrie, ils ont pour principale faiblesse leur manque de dynamisme. S'agissant du système YAMS, le problème réside plutot dans le domaine d'application de ce dernier, c'est-à-dire la CAO qui met en jeu des problèmes quelque peu différents de ceux de l'enseignement.

Ainsi, il s'agit pour nous, de mettre en œuvre les principes de la manipulation

directe dans un système de géométrie déclarative dédié à l'exploration de figures dynamiques du niveau secondaire.

Le problème de résolution des contraintes géométriques auquel nous faisons face se résume ainsi : Etant donné un ensemble d'objets et un ensemble de propriétés liant les objets entre eux, construire la(les) figure(s) répondant à cette spécification à partir d'un ensemble donné d'objets de base.

Nous présentons dans la section suivante les principales méthodes mises en œuvre pour résoudre ce problème.

1.3 Différentes approches de la résolution de contraintes géométriques

La résolution de contraintes géométriques est le thème central de nombreux domaines de recherches. Cette importance avait déjà été pressentie au début des années 80 par Borning A :

> *"Constraints will be taking an increasingly prominent position in our paradigms for programming in the years to come."*
> [Bor81] (page 386)

Au début des années 90, elle était déjà si grande que Kramer G écrivit :

> *"[...] in the CAD/CAM community [...] constraint systems have become* de rigueur *for "serious CAD systems"."*
> [Kra92b] (page xviii)

Aujourd'hui, ce problème se trouve au cœur par exemple des systèmes de CAO intelligents *(Intelligent CAD Systems)* [BR98, Sap91], des systèmes graphiques à base de contraintes *(Constraint Based Graphic Systems)* [MBB89, HN94], des modeleurs en infographie [LD95], des systèmes dédiés à l'enseignement de la géométrie [AIT93] ou encore en cinématique des robots [Kra92b].

Nous détaillons successivement dans les paragraphes qui suivent les principales approches, de type géométrique, algébrique, numérique, à base de décompositions ou coopératives, étudiées pour la résolution de contraintes géométriques. Puis, nous établissons un bilan de l'analyse de ces méthodes de résolution destiné à éclairer le lecteur sur notre approche décrite dans les parties II et III.

1.3.1 Approches géométriques

L'approche dite géométrique [Ald88, Sch93, Kra92a, CGZ97, GC98a] est une approche symbolique. Elle consiste à établir un plan de construction d'une figure à partir d'une description de celle-ci et d'une théorie de la géométrie sous une forme axiomatique. Le plan de construction établi peut être ensuite interprété pour aboutir à une solution numérique. L'architecture générale d'une telle approche est illustrée par la figure 1.4.

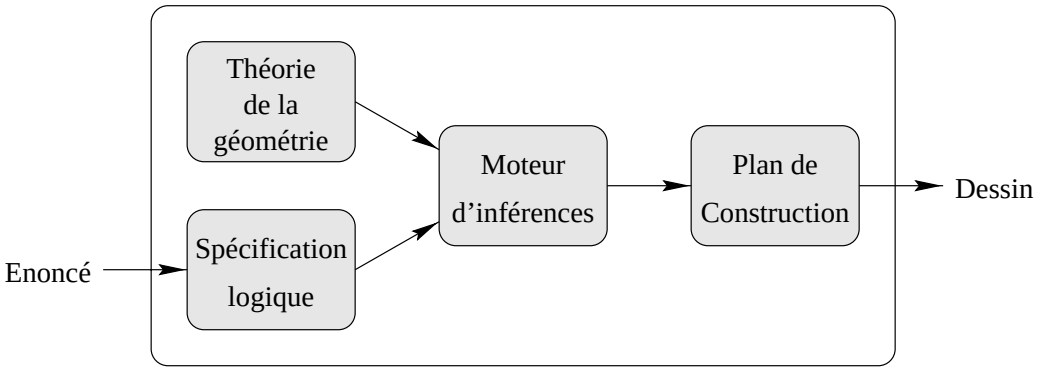

FIG. 1.4 – *Architecture générale des systèmes de résolution géométrique de contraintes.*

Buthion M [But75, But79] fut le premier à mettre en œuvre un système de résolution géométrique de problèmes de construction dans le prototype GEOM3.

Les avantages de cette approche sont multiples. Tout d'abord, elle évite de propager les erreurs liées aux calculs numériques, et elle résout bon nombre des problèmes de construction rencontrés dans l'enseignement de la géométrie. Ensuite, la séparation entre le raisonnement et l'univers modélisé facilite la définition de nouveaux objets et de nouvelles propriétés. Cette séparation facilite aussi la définition de nouvelles règles de production. De plus, et ceci nous semble être l'intérêt principal, dans la cadre d'un environnement de géométrie dynamique dédié à l'exploration de situations didactiques, l'interprétation séquentielle des plans de construction produits conduit à des animations fluides.

La principale difficulté rencontrée lors de l'établissement d'un plan de construction d'une figure réside dans l'existence de "boucles" dans la description d'une figure. Un exemple de telles figures est donné par le problème 1.6.

Nous décrivons dans les paragraphes suivants différentes méthodes géométriques

de résolution. Nous commençons par l'approche de Aldefeld B [Ald88], puis par celle de Schreck P [Sch93, Sch94] et enfin par l'approche par analyse des degrés de liberté de Kramer G [Kra92a, Kra92b].

Approche de Aldefeld

L'approche de Aldefeld B [Ald88] consiste à utiliser un algorithme de type chaînage avant pour résoudre le problème de la construction de figures géométriques. Cet algorithme s'appuie sur des heuristiques d'application des règles de production préétablies.

Si au cours de la résolution d'un problème, il est confronté à de l'indéterminisme, il repousse ce problème à la phase d'interprétation du plan de construction. Au cours de cette phase, il utilise la spécification de la figure attendue, donnée au travers d'une esquisse, pour choisir parmi les solutions produites la solution la plus "ressemblante". Le critère de ressemblance ici choisi est lié au nombre des contraintes de type même côté ou même séquence satisfaites.

Approche de Schreck

Schreck P propose dans [Sch93, Sch94] d'étendre les travaux de Buthion M [But75, But79] sur de nombreux points. Son approche se distingue de la précédente par le plan de construction produit et par la définition de la théorie de la géométrie utilisée.

Tout d'abord, il introduit dans les plans de construction produits des structures conditionnelles ($si \ldots alors \ldots sinon \ldots$). Allant de pair avec cette introduction, il adopte la définition de structures itératives ($pour \ldots dans \ldots faire \ldots$). L'intérêt de ces introductions réside dans la possible gestion des cas de figure.

Son approche repose aussi sur un univers géométrique modifiable. Il dissocie dans cet univers géométrique les connaissances des heuristiques d'application des règles. Ceci permet à un utilisateur averti de définir ses propres stratégies d'application des règles adaptées à ses problèmes.

L'auteur a mis en œuvre son approche dans le prototype Progé précédemment décrit au paragraphe 1.2.2 page 20.

Méthode par analyse des degrés de liberté

Kramer G propose dans [Kra92a, Kra92b] une approche symbolique singulière au problème de la résolution de contraintes géométriques. Il l'applique au domaine de la cinématique. Cette approche se base sur une analyse des degrés de liberté des éléments qui compose une figure. Son algorithme s'appuie sur trois tables. La première table, dite table fragmentaire de plan *(plan fragment table)*, contient la signature[6] de tout objet en fonction des contraintes et de l'état courant de ce dernier. La deuxième, dite table des lieux *(locus table)*, contient la description des lieux de chaque objet en fonction de sa signature. Enfin, la troisième table, dite table des lieux d'intersection *(locus intersection table)*, contient la description des lieux de tous les types d'intersection mettant en jeu deux objets de l'univers modélisé. L'algorithme, de type chaînage avant, consiste à examiner successivement les contraintes géométriques de la spécification de la figure et à réduire en conséquence les degrés de liberté des éléments en jeu à l'aide de la table fragmentaire de plan et de la table des lieux d'intersection. Pour pallier aux insuffisances de cette analyse locale, il est parfois nécessaire de recourir à une analyse globale du problème. Cette analyse consiste en une étude des lieux de points partiellement contraints. Elle est menée à l'aide de la table des lieux, et conduit à la localisation d'objets à l'intersection des lieux examinés. La terminaison de l'algorithme est ici assurée par la stricte décroissance des degrés de liberté de la figure.

L'inconvénient majeur de cette approche réside dans la création/mise-à-jour des tables de plan, des lieux et des lieux d'intersection.

1.3.2 Approches algébriques

La résolution de contraintes géométriques suivant une approche algébrique consiste dans un premier temps à produire le système d'équations correspondant aux contraintes, puis dans un deuxième temps à mettre sous forme triangulaire ce système. Une fois sous forme triangulaire, un système d'équations peut être simplement résolu numériquement.

Formellement, la première étape des approches algébriques consiste à produire le système (S) suivant :

6. La signature d'un objet est la liste de ses différents types de degré de liberté. Par exemple, la signature d'un cercle de rayon variable dans le plan est : 2 degrés de liberté de translation, 0 degré de liberté de rotation et 1 degré de liberté d'élongation.

$$\begin{cases} c_1(u_1, \ldots, u_n, x_1, \ldots, x_r) \\ \vdots \\ c_r(u_1, \ldots, u_n, x_1, \ldots, x_r) \end{cases}$$

où les c_i sont les formes algébriques des contraintes géométriques (polynômes en u_i et x_i), les u_i sont les paramètres des objets donnés (constantes), et les x_i sont les paramètres des objets à construire (variables).

La deuxième étape consiste à transformer ce système (S) en un système triangulaire équivalent (S') :

$$\begin{cases} f_1(u_1, \ldots, u_n, x_1) \\ f_2(u_1, \ldots, u_n, x_1, x_2) \\ \vdots \\ f_r(u_1, \ldots, u_n, x_1, \ldots, x_r) \end{cases}$$

Un algorithme simple de triangularisation consiste à successivement pseudo-diviser les polynômes $c_1, \ldots, c_n$ suivant les variables $x_r, \ldots, x_1$ pour obtenir un système sous la forme (S') à condition que le système (S) soit bien contraint.

Les deux principales techniques mises en œuvre pour trianguläriser un système d'équations sont axées sur les bases de Gröbner d'une part et l'algorithme de Ritt-Wu d'autre part.

Méthode axée sur les bases de Gröbner

L'idée de cette approche consiste à déterminer une base de Gröbner (G) correspondant au système d'équations (S) afin d'en extraire une forme triangulaire (S').

Une base de Gröbner (G) d'un système (S) est un système d'équations ayant la propriété que son idéal[7] est égal à celui du système (S), ce qui implique que les systèmes d'équations (G) et (S) aient les mêmes solutions.

Le premier algorithme mis en œuvre pour déterminer une base de Gröbner d'un système d'équations est dû à Buchberger B en 1965. Cet algorithme est par exemple décrit dans [HL93, Buc96].

Le principal handicap à l'utilisation de cette approche est la complexité exponen-

7. L'idéal I d'un ensemble $S = \{f_1, \ldots, f_r\}$ de polynômes sur l'anneau $K[x_1, \ldots, x_n]$ avec K un corps et $x_1, \ldots, x_n$ n variables indépendantes, noté $(f_1, \ldots, f_n)$, est l'ensemble des polynômes obtenus par combinaison linéaire des polynômes de (S). Formellement, $I = \{q_1 f_1 + \ldots + q_n f_n | q_i \in K[x_1, \ldots, x_n]\}$.

tielle de l'algorithme.

Méthode de Ritt-Wu

L'algorithme de triangularisation de Ritt-Wu diffère de l'approche axée sur les bases de Gröbner par sa manière de considérer les formes algébriques des contraintes géométriques. Dans cette approche, Ritt-Wu considèrent les polynômes multivariables comme des polynômes monovariables ayant des coefficients définis en les autres variables.

L'algorithme consiste à chercher une décomposition du système (S) en ensembles caractéristiques irréductibles de polynômes ; polynômes générateurs d'idéaux premiers de mêmes racines que le système (S).

C'est en 1977 que Wu W-T introduisit sa méthode. Plus tard, il reprit les travaux de Ritt J-F antérieurs aux siens à des fins de démonstration automatique. Les détails de cette méthode peuvent être trouvés dans l'ouvrage [Cho88].

Cette approche a été mise en œuvre dans le prototype de Chou S-C, Gao X-S et Zhang J-Z [GC98b].

L'inconvénient de cette méthode réside dans la complexité de l'algorithme de triangularisation.

1.3.3 Approches numériques

Dans les approches numériques, on trouve des méthodes exactes et des méthodes approchées. Celles-ci sont décrites successivement dans les paragraphes suivants.

Méthode exacte

L'idée consiste à définir les opérateurs arithmétiques exacts $+, -, \times, /, \sqrt{}, =$ et $>$ afin d'être en mesure de déterminer exactement les coordonnées des points d'intersection de deux cercles ou d'une droite et d'un cercle ; détermination qui conduit sur le corps des rationnels au calcul d'une racine carrée source d'une approximation. Pour ce faire, ces définitions reposent sur la notion d'*extension algébrique* [Bou96].

Sur le corps $Q[\sqrt{a_1}]$, corps des rationnels contenant l'extension $\sqrt{a_1}$, un nombre x est représenté par :

$$x = p + q\sqrt{a_1} \text{ avec } p, q, a_1 \in Q$$

Sur le corps $Q[\sqrt{a_1}]\ldots[\sqrt{a_{n-1}}][\sqrt{a_n}]$, corps des rationnels contenant les extensions successives $\sqrt{a_1}, \ldots, \sqrt{a_{n-1}}, \sqrt{a_n}$, un nombre x est représenté par :

$$x = p + q\sqrt{a_n} \text{ avec } p, q, a_n \in Q[\sqrt{a_1}]\ldots[\sqrt{a_{n-1}}]$$

Au début du processus de résolution, les paramètres des objets de la figure sont définis sur le corps $K_0 = Q$ des rationnels. Les calculs sont effectués sur ce corps tant que l'on n'arrive pas au calcul des points communs à deux cercles ou à une droite et un cercle. A ce moment, il arrive souvent que les coordonnées de tels points s'expriment à l'aide d'un radical du second degré $\sqrt{a_1}$ avec $a_1 \in K_0$. Les calculs se poursuivent alors dans $K_1 = K_0[\sqrt{a_1}]$, jusqu'à la résolution d'une autre équation quadratique qui peut introduire une extension $\sqrt{a_2}$ si cette dernière n'est pas une combinaison linéaire de l'extension déjà introduite. De la sorte, on forme au fur et à mesure de la résolution une suite de corps $K_0, K_1, \ldots, K_n$.

L'inconvénient principal de cette approche est sa complexité exponentielle en espace et sa complexité opérationnelle. Cependant, une étude pratique de Bouhineau D [Bou97] sur un corpus de 512 problèmes de géométrie extraits de [Cho88] montre d'une part que dans 86% des cas, la résolution introduit aucune extension et que les situations restantes introduisent au plus 3 extensions si l'on s'autorise des modifications locales, et d'autre part que ce pourcentage peut être augmenté à 96% et le nombre d'extensions introduites peut être réduit à 1 si l'on s'autorise des modifications majeures[8].

Cette approche a été mise en œuvre dans le prototype GéoSpécif précédemment décrit au paragraphe 1.2.2 page 18.

Méthode de Newton-Raphson

La méthode de Newton-Raphson [MS98] est la méthode de résolution la plus simple pour trouver les x tels que $f(x) = 0$ pour une fonction f continûment dérivable. Cet algorithme itératif est basé sur le principe que l'on considère qu'en un point d'une

8. Nous détaillons ces résultats dans la paragraphe 10.2.

courbe, cette dernière est bien approchée par la tangente en ce point, comme l'illustre la figure 1.5. Un pas de l'itération consiste à déterminer une valeur x_{n+1} comme le point d'intersection de la droite L et de l'axe des abscisses, où la droite L est la tangente à la fonction f au point x_n. Formellement la suite des points x_i est définie par :

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

L'algorithme s'arrête lorsque la valeur de x_i est considérée suffisamment proche d'une racine, c'est-à-dire formellement que le processus s'arrête en x_i si la condition $-\epsilon \leq f(x_i) \leq \epsilon$ est vérifiée.

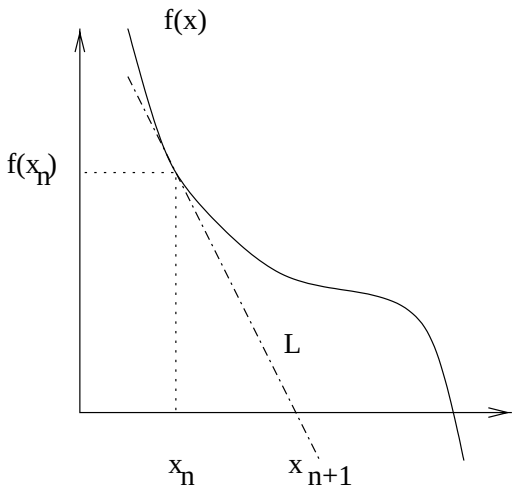

FIG. 1.5 – *Itération de la méthode de Newton-Raphson.*

L'atout majeur de cette méthode est quelle est facile à mettre en œuvre. Les deux principales faiblesses de cette dernière sont d'une part qu'elle ne permet pas d'obtenir toutes les solutions d'un système mais seulement la première, et d'autre part que l'algorithme peut diverger alors qu'une solution existe, même si la valeur initiale est proche de la racine cherchée.

Méthode par homotopie

Michelucci D et Lamure H semblent les premiers à avoir résolu des contraintes géométriques par homotopie [LM94]. Cette méthode consiste à résoudre un système $G(X) = 0$ avec $G = (g_1, g_2, \ldots, g_n)$ de n équations indépendantes à n inconnues, en supposant connue une solution S d'un système $F(X) = 0$ avec $F = (f_1, f_2, \ldots, f_n)$

"proche" de G. Cette solution connue peut être une approximation de la figure donnée par l'utilisateur, ou une première approximation de la figure par la méthode de Newton-Raphson. Le système F est défini par :

$$F(X) = G(X) - G(S)$$

A partir de G, F et S, est définie une interpolation linéaire H telle que :

$$H(t, X) = tG(X) + (1 - t)F(X)$$

Cette méthode itérative consiste à parcourir cette courbe homotopique H à partir du point $(t = 0, X = S)$ jusqu'à obtenir une solution en un point (t, X) où $t = 1$. Si le système n'a pas de solution, cette approche boucle.

Lors du cheminement de la courbe, l'algorithme peut faire face à une *bifurcation*. De telles bifurcations sont rencontrées lorsque par exemple à une étape de l'itération un point se trouve sur la médiatrice du segment formé par les deux solutions du plan qui satisfont les contraintes imposées à ce point. Pour pallier à ce problème, les auteurs proposent de perturber légèrement la figure.

Le principal intérêt de cette méthode réside dans son caractère prédictible, contrairement à la méthode de Newton-Raphson. Son principal inconvénient est sa lenteur du fait de l'utilisation de méthode itératives lors du cheminement de la courbe homotopique.

1.3.4 Autres approches

D'autres approches existent. Celles-ci consistent par exemple à décomposer un problème en sous-problèmes ou à aborder la construction automatique de figures géométriques suivant une approche coopérative.

Méthodes par décomposition

L'idée de la méthode par décomposition est de diviser un problème de construction en sous-problèmes plus simples. Le principal objectif visé par ces décompositions est de

briser les spécifications "cycliques" de certaines figures qui sont difficiles à appréhender par les précédentes méthodes de résolution. Un autre intérêt de ces approches est qu'elles accélèrent le processus de résolution. En effet, généralement la complexité des méthodes décrites est une fonction exponentielle de la taille du système. La réduction de ce système permet par conséquent un gain appréciable. Le classique exercice de CAO de l'exemple 1.6 illustre de telles spécifications cycliques.

Exemple 1.6 SPÉCIFICATION CYCLIQUE

Construire un hexagone ABCDEF à partir de la donnée des longueurs des côtés et des angles $\widehat{A}, \widehat{C}$ et $\widehat{E}$. Un exemple d'hexagone répondant à ce problème de construction est illustré par la figure 1.6.

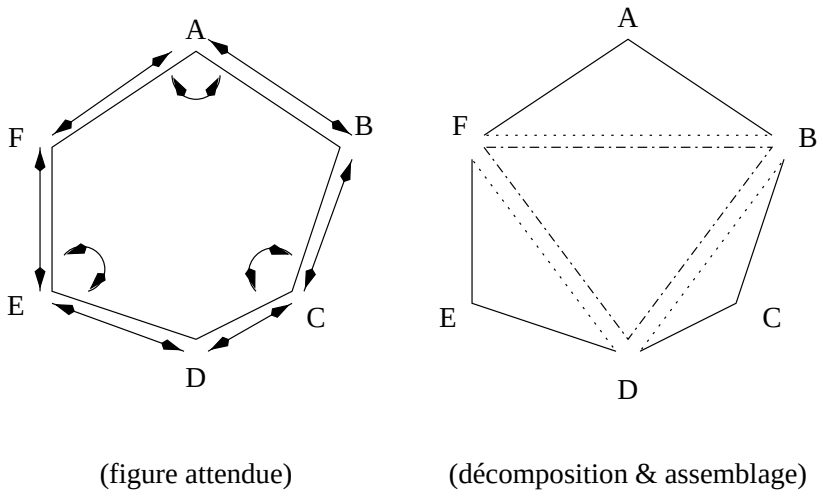

FIG. *1.6* – Exemple d'hexagone répondant au problème de construction.

Pour résoudre ce problème, les méthodes par décomposition isolent les triangles ABF, CBD, *et* EDF *pour finalement former l'hexagone demandé en considérant les points B, D et F comme des axes de rotation.*

On distingue deux courants dans ces méthodes par décomposition :

– L'approche descendante (Top-Down) : qui consiste à résoudre des bouts de figure aussi important que possible, aussi appelés *cluster*, puis à assembler les bouts résolus dès que possible.

  La figure 1.7 illustre trois cas de figures conduisant à l'assemblage des clusters en présence. Ces règles d'assemblage sont extraites de [BFH+95]. La première

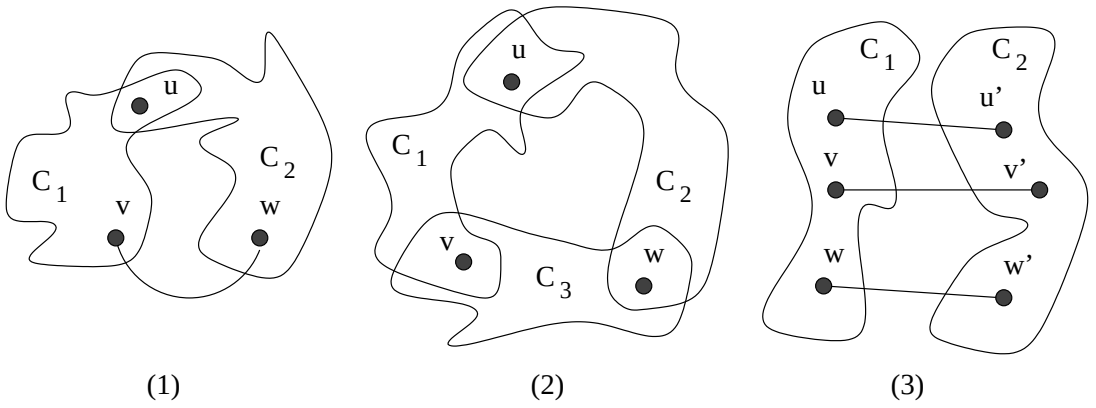

FIG. 1.7 – *Règles d'assemblage de clusters.*

règle d'assemblage correspond au cas où deux clusters C_1 et C_2, liés entre eux au travers d'une contrainte portant sur un élément de chacun, partagent de plus un même élément u. Le deuxième règle permet d'assembler trois clusters C_1, C_2 et C_3 adjacents, c'est-à-dire ayant un élément en commun deux à deux. Enfin la troisième règle permet d'assembler deux clusters liés par trois contraintes portant sur leurs éléments.

Cette approche a par exemple été adoptée par Dufourd J-F, Mathis P et Schreck P [DMS96] dans le système YAMS décrit dans la paragraphe 1.2.2 et par Bouma W, Fudos I, Hoffmann C, Cai J et Paige R [BFH+95].

– L'approche ascendante (Bottom-Up) : qui consiste à commencer par décomposer la figure en sous-figures élémentaires, puis à résoudre ces sous-figures, pour enfin les assembler.

  Cette décomposition peut être par exemple le résultat d'une analyse d'un système expert, ou le résultat d'une analyse du graphe des contraintes lié à la figure.

  Cette approche est par exemple la démarche suivie par Owen J-C [Owe91] et par Ait Aoudia S, Jegou R et Michelucci D [AAJM93].

Méthodes coopératives

L'idée des méthodes coopératives vient du constat que les différentes approches précédemment présentées se caractérisent par leur complémentarité. Plus précisément, nous avons vu que les approches géométriques et algébriques sont robustes et exactes mais elles nécessitent beaucoup d'espace mémoire et de temps de calcul. A l'inverse,

les approches numériques sont peu gourmandes en espace mémoire, et généralement rapides mais elles peuvent diverger. C'est pourquoi il est apparu relativement naturellement le sentiment de nécessité d'un compromis.

Nous présentons dans les paragraphes qui suivent deux approches de la coopération pour la résolution de contraintes géométriques.

Tran Q-N détaille dans [Trâ'n96] une approche hybride de la résolution de contraintes géométriques dans le domaine de la CAO. Il applique cette méthode au problème de l'intersection de surfaces en 3 dimensions et plus.

Cette approche met en présence un résolveur algébrique basé sur la méthode de Ritt-Wu, un résolveur basé sur la méthode de Newton-Raphson et un algorithme de calcul des racines de polynômes.

Dufourd J-F, Mathis P et Schreck P proposent [DMS96], outre leur approche par décomposition présentée dans le paragraphe 1.3.4, d'adopter une approche coopérative à chaque étape d'une résolution locale. Pour ce faire, ils utilisent une architecture supervisée de type *tableau noir* composée de deux agents géométriques aux stratégies de résolution différentes et d'un agent numérique. Ces stratégies sont une recherche en profondeur d'abord, et une recherche bornée en largeur. L'agent maître a pour charge d'une part d'activer successivement les agents suivant un schéma d'activation préétabli et d'autre part d'assembler les sous-systèmes résolus contenus dans le tableau noir.

1.3.5 Où placer GDRev dans la résolution de contraintes géométriques

Notre sentiment est que l'approche coopérative, encore trop peu explorée dans le cadre spécifique de la construction de figures géométriques, est d'un intérêt manifeste. Et bien que la diversité des approches mises en jeu (géométriques, algébriques, numériques exactes, numériques approchée) ne soit pas le gage d'un pouvoir de résolution plus grand et surtout d'une rapidité de résolution accrue, elle apparaît pratiquement comme adéquate. D'autant plus que les problèmes géométriques abordés sont des problèmes non-linéaires particuliers pour lesquels la définition de résolveurs adaptés au domaine est avantageuse. L'étude rapportée dans le chapitre 10 en est le témoignage.

Nos préférences en matière de construction penchent en premier lieu vers les approches symboliques qui fournissent des solutions plus propres à l'animation, puis

numériques exactes. Néanmoins, la prise en compte des approches numériques approchées est symptomatique du fait que nous nous contentons éventuellement d'obtenir une construction même approchée, du moment que le dessin correspondant soit acceptable. Il s'agit par conséquent d'établir autant que possible un plan de construction d'une figure en s'appuyant sur un dessin de la figure obtenu par résolution coopérative du système d'équations correspondant.

1.4 Conclusion

Après avoir montré l'intérêt de la géométrie déclarative vis à vis de la géométrie impérative, nous avons détaillé différentes approches de la résolution de contraintes géométriques. Celles-ci ont montré leurs forces mais aussi leurs faiblesses. Elles nous ont conduit à nous intéresser aux concepts de la coopération. Celui-ci est abordé dans le chapitre suivant en examinant les différentes approches mises en œuvre dans les langages de programmation logique avec contraintes.

Chapitre 2

Systèmes de coopération de résolveurs

A la fin des années 80, le paradigme de la programmation logique avec contraintes [MS98] est né de la croisée des deux paradigmes déclaratifs que sont la programmation logique et la satisfaction de contraintes. D'une certaine manière, elle peut être interprétée comme une généralisation de la programmation logique dans laquelle l'algorithme d'unification est etendu par des algorithmes de résolution de contraintes portant sur d'autres domaines que les arbres.

Pour répondre à des besoins spécifiques, de nombreux résolveurs ont été définis et mis en œuvre efficacement. Ils manipulent par exemple des domaines continus ou discrets suivant des méthodes symboliques ou numériques. Mais, devant la difficulté présentée par la résolution de certaines contraintes, il n'est pas étonnant que de nombreux travaux concernant la coopération de résolveurs aient été menés durant ces dernières années. L'idée est simple. Elle consiste à réutiliser les résolveurs efficaces déjà mis en œuvre sur des domaines de calcul spécifiques afin de les assembler pour former un tout complémentaire dans lequel chacun des composants (résolveurs) pallierait aux limites des autres.

Nous examinons dans ce chapitre les différentes approches de la coopération de résolveurs mises en œuvre dans les langages de programmation avec contraintes. Dans la section 2.1, nous cataloguons ces différentes approches. Puis, nous présentons successivement les principales propositions émanant des différents thèmes catalogués, c'est-à-dire la combinaison de résolutions dans la section 2.2, l'intégration de résolveurs dans la section 2.3, et la coopération de résolveurs dans la section 2.4. Nous situons GDRev parmi ces différentes approches dans la section 2.5. Enfin, la section 2.6 conclue ce

chapitre.

2.1 Classification des approches coopératives

Les différents travaux concernant la coopération de résolveurs se répartissent à notre avis, selon les trois thèmes suivants :

- Combinaison de résolutions. Ce thème rassemble les travaux étudiant la manière d'associer des résolveurs existant défini sur des domaines de calcul distincts.
- Intégration de résolveurs. Ce thème rassemble les travaux consacrés aux interactions de deux résolveurs distincts, afin de pallier aux limites de l'un par l'autre.
- Coopération de résolveurs. Ce thème rassemble les travaux dans lesquels un ensemble de résolveurs interagissent afin de résoudre un système de contraintes sur un même domaine de calcul. Il s'agit de travaux qui ont généralement pour point commun de mettre en présence une collection de résolveurs modélisés comme des "agents" supervisés par un agent "maître".

Un critère de distinction pour les travaux de ces deux derniers thèmes peut s'exprimer par une notion de "flexibilité". Celle-ci est faible dans les approches consacrées à l'intégration de résolveurs, et elle est élevée dans celles consacrées à la coopération de résolveurs.

L'intérêt d'une flexibilité faible est de pouvoir définir une coopération étroite et efficace, alors que l'intérêt d'une flexibilité élevée est de pouvoir d'une part ajouter de nouveau résolveurs simplement, et d'autre part de réutiliser des résolveurs existant en les considérant comme des boîtes noires.

2.2 Combinaison de résolutions

L'idée de la combinaison de résolutions consiste plus précisément dans un premier temps à séparer un système de contraintes initial en sous-systèmes définis sur des domaines de calcul distincts. Puis il s'agit dans un deuxième temps de résoudre ces sous-systèmes à l'aide de résolveurs spécifiques afin de rassembler dans un troisième temps les solutions partielles produites pour former la solution globale.

Il se pose un problème si une variable appartient à deux sous-systèmes distincts. L'idée suivie pour résoudre ce problème consiste à considérer la variable commune comme une constante dans une théorie et comme une variable à déterminer dans l'autre.

L'approche la plus ancienne est celle de Nelson G et Oppen D-C [NO79]. Elle consiste à combiner des procédures de décision provenant de théories différentes en une seule. Pour ce faire, chaque procédure de décision propage les égalités entre variables qu'elle déduit de son ensemble courant de contraintes aux autres procédures de décision. La procédure de décision s'arrête lorsqu'une insatisfiabilité est mise en évidence, où que les différentes procédures de décision ne déduisent plus de formules. En présence d'une disjonction, les différentes procédures de décision sont appliquées récursivement.

Plus récemment, Kirchner H et Ringeissen C [KR94], et Baader F, Schulz K [BS95] se sont penchés sur ce thème afin d'apporter une solution au problème de cycle de dépendance entre variables contenus dans des théories disjointes.

2.3 Intégration de résolveurs

Les principales propositions d'intégration de résolveurs étudiées dans le domaine de la programmation avec contraintes sont celles de Chiu C-K et Lee J-H-M [CL94], de Beringer H et De Backer B [BdB95], de Colmerauer A [Col96], de Benhamou F et Granvillier L [Ben96, BG96, Gra97], et plus récemment de Rueher M et Solnon C [RS97]. Nous les présentons successivement dans les paragraphes suivant, ainsi que d'autres approches semblables.

2.3.1 Approche de Chiu C-K et Lee J-H-M

En 1994, Chiu C-K et Lee J-H-M présentent dans [CL94] leur approche de la coopération de résolveurs validée par leur prototype CIAL (Constaint Interval Arithmetic Language). Ce dernier met en présence deux algorithmes de résolution de contraintes numériques sur le domaine des intervalles. Le premier algorithme est un résolveur linéaire d'égalités basé sur l'algorithme d'élimination de Gauss généralisé sur le domaine des intervalles. Le deuxième est un résolveur non-linéaire avec l'algorithme dit de "narrowing".

Dans leur approche, les contraintes à résoudre sont décomposées en contraintes primitives à l'aide si besoin de variables intermédiaires. Ces dernières sont ensuite séparées en d'une part les contraintes linéaires et d'autre part les contraintes non-linéaires. Une fois les décompositions envoyées aux résolveurs respectifs, le résolveur linéaire est activé. Il résout son système de contraintes jusqu'à aboutir à un point fixe. Alors, le résolveur non-linéaire est activé pour de nouveau laisser la main au résolveur linéaire dès que le domaine d'une variable ne peut plus être réduit. Cette procédure s'arrête lorsqu'un point fixe est atteint, ou qu'une inconsistance est détectée.

2.3.2 Approche de Beringer H et De Backer B

Pour le langage de programmation logique avec contraintes ICE, Beringer H et De Backer B [BdB95] ont étudié l'intégration d'un résolveur de réduction de domaines, capable de manipuler les réels et les entiers, avec un résolveur linéaire sur les réels. Plus précisément, le résolveur linéaire sur les réels se compose pour les équations d'un algorithme de Gauss(-Jordan), et pour les inéquations d'un Simplexe révisé intégrant explicitement les bornes du domaine des variables. Dans leur proposition, la communication des bornes du domaine d'une variable apparaissant à la fois dans une contrainte linéaire sur les réels (c'est-à-dire dans une contrainte dont l'opérateur relationnel est préfixé du caractère #) et dans une contrainte de domaine (c'est-à-dire dans une contrainte dont l'opérateur relationnel est préfixé du caractère $) est automatique. Elle s'opère du résolveur linéaire vers le résolveur de réduction de domaines pour réduire à un singleton le domaine d'une variable, et inversement pour signaler la réduction du domaine d'une variable une fois un point fixe atteint par le résolveur de réduction de domaines.

2.3.3 Approche de Colmerauer A

Dans le langage de programmation Prolog IV, Colmerauer A met en présence un résolveur linéaire et un résolveur non-linéaire sur les intervalles. Le résolveur linéaire se compose d'un algorithme d'élimination de Gauss pour les équations, et d'un Simplexe pour les inéquations et diséquations. Le choix du résolveur utilisé pour résoudre les contraintes est laissé au programmeur. Ce dernier indique son choix pour chacune des contraintes en utilisant les opérateurs sur intervalles (.+., .-., .*. et ./.) ou les opérateurs sur les rationnels (+, -, * et /). La coopération se fait dans Prolog IV au travers de variables partagées. Celle-ci s'opère dans le sens du résolveur linéaire vers le résolveur

sur intervalles, ou inversement, dès lors que la variable partagée est instanciée, c'est-à-dire dans le cas des intervalles, dès lors que le domaine de la variable partagée est réduit à un singleton.

2.3.4 Approche de Benhamou F et Granvillier L

Benhamou F et Granvillier L [Ben96, BG96, Gra97] proposent d'intégrer un résolveur non-linéaire exact à un résolveur non-linéaire sur les intervalles. Leur objectif, validé par leur prototype COSINUS, est la définition d'un résolveur de contraintes polynomiales. Plus précisément, ils proposent d'ajouter au système de contraintes courant des équations redondantes par le biais d'un traitement préliminaire. Ce traitement associe le calcul de bases de Gröbner partielles à des algorithmes de substitution et de factorisation de polynômes afin d'aboutir à un système de contraintes ayant le même ensemble de solutions que le système initial. Suite à cela, des méthodes de Newton combinées à des techniques d'énumération des solutions sont appliquées. L'intérêt de ces manipulations ressort lors de l'application des méthodes sur intervalles qui se trouvent grandement accélérées.

2.3.5 Approche de Rueher M et Solnon C

Par opposition aux coopérations "passives" déjà présentées, Rueher M et Solnon C proposent dans [RS97] une coopération "active" entre un résolveur linéaire et un résolveur non-linéaire sur intervalles. Le caractère actif/passif fait référence à la fréquence avec laquelle le résolveur sur intervalles signale au résolveur linéaire la réduction des bornes du domaine d'une variable. Dans une intégration passive, celle-ci s'opère aussitôt que le résolveur sur intervalles a atteint un point fixe dans son processus de résolution. Alors que dans une intégration active, celle-ci se fait dès que ces dernières ont été "significativement" réduites.

Un aspect crucial dans une telle proposition est l'appréciation du caractère significatif d'une réduction. En effet, si le signalement de la réduction des bornes du domaine d'une variable est trop fréquent, on court le risque de ralentir considérablement le résolveur linéaire. A l'inverse, si cette fréquence est trop faible, le gain d'une telle approche est négligeable.

Pour valider leur approche, les auteurs ont mis en œuvre en Oz [Smo95] un prototype appelé CCC. Dans ce prototype, les auteurs ont choisi de signaler les réductions

des bornes du domaine des variables lorsque celles-ci l'ont au moins été de 10%, ou lorsque le résolveur sur intervalles a atteint un point fixe. Ce choix qui peut paraître élevé, c'est-à-dire conduire à une surcharge du résolveur linéaire, donne des résultats très encourageants.

2.3.6 Autres approches

D'autres approches ont été proposées dans le même sens d'un certain point de vue, comme par exemple celles-ci de Czapor S-R [Cza89], ou de Pesant G et Boyer M [PB94, Pes95].

Approche de Czapor S-R

Afin de s'attaquer plus efficacement à la résolution d'équations algébriques, Czapor S-R [Cza89] propose de combiner le calcul de bases de Gröbner à un algorithme de factorisation. L'objectif de ces factorisations est d'obtenir des bases réduites.

Approche de Pesant G et Boyer M

L'approche de Pesant G et Boyer M [PB94, Pes95] consiste à associer en prétraitement à un résolveur linéaire de type Gauss et/ou Simplexe, un algorithme d'encadrement d'équations quadratiques. Cet algorithme interprète géométriquement les contraintes quadratiques afin de substituer à chacune d'elles un encadrement par des enveloppes convexes interne et externe définies en termes d'inéquations linéaires. Cette approche validée par le système Quad-CLP($\mathcal{R}$) donne des résultats intéressants pour les contraintes quadratiques visées.

2.4 Coopération de résolveurs

Les principales approches coopératives que nous présentons successivement dans les paragraphes suivants sont celles de Monfroy E, Rusinowitch M et Schott R [MRS96, Mon96], de Marti P et Rueher M [Rue94, MR95, Mar96], et enfin de Hong H [Hon94].

2.4.1 Approche de Monfroy E, Rusinowitch M et Schott R

Monfroy E, Rusinowitch M et Schott R se sont attaqués au problème de la résolution de contraintes non-linéaires en proposant d'utiliser une procédure de résolution préétablie qui met en présence un résolveur linéaire, un résolveur non-linéaire exact (Base de Gröbner) et un résolveur exact de polynômes en une variable développé en Maple [MRS96, Mon96]. Cet ensemble est mis en œuvre dans le prototype COSAC (Constraint System Architecture). Ce prototype est basé sur la plateforme ECLiPSe qui assure le rôle central de synchronisation des différents résolveurs. Cette procédure de résolution consiste à appliquer le résolveur linéaire autant que possible. Puis, il s'agit de transformer le système de contraintes restant par le calcul de base de Gröbner afin d'aboutir à des équations polynomiales en une variable solubles par le dernier résolveur.

2.4.2 Approche de Marti P et Rueher M

Par opposition, Marti P et Rueher R [Rue94, MR95, Mar96] proposent une "libre" coopération entre des résolveurs hétérogènes dans une architecture "multi-agent", où un agent est un résolveur, c'est-à-dire un processus muni de capacités de résolution, muni de capacités de communication et gérant ses propres données. Dans leur prototype SDRC les différents agents sont un résolveur linéaire, un résolveur non-linéaire sur intervalles, et un résolveur non-linéaire exact (Base de Gröbner). Ceux-ci ne communiquent pas directement entre eux, mais par l'intermédiaire d'un agent appelé "moniteur". L'envoi d'un message par un résolveur se fait dès lors que ce dernier a atteint un point fixe dans son processus de résolution. L'agent moniteur a pour charge de diviser le système de contraintes initial en sous-systèmes dédiés aux différents résolveurs, de centraliser les messages envoyés, de répartir les messages reçus et enfin de gérer les points de choix liés aux résolveurs non-déterministes.

2.4.3 Approche de Hong H

Dans son rapport [Hon94], l'auteur s'attache plus particulièrement à étudier la confluence de la coopération de résolveurs, c'est-à-dire à apporter une réponse à l'interrogation portant sur la dépendance des solutions d'un système d'équations vis-à-vis de l'ordre d'application des différents résolveurs. Pour son étude, l'auteur s'appuie sur le modèle de coopération qui consiste à appliquer successivement les différents

résolveurs en présence jusqu'à l'obtention d'un point fixe.

2.5 Où placer GDRev dans la coopération de résolveurs

Tout d'abord, il convient de remarquer que la résolution des problèmes de construction de figures géométriques issus des manuels scolaires de collèges et lycées délimite un cadre particulier de problèmes non-linéaires ayant le même "aspect général".

Notre conviction est que l'approche coopérative est prometteuse et avantageuse d'un point de vue génie logiciel. Le choix de cette approche coopérative se justifie aussi par la prise en compte du domaine d'application qu'est la géométrie qui peut en résulter. Cette prise en compte permet de définir des résolveurs adaptés. Enfin, ce choix se justifie aussi par la possibilité d'utiliser le système opérationnel Prolog IV qui dispose de plusieurs résolveurs.

Notre approche met en présence des résolveurs linéaire exact (le résolveur linéaire de Prolog IV), quadratique partiel (mis en œuvre en Prolog IV), sur intervalles (le résolveur sur intervalles de Prolog IV) et symbolique (mis en œuvre en Prolog IV). Elle se caractérise par conséquent par une approche à flexibilité élevée.

Elle se distingue des travaux présentés dans ce chapitre et dans le chapitre précédent par la diversité des résolveurs mis en jeu dans le processus de construction. En particulier, son originalité réside dans l'utilisation des résolveurs quadratiques partiel, complétion de propriétés (dont l'objectif est d'ajouter des contraintes géométriques redondantes) et complétion d'objets (dont l'objectif est d'ajouter des objets intermédiaires facilitant la construction de la figure), qui sont symptomatiques de la prise en compte d'un domaine d'application précis : les constructions géométriques.

Nous présentons en détail les résolveurs impliqués dans GDRev dans le chapitre 7, et nous explicitons leurs interactions dans le chapitre 8.

2.6 Conclusion

Nous avons présenté dans ce chapitre ce qui nous semble être les principales approche du thème de recherche de la coopération de résolveurs. Le nombre important de propositions et d'axes de recherche dans ce domaine souligne son importance et son

intérêt. D'ailleurs, afin de rassembler les travaux déjà entamés et d'étendre la flexibilité et l'adaptabilité des approches, une proposition de groupe de travail sur le thème de la "Collaboration de solveurs" a été faite récemment.

Deuxième partie

Définition du système GDRev

Chapitre 3

Présentation informelle de GDRev

GDREV est un système déclaratif pour la géométrie dynamique. Ce système est destiné plus particulièrement à la résolution des problèmes de construction de figures géométriques issus des manuels scolaires de collège et de lycée, mais il permet aussi de construire des figures plus compliquées, comme des figures fractales.

Avant de définir dans cette deuxième partie de ce manuscrit l'interface d'un tel environnement de programmation géométrique déclarative, nous introduisons informellement dans ce chapitre ce système au travers d'un exemple d'activité géométrique. Nous présentons brièvement dans la section 3.1 l'architecture générale du système GDRev. Puis, nous détaillons dans la section 3.2 un exemple de scénario de découverte à l'aide de GDRev. Les aspects liés à la procédure de résolution sont considérés dans la troisième partie de ce manuscrit.

3.1 Architecture globale

GDRev est un système fonctionnant sur PC sous Windows 95/98/NT 4.0. Une impression écran du système est donnée par la figure 3.1. Sans entrer dans les détails, on distingue sur cette figure une fenêtre principale dans laquelle on discerne les éléments suivants :

- Des menus : à travers lesquels l'utilisateur a accès aux fonctions de haut niveau de GDRev relatives aux fichiers, à l'édition, aux figures, aux préférences du système, à l'affichage du système, aux fenêtres d'affichage et à l'aide en ligne.

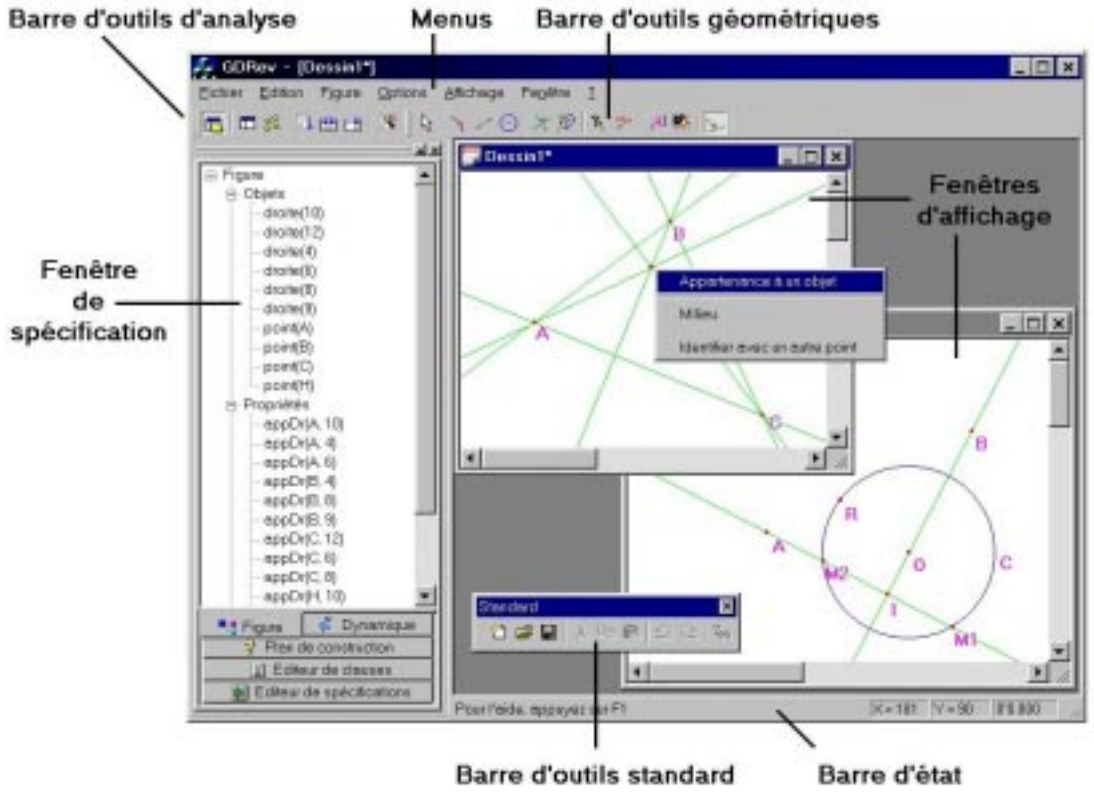

FIG. 3.1 – *Vue du système GDRev.*

– Des barres de contrôle : à travers lesquelles l'utilisateurs a accès aux principales fonctionnalités du système. Parmi ces barres de contrôle, on reconnaît :

  – Une barre d'outils standard : qui regroupe les principales fonctions des menus fichier et édition.

  – Une barre d'outils d'analyse : qui regroupe les fonctions relatives à l'analyse des spécifications éditées dans la fenêtre de spécification.

  – Une barre d'outils géométriques : qui regroupe par catégorie toutes les fonctions de création, de construction, d'annotation et d'affichage des éléments d'une figure.

  – Une barre d'état : dans laquelle sont affichés les messages d'aide, les coordonnées de la souris dans la fenêtre d'affichage active, et enfin le temps nécessaire à la (re)construction de la figure.

Chacune de ces barres de contrôle peut être ancrée à un côté de la fenêtre principale, comme l'illustre par exemple la barre d'outils géométriques sur la figure 3.1,

ou flottante, comme l'illustre la barre d'outils standard.

- Un ensemble de fenêtres d'affichage : dans lesquelles les dessins correspondant aux figures spécifiées sont affichés.

- Une fenêtre de spécification : dans laquelle s'affiche et/ou est éditée la spécification de la figure courante. Cette fenêtre a aussi la faculté d'ancrage à un côté de la fenêtre principale, comme nous pouvons le voir sur la figure 3.1. Elle se compose de 5 onglets relatifs à la fenêtre d'affichage active. Ces onglets sont intitulés :

  - Figure. L'utilisateur retrouve ici la spécification géométrique de la figure courante. Les différents langages géométriques utilisés pour décrire une figure sont détaillés dans la chapitre 4.
  - Dynamique. L'utilisateur retrouve ici la spécification dynamique de la figure courante. Les différents états caractérisant les objets d'une figure dynamique, ainsi que les fonctionnalités capables de manipuler ces aspects, sont détaillés dans le paragraphe 5.2.1 page 79.
  - Editeur de spécifications. L'utilisateur peut ici éditer d'une manière déclarative les spécifications de sa figure dans les langages décrits dans le chapitre 4.
  - Editeur de clauses. L'utilisateur peut ici éditer des clauses, dont les concepts sont détaillés dans le chapitre 4.
  - Plan de construction. L'utilisateur retrouve ici une construction impérative de la figure courante.

Pour spécifier logiquement une figure géométrique, nous avons défini le langage ELDL (Extended Logical Description Language) du premier ordre. Ce langage comprend des atomes de typage qui expriment le type et les composants d'un objet, et des atomes de propriété qui expriment les relations géométriques liant des objets. La spécification d'une figure en ELDL est la conjonction d'atomes.

3.2 Exemple de scénario

Le problème de construction abordé ici est une activité d'exploration que l'on peut imaginer autour de la notion d'orthocentre d'un triangle.

Exemple 3.2 ORTHOCENTRE D'UN TRIANGLE

Construire un triangle ABC pourvu de ses hauteurs D_{AH}, D_{BH}, D_{CH}, et de son orthocentre H à partir des points A, B et C. Le dessin attendu est reproduit sur la figure 3.2.

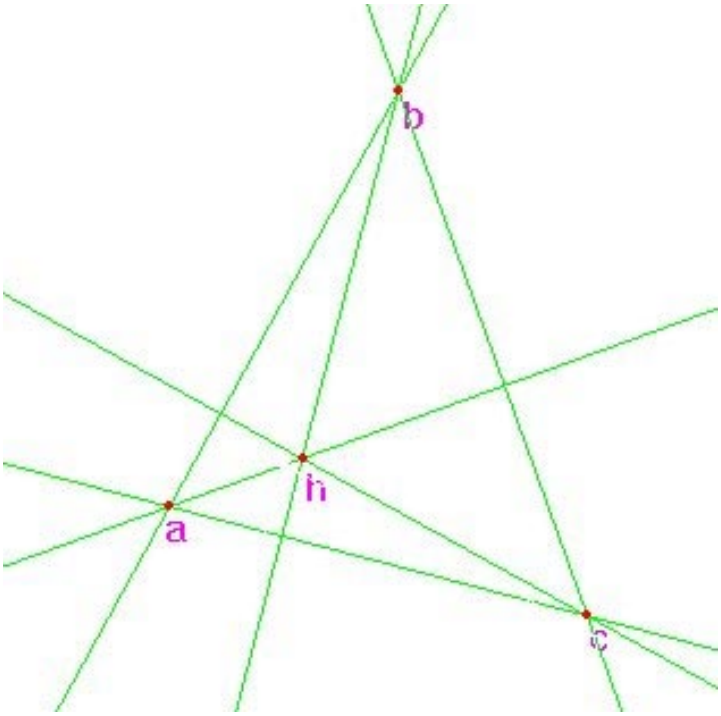

FIG. *3.2* – Triangle et orthocentre.

Dans le langage ELDL, une spécification géométrique répondant à cet énoncé est la suivante :

$$\begin{aligned}
&\mathsf{point(A)} \wedge \mathsf{point(B)} \wedge \mathsf{point(C)} \wedge \mathsf{point(H)} \wedge \\
&\mathsf{droite}(\mathsf{D}_{AB}) \wedge \mathsf{droite}(\mathsf{D}_{AC}) \wedge \mathsf{droite}(\mathsf{D}_{BC}) \wedge \\
&\mathsf{droite}(\mathsf{D}_{AH}) \wedge \mathsf{droite}(\mathsf{D}_{BH}) \wedge \mathsf{droite}(\mathsf{D}_{CH}) \wedge \\
&\mathsf{appDr}(\mathsf{A}, \mathsf{D}_{AB}) \wedge \mathsf{appDr}(\mathsf{A}, \mathsf{D}_{AC}) \wedge \mathsf{appDr}(\mathsf{A}, \mathsf{D}_{AH}) \wedge \\
&\mathsf{appDr}(\mathsf{B}, \mathsf{D}_{AB}) \wedge \mathsf{appDr}(\mathsf{B}, \mathsf{D}_{BC}) \wedge \mathsf{appDr}(\mathsf{B}, \mathsf{D}_{BH}) \wedge \\
&\mathsf{appDr}(\mathsf{C}, \mathsf{D}_{AC}) \wedge \mathsf{appDr}(\mathsf{C}, \mathsf{D}_{BC}) \wedge \mathsf{appDr}(\mathsf{C}, \mathsf{D}_{CH}) \wedge \\
&\mathsf{appDr}(\mathsf{H}, \mathsf{D}_{AH}) \wedge \mathsf{appDr}(\mathsf{H}, \mathsf{D}_{BH}) \wedge \mathsf{appDr}(\mathsf{H}, \mathsf{D}_{CH}) \wedge \\
&\mathsf{perp}(\mathsf{D}_{AB}, \mathsf{D}_{CH}) \wedge \mathsf{perp}(\mathsf{D}_{AC}, \mathsf{D}_{BH}) \wedge \mathsf{perp}(\mathsf{D}_{BC}, \mathsf{D}_{AH})
\end{aligned}$$

L'acquisition de cette spécification peut être fait suivant l'une et/ou l'autre des deux approches suivantes :

- Approche textuelle. Cette approche consiste à analyser la spécification géométrique préalablement éditée dans l'onglet "Editeur de spécifications" de la fenêtre de spécification, puis à donner la spécification dynamique de la figure, c'est-à-dire à placer les points de base A, B et C de la figure.

- Approche graphique. Cette approche consiste à utiliser les outils mis à disposition dans la barre des outils géométriques à la manière de Cabri-Géomètre.

  Par exemple, un utilisateur peut commencer par :

  1. *Créer les points A, B et C.*
  2. *Construire à partir de ces points les droites D_{AB}, D_{AC} et D_{BC}.*
  3. *Construire à partir des objets déjà construits ou créés les hauteurs D_{AH} et D_{BH}.*
  4. *Construire le point d'intersection H des précédentes hauteurs.*
  5. *Et enfin construire la hauteur D_{CH}.*

  Pour achever cette construction, c'est-à-dire ajouter à la spécification de la figure la propriété d'appartenance du point H à la hauteur D_{CH}, l'utilisateur dispose d'une fonction *Ajouter propriété* dans la barre d'outils géométriques. Pour cela, il lui suffit de sélectionner un des objets argument de la propriété, puis de choisir dans le menu déroulant apparaissant la propriété désirée (voir figure 3.1), puis enfin de sélectionner les autres arguments de la propriété.

  En utilisant cette méthode, l'utilisateur définit à la fois la spécification géométrique et la spécification dynamique de sa figure.

  Dans l'exemple précédent, les objets de base sont les objets créés, et les objets qui ne sont pas de base sont les objets construits.

La résolution de cette spécification géométrique avec la donnée des points A, B et C conduit à une figure dynamique qui permet à l'utilisateur d'observer les positions du point H en déplaçant le point A (resp. B) (resp. C) avec les points B et C fixes (resp. A et C) (resp. A et B).

L'intérêt de l'approche déclarative apparaît si l'utilisateur manifeste l'intention d'animer le point H avec par exemple les points B et C fixes, c'est-à-dire si l'utilisateur

manifeste l'intention de modifier la spécification dynamique de sa figure. Pour ce faire, il dispose des fonctions *Libérer point* et *Fixer point* dans la barre d'outils géométriques.

Un exemple de transition entre la spécification dynamique où les points A, B et C sont donnés et la spécification dynamique où les points B, C et H sont donnés, consiste à libérer le point A dans un premier temps, puis à fixer le point H par un clic du bouton gauche de la souris dans la fenêtre d'affichage correspondante dans un deuxième temps.

A tout moment, l'utilisateur peut demander la recherche d'une construction impérative de sa figure à l'aide de la règle et du compas en sélectionnant l'item *Plan de construction* du menu *Figure*, ou en activant le bouton correspondant de la barre d'outils d'analyse. Ainsi, à partir de la dernière configuration dynamique, un plan de construction possible, produit par GDRev grâce à cette fonctionnalité, est rapporté ci-dessous.

1. *Créer le point B,*
2. *Créer le point C,*
3. *Créer le point H,*
4. *Construire la droite D_{BC} passant par le point B et le point C,*
5. *Construire la droite D_{AH} passant par le point H et perpendiculaire à la droite D_{AC},*
6. *Construire la droite D_{BH} passant par le point B et le point H,*
7. *Construire la droite D_{CH} passant par le point C et le point H,*
8. *Construire la droite D_{AC} passant par le point C et perpendiculaire à la droite D_{BH},*
9. *Construire la droite D_{AB} passant par le point B et perpendiculaire à la droite D_{CH},*
10. *Construire le point A à l'intersection de la droite D_{AB} et de la droite D_{AC}.*

L'intérêt d'une telle fonctionnalité est qu'elle forme un pont entre les approches impérative et déclarative de la géométrie dynamique. Nous remarquons que dans le cas présent, la construction est relativement triviale. Une telle simplicité ne se rencontre pas toujours, comme nous l'expliquons dans le chapitre 7.

Chapitre 4

Langages de spécification de figures dynamiques de GDRev

La définition d'un environnement de programmation géométrique déclarative nécessite la définition de multiples langages de types et de niveaux différents. Certains, constituant l'"interface", permettent les échanges entre l'utilisateur et le système. Les autres au contraire sont transparents à l'utilisateur et permettent une représentation précise des connaissances nécessaires aux mécanismes de résolution. Dans leur ensemble, ils définissent un certain pouvoir de description qui constitue un indice de l'utilisation du système et de son applicabilité à différents domaines.

Dans ce chapitre, nous détaillons les langages de spécification de GDRev, et nous montrons comment spécifier dans ces langages des exercices de construction. Nous répertorions rapidement dans la section 4.1 les différents langages présents dans le système GDRev et nous présentons leur hiérarchie. Ensuite, nous décrivons plus précisément dans la section 4.2 les langages logiques de GDRev.

4.1 Différents niveaux de langages

Le paragraphe 4.1.1 classifie les différents langages par catégorie. Les paragraphes 4.1.2 et 4.1.3 présentent pour chacune de ces catégories une hiérarchie organisant les langages les composant.

4.1.1 Catégories de langages

De l'analyse et de l'utilisation des systèmes existants de géométrie dynamique, il ressort le besoin de définir des langages dans les trois catégories suivantes :

- les langages géométriques. Ils permettent de définir la spécification géométrique d'une figure, c'est-à-dire la spécification d'un ensemble d'objets et des propriétés qui les lient entre eux. Dans l'idéal, les propriétés que nous attendons de ces langages sont d'une part qu'ils permettent à un utilisateur de spécifier la figure qu'il attend dans le cas de solutions multiples, et d'autre part qu'ils supportent la définition d'objets complexes à l'image des macros construction de Cabri-Géomètre ou des scripts de Geometer's Sketchpad. De telles constructions doivent aussi permettre la construction de nouveaux objets si certains éléments de la figure vérifient une propriété particulière.

- les langages d'affichage. Ils permettent de spécifier les paramètres d'affichage des objets d'une figure, comme par exemple la couleur, l'épaisseur, le style ou encore la visibilité. En gardant à l'esprit le caractère dynamique d'une figure, il s'en suit la nécessité de pouvoir spécifier dynamiquement les paramètres d'affichage des éléments de cette dernière. Par exemple, nous souhaitons pouvoir spécifier que les représentations des points vérifiant une certaine propriété soient une croix et que celles de ceux ne vérifiant pas cette propriété soient un gros rond plein.

- les langages d'animation. Ils permettent de définir la spécification dynamique d'une figure. De même que pour les langages d'affichage, de telles spécifications comprennent non seulement les objets d'une figure à partir desquels celle-ci est construite, mais aussi les conditions de l'animation des différents objets. Par exemple, on pourrait aimer spécifier qu'un point doit être considéré comme libre si deux autres objets vérifient une propriété particulière.

Dans chacune de ces catégories, on peut distinguer deux niveaux de langage. Le premier type correspond à la définition de spécifications "élémentaires" comme la définition d'un objet, la spécification de la couleur d'un objet, la spécification des coordonnées d'un objet .... Le deuxième type correspond à la définition de spécifications "composées" comme la construction de nouveaux objets si certains éléments de la figure vérifient une propriété particulière, la modification de l'aspects des points suivant qu'ils vérifient ou non une propriété particulière, la libération des coordonnées d'un objet si deux autres objets vérifient une propriété particulière, ...

Dans la suite de ce manuscrit, nous nous intéressons plus particulièrement aux langages géométriques.

Tout d'abord, nous nous intéressons aux différents niveaux qui caractérisent les langages que nous avons qualifiés de "élémentaires", puis à ceux des langages que nous avons qualifiés de "composés"

4.1.2 Niveaux des langages "élémentaires"

On peut penser acquérir les "élémentaires" spécifications géométriques d'une figure au travers d'un langage de description textuel, ou d'un langage de description graphique, ou encore d'un langage d'interface. Nous avons appelé de tels langages :

- Student Construction Language (SCL). Il permet de décrire une figure à travers un langage d'interface utilisant des menus, boutons, ...
- Classroom Description Language (CDL). Il permet de décrire une figure à travers un langage textuel.
- Drawing Description Language (DDL). Il permet de décrire une figure à partir d'un dessin réalisé à main levée.

Pour définir ces langages, il convient de proposer un langage au caractère logique dans lequel les spécifications décrites dans l'un ou l'autre des langages précédents sont traduites. L'objectif de cette définition est de s'appuyer sur une base solide permettant de décrire précisément les figures à construire par des mécanismes de résolution de contraintes géométriques. Ce langage est le :

- Logical Description Language (LDL). Il permet de décrire sous la forme de formules logiques des figures.

L'organisation hiérarchique de ces différents langages élémentaires est illustrée par le diagramme de la figure 4.1.

L'étude et la mise en œuvre d'un langage tel que DDL ne sont pas abordées ici. Signalons qu'une étude a été menée par Nebon F [Neb96] sur l'applicabilité de l'apprentissage automatique de concepts à l'acquisition automatique d'une spécification logique géométrique à partir d'un dessin. Cette étude a conduit à la mise en œuvre d'un prototype de langage DDL.

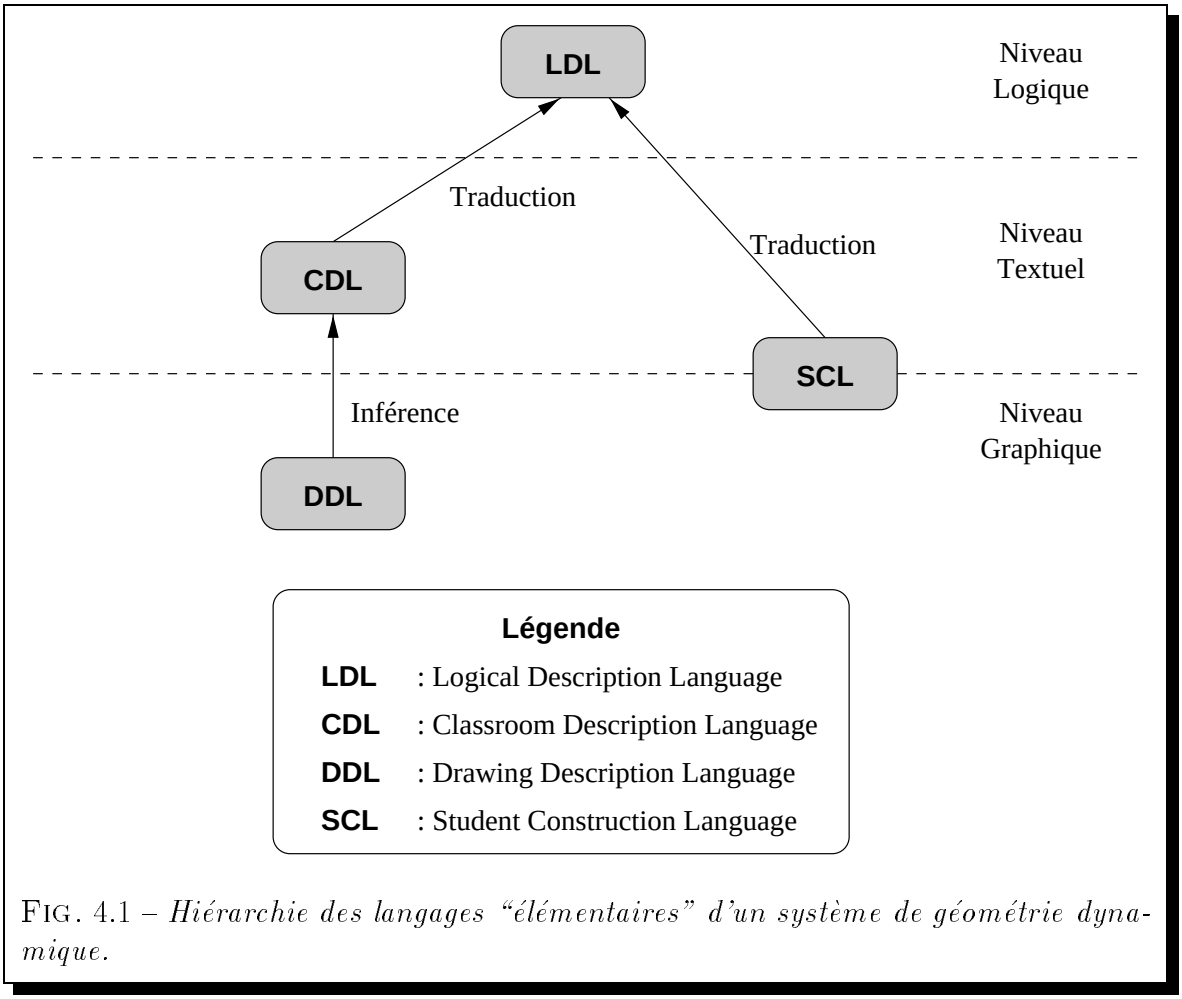

FIG. 4.1 – *Hiérarchie des langages "élémentaires" d'un système de géométrie dynamique.*

4.1.3 Niveaux des langages "composés"

Il semble clair que l'on peut à nouveau penser acquérir les spécifications géométriques d'une figure "composée" au travers d'un langage de description textuel, ou d'un langage de description graphique, ou encore d'un langage d'interface.

Afin de toujours nous appuyer sur un langage de base solide, il convient d'intégrer en premier lieu les extensions désirées au langage logique LDL, pour ensuite répercuter ces extensions aux différents langages textuels ou graphiques. Ceci nous conduit à la définition des langages que avons appelés :

- Extended Logical Description Language (ELDL). Il permet d'étendre le langage LDL en intégrant au niveau des spécifications logiques les deux fonctionnalités suivantes :
  - la modularité et la récursivité : qui permettent de décrire des objets complexes

en termes d'objets déjà définis.

- la négation : qui permet de spécifier que des objets ne vérifient pas une certaine propriété.

- Extented Student Construction Language (ESCL). Il permet d'étendre le langage SCL en supportant au niveau de l'interface les deux fonctionnalités ajoutées au niveau logique.

- Extented Classroom Description Language (ECDL). Il permet d'étendre le langage CDL en supportant au niveau textuel les deux fonctionnalités ajoutées au niveau logique

- Extented Drawing Description Language (EDDL). Il permet d'étendre le langage DDL en supportant la description des figures récursives à partir de la donnée d'un ensemble de dessins réalisés à main levée.

L'organisation hiérarchique des langages dans leur ensemble est illustrée par le diagramme de la figure 4.2.

De même, l'étude du langage EDDL n'est pas abordée ici. Signalons qu'une étude a été menée par Coste P [Cos97] sur l'applicabilité de l'apprentissage automatique de concepts à l'acquisition automatique d'une spécification logique géométrique récursive à partir d'un ensemble de dessins. Cette étude a aussi conduit à une mises en œuvres d'un prototype de langage EDDL.

L'étude que nous menons ici porte exclusivement sur les langages géométriques LDL et ELDL. Le chapitre 5 est consacré aux langages SCL et ESCL.

4.2 Langages géométriques

Les paragraphes qui suivent sont consacrés à la description des langages géométriques LDL et ELDL de la hiérarchie décrite dans les sections précédentes.

4.2.1 Langage LDL

Le langage logique géométrique LDL définissant les figures est un langage du premier ordre. Il constitue un langage central de notre système. C'est au travers de formules exprimées dans ce langage que les différents résolveurs de notre environnement

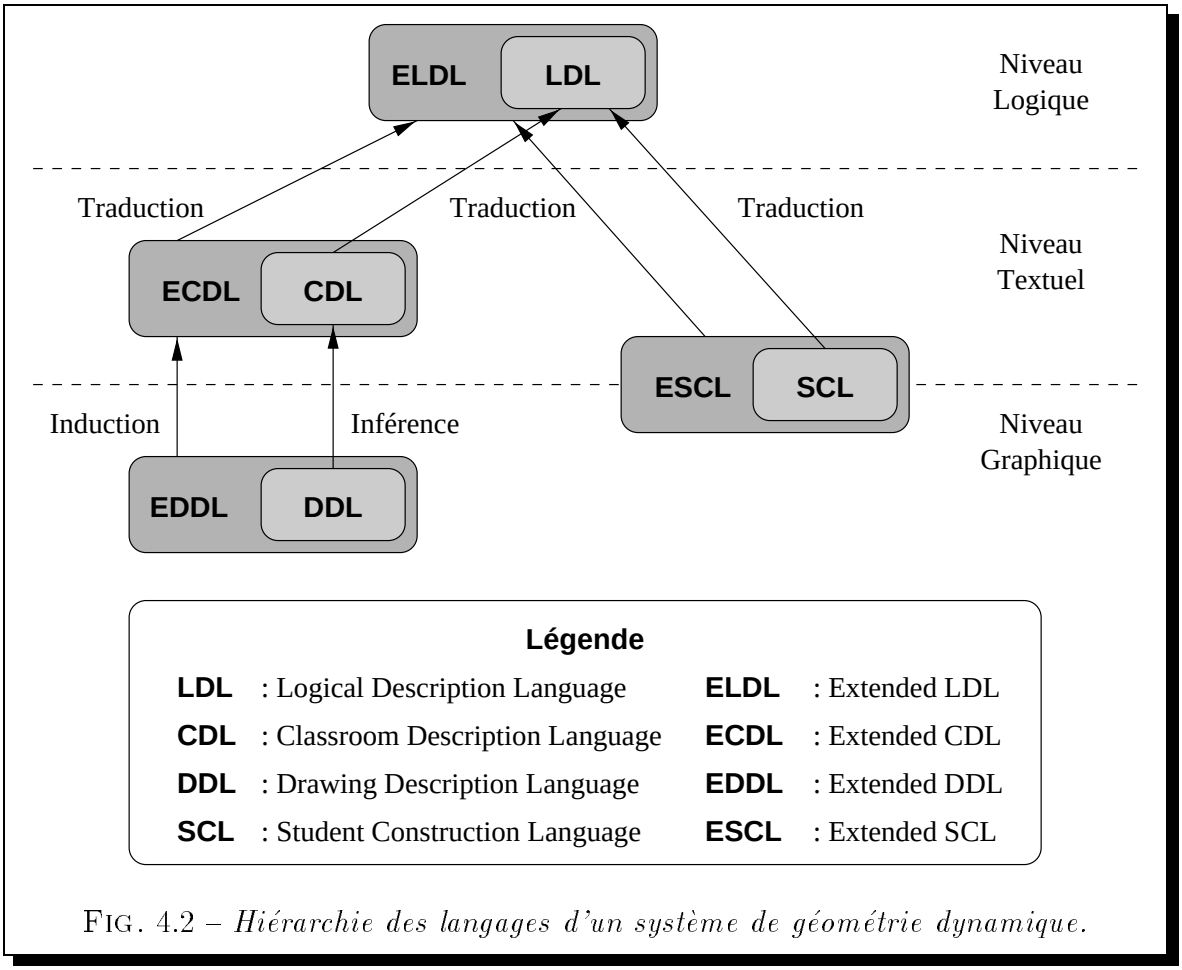

FIG. 4.2 – *Hiérarchie des langages d'un système de géométrie dynamique.*

communiquent afin de concourir à la construction d'une figure, comme nous l'expliquons dans le chapitre 8.

Dans un premier paragraphe, nous décrivons le langage logique géométrique LDL tiré des travaux de [Des94]. Ensuite, nous illustrons ce langage par quelques exemples de spécification.

Présentation du langage LDL

Le langage logique géométrique LDL se décompose en deux classes d'atome. Ces deux classes d'atome sont :

- les atomes de typage : qui expriment le type et les composants d'un objet.
- les atomes de propriété : qui expriment les propriétés liant les éléments de la figure.

Le tableau 4.1 récapitule les différents atomes de typage et les atomes de propriété du langage, et détaille pour chacun d'eux leurs arguments et leur signification.

Atome	**Signification**
TYPAGE	
point(p)	p est un point
droite(l)	l est une droite
demiDroite(h, p, l)	h est une demi-droite d'origine le point p ...
	... et de support la droite l
segment(s, p_1, p_2, l)	s est un segment d'extrémité les points p_1 et p_2 ...
	... et de support la droite l.
cercle(c, o, r)	c est le cercle de centre le point o et de rayon la distance r
distance(d)	d est une distance
PROPRIÉTÉ	
appDr(p, l)	le point p appartient à la droite l
appDD(p, h)	le point p appartient à la demi-droite l
appSeg(p, s)	le point p appartient au segment s
appCc(p, c)	le point p appartient au cercle c
par(l_1, l_2)	les droites l_1 et l_2 sont parallèles
perp(l_1, l_2)	les droites l_1 et l_2 sont perpendiculaires
invSens(h_1, h_2)	les demi-droites h_1 et h_2 ont sens inverse
memeSens(h_1, h_2)	les demi-droites h_1 et h_2 ont même sens
distPP(d, p_1, p_2)	d est la distance séparant les points p_1 et p_2
infDist(d_1, d_2)	la distance d_1 est strictement inférieure ...
	... à la distance d_2
demiDist(d_1, d_2)	la distance d_1 est égale à $\frac{1}{2}$ fois la distance d_2
milieu(p, p_1, p_2)	le point p est le milieu des points p_1 et p_2
egalPt(p_1, p_2)	les points p_1 et p_2 sont égaux
egalDr(d_1, d_2)	les droites d_1 et d_2 sont égales
egalDD(h_1, h_2)	les demi-droites h_1 et h_2 sont égales
egalSeg(s_1, s_2)	les segments s_1 et s_2 sont égaux
egalCc(c_1, c_2)	les cercles c_1 et c_2 sont égaux
egalDist(d_1, d_2)	les distances d_1 et d_2 sont égales

TAB. 4.1 – *Types et propriétés du langage LDL*

Dans ce langage logique du premier ordre, une formule décrivant une figure satisfait les trois contraintes suivantes :

1. Elle est une conjonction de formules de base. C'est-à-dire elle est de la forme :

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n$$

   où P_i est un atome, c'est-à-dire soit un atome de typage, soit un atome de propriété.

2. Elle est close.

3. A tout identificateur est associé un et un seul atome de typage.

Les choix des composants des atomes de typage ont été guidés par l'importance qu'ils nous semblent apporter aux démonstrations possibles. Comme nous l'expliquons dans le chapitre 7, les résolveurs employés sont sensibles aux objets présents dans une spécification. Ainsi, nous estimons que la désignation d'une demi-droite h ($demiDroite(h, p, l)$) signifie implicitement la construction de la droite l qui la supporte et du point p qui marque son extrémité. De même, la construction d'un segment s ($segment(s, p_1, p_2, l)$) signifie implicitement la construction des deux points p_1 et p_2 qui constituent ses extrémités et de la droite l qui le supporte, avec la contrainte supplémentaire que $p_1 \leq p_2$ dans l'ordre lexicographique usuel. Aussi, la construction d'un cercle c ($cercle(c, o, r)$) signifie implicitement la construction du point o qui est son centre et de la distance r qui est le rayon de ce dernier.

Exemples de spécifications LDL

Afin de clarifier nos propos, nous présentons les formules logiques géométriques correspondant à deux exercices de construction. Le premier est tiré du livre de mathématiques de collège [TDVP88].

Exemple 4.3 Exercice de collège

On considère trois points A, B et C d'un cercle de centre O. On désigne par A', B' et C' les milieux des côtés $[BC]$, $[CA]$ et $[AB]$, et I, J et K les symétriques de O par rapport aux points A', B' et C'. Construire le triangle IJK.

Cette construction est illustrée par la figure 4.3.

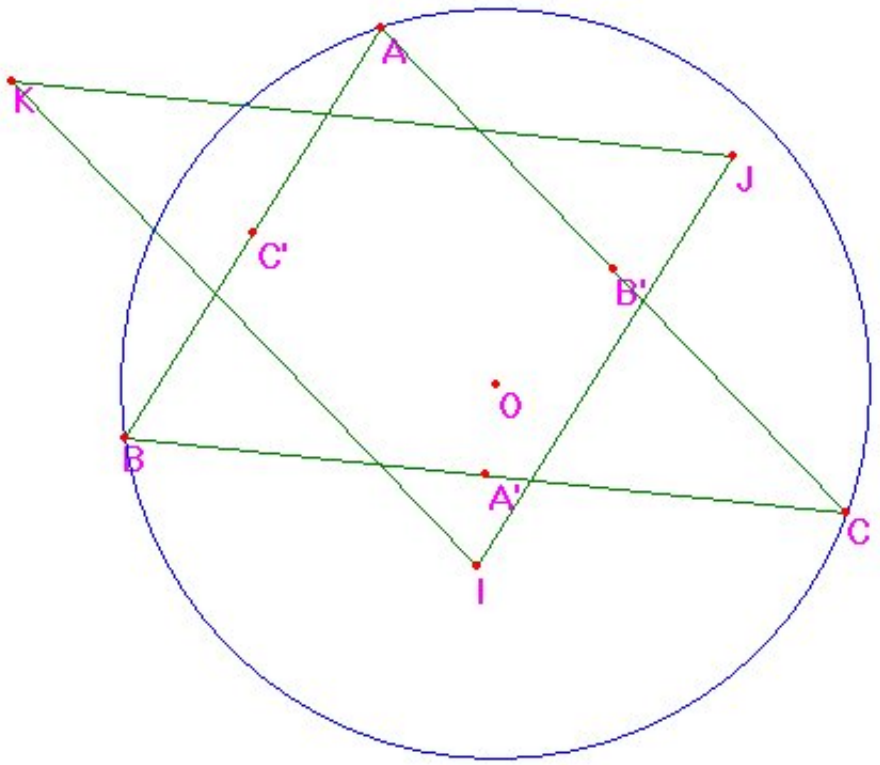

FIG. *4.3* – Exercice de construction de collège.

La spécification LDL de cette figure est :

$$
\begin{aligned}
&\mathit{point}(A) \wedge \ \mathit{point}(B) \wedge \ \mathit{point}(C) \wedge \\
&\mathit{cercle}(c, O, r) \wedge \ \mathit{point}(O) \wedge \ \mathit{distance}(r) \wedge \\
&\mathit{appCc}(A, c) \wedge \ \mathit{appCc}(B, c) \wedge \ \mathit{appCc}(C, c) \wedge \\
&\mathit{point}(A') \wedge \ \mathit{point}(B') \wedge \ \mathit{point}(C') \wedge \\
&\mathit{segment}(s_{AB}, A, B, I_{AB}) \wedge \ \mathit{segment}(s_{AC}, A, C, I_{AC}) \wedge \\
&\mathit{segment}(s_{BC}, B, C, I_{BC}) \wedge \\
&\mathit{milieu}(A', B, C) \wedge \ \mathit{milieu}(B', C, A) \wedge \ \mathit{milieu}(C', A, B) \wedge \\
&\mathit{point}(I) \wedge \ \mathit{point}(J) \wedge \ \mathit{point}(K) \wedge \\
&\mathit{milieu}(A', I, O) \wedge \ \mathit{milieu}(B', J, O) \wedge \ \mathit{milieu}(C', K, O) \wedge \\
&\mathit{segment}(s_{IJ}, I, J, I_{IJ}) \wedge \ \mathit{segment}(s_{IK}, I, K, I_{IK}) \wedge \\
&\mathit{segment}(s_{JK}, J, K, I_{JK}).
\end{aligned}
$$

Cet autre exemple montre comment définir une contrainte issue du domaine de la CAO.

Exemple 4.4 LONGUEUR

Construire un segment de longueur donnée ayant comme extrémité un point donné et parallèle à une direction donnée. La figure 4.4 illustre la figure attendue.

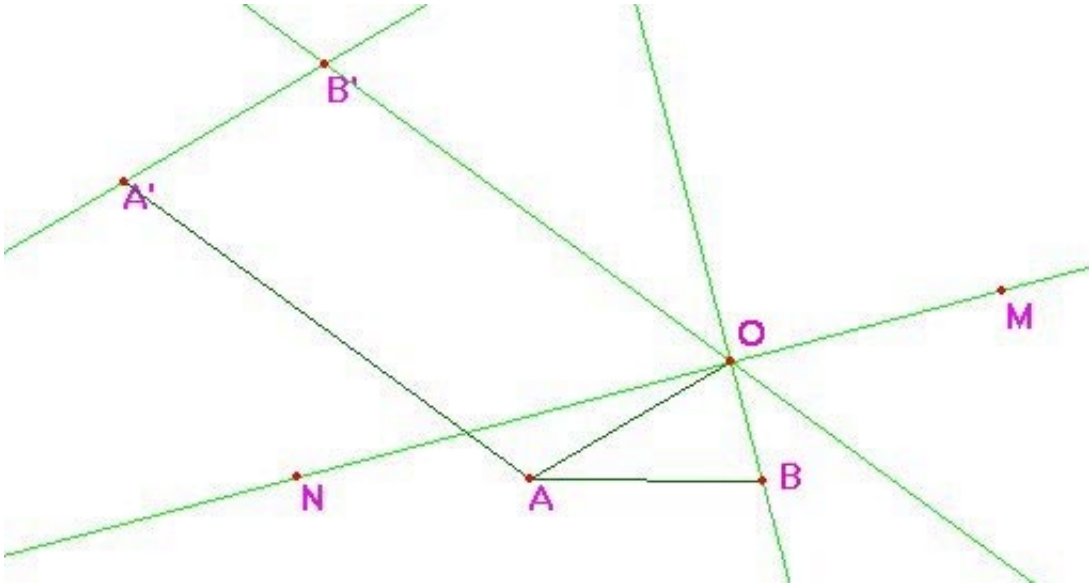

FIG. *4.4* – Segment de longueur donnée.

Cette construction s'appuie sur les notions de rectangle et de parallélogramme. En effet, rappelons-nous d'une part que les diagonales d'un rectangle sont de même longueur et se coupent en leur milieu, et d'autre part que les côtés opposés d'un parallélogramme ont même longueur. Ainsi, en construisant un rectangle de diagonale NB centré en A, il est possible de construire un parallélogramme $AA'B'o$, afin de résoudre le problème posé.

La spécification LDL de cette figure est :

$$point(A) \wedge\ point(B) \wedge\ segment(s_{AB}, A, B, I_{AB}) \wedge$$
$$point(N) \wedge\ milieu(A, N, B) \wedge$$
$$point(M) \wedge\ droite(I_{MN}) \wedge\ appDr(M, I_{MN}) \wedge\ appDr(N, I_{MN}) \wedge$$
$$droite(I_{Bo}) \wedge\ appDr(B, I_{Bo}) \wedge\ perp(I_{Bo}, I_{MN}) \wedge$$

$point(o) \wedge\ appDr(o, l_{MN}) \wedge\ appDr(o, l_{Bo}) \wedge$

$segment(s_{Ao}, A, o, l_{Ao}) \wedge\ segment(s_{AA'}, A, A', l_{AA'}) \wedge$

$droite(l_{oB'}) \wedge\ appDr(o, l_{oB'}) \wedge\ appDr(B', l_{oB'}) \wedge\ par(l_{oB'}, s_{AA'}) \wedge$

$droite(l_{A'B'}) \wedge\ appDr(A', l_{A'B'}) \wedge\ appDr(B', l_{A'B'}) \wedge\ par(l_{A'B'}, s_{Ao}) \wedge$

$point(A') \wedge\ point(B') \wedge\ segment(s_{A'B'}, A', B', l_{A'B'}).$

Cette spécification a la particularité que les équations correspondant aux objets et aux propriétés qui la composent soient linéaires.

4.2.2 Langage ELDL

Nous nous intéressons à une extension du langage LDL, le langage ELDL, pour répondre à deux problèmes importants. Nous exposons d'abord la problématique étudiée et les solutions que nous proposons. Puis, nous décrivons la sémantique du langage défini. Enfin, nous donnons des exemples de spécifications ELDL.

Le détail de l'interpréteur associé à ce langage est donné dans le chapitre 9.

Motivations des extensions du langage LDL

Les améliorations que nous souhaitons apporter au langage LDL portent sur les deux points suivants :

- l'introduction de l'implication. Il ressort de l'analyse et de l'expérimentation du langage LDL tout le bénéfice que nous pourrions tirer de la définition d'objets en terme d'objets plus élémentaires. Des illustrations de ces spécifications se retrouvent par exemple dans les macros constructions de Cabri-Géomètre. Les bénéfices escomptés sont une plus grande simplicité, une plus grande clarté des définitions, la possible réutilisation de définitions et la possibilité de construire des figures récursives.

  Pour ce faire, nous proposons d'étendre les définitions des formules par l'introduction de l'implication en autorisant la définition de "règles". Une règle est un énoncé, donné par l'utilisateur, de la forme :

$$A \leftarrow B_1, B_2, \ldots, B_n \text{ avec } n \geq 0$$

où A et les B_i sont des atomes. A est la tête de la règle et les B_i forment le corps de celle-ci.

Exemple 4.5 EXEMPLE DE RÈGLE

La règle suivante définit la nouvelle relation symetrique *en terme de relations déjà existantes du langage LDL.*

$$\begin{aligned} &symetrique(A, B, C) \leftarrow \\ &\quad point(A) \wedge\ point(B) \wedge\ point(C) \wedge \\ &\quad milieu(C, B, A). \end{aligned}$$

Exemple 4.6 AUTRE EXEMPLE DE RÈGLE

La règle suivante définit la nouvelle relation mediatrice *en terme de relations déjà existantes du langage LDL.*

$$\begin{aligned} &mediatrice(A, B, D) \leftarrow \\ &\quad point(A) \wedge\ point(B) \wedge\ droite(D) \wedge \\ &\quad point(i) \wedge\ milieu(i, A, B) \wedge \\ &\quad droite(I) \wedge\ appDr(A, I) \wedge\ appDr(B, I) \wedge\ appDr(i, I) \wedge \\ &\quad appDr(i, D) \wedge\ perp(D, I). \end{aligned}$$

Exemple 4.7 EXEMPLE DE COMPOSITION DE RÈGLES

La première règle définit la relation triangle *en terme de relations déjà existantes du langage LDL. La deuxième règle définit la nouvelle relation* triangleRectangleEnA *en terme de relations déjà existantes du langage LDL et de la relation du langage ELDL* triangle.

$$\begin{aligned} &triangle(A, B, C, D_{AB}, D_{AC}, D_{BC}) \leftarrow \\ &\quad point(A) \wedge\ point(B) \wedge\ point(C) \wedge \\ &\quad droite(D_{AB}) \wedge\ droite(D_{AC}) \wedge\ droite(D_{BC}) \wedge \\ &\quad appDr(A, D_{AB}) \wedge\ appDr(A, D_{AC}) \wedge \\ &\quad appDr(B, D_{AB}) \wedge\ appDr(B, D_{BC}) \wedge \\ &\quad appDr(C, D_{AC}) \wedge\ appDr(C, D_{BC}). \end{aligned}$$

$$triangleRectangleEnA(A, B, C, D_{AB}, D_{AC}, D_{BC}) \leftarrow$$

triangle(A, B, C, D_{AB}, D_{AC}, D_{BC}) $\wedge$
perp(D_{AB}, D_{AC}).

Le concept de règle permet d'aller plus loin en autorisant des définitions récursives de relations[1]. Des exemples de telles définitions se retrouvent dans les figures fractales comme le flocon de Koch illustré par la figure 1.2 de la section 1.2.1 page 16.

Pour les mécanismes de résolution, et plus particulièrement pour le démonstrateur automatique utilisé [Ost97b], de telles règles sont représentées sous la forme d'axiomes qui complètent ainsi la théorie de la géométrie donnée en annexe A.

– l'introduction de la négation. Notre première motivation réside notre volonté de pouvoir dissocier par exemple les points d'intersection d'une droite et d'un cercle. L'objectif de cette introduction est aussi de pouvoir spécifier une théorie axiomatique de la géométrie comprenant des axiomes simples comme le premier postulat d'Euclide qui énonce que par deux points distincts ne passe qu'une seule droite.

  Dans [Des94], Desmoulins C apporte une première solution au traitement de la négation qu'il mis en œuvre dans TALC, un tuteur pour la construction de figures géométriques. Pour satisfaire la forme restreinte à des clauses de Horn de ses axiomes de la géométrie, il transforme tout littéral négatif $\neg P$ en un littéral positif neg_P. Cette approche n'est malheureusement pas satisfaisante. Ces insuffisances portent sur le fait que les propriétés P et $\neg P$ ne sont pas exclusives et sur l'impossibilité de déduire :

$$\{P \Rightarrow Q, \neg P \Rightarrow Q\} \vdash Q$$

  Pour prendre en compte partiellement la négation, nous introduisons l'atome $non(P)$ d'arité 1, dont la sémantique est la négation de l'atome P. Toutefois, nous restreignons l'emploi de cette négation partielle aux littéraux du langage LDL. Cette introduction a pour conséquence de lever la restriction portant sur les axiomes d'une théorie de la géométrie à être sous la forme de clauses de Horn,

1. En attendant la solution qui est donnée dans la section 4.2.2 page 69, un exercice intéressant pour un lecteur averti consiste à imaginer un moyen de définir géométriquement le niveau de récurrence d'une figure dynamique si possible à l'aide de propriétés mettant en jeu uniquement des équations linéaires en considérant que le modèle computationnel de telles règles est semblable à celui de Prolog.

et nous conduit aussi à utiliser un démonstrateur basé sur les domaines d'incertitude [Ost97a]. Les formulations autorisées dans ce système sont des clauses où la tête est un littéral positif et où le corps est une conjonction de littéraux positifs ou négatifs.

La mise en œuvre de ces extensions sont décrites dans le paragraphe 9.4.

Interprétation des formules ELDL

Notre vision d'une règle est qu'elle représente un moyen de définir un objet complexe à l'aide d'autres objets plus simples. Dans la règle précédente, le symbole $\leftarrow$ est utilisé pour noter l'implication logique.

A titre d'illustration, en reprenant les exemples précédents, la règle :

- *symetrique* se lit :

  Le point A est le symétrique du point B par rapport au point C si C est le milieu des points B et A.

  De la sorte, nous définissons la relation *symétrique* en terme de l'atome de typage LDL *point* et de l'atome de propriété LDL *milieu*.

- *mediatrice* se lit :

  Pour tout A, B et D, D est la médiatrice des points A et B, si A et B sont des points, si D est une droite, et s'il existe un point i milieu de AB tel qu'il existe une droite l passant respectivement par les points A, B et i et perpendiculaire à la droite D.

  Ainsi, nous avons formé la relation *mediatrice* en terme des atomes de typage LDL *point* et *droite* et des atomes de propriété LDL *appDr*, *milieu* et *perp*.

- *triangle* se lit :

  D_{AB}, D_{AC} et D_{BC} forment les côtés d'un triangle de sommet A, B et C, si D_{AB}, D_{AC} et D_{BC} sont des droites passant respectivement par les points A et B, A et C, et B et C.

  De la sorte, nous définissons la relation *triangle* en terme des atomes de typage LDL *point* et *droite* et de l'atome de propriété LDL *appDr*.

– *triangleRectangleEnA* se lit :

> *D_{AB}, D_{AC} et D_{BC} forment les côtés d'un triangle de sommet A, B et C et rectangle en A, si A, B, C, D_{AB}, D_{AC} et D_{BC} forment un triangle, et si D_{AB} est perpendiculaire à D_{AC}.*

Ainsi, nous avons formé la relation *triangleRectangleEnA* en terme de la règle ELDL *triangle* et de l'atome de propriété LDL *perp*.

Nous pouvons remarquer que dans le cadre de la géométrie considérée, cette relation d'implication s'avère souvent être une relation d'équivalence.

Exemples de spécifications ELDL

Nous présentons dans cette partie successivement deux exemples de spécification ELDL.

Exemple 4.8 ANGLE

Construire deux droites sécantes D et D′ en un point A′ donné telles que celles-ci forment un angle α donné par deux droites, et telles que la direction de la droite D soit donnée. La figure 4.8 illustre le résultat attendu.

Dans cette figure, la direction de la droite D est donnée par la droite BM, et l'angle à reproduire est donné par les droites AB et AC.

Cette construction s'appuie sur la notion d'arc capable, puisque rappelons nous que tous les points qui constituent un arc regardent le segment formé par ses extrémités sous le même angle. Pour construire l'arc capable de l'angle $\widehat{BAC}$, nous commençons par déterminer le centre du cercle circonscrit au triangle BAC. Une fois ce cercle déterminé, nous construisons sur ce dernier un point appelé L suivant une direction donnée (ici donné par la droite BM). Enfin, nous rapportons l'angle construit au point désiré.

Pour simplifier le problème, nous considérons que l'intersection de la droite BM et de l'arc BAC n'est pas réduite au seul point B.

Ainsi, la spécification ELDL de cette figure est :

$point(A) \wedge\ point(B) \wedge\ point(C) \wedge$

$droite(D_{AB}) \wedge\ appDr(A, D_{AB}) \wedge\ appDr(B, D_{AB}) \wedge$

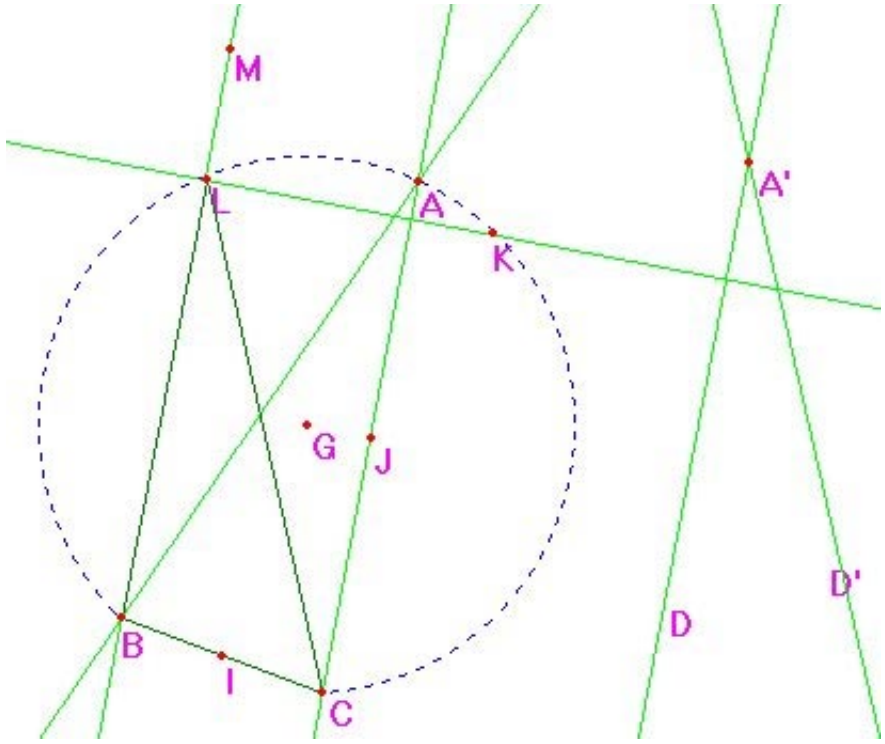

FIG. *4.8* – Droites formant un angle donné.

droite(D_{AC}) ∧ *appDr*(A, D_{AC}) ∧ *appDr*(C, D_{AC}) ∧
point(M) ∧ *point*(A') ∧ *droite*(D) ∧ *droite*(D')
angle(A', D, D', A, B, C, M).

Sachant que :

angle(A', D, D', A, B, C, M) ←
point(A') ∧ *droite*(D) ∧ *droite*(D') ∧ *point*(M) ∧
point(A) ∧ *point*(B) ∧ *point*(C) ∧
point(G) ∧
gravite(A, B, C, G) ∧
point(K) ∧
milieu(G, B, K) ∧
droite(D_{BM}) ∧ *appDr*(B, D_{BM}) ∧ *appDr*(M, D_{BM}) ∧
droite(D_{KL}) ∧ *appDr*(K, D_{KL}) ∧ *perp*(D_{KL}, D_{BM}) ∧
point(L) ∧ *appDr*(L, D_{KL}) ∧ *appDr*(L, D_{BM}) ∧
segment(s_{CL}, C, L, l_{CL}) ∧

$appDr(A', D) \wedge\ par(D, D_{BM}) \wedge$
$appDr(A', D') \wedge\ par(D', I_{CL}).$

$gravite(A, B, C, G) \leftarrow$
$point(A) \wedge\ point(B) \wedge\ point(C) \wedge$
$droite(M_{BC}) \wedge\ droite(M_{AC}) \wedge$
$mediatrice(B, C, M_{BC}) \wedge$
$mediatrice(A, C, M_{AC}) \wedge$
$point(G) \wedge\ appDr(G, M_{BC}) \wedge\ appDr(G, M_{AC}).$

$mediatrice(A, B, M) \leftarrow$
$point(A) \wedge\ point(B) \wedge$
$point(I) \wedge\ milieu(I, A, B) \wedge$
$droite(D_{AB}) \wedge\ appDr(A, D_{AB}) \wedge\ appDr(B, D_{AB}) \wedge$
$droite(M) \wedge\ appDr(I, M) \wedge\ perp(M, D_{AB}).$

Cet exemple illustre comment spécifier en ELDL une contrainte géométrique typique en CAO.

Cette spécification est d'autant plus intéressant qu'elle conduit à une construction mettant en jeu des contraintes exprimables par des équations linéaires.

Exemple 4.9 PUISSANCE

Construire géométriquement la puissance $n^{ième}$ de a. La figure 4.9 illustre le résultat attendu.

Sur cette figure, le point J représente le point de coordonnées $(0, 1)$ et le point A le point de coordonnées $(a, 0)$.

Cette construction s'appuie sur la construction élémentaire d'un produit telle qu'elle est décrite dans l'ouvrage [Car88]. Ainsi, en construisant la droite d_A passant par le point A et perpendiculaire à la droite JA, on détermine un point A' sur l'axe Oj ayant pour ordonnée a^2. L'application successive de cette construction élémentaire permet de construire la puissance $n^{ième}$ de a.

La construction de cette figure nécessite aussi d'être en mesure de définir d'une manière purement géométrique, c'est-à-dire sans introduire la notion d'entier, le niveau de récurrence d'une telle figure dynamique récursive. Pour ce faire, à côté de la figure définissant géométriquement la puissance $n^{ième}$ d'un nombre, nous avons défini une

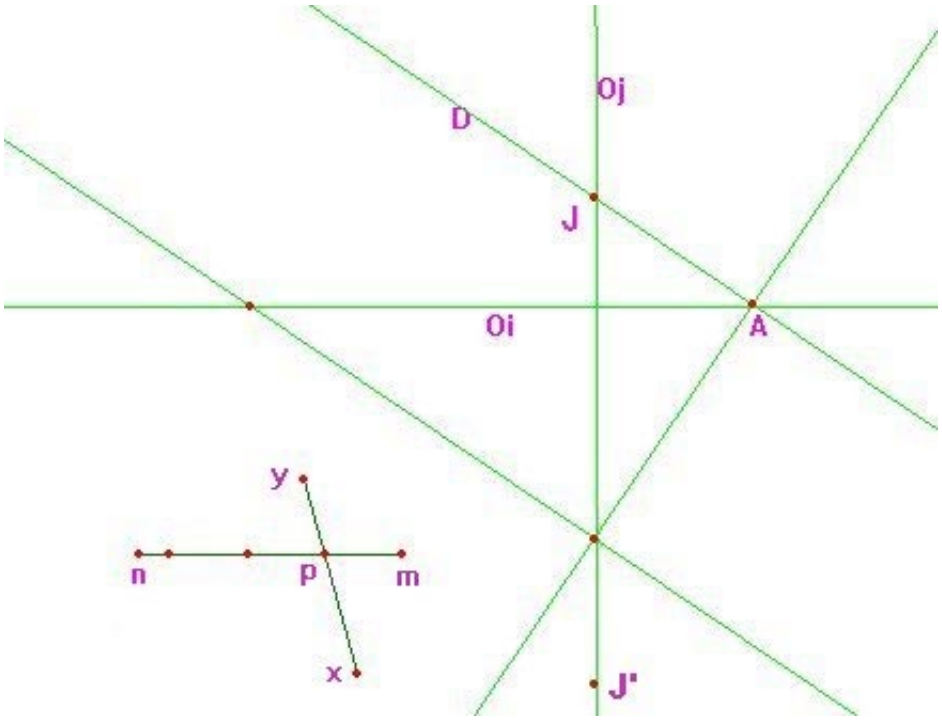

FIG. *4.9* – Puissance n^{ième} de *a*.

sorte de "curseur". Ce dernier est formé d'un segment $[nm]$ *sur lequel nous avons placé un point* p*. Le niveau de récurrence de la figure dynamique récursive correspond au nombre entier de fois qu'il est possible de découper le segment* $[np]$ *en segments de longueur* $[pm]$*.*

Il convient de remarquer que pour rendre de plus cette construction linéaire, c'est-à-dire pour que cette formulation fasse intervenir uniquement des propriétés exprimables par des équations linéaires, nous avons utilisé la propriété de milieu.

La spécification ELDL de cette figure est :

$$
\begin{aligned}
&\mathit{point}(J) \wedge \ \mathit{point}(J') \wedge \ \mathit{droite}(Oi) \wedge \ \mathit{droite}(Oj) \wedge \ \mathit{point}(A) \wedge \\
&\mathit{appDr}(J, Oj) \wedge \ \mathit{appDr}(J', Oj) \wedge \ \mathit{appDr}(A, Oi) \wedge \ \mathit{perp}(Oi, Oj) \wedge \\
&\mathit{droite}(D) \wedge \ \mathit{appDr}(J, D) \wedge \ \mathit{appDr}(A, D) \wedge \\
&\mathit{point}(n) \wedge \ \mathit{point}(m) \wedge \ \mathit{point}(p) \wedge \ \mathit{point}(x) \wedge \ \mathit{point}(y) \wedge \\
&\mathit{segment}(s_{nm}, n, m, l_{nm}) \wedge \ \mathit{segment}(s_{xy}, x, y, l_{xy}) \wedge \\
&\mathit{appSeg}(p, s_{nm}) \wedge \ \mathit{appSeg}(p, s_{xy}) \wedge \\
&\mathit{puissance}(Oi, Oj, D, A, n, m, p).
\end{aligned}
$$

Sachant que :

puissance(Oi, Oj, D, A, n, m, p) $\leftarrow$
droite(Oi) $\wedge$ *droite(Oj)* $\wedge$ *droite(D)* $\wedge$ *point(A)* $\wedge$
point(n) $\wedge$ *point(m)* $\wedge$ *point(p)* $\wedge$ *point(o)* $\wedge$
segment(s, n, m, I) $\wedge$ *milieu(p, m, o)* $\wedge$ *appSeg(o, s)* $\wedge$
droite(D′) $\wedge$ *perp(D, D′)* $\wedge$ *appDr(A, D′)* $\wedge$
point(A′) $\wedge$ *appDr(A′, D′)* $\wedge$ *appDr(A′, Oi)* $\wedge$
non(egalPt(A, A′)) $\wedge$
puissance(Oi, Oj, D′, A′, n, p, mo).

puissance(Oi, Oj, D, A, n, m, p) $\leftarrow$
droite(Oi) $\wedge$ *droite(Oj)* $\wedge$ *droite(D)* $\wedge$ *point(A)* $\wedge$
point(n) $\wedge$ *point(m)* $\wedge$ *point(p)* $\wedge$ *point(o)* $\wedge$
segment(s, n, m, I) $\wedge$ *milieu(p, m, o)* $\wedge$ *appSeg(o, s)* $\wedge$
droite(D′) $\wedge$ *perp(D, D′)* $\wedge$ *appDr(A, D′)* $\wedge$
point(A′) $\wedge$ *appDr(A′, D′)* $\wedge$ *appDr(A′, Oj)* $\wedge$
non(egalPt(A, A′)) $\wedge$
puissance(Oi, Oj, D′, A′, n, p, mo).

puissance(Oi, Oj, D, A, n, m, p) $\leftarrow$
droite(Oi) $\wedge$ *droite(Oj)* $\wedge$ *droite(D)* $\wedge$ *point(A)* $\wedge$
point(n) $\wedge$ *point(m)* $\wedge$ *point(p).*

Cette figure dynamique peut être animée de façons distinctes.

- Un premier type d'animations résulte de la déformation de l'un des objets Oi, Oj, D, J et A. De telles animations ne modifient pas le niveau de récurrence de le figure, et conduisent à la construction géométrique de la même puissance $n^{ième}$ de différents nombres.

- A l'inverse, la déformation des objets n, m, x ou y, c'est-à-dire par conséquent à la déformation de p, définit un autre type d'animations où le niveau de récurrence de la figure est modifié.

Ce dernier type d'animation, rendu possible par la définition de clauses interprétées à la Prolog, est nouveau dans le domaine de la géométrie dynamique.

Chapitre 5

Définition des langages d'interface de GDRev

La première chose qu'un utilisateur perçoit d'un système est son interface. Cette primeur fait d'elle un élément déterminant dans le rejet ou l'utilisation du logiciel par l'usager. Ce caractère déterminant fait par conséquent de sa conception une composante cruciale dans le processus de développement du logiciel. Outre ses qualités esthétiques, ses qualités ergonomiques et didactiques modulent la quantité et surtout la qualité des interactions de l'utilisateur avec le système. Ainsi, des représentations judicieuses, suggestives et bien organisées conduisent à une exploration profonde des fonctionnalités du système. Dans le domaine de l'apprentissage avec ordinateur, cela conduit à de meilleures possibilités d'enseignement.

L'importance des interfaces est d'autant plus grande que l'évolution de la puissance des ordinateurs et la vulgarisation de l'usage de la souris ont considérablement accru leurs possibilités. Des interfaces fondées sur le concept de commandes textuelles interprétées, nous sommes passé aux interfaces fondées sur le concept de la manipulation directe des objets modélisés [Shn83].

Nous décrivons dans ce chapitre les langages d'interface de GDRev fondés sur les principes de la manipulation directe, à savoir le langage d'édition de spécifications et le langage de construction de spécifications ESCL déjà introduit au paragraphe 4.1.3. Plus particulièrement, nous détaillons les aspects géométriques et les aspects d'animation de ces langages. Nous rappelons d'abord dans la section 5.1 les principes fondamentaux des interfaces dites de manipulation directe. Puis, nous détaillons dans la section 5.2 les principales fonctionnalités d'animation. Nous explicitons ensuite dans la section 5.3 les principales fonctionnalités de construction. Enfin, nous présentons

dans la section 5.4 les principales autres fonctionnalités de GDRev dont en particulier la vérification de propriétés.

5.1 Principes des interfaces de manipulation directe

L'histoire des interfaces de manipulation directe [Shn83, Nan90, FvDFH90] commence au début des années 80 avec les projets Alto et Star menés au sein des laboratoires de recherche de Rank Xerox. L'interface de ces machines était fondée sur la métaphore du bureau. Des icons représentaient les différents éléments d'un bureau sur lesquels l'utilisateur pouvait agir à l'aide d'une souris. Cette métaphore fut reprise par la suite par Apple dans ses machines Lisa [Wil83] et Macintosh [Wil84]. Aujourd'hui, les systèmes intégrant les principes de la manipulation directe sont largement répandus dans le monde des PC sous Windows 3.1x/95/98/NT, dans le monde des stations Unix sous X-Windows, et bien sur dans le monde Apple. Dans le domaine des systèmes de géométrie dynamique, Cabri-Géomètre nous semble être le meilleur exemple d'intégration de ces principes.

Nous présentons dans les paragraphes suivant les principes des interfaces de manipulation directe. Au cours de ces présentations, les problèmes qu'ils impliquent dans le cadre de la programmation géométrique déclarative sont soulignés, et les solutions que nous proposons d'y apporter sont déclinées en nous appuyant sur Cabri-Géomètre. Ensuite, nous précisons les différents modes d'interaction possibles avec leurs avantages et leurs inconvénients. Nous reportons les lecteurs désireux d'en connaître davantage sur la question à l'ouvrage de Nanard J [Nan90].

5.1.1 Représentation permanente des objets

Ce principe veut que tout objet défini soit représenté à l'écran à tout moment. Il est établi afin que l'utilisateur puisse faire le lien entre les représentations qu'il perçoit et les objets du système qu'il a définis.

Dans un système de géométrie dynamique déclarative, il n'est pas garanti qu'une figure puisse être construite étant donné ses spécifications géométriques et dynamiques. Il s'ensuit naturellement que les objets non construits n'ont pas lieu d'être représentés dans la fenêtre dessin. Pourtant, de tels objets doivent rester accessibles à l'utilisateur

afin qu'il puisse opérer des actions sur ceux-ci.

Suivant cette dernière remarque, les objets géométriques d'un système déclaratif se doivent d'avoir une double représentation. La première est celle que l'utilisateur perçoit au travers du dessin. La deuxième est celle qui a un caractère permanent. Avec cette double représentation, l'utilisateur peut accéder à tout élément d'une figure quelque soit sa spécification dynamique, et ce sans avoir à se souvenir du nom qu'il lui a attribué, si tel est le cas. Cette deuxième représentation présente aussi l'intérêt de renforcer les retours d'informations visuelles de tels systèmes comme nous le présentons dans le paragraphe 5.1.6.

Dans GDRev, cette deuxième représentation se retrouve dans les onglets *Figure* et *Dynamique* de la fenêtre de spécification.

5.1.2 Action physique

Ce principe est la clé de voûte des interfaces de manipulation directe. Il se traduit autant que faire se peut par l'élimination des boîtes de dialogue, et conduit par conséquent à l'utilisation massive des accessoires de pointage comme la souris ou le crayon électronique. De la sorte, l'utilisateur désigne directement les objets sur lesquels il souhaite agir. Ces désignations directes conduisent à ce que Nanard J [Nan90] appelle *l'impression d'engagement directe.*

> *"L'impression d'engagement direct correspond à ce que l'usager ressent lorsqu'il peut agir directement et librement sur les représentations des objets de son propre monde et percevoir de façon immédiate leurs réactions."*
> [Nan90] (page 42)

Il s'ensuit que les interfaces de systèmes comme GéoSpécif ou GéoPlan ne peuvent revendiquer mettre en œuvre ce principe dans la mesure où ils imposent à l'utilisateur de nommer tous les objets parce que par exemple la construction de l'intersection d'objets passe par la dénomination de ceux-ci.

5.1.3 Rapidité

Un problème posé par les actions physiques menées par l'utilisateur sur les objets du système est la perception immédiate des réactions de ce dernier. La vitesse de réaction de celui-ci est un facteur déterminant puisqu'elle conduit l'utilisateur à penser qu'il

agit effectivement sur les objets. Cette rapidité de réaction passe par le développement de structures de données et d'algorithmes optimisés et aussi par l'utilisation d'un ordinateur doté d'une très grande puissance de calcul.

5.1.4 Continuité

Le principe de continuité est important dans le cadre des interfaces de manipulation directe. Il veut par exemple que le déplacement infinitésimal d'un objet de base ne produise pas une déformation importante de l'allure de la figure. En effet, des changements brutaux peuvent être la source de perturbations et impliquent quoi qu'il en soit un effort cognitif supplémentaire à l'usager.

Dans le cadre d'un système de géométrie déclarative, ce principe est difficile à mettre en œuvre. En effet, la modification de la spécification dynamique des éléments d'une figure peut conduire à la non reconstruction de certains objets de cette dernière. Ces objets non reconstruits ne devant être affichés, il s'ensuit une modification profonde de l'allure de la figure qui peut être la sources d'incompréhensions pour des utilisateurs novices.

Il convient toutefois de signaler que la discontinuité de Cabri-Géomètre, liée à l'existence ou non d'objets, permet de construire des figures spectaculaires, comme les polyèdres convexes, gérant les lignes cachées ; figures ayant de la sorte un caractère discret [Pay93].

5.1.5 Réversibilité

Ce principe fait référence à la capacité du système à revenir dans l'état précédent la dernière action entreprise par l'utilisateur, ou dans les différents états de celui-ci depuis le commencement de la session de travail. Cette fonctionnalité est mise en œuvre par la commande *Annuler* que l'on trouve dans le menu *Edition*. Son importance est grande puisqu'elle permet à l'utilisateur de revenir sur toute action en encourageant par là même la libre exploration des fonctionnalités des logiciels. Ses caractéristiques sont par conséquent qu'elle doit être directement accessible par une commande unique et sans exceptions.

Dans le cadre des systèmes de géométrie dynamique, un aspect de la réversibilité de la commande d'animation est l'animation inverse. Sa réalisation est d'autant plus appréciée qu'elle renforce ainsi la cohérence du système.

5.1.6 Retours d'informations visuelles

Ce principe concerne les effets visuels utilisés en réponse aux actions de l'utilisateur. L'importance de tels effets est considérable pour rendre compte à l'utilisateur de l'opération qu'il est en train ou qu'il cherche à accomplir. Nous avons mentionné dans le paragraphe 1.2.1 que son absence dans le système Geometer's Sketchpad nuit à l'utilisation de ce dernier. Sa manifestation est par exemple le clignotement des objets sélectionnés, le changement de la forme du pointeur de la souris, des messages suggestifs proche de ce dernier, ou enfin ce que Coutaz J explicite dans [Cou91] sous l'appellation de *forme élastique*.

> *"Tant que le bouton reste enfoncé, une expression d'entrée partielle (la localisation actuelle de la souris) est entrelacée avec la production d'une expression de sortie (la forme élastique qui traduit la taille de l'objet si le bouton était relâché à cet instant)."*
> [Cou91] (page 58)

Ce concept de forme élastique est par exemple appliqué dans Cabri-Géomètre dans notamment les outils de création de segment, droite, triangle, polygone et cercle.

Un autre aspect du retour d'informations a été mis en œuvre dans le cadre de GDRev. Il est lié à la double représentation des objets dans l'interface comme cela a été présenté dans le paragraphe 5.1.1. Il consiste, outre les messages suggestifs proche du pointeur de la souris, à mettre en évidence la représentation permanente d'un objet lorsque celui-ci est pointé. Il consiste aussi inversement à mettre en évidence un objet sur un dessin lorsque sa représentation permanente est désignée, et ce quel que soit la primitive de construction utilisée.

5.1.7 Simplicité

L'affirmation selon laquelle l'interface d'un système doit être la plus simple possible est une lapalissade. Pourtant, sa prise en compte n'est pas chose facile. Elle consiste à faciliter la tâche des utilisateurs novices, sans pour autant alourdir celle des utilisateurs experts, ni restreindre les fonctionnalités du système. Elle nécessite la définition du modèle conceptuel de l'utilisateur. Cette modélisation doit permettre de définir les actions par défaut que doit effectuer le système ; actions qui doivent cependant être redéfinissables au travers des préférences du système pour les utilisateurs experts.

Un exemple de simplification est la définition de commandes génériques. Par exemple dans le cadre des systèmes de géométrie dynamique, une telle commande est celle qui fournit un unique outil de construction des points d'intersection de deux objets ; il reste au système la charge d'utiliser la méthode adéquate. En effet, on trouve notamment parmi les fonctionnalités attendues de tels systèmes celles de construction du point d'intersection de deux droites, d'une droite et d'un segment, d'une droite et d'un cercle, ..., et la démarche qui consisterait à associer à chaque fonctionnalité une entrée différente dans l'interface s'incrirait bien sûr à l'encontre du principe de simplicité.

Un exemple de tâche fastidieuse est l'activation d'une commande accessible dans un menu. Une première manière de simplifier cette tâche est d'associer à chacune des principales commandes un raccourci clavier, c'est-à-dire une combinaison unique de touches. Une autre manière consiste à regrouper celles-ci dans un ensemble de barres d'outils personnalisables.

Enfin, un autre moyen de simplifier la tâche des utilisateurs est de lui permettre de définir des procédures comme dans les langages de programmation. En effet, l'utilisation de celles-ci facilite grandement l'exécution de longues tâches répétitives.

5.1.8 Mode d'interaction

Dans leur très grande majorité, les logiciels interagissent avec les utilisateurs suivant un schéma "postfixé" *Objet/Verbe*. C'est-à-dire que l'exécution d'une commande commence par la sélection des objets sur laquelle elle porte, puis l'activation de celle-ci. Par exemple dans les traitements de textes, pour faire ressortir un mot en gras, l'utilisateur commence par sélectionner le mot en question, puis active la commande *gras*. Ce mode d'interaction est le mode suivi dans Geometer's Sketchpad.

L'autre schéma d'interaction est celui du *Verbe/Objet*. C'est-à-dire que l'exécution d'une commande commence par l'activation de celle-ci, puis la sélection des objets sur laquelle elle porte. Ce mode d'interaction est celui qui a été choisi pour Cabri-Géomètre.

Le débat entre les partisans de ces deux approches reste actuel. La principale contrainte du premier schéma d'interaction est qu'il impose la connaissance du type et du nombre des arguments d'une commande. Cette contrainte limite l'exploration des fonctionnalités des systèmes par les usagers novices. Un autre inconvénient de ce schéma est qu'il ne permet pas de retour d'informations sur l'action en cours,

puisqu'elle n'est connue qu'en définitive. A l'inverse, le second schéma d'interaction permet le tels retours d'informations suggestifs, facilitant par là même l'utilisation des systèmes. C'est pourquoi, nous pensons que le schéma *Verbe/Objet* est bien mieux adapté que le schéma *Objet/Verbe* dans de nombreuses situations, et en particulier dans le cadre d'un système de géométrie. Le succès de Cabri-Géomètre sur le marché des logiciels d'enseignement de la géométrie, nous laisse supposer qu'une majorité de personnes pense comme nous.

5.2 Principales fonctionnalités d'animation des figures

Nous venons de voir les principes des interfaces de manipulation directe et l'intérêt de la définition de celles-ci pour les systèmes informatiques dans leur ensemble. Dans le cadre de systèmes dédiés à l'enseignement de la géométrie, cette réalité est d'autant plus importante qu'elle permet une meilleure appropriation des figures construites. De telles interfaces ont déjà été définies avec succès dans le cadre de systèmes de géométrie dynamique impérative. Cabri-Géomètre nous semble en être le meilleur exemple. Nous proposons ici la définition d'un ensemble de fonctionnalités d'animation dans le cadre d'un système de géométrie déclarative. Celles-ci sont décrites dans les paragraphes qui suivent. Nous les avons conçues par extension de celles de Cabri-Géomètre en tenant compte des possibilités nouvelles introduites par l'aspect déclaratif. Nous distingons d'abord les opérations portant sur le degré d'instanciation des objets, pour aborder ensuite celles portant sur l'animation proprement dite.

5.2.1 Manipulation du degré d'instanciation des objets

Les principales fonctionnalités qui distinguent un système de géométrie dynamique impérative d'un système de géométrie dynamique déclarative sont celles qui permettent de modifier à tout moment la spécification dynamique d'une figure, c'est-à-dire les objets à partir desquels le système doit reconstruire la figure. Avant de définir ces fonctionnalités, il faut définir les différents états qui caractérisent un objet d'une figure.

Etats des objets d'une figure

Parmi les différents types d'objets présents dans le système, nous avons considéré que seuls les objets de type point peuvent être des objets de base d'une figure, c'est-à-dire donnés par l'utilisateur. Cette position restrictive n'est pas définitive, et l'animation d'autres objets sera étudiée dans une étape ultérieure. Notons simplement que l'animation de points suffit à obtenir toutes les animations possibles d'une figure à condition d'introduire suffisamment de points.

La figure 5.1 illustre les différents états des objets de type point que nous proposons dans le système GDRev.

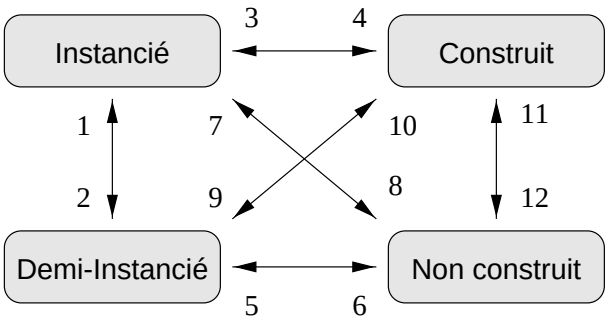

FIG. 5.1 – *Graphe des états d'un objet de type point.*

Un point peut être :

- instancié, c'est-à-dire donné par l'utilisateur. Un tel point correspond à un point de base dans Cabri-Géomètre.

- demi-instancié, c'est-à-dire assujetti par l'utilisateur à appartenir à une droite, une demi-droite, un segment ou un cercle construit. Une telle contrainte réduit de moitié les degrés de liberté du point qui peut néanmoins être source d'animations sur son support. Un tel point correspond à un point sur objet dans Cabri-Géomètre.

- construit, c'est-à-dire déterminé par le système en respectant la spécification géométrique. Un tel point correspond à un point sur deux objets dans Cabri-Géomètre, dans le cas de figures bien contraintes.

- non construit, c'est-à-dire indéterminé par le système. Un tel point n'a pas d'équivalent dans Cabri-Géomètre.

La transition 1 correspond à l'instanciation d'un point demi-instancié. Cette action a pour conséquence de rigidifier la figure, c'est-à-dire limiter, voire interdire, l'animation de cette dernière. A l'inverse, la transition 2 illustre le passage d'un point d'un état instancié à un état demi-instancié. La transition 3 (resp. 7) correspond à l'instanciation d'un point construit (resp. non construit). Les transitions 4 et 8 correspondent à la désinstanciation d'un point donné, suffisamment contraint pour être reconstruit par le système dans le premier cas, et insuffisamment contraint dans le deuxième cas. Un point non construit devient demi-instancié, ce qui correspond à la transition 5, lorsque l'utilisateur place ce dernier sur une droite, une demi-droite, un segment ou un cercle construit. Il redevient non construit, transition 6, par l'opération inverse de désinstanciation. Un point construit peut être demi-instancié sur un de ses supports, transition 9. Cette action a pour conséquence de rigidifier la figure. Un point demi-instancié devient construit, transition 10, après libération de sa contrainte de demi-instanciation.

Toutes ces changements correspondent à des actions de l'utilisateur sur le point proprement dit. A l'opposé, les transitions 11 et 12 correspondent à des modifications d'état de points d'une figure suite à des actions de l'utilisateur sur d'autres points de cette dernière. La transition 11 illustre le passage d'un état non construit à un état construit suite à l'ajout de propriétés, ou l'instanciation d'un point. Le passage inverse, illustré par la transition 12, peut quant à lui faire suite à la désinstanciation d'un point.

Le graphe des états possibles d'un objet de type autre que point correspond au sous graphe de la figure 5.1 constitué des états construit et non construit reliés entre eux par les transitions 11 et 12.

Fonctionnalités de manipulation des spécifications dynamiques

Les deux fonctionnalités décrites ci-dessous permettent à un utilisateur de modifier à tout moment la spécification dynamique d'une figure, c'est-à-dire les objets à partir desquels le système doit reconstruire cette dernière.

L'opération d'*instanciation* permet d'instancier (fixer), mais aussi de demi-instancier (fixer à moitié), un point. D'une manière similaire, l'opération de *désinstanciation* permet de libérer "complètement", mais aussi "partiellement", un point.

- L'opération de désinstanciation. Pour libérer "complètement" un point (transition 4, 6, 8 ou 10), il suffit de le désigner par exemple par un clic du bouton gauche

de la souris.

Une libération "partielle" d'un point (transition 2) doit aussi être obtenue par une simple désignation par exemple à l'aide d'un clic du bouton gauche de la souris en maintenant la touche *Ctrl* enfoncée.

- L'opération d'instanciation. L'instanciation d'un point demi-instancié (transition 1) ou construit (transition 3) se réduit à la désignation de ce dernier sur le dessin par exemple par un clic du bouton gauche de la souris.

  L'instanciation d'un point non construit (transition 7) se révèle plus délicate. Elle nécessite que l'utilisateur sélectionne au préalable la représentation permanente de l'objet qu'il souhaite fixer, puis fixe les coordonnées de ce dernier par exemple par un clic du bouton gauche de la souris dans la fenêtre d'affichage correspondante.

  Pour procéder à la demi-instanciation d'un point non construit (transition 5) (resp. construit (transition 9)), il s'agit de procéder de la même manière en désignant dans le deuxième temps l'objet support. Cette désignation n'étant pas exacte, nous sommes confrontés au problème de déterminer sur l'objet le point visé par l'utilisateur. Une première idée consiste à considérer l'abscisse ou l'ordonnée du point défini par le clic du bouton de la souris, et calculer l'autre coordonnée en conséquence. Cette approche, très facile à mettre en œuvre, n'est malheureusement pas acceptable dans la mesure où les calculs menés peuvent conduire à une inconsistance. Une solution à ce problème est la suivante. La demi-instanciation d'un point sur :

  - une droite, consiste à considérer le projeté orthogonal du clic sur cette dernière.
  - une demi-droite, consiste à considérer le projeté orthogonal du clic sur cette dernière si celui-ci appartient à celle-ci, ou l'extrémité de la demi-droite dans le cas contraire.
  - un segment, consiste à considérer le projeté orthogonal du clic sur le segment s'il appartient à ce dernier, ou l'extrémité du segment la plus proche du clic dans le cas contraire.
  - un cercle, consiste à considérer le point d'intersection de ce dernier et de la demi-droite d'extrémité le centre du cercle et passant par le point défini par le clic.

Cette approche nous semble la plus appropriée et correspondre à l'attente de tout utilisateur. C'est pourquoi nous l'avons adopté dans notre mise en œuvre.

Ces deux fonctionnalités sont regroupées dans GDRev dans le bouton à l'extrême droite de la barre d'outils géométriques sous les appellations *Fixer point* et *Libérer point*.

5.2.2 Déformation de la figure

Nous décrivons dans les paragraphes qui suivent les aspects liés à la déformation de la figure. Nous commençons par la fonctionnalité d'animation, c'est-à-dire la fonctionnalité qui permet de déformer la figure, puis nous examinons le traitement des cas dégénérés, et enfin nous décrivons notre approche pour actualiser les points demi-instanciés.

Fonctionnalité d'animation

Rappelons que la fonctionnalité d'animation, qui permet à tout moment d'animer les figures construites par manipulation directe, est la fonctionnalité la plus importante dans un système de géométrie dynamique. Cette importance est liée au fait qu'elle n'a pas d'équivalent dans l'univers du papier/crayon. Sa mise en œuvre dans les micro-mondes de géométrie a donné à ceux-ci leurs lettres de noblesse en faisant de ces derniers bien plus qu'une simple transposition informatique de l'univers du papier/crayon. Elle donne un accès aux éléments du dessin qui peuvent être saisis et déplacés, tout en conservant les relations géométriques spécifiées. Les objets d'une figure qu'il est possible de déplacer sont les points instanciés et les points demi-instanciés. Les premiers peuvent être déplacés d'une manière quelconque, alors que les derniers sont assujettis à appartenir à un objet support. Cette saisie et ce déplacement se font à l'aide de la souris.

Dans Cabri-Géomètre, le mode opératoire à suivre pour déplacer un objet de base consiste à sélectionner ce dernier par un clic du bouton gauche de la souris et à déplacer la souris tout en maintenant le bouton enfoncé. A chaque nouvelle position, la figure est effacée, puis les paramètres des objets de cette dernière sont recalculés, et enfin la figure est réaffichée en temps réel. En relâchant le bouton, l'utilisateur manifeste son désir de ne plus déplacer l'objet précédemment saisi.

L'approche que nous avons choisie dans GDRev est un peu différente. Cette dif-

férence porte sur le fait qu'à la suite de la sélection d'un point et au déplacement de celui-ci par un mouvement de la souris en maintenant le bouton gauche enfoncé, le relâchement de ce dernier ne signifie pas la fin de l'animation du point. La marque de fin est obtenue par un clic du bouton droit de la souris, qui précise ainsi la dernière position du point sélectionné.

Il est aussi possible dans GDRev de déplacer un point d'une manière discrète, c'est-à-dire de déplacer un point d'une position à une autre sans passer par les positions intermédiaires. Pour procéder à un tel déplacement, il suffit de désigner le point à animer par un clic du bouton gauche de la souris, puis de désigner la nouvelle position du point par un autre clic du bouton gauche de la souris. L'intérêt de tels déplacements est d'éviter d'éventuels déplacements trop lents dans un environnement déclaratif, au détriment, nous l'admettons, du principe de continuité des interfaces de manipulation directe.

Cas de dégénérescence

Au cours de la construction de la figure ou de la déformation de celle-ci, il peut arriver que les spécifications deviennent inconsistantes, c'est-à-dire qu'il devient impossible de déterminer les paramètres réels d'un objet vérifiant les propriétés géométriques imposées par l'utilisateur étant donné les coordonnées des objets de base. Un exemple d'une telle inconsistance est la construction du point d'intersection de deux droites distinctes parallèles.

L'approche suivie dans Cabri-Géomètre est d'autoriser de telles définitions et de telles déformations de la figure. Les objets en cause ne sont alors plus affichés ainsi que leurs descendants, mais ils existent en tant que tels dans la définition de la figure. Si à la suite d'autres déformations ces objets peuvent être déterminés, alors ils retrouvent leur représentation graphique.

Notre approche est différente. Elle s'inspire du caractère logique d'une spécification, et à l'instar de Prolog, conduit à un échec lors d'une inconsistance. Cet échec est signalé à l'utilisateur par un message.

Actualisation des points demi-instanciés

A l'occasion de la déformation de la figure, il se pose aussi le problème de la reconstruction des points demi-instanciés. Le problème est que de telles points ne sont

pas suffisamment contraints pour être reconstruits.

L'approche suivie dans Cabri-Géomètre est de faire dépendre de tels points non seulement de l'objet support, mais aussi de propriétés par défaut résultant de la construction graphique du point sur l'objet. Ces propriétés n'ont pas de caractère purement géométrique. Elles ont été conçues en prenant en compte d'une part le principe de continuité adopté dans la définition du système, et d'autre part le moindre coût des calculs qu'ils impliquent. La figure 5.2 illustre les propriétés utilisées pour un point sur droite, sur demi-droite, sur segment et sur cercle.

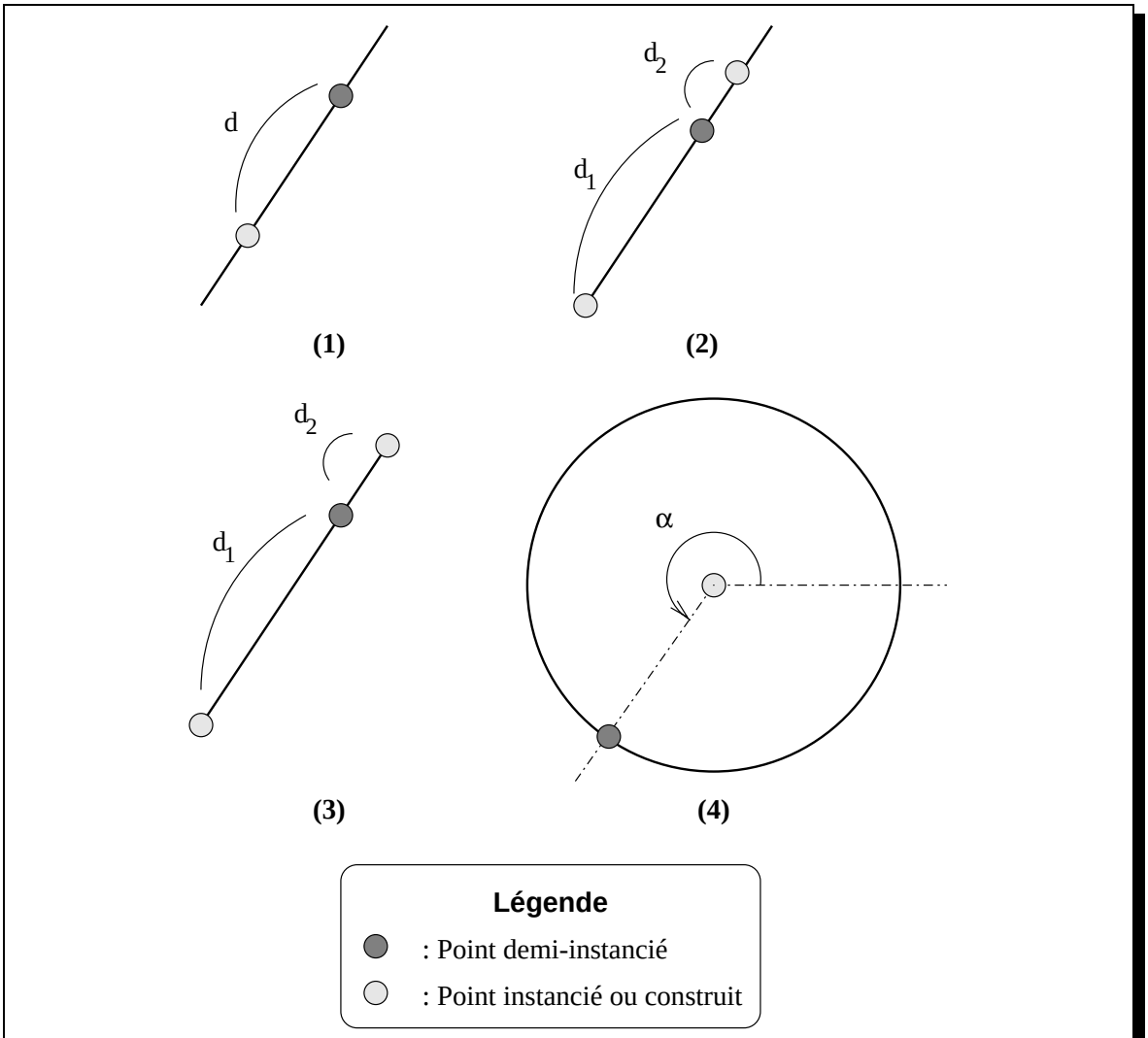

FIG. 5.2 – *Propriétés conservées lors de l'actualisation des points demi-instanciés.*

Dans le cas :

– d'une droite, illustré par la figure 5.2 (1), la distance d est constante.

- d'une demi-droite, illustré par la figure 5.2 (2), le rapport des distances $\frac{d_1}{d_2}$ est constant.
- d'un segment, illustré par la figure 5.2 (3), le rapport des distances $\frac{d_1}{d_2}$ est constant.
- d'un cercle, illustré par la figure 5.2 (4), l'angle α formé entre la demi-droite horizontale d'extrémité le centre du cercle et la demi-droite d'extrémité le centre du cercle et passant par le point demi-instancié est constant.

L'approche que nous avons suivie est celle retenue dans Cabri-Géomètre dans la mesure où elle nous semble naturelle et d'un coût négligeable vis-à-vis des animations qu'elle permet.

5.3 Principales fonctionnalités de construction des figures

Nous présentons successivement les principales fonctionnalités de GDRev relatives à l'édition des spécifications des figures, à la construction des spécifications des figures, et à la manipulation des propriétés des éléments des figures.

5.3.1 Edition des spécifications

Une première manière de donner au système GDRev les spécifications géométriques d'une figure est de l'éditer dans la fenêtre de spécification dans le langage CDL et/ou ECDL, puis d'analyser les spécifications éditées grâce aux fonctions d'analyse mises à disposition. Avec cette approche, l'utilisateur indique uniquement la spécification géométrique de la figure qu'il attend. Il doit ensuite par conséquent utiliser les fonctions d'instanciation détaillées dans le paragraphe 5.2.1 pour aboutir à une figure dynamique. Nous présentons dans ce qui suit cette première approche.

Saisie des spécifications

Pour aider un utilisateur dans une tâche fastidieuse d'édition, une fonctionnalité de coloration syntaxique automatique s'avère particulièrement efficace. Une telle fonctionnalité permet tout d'abord d'éviter bon nombre de fautes de frappe sur les mots

clés des langages. De plus, elle permet de mettre en évidence les différentes catégories de spécification. Enfin, elle conduit à des spécifications présentées de manière uniforme. Son intérêt s'avère plus grand encore si le choix des couleurs utilisées dans cette coloration syntaxique revient à l'utilisateur au travers des préférences du système.

GDRev dispose d'une telle fonctionnalité de coloration syntaxique automatique ce qui lui confère ces nombreux atouts. De plus, afin de clairement dissocier les spécifications géométriques d'une figure des spécifications des clauses, qui peuvent être réutilisées, l'édition de ces dernières se fait dans des fenêtres différentes. Celles-ci sont respectivement l'onglet *Editeur de spécifications* et *Editeur de clauses* de la fenêtre de spécification.

Fonctions d'analyse

Les spécifications éditées nécessitent d'être analysées afin d'une part de vérifier leur syntaxe et leur sémantique, et d'autre part de construire les structures de données correspondantes[1]. Pour ce faire, l'utilisateur dispose d'un ensemble de fonctions d'analyse accessible dans le menu *Figure*, ou dans la barre d'outils d'analyse. Celle-ci est illustrée par la figure 5.3.

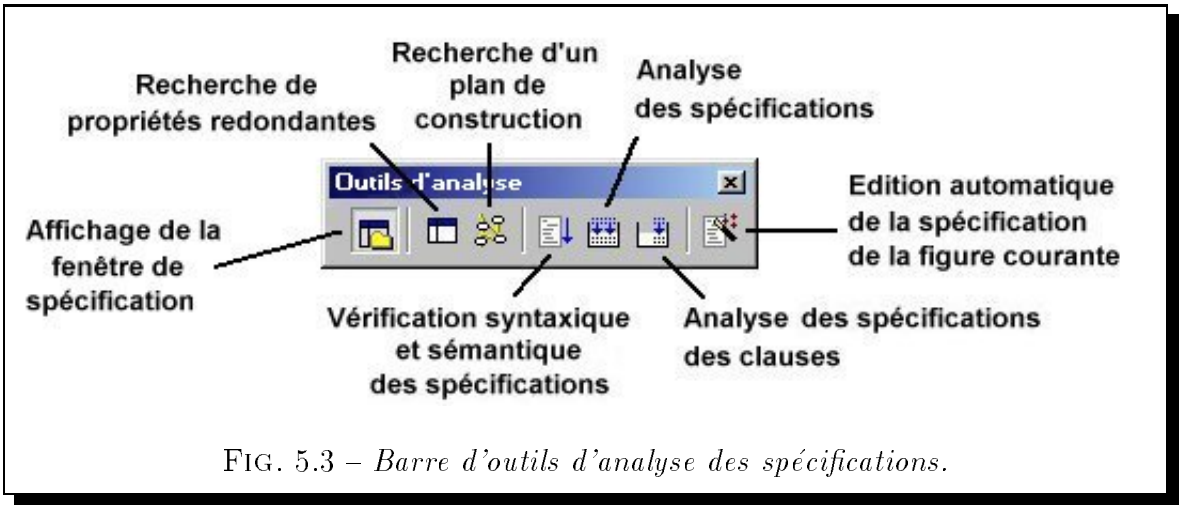

FIG. 5.3 – *Barre d'outils d'analyse des spécifications.*

Ces fonctions concernent :

– L'édition automatique de la spécification de la figure courante. En activant cette fonction, l'utilisateur demande au système d'éditer dans l'onglet correspondant de la fenêtre de spécification, la spécification géométrique de la figure courante.

1. Les structures de données définies sont explicitées dans le chapitre 6 page 103.

- L'analyse des spécifications des clauses. Cette fonction analyse les spécifications des clauses éditées dans l'onglet *Editeur de clauses* de la fenêtre de spécification. Suite à cette analyse, les clauses définies sont mises à disposition dans la barre d'outils géométriques comme tout autre outil.

- L'analyse des spécifications. Cette fonction analyse d'une part des spécifications des clauses éditées dans l'onglet *Editeur de clauses* de la fenêtre de spécifications, et d'autre part des spécifications géométriques de la figure éditées dans l'onglet *Editeur de spécifications* de la fenêtre de spécification. Cette dernière analyse conserve autant que possible les spécifications dynamiques des objets de la figure courante.

- La vérification syntaxique et sémantique des spécifications. Cette fonction vérifie la syntaxe et la sémantique des spécifications des clauses et des spécifications géométriques qui ont été éditées. Cette fonction ne modifie pas la figure courante, mais signale les erreurs commises dans les spécifications éditées.

Les fonctions d'analyse ne prouvent pas l'équivalence des clauses éditées. Cet aspect nous semble être une perspective intéressante.

5.3.2 Construction d'une spécification

Une deuxième manière de fournir la spécification géométrique d'une figure est de la construire en suivant une approche impérative à l'aide d'outils mis à disposition dans une barre d'outils géométriques. GDRev dispose d'un tel langage qui constitue les langages SCL et ESCL introduit dans le chapitre 4. Avec cette approche, l'utilisateur indique non seulement la spécification géométrique de la figure qu'il attend, mais aussi la spécification dynamique des éléments qui la composent. Nous présentons d'abord les primitives de construction d'objets et de propriétés puis celles de construction de clauses.

Primitives de construction

Le langage ESCL de construction de figures géométriques que nous décrivons ici est basé sur celui de Cabri-Géomètre II. Ce choix n'est pas seulement lié à notre participation au projet Cabri-Géomètre. Il découle des indéniables qualitées didactiques de son interface, issue d'une étude pluridisciplinaire réunissant des professeurs de collège et

lycée, des didacticiens et des informaticiens. La très large diffusion de Cabri-Géomètre est aussi un atout incontestable puisqu'elle permet de supposer une prise en main rapide de notre système.

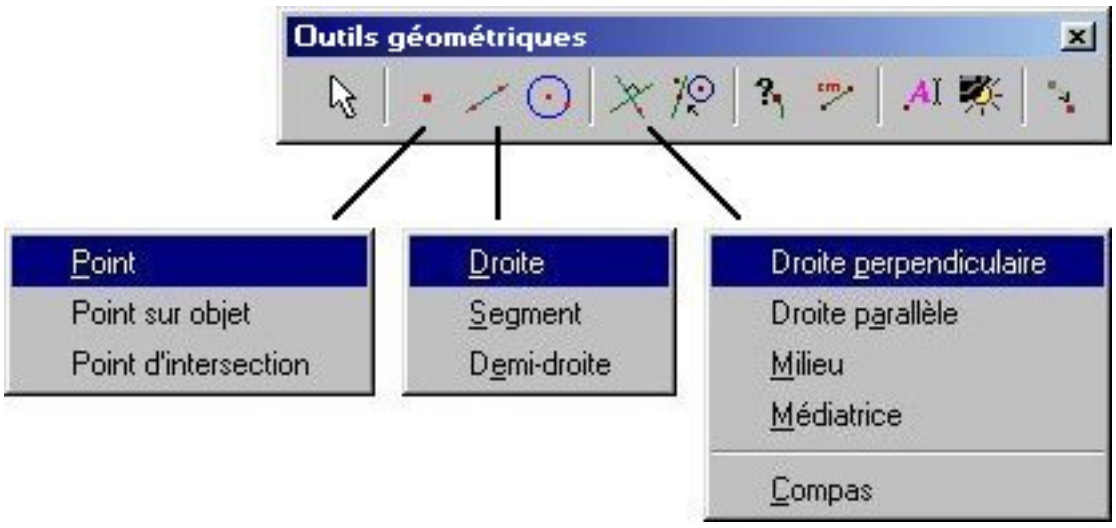

FIG. 5.4 – *Barre d'outils de construction.*

Les primitives de construction ESCL de GDRev sont accessibles au travers de la barre d'outils géométriques illustrée par la figure 5.4. Le deuxième, troisième, quatrième et cinquième bouton de cette barre d'outils regroupent par thème les fonctionnalités de construction de figures. Ces boutons sont respectivement dédiés au point, aux objets rectilignes, au cercle et aux autres constructions.

Rappelons nous que dans GDRev, une construction est autorisée, suivant un schéma d'interaction *Verbe/Objet*, qu'à la condition que celle-ci soit valide numériquement. A l'inverse de Cabri-Géomètre, la sélection des arguments d'un outil de construction suit un ordre préétabli. De plus, les outils de constructions ne permettent actuellement pas la création des points sur objet ou points sur deux objets à la volée. Les primitives *Droite*, *Demi-Droite*, *Segment*, *Cercle*, *Droite Perpendiculaire*, *Droite Parallèle*, *Milieu*, *Médiatrice* et *Compas* supportent néanmoins la création à la volée de point de base, c'est-à-dire de points instanciés.

Les tableaux 5.1 et 5.2 détaillent les formules LDL correspondant aux primitives de construction du langage ESCL. Dans ces tableaux, l'appellation a_i fait référence au $i^{ième}$ argument de la primitive de construction, et les appellations *i, j, ...* font référence aux objets construits. Ceux-ci peuvent être nommés par l'utilisateur, en utilisant tout le jeu de caractères offert par le clavier de l'ordinateur, à l'aide de l'opération *Nommer*,

mais ils sont quoi qu'il en soit identifiés de manière unique par le système.

Primitive de construction	**Arguments**	**Formule LDL correspondante**
OBJETS PONCTUELS		
point	aucun	point(i)
point sur objet	droite	point(i) $\wedge$ appDr(i, a_1)
	demi-droite	point(i) $\wedge$ appDD(i, a_1)
	segment	point(i) $\wedge$ appSeg(i, a_1)
	cercle	point(i) $\wedge$ appCc(i, a_1)
point d'intersection	droite, droite	point(i) $\wedge$ appDr(i, a_1) $\wedge$ appDr(i, a_2)
	droite, demi-droite	point(i) $\wedge$ appDr(i, a_1) $\wedge$ appDD(i, a_2)
	droite, segment	point(i) $\wedge$ appDr(i, a_1) $\wedge$ appSeg(i, a_2)
	droite, cercle	point(i) $\wedge$ appDr(i, a_1) $\wedge$ appCc(i, a_2)
	demi-droite, demi-droite	point(i) $\wedge$ appDD(i, a_1) $\wedge$ appDD(i, a_2)
	demi-droite, segment	point(i) $\wedge$ appDD(i, a_1) $\wedge$ appSeg(i, a_2)
	demi-droite, cercle	point(i) $\wedge$ appDD(i, a_1) $\wedge$ appCc(i, a_2)
	segment, segment	point(i) $\wedge$ appSeg(i, a_1) $\wedge$ appSeg(i, a_2)
	segment, cercle	point(i) $\wedge$ appSeg(i, a_1) $\wedge$ appCc(i, a_2)
	cercle, cercle	point(i) $\wedge$ appCc(i, a_1) $\wedge$ appCc(i, a_2)
OBJETS RECTILIGNES		
droite	point, point	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ appDr(a_2, i)
demi-droite	point, point	demiDroite(i, a_1, l) $\wedge$ appDD(a_2, i)
segment	point, point	segment(i, a_1, a_2, l)
OBJETS NON RECTILIGNES		
cercle	point, point	cercle(i, a_1, r) $\wedge$ appCc(a_2, i) $\wedge$ distPP(r, a_1, a_2)

TAB. 5.1 – *Primitives de construction du langage ESCL*

Les spécifications dynamiques de ces primitives résultent du fait que la primitive

Primitive de construction	Arguments	Formule LDL correspondante
AUTRES		
droite perpendiculaire	point, droite	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ perp(i, a_2)
	point, demi-droite	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ perp(i, a_2)
	point, segment	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ perp(i, a_2)
droite parallèle	point, droite	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ par(i, a_2)
	point, demi-droite	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ par(i, a_2)
	point, segment	droite(i) $\wedge$ appDr(a_1, i) $\wedge$ par(i, a_2)
milieu	point, point	point(i) $\wedge$ milieu(i, a_1, a_2)
	segment	point(i) $\wedge$ segment(a_1, p_1, p_2, l) $\wedge$ milieu(i, p_1, p_2)
médiatrice	point, point	point(i) $\wedge$ milieu(i, a_1, a_2) $\wedge$ droite(j) $\wedge$ appDr(a_1, j) $\wedge$ appDr(a_2, j) $\wedge$ droite(k) $\wedge$ appDr(i, k) $\wedge$ perp(j, k)
	segment	point(i) $\wedge$ segment(a_1, p_1, p_2, l) $\wedge$ milieu(i, p_1, p_2) $\wedge$ droite(k) $\wedge$ appDr(i, k) $\wedge$ perp(a_1, k)
compas	segment, point	segment(a_1, p_1, p_2, l) $\wedge$ distance(d) $\wedge$ distPP(d, p_1, p_2) $\wedge$ cercle(i, a_2, d)

TAB. 5.2 – *Primitives de construction du langage ESCL*

Point crée des points instanciés, la primitive *Point sur objets* crée des points demi-instanciés et toutes les autres primitives conduisent à des objets construits. Toutefois, les points créés à la volée par les primitives *Droite*, *Demi-Droite*, *Segment*, *Cercle*, *Droite Perpendiculaire*, *Droite Parallèle*, *Milieu*, *Médiatrice* et *Compas* sont des points instanciés.

Afin d'illustrer nos propos, considérons l'exemple 5.5 suivant.

Exemple 5.5 CONSTRUCTION D'UNE SPÉCIFICATION

Construire le centre de gravité d'un triangle ABC. Le résultat de cette construction est illustré par la figure 5.5.

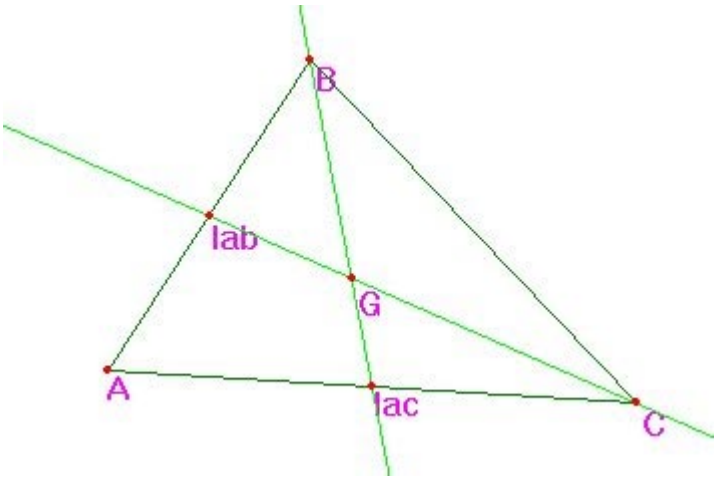

FIG. 5.5 – Construction du centre de gravité d'un triangle.

Une construction peut commencer par :

1. *Créer les points A, B et C à l'aide de la primitive Point.*
2. *Construire à partir de ces points les segments S_{AB}, S_{AC} et S_{BC} à l'aide de la primitive Segment.*
3. *Construire à partir des objets déjà construits ou créés les milieux I_{AB} et I_{AC} des segments S_{AB} et S_{AC} à l'aide de la primitive Milieu.*
4. *Construire les médianes M_B et M_C issues respectivement des points B et C à l'aide de la primitive Droite.*
5. *Et enfin construire le centre de gravité G à l'aide de la primitive Point d'intersection.*

La spécification LDL de cette figure est :

$$
\begin{aligned}
&\mathit{point}(A) \wedge \ \mathit{point}(B) \wedge \ \mathit{point}(C) \wedge \\
&\mathit{segment}(S_{AB}, A, B, I_{AB}) \wedge \ \mathit{segment}(S_{AC}, A, C, I_{AC}) \wedge \\
&\mathit{segment}(S_{BC}, B, C, I_{BC}) \wedge \\
&\mathit{point}(I_{AB}) \wedge \ \mathit{point}(I_{AC}) \wedge \ \mathit{milieu}(I_{AB}, A, B) \wedge \ \mathit{milieu}(I_{AC}, A, C) \wedge \\
&\mathit{droite}(M_B) \wedge \ \mathit{droite}(M_C) \wedge \\
&\mathit{appDr}(B, M_B) \wedge \ \mathit{appDr}(I_{AC}, M_B) \wedge
\end{aligned}
$$

$appDr(C, M_C) \wedge \ appDr(I_{AB}, M_C) \wedge$

$point(G) \wedge \ appDr(G, M_B) \wedge \ appDr(G, M_C).$

La spécification dynamique de cette figure précise de plus que les points A, B et C sont des points instanciés, et que tous les autres objets sont des objets construits.

Primitives de construction de clauses

Nous décrivons maintenant les aspects liés à la notion de clause introduite dans le paragraphe 4.2.2 page 63. Ces aspects concernent la saisie d'une clause et l'exécution de celle-ci.

Saisie d'une clause Nous avons expliqué dans le paragraphe 1.2.1 page 15 ce que Cabri-Géomètre extrait de la spécification d'une figure pour former une macro construction suite à la donnée d'objets initiaux et d'objets finaux.

La définition que nous avons donnée dans le paragraphe 4.2.2 des clauses qui remplissent un rôle analogue à celui des macros constructions mais en géométrie dynamique déclarative, ne permet pas de procéder de la même manière. Le problème difficile auquel nous devons faire face est qu'il n'existe pas de structure de dépendance orientée de laquelle on peut extraire un sous graphe. En géométrie dynamique déclarative, la spécification d'une figure peut toutefois être vue sous la forme d'un graphe, où les nœuds de celui-ci sont les objets de cette dernière, et où deux nœuds sont reliés entre eux par une arête si et seulement si ils sont les arguments d'une même propriété. De plus, les ensembles d'objets initiaux et d'objets finaux ne sont pas disponibles dans une optique déclarative. Une clause comporte des arguments, et à partir d'eux il faut d'extraire de la spécification de la figure les objets intermédiaires que l'utilisateur souhaite voir intervenir dans sa clause. Une fois l'ensemble des objets intermédiaires connus, une idée consiste à considérer parmi les propriétés de la figure exclusivement celles faisant intervenir les objets sélectionnés comme argument de la clause ou les objets intermédiaires.

Il n'y a pas de règle reconnue pour résoudre ce problème, c'est-à-dire pour déterminer l'ensemble de ces objets intermédiaires. Une première idée consiste à considérer comme objets intermédiaires tous les objets de la figure liés directement ou indirectement aux objets sélectionnés comme argument, privés de ces derniers. De la sorte, le graphe extrait est le sous-graphe connexe pour les arguments de la clause du graphe

original. Cette première approche a l'avantage d'être simple à mettre en œuvre et de convenir dans de nombreux cas.

Une deuxième idée consiste à requérir de l'utilisateur la donnée des objets intermédiaires qu'il souhaite voir intervenir dans sa clause. De la sorte, le graphe extrait exprime la fermeture des propriétés sur les objets arguments et intermédiaires. L'intérêt d'une telle approche est qu'elle offre à l'utilisateur une approche prédictible. Son principale inconvénient est qu'elle lui impose une tâche de sélection supplémentaire.

Une troisième idée repose sur une notion de chemin dans un graphe. Muni du graphe d'une figure, nous pouvons définir la notion d'ensemble d'objets intervenant dans un chemin de longueur i comme étant l'ensemble des objets intervenant dans un chemin de longueur au plus i reliant deux arguments distincts de la clause. La saisie d'une clause peut se résumer alors à déterminer la plus petite longueur de chemin telle que la clause alors formée soit valide.

La question se pose maintenant de déterminer le critère de validité d'une clause ayant n arguments. Nous proposons que celui-ci soit qu'une clause est valide si tous les objets de cette dernière peuvent être construits à partir de la donnée de $n-1$ de ces arguments.

Pour illustrer nos propos, considérons l'exemple 5.6 suivant.

Exemple 5.6 SAISIE D'UNE CLAUSE

Considérons que la figure construite par l'utilisateur soit celle de l'orthocentre d'un triangle. Le problème consiste à extraire une clause reliant les points A, B, C et H. Notre vision sous forme de graphe de cette figure est illustrée par la figure 5.6.

Dans cet exemple, la plus petite longueur de chemin reliant deux arguments de la clause est 2. Elle conduit à considérer comme objets intermédiaires les droites D_{AB}, D_{AC}, D_{BC}, D_{AH}, D_{BH} et D_{CH}. La clause formée est la suivante :

$$
\begin{aligned}
&\mathit{orthocentre}(A, B, C, H) \leftarrow \\
&\quad \mathit{point}(A) \wedge \ \mathit{point}(B) \wedge \ \mathit{point}(C) \wedge \ \mathit{point}(H) \wedge \\
&\quad \mathit{droite}(D_{AB}) \wedge \ \mathit{droite}(D_{AC}) \wedge \ \mathit{droite}(D_{BC}) \wedge \\
&\quad \mathit{appDr}(A, D_{AB}) \wedge \ \mathit{appDr}(A, D_{AC}) \wedge \\
&\quad \mathit{appDr}(B, D_{AB}) \wedge \ \mathit{appDr}(B, D_{BC}) \wedge \\
&\quad \mathit{appDr}(C, D_{AC}) \wedge \ \mathit{appDr}(C, D_{BC}) \wedge \\
&\quad \mathit{droite}(D_{AH}) \wedge \ \mathit{droite}(D_{BH}) \wedge \ \mathit{droite}(D_{CH}) \wedge \\
&\quad \mathit{appDr}(A, D_{AH}) \wedge \ \mathit{appDr}(H, D_{AH}) \wedge
\end{aligned}
$$

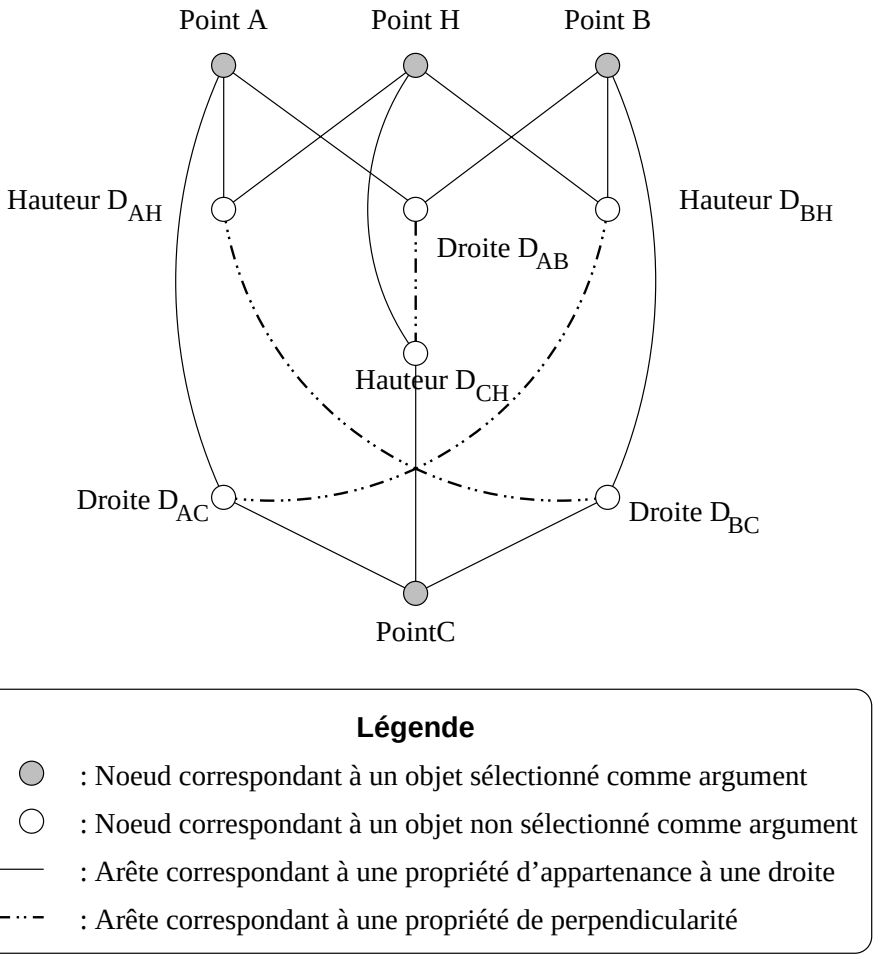

FIG. *5.6* – Vision sous forme de graphe de la figure.

$$appDr(B, D_{BH}) \wedge\ appDr(H, D_{BH}) \wedge$$
$$appDr(C, D_{CH}) \wedge\ appDr(H, D_{CH}) \wedge$$
$$perp(D_{AB}, D_{CH}) \wedge\ perp(D_{AC}, D_{BH}) \wedge\ perp(D_{BC}, D_{AH}).$$

Elle est valide puisque tous les objets de cette dernière peuvent être construits soit à partir des points A, B et C, ou des points A, B et H, ou des points A, C et H, ou enfin des points B, C et H.

Exécution d'une clause Les clauses saisies sont mises à disposition de l'utilisateur dans la barre d'outils géométriques, comme tout autre outil. Elles suivent le même mode d'interaction que les autres primitives, à savoir *Verbe/Objet*. De même que toute primitive de construction, la sélection des arguments d'une clause suit l'ordre de celle-

ci. Enfin, l'utilisateur spécifie qu'il ne souhaite pas sélectionner un argument par un double clic sur le bouton droit de la souris.

5.3.3 Manipulation des propriétés

Un aspect important dans le cadre d'un environnement déclaratif pour la géométrie est la faculté du système à supporter la manipulation des propriétés géométriques des éléments d'une figure. Ces manipulations consistent en l'ajout ou la suppression de propriétés.

Ajout de propriétés

Par l'ajout de propriétés liant des éléments d'une figure, un utilisateur peut par exemple manifester son intention d'explorer une nouvelle situation géométrique à partir de la situation courante. Par exemple, en partant de la spécification d'un quadrilatère, l'ajout de propriétés de parallélisme conduit à la spécification d'un parallélogramme. Il peut aussi par exemple chercher à aider les mécanismes de résolution (cf paragraphe 7.5). Enfin, cela permet clairement une approche "compositionelle" pour construire une figure. A partir de deux figures indépendantes, il devient possible d'en composer une troisième en indiquant par exemple qu'un point de l'une appartient à une droite de l'autre.

Comme pour la construction d'une spécification, l'ajout de propriétés dans GDRev peut être réalisé suivant une approche :

- textuelle, qui consiste dans un premier temps à éditer dans l'onglet adéquat de la fenêtre de spécification les propriétés désirées dans le langage CDL et/ou ECDL puis dans un deuxième temps à analyser la nouvelle spécification.

- graphique, qui consiste à utiliser l'opération *Ajouter propriété* placée dans le bouton à l'extrême droite de la barre d'outils géométriques. Pour ce faire, nous proposons que l'utilisateur désigne par un clic du bouton gauche de la souris un des arguments de la propriété, puis choisisse dans le menu déroulant alors affiché au niveau du pointeur de la souris[2] la propriété désirée, et enfin achève de désigner les arguments de cette dernière.

2. De tels menus sont aussi appelés des *popup menus*.

Nous pensons qu'avant de chercher à ajouter une quelconque propriété, le système se doit de vérifier approximativement si les objets sélectionnés vérifient ou non la propriété considérée. Cette vérification approximative a pour but de limiter les changements brutaux de l'allure de la figure, qui peuvent être perturbateurs pour l'utilisateur. Dans l'affirmative, le système a à charge de procéder à la reconstruction de la figure ayant sa spécification augmentée de la propriété additionnelle. Pour ce faire, plusieurs approches peuvent être considérées. Nous les décrivons sur l'exemple 5.7 suivant.

Exemple 5.7 AJOUT DE PROPRIÉTÉ

On considère un point A, et un cercle C centré en un point O donné et passant par un point B donné. Il s'agit d'ajouter la propriété d'appartenance du point A au cercle C.

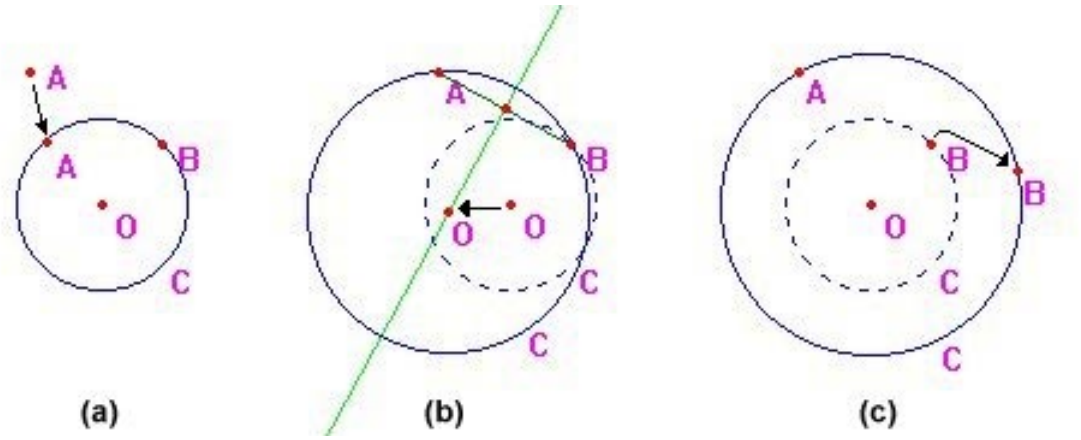

FIG. *5.7* – Différents scénarios d'ajout de propriétés.

Une séquence opératoire pour ajouter cette propriété est :

1. *Sélectionner le point A.*
2. *Choisir l'item d'appartenance à un objet dans le popup menu affiché.*
3. *Sélectionner le cercle C.*

Sachant que les objets donnés sont les points A, O et B, et les objets construits sont le cercle C et son rayon R, nous pouvons imaginer trois scénarios aboutissant à l'ajout de cette propriété. Ces scénarios sont illustrés par la figure 5.7.

Le scénario (a) consiste à :

1. *Désinstancier le point A*

2. *Ajouter la propriété :* appCc(A, C)
3. *Demi-instancier le point A*
4. *Instancier le point A*

Le scénario (b) consiste à :

1. *Désinstancier le point O*
2. *Ajouter la propriété :* appCc(A, C)
3. *Demi-instancier le point O*
4. *Instancier le point O*

Le scénario (c) consiste à :

1. *Désinstancier le point B*
2. *Ajouter la propriété :* appCc(A, C)
3. *Demi-instancier le point B*
4. *Instancier le point B*

La question se pose de décider du scénario à employer dans le cas général en maintenant comme invariante la spécification dynamique des éléments de la figure. La réponse que nous apportons à ce problème trouve son origine dans la séquence opératoire exécutée par l'utilisateur. En effet, il est possible de dissocier les scénarios possibles en deux catégories suivant qu'ils conduisent à la modification du premier argument sélectionné, ou qu'ils conduisent à la modification des arguments sélectionnés ensuite. Sur notre exemple, cela revient à dissocier les scénarios conduisant à la modification du point A, c'est-à-dire le scénario *(a)*, des scénarios conduisant à la modification du cercle C, c'est-à-dire les scénarios *(b)* et *(c)*. Nous appelons respectivement ces catégories :

- Top-Down, qui consiste à ajuster préférentiellement le premier objet sélectionné lors de la phase de sélection des arguments de la propriété.
- Bottom-Up, qui consiste à ajuster préférentiellement les derniers objets sélectionnés lors de la phase de sélection des arguments de la propriété.

Notre préférence penche en faveur de l'approche top-down. En effet, nous pensons que bien qu'une propriété soit une relation est non une fonction, en désignant un premier objet, l'utilisateur désigne l'objet qu'il souhaite voir ajuster. Conscient du caractère discutable de ce choix, ce dernier est laissé en définitive à l'utilisateur au travers des préférences du système.

En nous plaçant dans l'une ou l'autre de ces catégories, nous pouvons remarquer que deux cas de figure peuvent se présenter :

- l'objet sélectionné est un point instancié ou demi-instancié, et alors le nombre des scénarios est réduit à 1.
- l'objet sélectionné est soit un point construit, soit un objet de type droite, demi-droite, segment ou cercle, et alors le nombre des scénarios est strictement supérieur à 1.

Dans ce deuxième cas de figure, reste encore à décider du scénario à employer. Pour répondre à cette question, rappelons nous qu'une figure peut être vue sous la forme d'un graphe, où les nœuds de celui-ci sont les objets de cette dernière, et où deux nœuds sont reliés entre eux par une arête si et seulement si ils sont les arguments d'une même propriété. Notre problème est de choisir le point instancié à déplacer pour ajuster l'objet sélectionné. Cette recherche peut être menée sur cette structure graphique suivant une approche en profondeur d'abord, ou une approche en largeur d'abord. Ce choix est aussi laissé à l'utilisateur au travers des préférences du système.

De toute façon, dans la mesure où il est exigé que la propriété ajoutée soit approximativement respectée sur le dessin, l'utilisateur devrait dans la majorité des cas ne rien ressentir qui soit perturbateur. L'algorithme que nous utilisons est celui que nous avons proposé dans le chapitre 11 dans le cadre de la construction d'un "préceptoriel".

Suppression de propriétés

Par la suppression de propriétés, un utilisateur peut manifester son intention d'explorer une nouvelle situation géométrique à partir de la situation courante, ou chercher par exemple à déterminer la spécification minimale requise par le système pour construire une figure dynamique désirée.

Comme pour l'ajout, la suppression de propriétés peut être opérée suivant les deux

modes suivants :

- textuel, qui consiste dans un premier temps à effacer dans l'onglet adéquat de la fenêtre de spécification les propriétés non désirées éditées dans le langage CDL et/ou ECDL puis dans un deuxième temps à analyser la nouvelle spécification.

- graphique, qui consiste à utiliser l'opération *Supprimer propriété* placée dans le bouton à l'extrême droite de la barre d'outils géométriques. Pour ce faire, nous proposons que l'utilisateur sélectionne un des objets intervenant dans la propriété non désirée par un clic du bouton gauche de la souris, puis choisisse dans le popup menu affiché la propriété non désirée.

  Cette définition de la fonction de suppression graphique de propriété découle du caractère relationnel, et non fonctionnel, d'une propriété géométrique.

5.4 Autres fonctionnalités

Enfin, nous présentons ici les principales autres fonctionnalités de GDRev relatives d'une part à la trace d'un objet et d'autre part à la vérification de propriétés.

5.4.1 Trace d'un objet

Dans le paragraphe 1.1 page 12, nous avons présenté la notion de lieu et nous avons indiqué l'importance de celle-ci. Pour déterminer des lieux, Cabri-Géomètre offre deux fonctionnalités :

- La trace. Par cette fonctionnalité, l'utilisateur spécifie au système de ne pas effacer les objets dont il souhaite la trace au cours des déformations de la figure. Cet affichage est maintenu tant que l'utilisateur n'invoque pas la commande *Tout redessiner* ou qu'un objet de la figure est sélectionné à l'aide de cette fonctionnalité.

- Le lieu. Par cette fonctionnalité, l'utilisateur spécifie au système de construire et d'afficher le lieu d'un objet, ou l'enveloppe de celui-ci dans le cas d'objets non ponctuels, lorsqu'un point sur objet parcours son support. Les lieux construits sont des objets sur lesquels l'utilisateur peut s'appuyer pour poursuivre sa construction.

Nous pensons que la complémentarité des deux fonctionnalités présentées fait que leur mise en œuvre dans un système de géométrie dynamique est un atout fort appréciable. GDRev dispose actuellement de la fonction de trace. Celui-ci est disponible dans la barre d'outils géométriques suivant les même modalités que Cabri-Géomètre. Nous espérons la compléter dès que possible avec une fonctionnalité de lieu.

5.4.2 Vérification de propriétés

La validation de propriétés géométriques a été, et reste, un domaine de recherches particulièrement actif. Les principaux prototypes mis en œuvre dans cette optique dans le domaine des systèmes de géométrie dynamique sont les systèmes TALC [Des94], Cabri-Euclide [Lue97], Mentoniezh [Nic89, Tri96b, Py96] et Chypre [Ber94].

Cabri-Géomètre II dispose d'outils de vérification de propriété. Les propriétés qu'il est possible de vérifier sont l'alignement, le parallélisme, la perpendicularité, l'équidistance et enfin l'appartenance. La véracité de ces propriétés est donnée suivant une approche numérique. Cette approche est basée sur le principe qu'une propriété est vérifiée si elle l'est lors de tout déplacement de tout objet de base de la figure. Pratiquement, cette vérification est faite sur un ensemble fini de dessins obtenus par déplacements aléatoires d'objets de base. Cette approche, bien qu'incomplète, fournit d'excellents résultats.

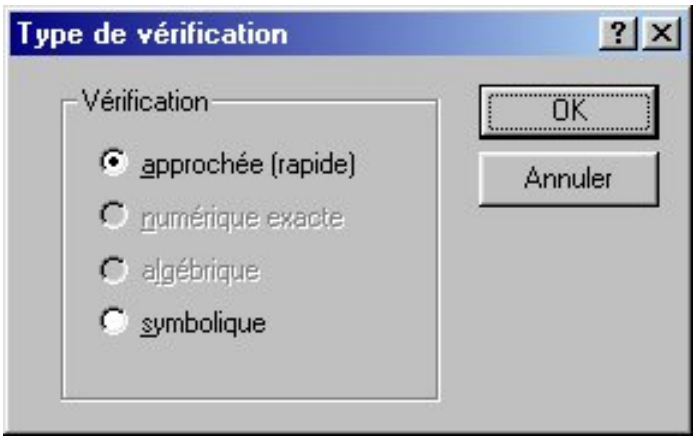

FIG. 5.8 – *Choix du type de la vérification d'une propriété.*

Notre approche se distingue de celle de Cabri-Géomètre sur :

– les propriétés qu'il est possible de vérifier. Celles-ci sont les propriétés d'appar-

tenance, de parallélisme, de perpendicularité, de milieu et enfin d'égalité. Le choix de ces propriétés est issu du langage logique sur lequel est bâti GDRev, c'est-à-dire le langage LDL.

– l'approche de la validation de propriété. Nous proposons dans GDRev un ensemble de méthodes de validation de propriété. Celles-ci comprennent une méthode numérique approchée, qui a la qualité d'être rapide mais incomplète, et une méthode symbolique, qui à la qualité d'être exacte mais possiblement très lente et incomplète pratiquement [Ost97b].

  Le choix de la méthode employée pour valider une propriété se fait au travers de la boîte de dialogue illustrée par la figure 5.8.

  Nous envisageons de compléter ces méthodes par une méthode numérique exacte, qui à la qualité d'être exacte sur un dessin.

Si la propriété n'est pas vérifiée, il semble naturel que le système fournisse un contre-exemple mettant clairement en évidence le caractère faux de cette dernière dans le cas général. Cette fonctionnalité n'est pas encore disposition dans GDRev. Toutefois, un prototype, dénommé SPhinx pour Système Préceptoriel, a fait l'objet d'une étude dans ce sens. Il est présenté plus en détails dans le chapitre 11 page 205.

Chapitre 6

Mise en oeuvre des langages d'interface de GDRev

La mise en œuvre des langages d'interface de GDRev constitue une part très importante quantitativement du temps consacré à notre travail et dont le succès constitue au final la preuve de la faisabilité d'une interface de manipulation directe pour un système de géométrie dynamique déclarative. Cette réalisation a conduit à se poser le problème de la définition de structures de données adéquates, de leur l'organisation, et de l'écriture des algorithmes les utilisant.

Nous présentons dans ce chapitre les structures de données et les algorithmes utilisés par GDRev. Nous introduisons dans la section 6.1 le modèle d'architecture adopté dans la mise en œuvre de ce système interactif. Nous explicitons ensuite dans la section 6.2 les contraintes logicielles et matériels auxquelles nous avons dues nous plier. Nous détaillons dans la section 6.3 les structures de données géométriques. Nous exposons dans la section 6.4 les aspects liés à la déformation de la figure. Plus particulièrement, nous abordons le cas où un plan de construction a pu être établi. Enfin, nous évaluons dans la section 6.5 cette mise en œuvre à l'aide de données chiffrées.

6.1 Modèle d'architecture

Plusieurs modèles d'architecture ont été définis pour les interfaces homme-machine. Les premiers modèles proposent une approche séquentielle qui restreint les interactions entre l'utilisateur et le système. Plus récemment, les modèles multi-agents, comme le modèle PAC *(Présentation Abstraction Contrôle)* [Cou90], ont une approche événe-

mentielle qui permet de mieux prendre en compte les interactions de l'utilisateur avec le système. Ils se caractérisent par une organisation structurée et modulaire facilitant la mise en œuvre et la maintenance des systèmes, une communication par évènements et une indépendance du modèle vis-à-vis de l'environnement de programmation choisi[1].

Nous décrivons successivement dans les paragraphes suivants le modèle PAC, qui est une référence dans le domaine des architectures d'interface. Puis, nous détaillons le modèle que nous avons suivi pour GDRev qui est basé sur le modèle PAC.

6.1.1 Modèle PAC

Le modèle PAC [Cou90] est destiné à structurer un système interactif en une hiérarchie d'agents réactifs. Un agent réactif est un système de traitement d'informations de type stimuli/récepteur. Ainsi, les agents du modèle PAC interagissent au travers d'envois de messages. Ils sont capables d'émettre des messages et de réagir à des messages extérieurs, pouvant provenir de l'utilisateur ou d'autres agents réactifs.

Un agent du modèle PAC est constitué :

- d'une présentation, qui définit à tout instant l'image de l'objet.
- d'une abstraction, qui définit la composante fonctionnelle de l'objet ainsi que ses attributs.
- d'un contrôle, qui d'une part maintient la cohérence entre la partie abstraction et la partie présentation, et d'autre part gère les communications de cet agent avec les autres agents.

Un agent réactif peut être élémentaire, ou composé d'agents réactifs. De tels agents composés réagissent de manière similaire aux agents élémentaires. Toutefois, le comportement d'un agent composé dépend du contrôle de l'agent de plus haut niveau, mais aussi des agents le composant, et sa présentation hérite de la présentation des agents le composant.

1. Cette dernière remarque n'est pas toujours valide. Elle l'est cependant dans le cas du modèle PAC.

6.1.2 Modèle de GDRev

Le modèle de GDRev reprend le modèle d'architecture PAC pour ses qualités de modularité, de structuration et d'indépendance vis-à-vis du langage de programmation choisi.

Dans cette approche, un objet géométrique est vu comme un agent réactif tel que sa composante :

- présentation soit la représentation de l'objet,
- abstraction soit les paramètres géométriques, d'affichage et d'animation de l'objet,
- contrôle définisse le comportement de l'objet suite à des actions de l'utilisateur.

Nous avons apporté deux adaptations à ce modèle. Elles concernent :

- La gestion de la communication avec l'utilisateur. En effet, la communication entre le système et l'utilisateur est parfois ambiguë. De telles ambiguïtés se présentent par exemple lorsque ce dernier approche le pointeur de la souris d'un objet et qu'un ou que plusieurs objets de même type sont à proximité. Afin de pouvoir déterminer précisément l'objet pointé par l'utilisateur, nous introduisons un module de gestion des ambiguïtés. Celui-ci a pour charge d'inviter l'utilisateur à préciser son choix si plusieurs agents ont réagi.

  Si le type de l'objet attendu est multiple et comprend le type point, alors si un seul objet de type point se trouve à proximité du pointeur de la souris, l'ambiguïté est levée en considérant le point comme prioritaire. Ce choix se justifie par la taille d'un objet de type point relativement à celle d'un objet d'un autre type.

- La présentation des agents. Cette adaptation est liée à la double représentation des objets dans le cadre d'un système de géométrie dynamique déclarative. Cela nous a conduit naturellement à associer à chaque agent PAC deux composants de représentation, chacun étant lié à une représentation particulière de l'objet.

6.2 Contraintes de la mise en oeuvre

Pour le développement des langages d'interface de GDRev, que nous avons détaillés dans le chapitre 5, nous avons dû nous plier à des contraintes d'ordre :

- Matériel. Cette contrainte est liée au choix du langage de programmation utilisé pour mettre en œuvre les méthodes de résolution des contraintes géométriques que nous avons définies et que nous présentons dans la troisième partie de ce manuscrit. Ce langage est le langage de programmation avec contraintes Prolog IV qui fonctionne sur PC sous Windows 32 bits, c'est-à-dire sous Windows 95/98/NT, et sur station Unix sous SunOs ou Solaris. Ceci exclus par conséquent toute mise en œuvre sur les machines Apple.

- Logiciel. Cette contrainte est aussi liée au langage de programmation choisi pour mettre en œuvre les méthodes de résolution des contraintes géométriques. En effet, les moyens mis à disposition des programmeurs pour faire interagir des programmes Prolog IV avec des programmes écrit dans d'autres langages de programmation concernent ceux écrit dans les langages Tcl/Tk, C/C++ ou plus récemment Java.

- Génie logiciel. Cette contrainte implique que notre réalisation ait la propriété d'être structurée et modulaire, afin de faciliter l'ajout d'objets ou de fonctionnalités au système ainsi que la maintenance de ce dernier.

Notre choix s'est porté en faveur des machines PC sous Windows. Ce choix a été dicté par la plus large expérimentation et diffusion qui peut en résulter.

Le choix du langage de programmation pour l'interface fait suite à une première ébauche des langages d'interface de GDRev réalisée en Tcl/Tk. Celle-ci nous a permis de nous convaincre de la justesse de nos définitions des langages. Cependant, les contraintes d'efficacité posées par les interfaces de manipulation directe nous ont conduit à abandonner ce langage de programmation interprété au profit du langage de programmation compilé C++ (Visual). Ce choix comporte d'importants avantages. Tout d'abord, son caractère orienté objet[2] permet d'une part de satisfaire aux contraintes de génie logiciel, et d'autre part de supporter la notion d'agent réactif du

2. Nous n'abordons pas ici les aspects liés au modèle objet. Nous supposons que le lecteur est au fait de ceux-ci. Dans le cas contraire, nous reportons ce dernier à la littérature abondante qui traite du sujet.

modèle PAC. Ensuite, il nous offre la possibilité d'user de la librairie MFC (Microsoft Foundation Classes) comportant de nombreuses classes prédéfinies. La contrepartie à l'utilisation de cette librairie se traduit par un affichage parfois insatisfaisant. Par exemple, la définition d'un segment sur une droite n'aboutit pas à une parfaite superposition de celui-ci sur celle-ci à l'écran. Nous estimons néanmoins ce choix pratique le meilleur dans les circonstances présentes.

6.3 Hiérarchie des classes géométriques

Dans Cabri-Géomètre [Bau90, Lab95], une distinction est faite entre le type d'un objet, c'est-à-dire point, droite, ..., et la classe d'un objet, c'est-à-dire droite passant par deux points, droite perpendiculaire à une ligne passant par un point, ... Cette distinction se retrouve jusque dans les structures de données définies, ce qui conduit à introduire dans la définition d'un objet ses relations géométriques. Cette distinction trouve sa justification dans le fait qu'un objet a des attributs et des méthodes particuliers suivant les propriétés qui les lient aux autres objets de la figure. Ainsi, le type d'un objet définit par exemple sa représentation et la fonction de calcul de la distance séparant l'objet du pointeur de la souris, et la classe d'un objet définit par exemple son type, le nombre de ses constituants et la méthode de (re)calcul de ses paramètres en fonction des paramètres de ses constituants.

Dans GDRev, une telle approche n'a pas de raison d'être. En effet, dans une approche déclarative, un objet n'a pas d'attributs ou de méthodes particuliers suivant les propriétés qui le lient aux autres objets de la figure. De plus, il n'est pas rare en géométrie déclarative d'être en présence d'une spécification surcontrainte. Le problème est que de telles spécifications sont mal prises en compte dans un telle approche. Enfin, cette approche implique un surcoût si la spécification géométrique de la figure est modifiée. Or, notre approche déclarative favorise la modification des spécifications géométriques des figures afin d'en explorer tous les aspects pour une meilleure appropriation. C'est pourquoi nous proposons de dissocier les concepts d'objets et de propriétés dans les structures de données d'un système de géométrie dynamique déclarative. Et, nous définissons par conséquent d'un côté un ensemble de classes associé aux objets géométriques, et d'un autre côté une classe associée aux propriétés géométriques. Toutefois, cette dissociation ne nous interdit pas d'organiser les données suivant une hiérarchie basée sur les attributs et les méthodes communs aux différents objets.

Nous présentons dans le paragraphe suivant cette hiérarchie des classes des objets géométriques. Puis, nous décrivons la classe que nous avons définie pour la notion de propriété géométrique.

6.3.1 Hiérarchie des classes des objets

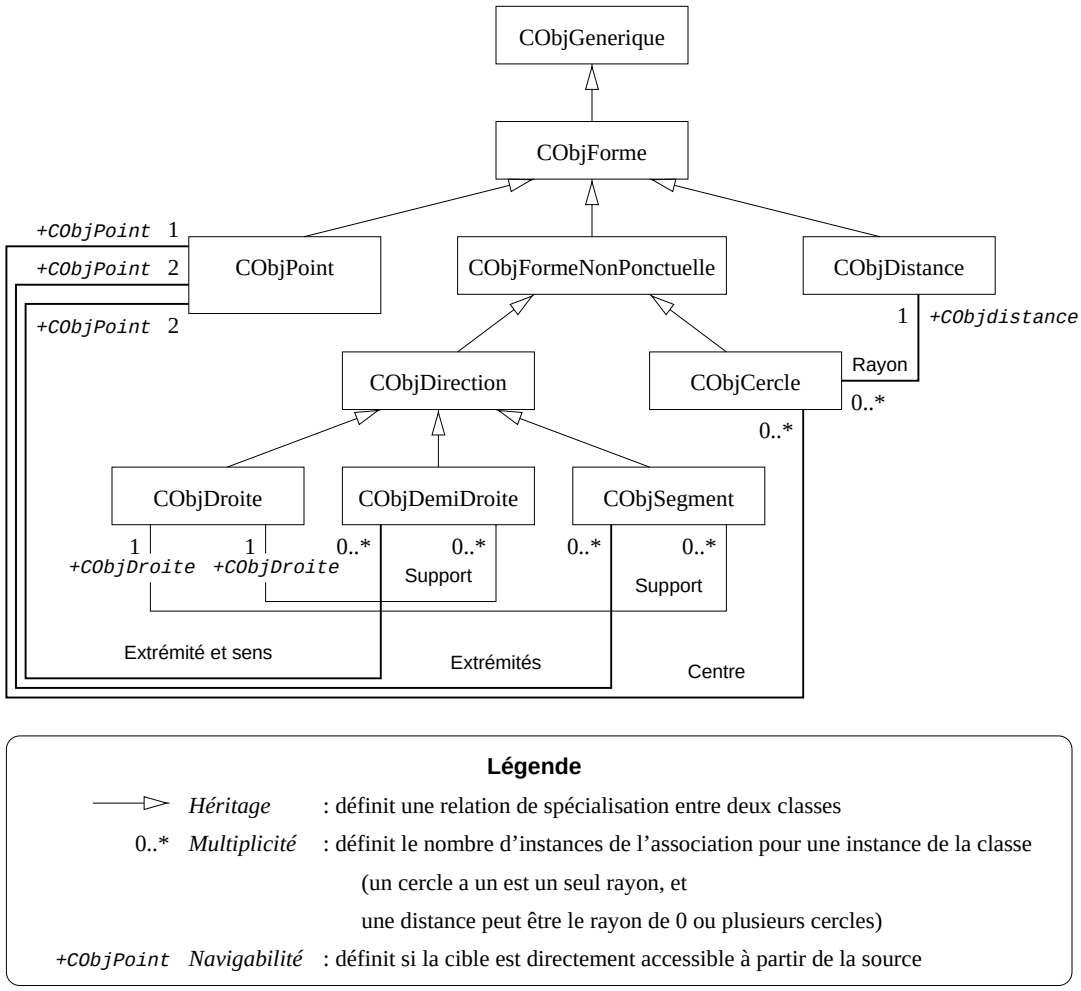

FIG. 6.1 – *Diagramme de la hiérarchie des classes des objets de GDRev.*

La figure 6.1 illustre la hiérarchie des classes que nous avons mise en œuvre dans GDRev dans le formalisme UML [KMPR98]. La justification d'une telle hiérarchie découle du caractère commun de certains attributs et de certaines méthodes associés aux classes. Ainsi, la classe :

– `CObjGenerique` est la racine de cette hiérarchie arborescente.

Les attributs de cette classe comportent par exemple un entier identifiant de manière unique les objets définis et un drapeau indiquant si l'objet est visible ou non.

Les méthodes virtuelles de cette classe, c'est-à-dire les méthodes spécifiques aux classes qui héritent de celle-ci, sont par exemple la procédure d'affichage, la procédure de sauvegarde et la fonction de calcul de la distance séparant l'objet du pointeur de la souris.

- `CObjForme` est la racine de l'arborescence associée aux objets géométriques de GDRev.

  Un attribut commun à tout objet géométrique est par exemple le nom que l'utilisateur donne à ce dernier

- `CObjPoint` est la classe des objets de type point qui sont caractérisés par leur abscisse x et ordonnée y.

- `CObjDistance` est la classe des objets de type distance qui sont caractérisés par leur mesure d.

- `CObjFormeNonPonctuelle` est la racine de l'arborescence associée aux objets non ponctuels, c'est-à-dire aux droites, aux demi-droites, aux segments et aux cercles.

  Les attributs spécifiques à cette classe sont par exemple l'épaisseur et le style du trait (plein, pointillé, ...) associé à la représentation affichée sur le dessin de l'objet.

- `CObjCercle` est la classe des objets de type cercle caractérisés par leur centre (un point) et leur rayon (une distance).

- `CObjDirection` est la racine de l'arborescence associée aux objets rectilignes, c'est-à-dire aux droites, aux demi-droites et aux segments.

- `CObjDroite` est la classe des objets de type droite caractérisés par leur coefficients a, b et c de l'équation $ax + by + c = 0$.

- `CObjDemiDroite` est la classe des objets de type demi-droite caractérisés par leur extrémité (un point) leur support (une droite) et leur orientation définie par un point.

- `CObjSegment` est la classe des objets de type segment caractérisés par leurs extrémités (deux points) et leur support (une droite).

Cette organisation hiérarchique a de nombreux avantages. Tout d'abord, elle est très intuitive. Ensuite, elle est concise et aussi facilement évolutive. Enfin, elle permet de satisfaire aux contraintes d'une mise en œuvre structurée, modulaire et facile à maintenir.

6.3.2 Classe des propriétés

Du fait que nous distinguons les objets de ses propriétés qui le lient aux autres objets d'une figure, une structure définissant une propriété se résume à une entité permettant d'accéder à des objets, identifiée par un foncteur et caractérisée par un drapeau indiquant si la propriété est un littéral LDL ou la négation d'un littéral LDL.

```
class CPropriete: public CObject
{
// Attributs
public:
    /* Drapeau indiquant s'il s'agit d'un litteral, ou de la negation d'un litteral */
    BOOL    m_bSigne;

    /* Chaine de caracteres du foncteur de la propriete */
    CString  m_strProp;

    /* Nombre d'attributs de la propriete */
    int      m_iNbAttributs;

    /* Tableau de pointeurs vers les attributs de la propriete */
    CTypedPtrArray<CObArray, CObjForme*> m_oaAttributs;

// Operations
public:
    ...
};
```

TAB. 6.1 – *Classe CMyPropriete.*

La définition que nous proposons d'une classe associée au concept de propriété découle de cette remarque. Celle-ci, dénommée classe `CPropriete`, est illustrée par le tableau 6.1. Cette classe est constituée d'un attribut de type chaîne de caractères *m_strProp* identifiant le foncteur de la propriété, liant les *m_iNbAttributs* objets de

cette dernière, pointés par les éléments du tableau *m_oaAttributs*, ou de la négation de propriété suivant la valeur du drapeau booléen *m_bSigne*.

6.4 Classe d'un plan de construction

Parmi les méthodes de résolution des contraintes géométriques que nous présentons dans la troisième partie de ce manuscrit, l'une d'elle consiste à déterminer un plan de construction de la figure à l'aide de la règle et du compas. Si un tel plan de construction a pu être établi, l'actualisation des paramètres des objets de la figure au cours d'une déformation est opéré suivant une approche procédurale. Cette approche consiste à recalculer les paramètres des objets les uns après les autres en suivant les étapes du plan de construction établi.

Nous décrivons successivement dans les paragraphes suivants la structure de dépendance utilisée pour décrire un plan de construction, ainsi que l'algorithme de propagation, repris de Cabri-Géomètre, utilisé pour actualiser les paramètres des objets lors d'animations.

6.4.1 Structure de dépendance

La relation que nous avons choisie pour représenter un plan de construction est la relation de dépendance $\mathcal{R}$, utilisée dans Cabri-Géomètre, telle que deux objets x et y sont liés par la relation $x\mathcal{R}y$ si la construction de l'objet x dépend directement de la construction de l'objet y. Symétriquement, on dit que si deux objets x et y sont tels que $x\mathcal{R}y$ alors l'objet y est un constituant de l'objet x. L'exemple 6.2 suivant illustre cette relation de dépendance.

Exemple 6.2 RELATION DE DÉPENDANCE

Nous considérons la figure dynamique correspondant à un triangle ABC pourvu de son orthocentre H, obtenu à partir des sommets A et B du triangle et du point H. Le graphe des objets et des propriétés de cette figure est illustré par la figure 6.2(a).

Un plan de construction possible de cette dernière consiste à :

1. Créer le point A,
2. Créer le point B,

3. Créer le point H,
4. Construire la droite D_{AB} passant par le point A et le point B,
5. Construire la droite D_{CH} passant par le point H et perpendiculaire à la droite D_{AB},
6. Construire la droite D_{AH} passant par le point A et le point H,
7. Construire la droite D_{BH} passant par le point B et le point H,
8. Construire la droite D_{AC} passant par le point A et perpendiculaire à la droite D_{BH},
9. Construire la droite Dbc passant par le point B et perpendiculaire à la droite D_{AH},
10. Construire le point A à l'intersection de la droites D_{AB} et de la droite D_{AC}.

Ce plan de construction conduit au graphe des dépendances illustré par la figure 6.2(b).

Pour établir de telles relations, nous avons défini une classe permettant de lier des instances d'objets de la hiérarchie des classes d'objets géométriques. Cette classe est la classe `CStructureDependance` que nous illustrons par le tableau 6.2.

Cette classe est constituée d'un attribut de type entier *m_iType* identifiant le type de la relation de dépendance, liant l'objet défini par cette relation, et pointé par *m_pObjDefini* à ces constituants au nombre de *m_iNbConstituants* et pointés par les éléments du tableau *m_oaConstituants*.

La principale méthode de cette classe est la fonction *RecalculerObjet*. Celle-ci se résume à appeler la fonction de calcul associée au type de la relation définie et pointée par l'attribut *pFctCalcul*. Ces fonctions de calcul sont les fonctions protégées *CalculDroite*, *CalculDroitePerpendiculaire*, ... spécifiques aux différentes relations de dépendance qui retournent faux si l'objet n'a pas pu être déterminé, et vrai sinon.

6.4.2 Propagation de la déformation

L'actualisation des objets d'une figure lors de l'animation d'un point de celle-ci à l'aide d'un plan de construction de cette dernière consiste à déterminer dans un étape préliminaire la liste des objets de la figure modifiés et l'ordre d'actualisation de ceux-ci.

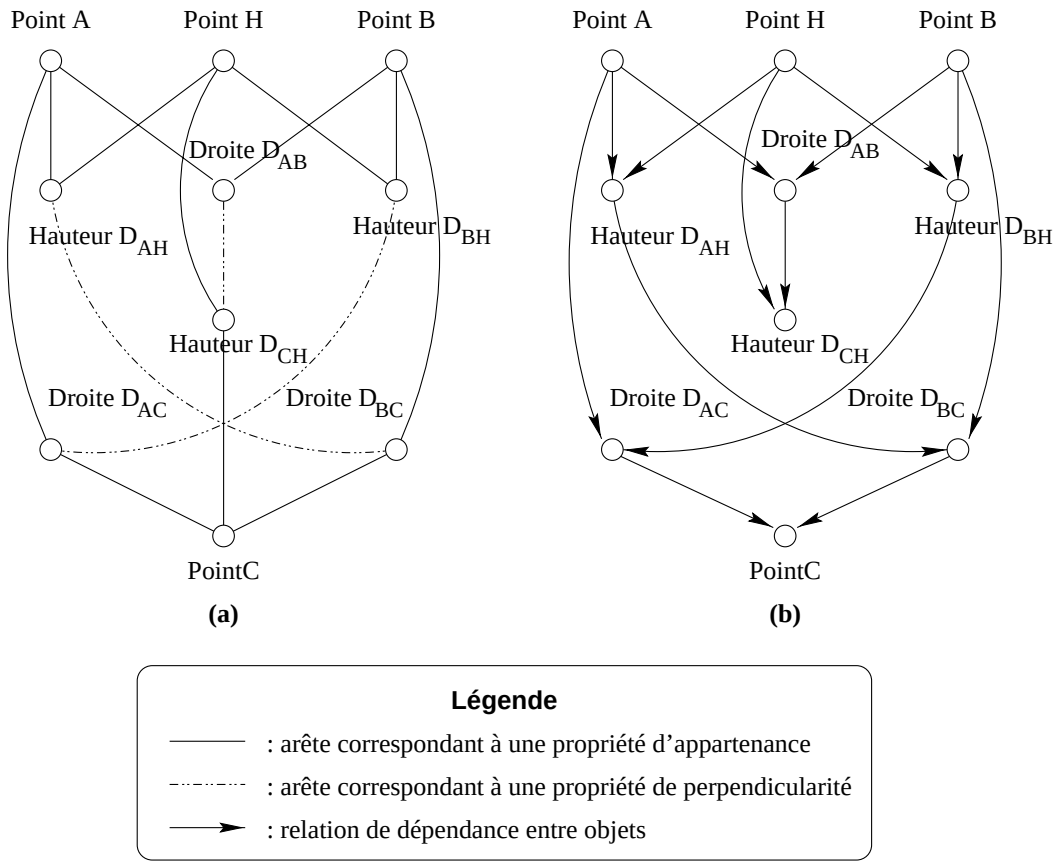

FIG. *6.2* – Graphe des propriétés et des dépendances d'un triangle et de son orthocentre.

Une fois cette liste établie, pour chacune des positions prises par le point déplacé au cours de l'animation il s'agit pour le système de :

1. recalculer les positions des objets,
2. effacer la figure,
3. réafficher les objets de la figure.

Cette étape préliminaire se ramène à un tri topologique sur le sous-graphe des dépendances issu de l'objet sélectionné. Une manière d'effectuer un tel tri topologique est de concaténer les listes des dépendances des objets déplacés à la suite les unes des autres en prenant garde lors de l'élimination des occurrences multiples de garder la dernière occurrence.

Afin d'illustrer ce propos, examinons l'exemple 6.3.

```
class CStructureDependance: public CObject
{
// Attributs
public:
     /* Type de la relation de dependance */
     int m_iType;

     /* Nombre de constituants de la relation de dependance */
     int m_iNbConstituants;

     /* Pointeur vers l'objet defini par la relation de dependance */
     CObjForme * m_pObjDefini;

     /* Tableau de pointeurs vers les constituants de l'objet defini
     par la relation de dependance */
     CTypedPtrArray<CObArray, CObjForme*> m_oaConstituants;

     /* Pointeur vers la fonction de calcul associe a la relation de dependance */
     BOOL (CMyStructureDependance::*pFctCalcul) ();

// Operations
public:
     /* Fonction de calcul d'un objet */
     BOOL RecalculerObjet( ) { return (this->*pFctCalcul)(); };
     ...

// Operations
protected:
     /* Fonction de calcul d'une droite passant par deux points */
     BOOL CalculDroite( );
     /* Fonction de calcul d'une droite perpendiculaire passant par un point */
     BOOL CalculDroitePerpendiculaire( );
     ...
};
```

TAB. 6.2 – *Classe CMyStructureDependance.*

Exemple 6.3 PROPAGATION DE LA DÉFORMATION

On considère la figure dynamique correspondant à un triangle ABC pourvu de son orthocentre H, obtenu à partir des sommets A et B du triangle et du point H. L'objet sélectionné pour une animation est le point H. La figure 6.3 donne le graphe des dé-

pendances des objets ainsi que le sous-graphe des objets modifiés suite au déplacement du point H.

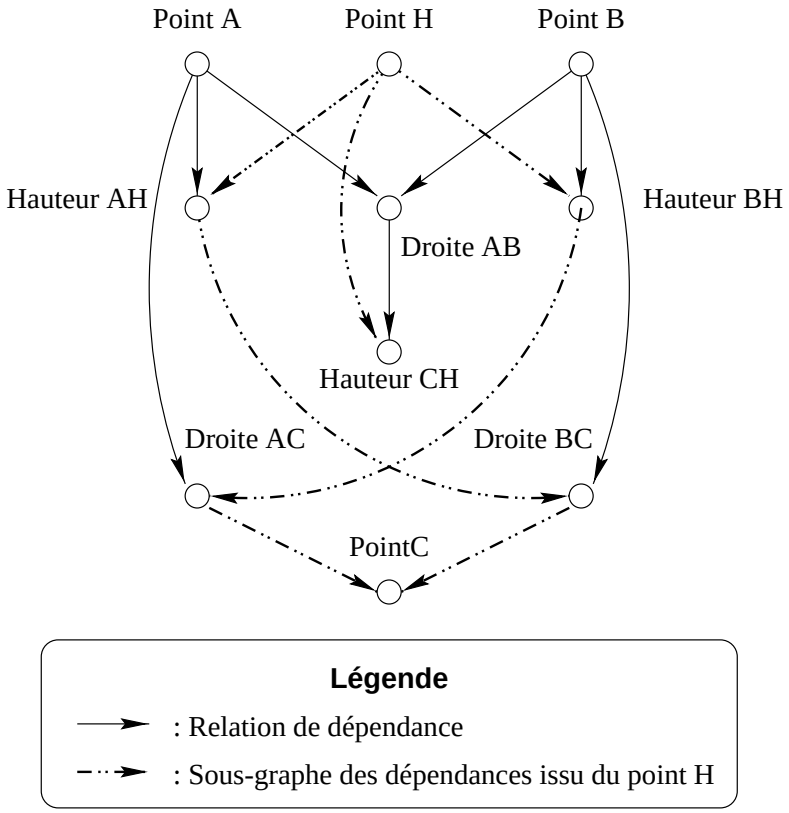

FIG. *6.3* – Graphe des dépendances d'un triangle et de son orthocentre.

Les listes des dépendances des différents objets de la figure sont reprises dans le tableau 6.3 .

La concaténation des listes des dépendances issues du point H donne :

Point H, Hauteur D_{AH}*, Hauteur* D_{BH}*, Hauteur* D_{CH}*, Droite* D_{BC}*, Droite* D_{AC}*, Point C, Point C.*

L'élimination des occurrences multiples conduit à la liste d'actualisation ordonnée :

Point H, Hauteur D_{AH}*, Hauteur* D_{BH}*, Hauteur* D_{CH}*, Droite* D_{BC}*, Droite* D_{AC}*, Point C.*

Les performances de cette approche sont difficilement mesurables. Elle donne une impression de fluidité et de continuité n'ayant aucune commune mesure avec les situations où de tels plans de construction n'ont pu être établi.

Objets	*Listes des dépendances*
LES POINTS	
Point A	*Hauteur* D_{AH}, *Droite* D_{AB}, *Droite* D_{AC}
Point B	*Hauteur* D_{BH}, *Droite* D_{AB}, *Droite* D_{BC}
Point C	
Point H	*Hauteur* D_{AH}, *Hauteur* D_{BH}, *Hauteur* D_{CH}
LES DROITES	
Droite D_{AB}	*Hauteur* D_{CH}
Droite D_{AC}	*Point C*
Droite D_{BC}	*Point C*
Hauteur D_{AH}	*Droite* D_{BC}
Hauteur D_{BH}	*Droite* D_{AC}
Hauteur D_{CH}	

TAB. *6.3* – Listes des dépendances d'un triangle construit à partir de ses sommets A et B et de son orthocentre H.

6.5 Aspects quantitatifs

Dans son ensemble, la mise en œuvre des langages d'interface de GDRev représente :

- l'écriture de 62.000 lignes de C++ contenues dans 315 fichiers,
- la définition de 200 classes d'objets,
- une mise en œuvre d'une durée de 8 mois homme,
- un code exécutable de 3.4 Mo,

Ces quelques chiffres confirment la place de plus en plus importante tenue par les interfaces dans le processus de développement d'un système. Ils sont à rapprocher des aspects quantitatifs de la mise en œuvre des méthodes de résolution des contraintes géométriques présentés dans le chapitre 9. L'utilisation des Microsoft Foundation Classes et des concepts de programmation objet ont permis d'effectuer une mise en œuvre relativement rapide, modulaire, facile a maintenir et à priori relativement correcte. Bien que cette dernière remarque mériterait d'être validée par une expérimentation de grande envergure.

Troisième partie

Définition du système de coopération de résolveurs

Chapitre 7

Agents mis en oeuvre

La notion d'"agent" a été utilisée dans de nombreux domaines de recherche en informatique comme par exemple en intelligence artificielle [Sho93, WJ95, FG96], en programmation concurrente [Sar93] et en robotique. Une caractéristique remarquable de cette notion est que jusqu'à présent aucun réel consensus n'a été obtenu sur sa définition. Ainsi, qu'entend-t'on par le terme agent ? Le sens que nous donnons dans ce manuscrit ne correspond pas aux définitions issues du domaine de l'intelligence artificielle. Notre vision d'un agent est celle d'un programme capable de communiquer avec d'autres agents (ou programmes) et capable de percevoir son environnement, c'est-à-dire l'état dans lequel se trouvent les autres agents.

Pour attaquer le problème de la résolution de contraintes géométriques, nous avons défini six agents. Nous détaillons dans ce chapitre ces différents agents et leurs rôles. Les trois premiers utilisent des méthodes générales de résolution d'équations. Les trois autres résultent de nos réflexions sur le domaine d'application à savoir la géométrie. Nous décrivons tout d'abord les trois premiers, dans la section 7.1 l'agent appelé agent linéaire, dans la section 7.2 l'agent appelé agent quadratique, et dans la section 7.3 l'agent appelé agent intervalle. Nous explicitons ensuite les trois autres, dans la section 7.4 l'agent appelé agent de complétion d'objets, dans la section 7.5 l'agent appelé agent de complétion de propriétés, et dans la section 7.6 l'agent appelé agent règle et compas. Les mécanismes de coopération entre ces agents sont décrits dans le chapitre 8.

7.1 Agent linéaire

Cet agent a pour charge de résoudre des équations linéaires ou pseudo-linéaires, c'est-à-dire des équations qui deviennent linéaires après l'instanciation de certaines de leurs variables. Ainsi, l'agent linéaire évalue les atomes de propriété appDr, appDD, appSeg, par, perp, invSens, memeSens et milieu (cf tableau 4.1 page 59). Pour ce faire, il adopte une approche numérique exacte en utilisant un résolveur basé sur l'algorithme d'élimination de Gauss.

Le problème de cette approche est qu'elle se limite aux systèmes d'équations linéaires. Cette limite interdit par exemple à priori les intersections de droites et de cercles connus.

7.2 Agent quadratique

Cet agent a pour charge de pallier aux limites de l'agent linéaire. Son algorithme permet de résoudre des équations quadratiques simples (équations à une seule variable) et constitue ainsi un prolongement de l'algorithme d'élimination de Gauss sur les contraintes quadratiques. Il évalue les atomes de propriété appCc et distPP (cf tableau 4.1 page 59) qui correspondent tous les deux à une équation de la forme :

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

si celle-ci peut s'écrire sous la forme :

$$a.z^2 + b.z + c = 0$$

où z est l'une des variables du quintuplet (x, x_0, y, y_0, r) et où a, b et c sont des constantes. Pour ce faire, il détermine les coefficients $a_0, b_0, a_1, b_1, a_2, b_2, a_3, b_3, a_4$ et b_4 tels que :

$$z = a_0.x + b_0 \quad (7.1)$$

$$z = a_1.x_0 + b_1 \quad (7.2)$$

$$z = a_2.y + b_2 \quad (7.3)$$

$$z = a_3.y_0 + b_3 \quad (7.4)$$

$$z = a_4.r + b_4 \quad (7.5)$$

A partir de ces coefficients, les constantes a, b et c se calculent comme suit :

$$a = (a_0 - a_1)^2 + (a_2 - a_3)^2 - a_4^2 \quad (7.6)$$

$$b = 2 \times ((a_0 - a_1) \times (b_0 - b_1) + (a_2 - a_3) \times (b_2 - b_3) - a_4 \times b_4) \quad (7.7)$$

$$c = (b_0 - b_1)^2 + (b_2 - b_3)^2 - b_4^2 \quad (7.8)$$

Bien sur cet algorithme n'est pas complet. Il se révèle toutefois utile. Il permet par exemple de déterminer l'intersection d'une droite et d'un cercle connus. Son inconvénient est qu'il introduit des erreurs numériques si le résultat fourni est sous la forme d'un nombre flottant. Pour limiter l'introduction de ces erreurs, une idée consiste à retarder autant que possible les appels à cet agent en s'inspirant du principe de résolution naïf [Col93] du résolveur du langage de programmation logique avec contraintes PrologIII. Une autre idée consiste, pour rester dans une arithmétique exacte, à intégrer dans la représentation de tous les nombres construits la racine carrée calculée par la résolution du trinôme comme cela est présenté dans le paragraphe 1.3.3.

7.3 Agent intervalle

Nous décrivons dans les paragraphes suivants un agent dit "intervalle". Nous commençons par exposer d'une part les objectifs de cet agent et d'autre part l'intérêt d'user d'une telle approche. Puis, nous décrivons succinctement les principes des résolveurs sur intervalles afin d'en faire ressortir des limites. Enfin, nous illustrons nos propos par un exemple de résolution.

7.3.1 Objectifs de l'agent intervalle

Cet agent a pour charge de déterminer par approximations successives les solutions du système d'équations issu de la spécification de la figure suivant une approche sur le domaine des intervalles [Cle87]. Il évalue tous les atomes de propriété (cf tableau 4.1 page 59).

Les intérêts de cette approche sont multiples. Tout d'abord, elle permet de s'attaquer à des problèmes de construction difficiles qualitativement, c'est-à-dire de s'attaquer à des figures contenant peu d'objets reliés entre eux principalement par des contraintes géométriques se traduisant par des équations quadratiques. Un autre avantage de cette approche est qu'elle est correcte, c'est-à-dire qu'elle n'introduit par d'er-

reur de calcul. Enfin, cette approche est en général rapide [LGRT96].

7.3.2 Résolveurs sur intervalles

L'idée fondamentale des résolveurs sur intervalles est de manipuler des variables auxquelles sont associées non plus uniquement une valeur exacte, mais un domaine de valeurs, un intervalle.

L'intérêt des méthodes de calculs sur intervalles est double. D'une part, elles fournissent un ou une union d'intervalles résultats en garantissant l'encadrement de la ou des solutions correctes. D'autre part, elle permettent de s'attaquer à de nouveaux problèmes et d'étudier non plus seulement des valeurs particulières de variables, mais les domaines de variation de ces dernières.

Pour ce faire, l'algorithme de consistance [Dav87, LR97, CDR98] généralement utilisé est un algorithme de point fixe de type AC-3 [Mac77]. Cet algorithme est décrit dans le tableau 7.1. Dans cet algorithme, la fonction $\tilde{C}$ est l'extension aux intervalles de la fonction de calcul associée à la contrainte C. La notation $D_\emptyset$ représente les domaines des variables réduit au domaine vide. Ceci se produit lorsque le système de contraintes est inconsistant.

```
1   FILTRAGE(C_1, ..., C_n : contraintes,
2            D : domaines des variables des contraintes)
3
4   queue ← { C_1, ..., C_n }
5   Tant Que D ≠ D_∅ et queue ≠ ∅ Faire
6       DepilerContrainte(queue, C)
7       D' ← C̃(D)
8       Si D' ≠ D Alors
9           queue ← queue ∪ { C' | C' ∈ { C_1, ..., C_n } ∧ C' ≠ C ∧
10                                var(C) ∩ var(C') ≠ ∅ }
11          D ← D'
14      FinSi
15  FinTantQue
16  Filtrage ← D
```

TAB. 7.1 – *Algorithme de propagation d'intervalles.*

Une première limite de cette approche est qu'elle peut conduire à une explosion combinatoire. En effet, les extensions des fonctions de calcul associées aux contraintes

ne préservant pas toujours la convexité des domaines, comme par exemple la division, il s'ensuit que les domaines calculés par la fonction $\tilde{C}$ ne sont pas un domaine mais une union de domaines, générant ainsi des points de choix. Pour pallier à cette limite, une idée consiste à approcher cette union par un intervalle englobant [Lho93, Ben94, BMvH94, BO97].

Une fois un point fixe atteint, une possibilité pour obtenir des domaines réduits pour les différentes variables consiste à avoir recours à des énumérations d'intervalles. Par de telles énumérations, on force le résolveur à tenter de nouveau de réduire les bornes du domaine d'une variable sur une première moitié de ce dernier, puis sur l'autre.

Enfin, une autre limite de cette approche réside dans sa très grande dépendance vis-à-vis de la forme des contraintes[1] comme l'illustre l'exemple 7.1.

Exemple 7.1 DÉPENDANCE DES APPROCHES SUR LES INTERVALLES

Le calcul du domaine de la fonction :

$$f(x) = x - x \text{ avec } x \in [0,1]$$

donne $f(x) \in [-1,1]$ *alors qu'il paraît facile de déduire que le domaine de* $f(x)$ *est réduit à 0. Ceci est lié à l'interprétation de la fonction* $f(x)$ *comme étant la fonction* $f(x) = x - y$ *avec* $x, y \in [0,1]$.

Pour pallier à cette limite, une idée consiste à user d'algorithmes formels comme nous l'avons présenté dans le chapitre 2.

7.3.3 Exemple de résolution sur intervalles

Afin d'illustrer l'intérêt de la définition de cet agent, nous considérons le problème suivant seulement soluble par ce dernier.

Exemple 7.2 RÉSOLUTION SUR INTERVALLES

Construire un triangle ABM à partir de l'angle $\widehat{AMB}$ *et de la longueur des médianes issues des sommets A et B. La figure 7.2 illustre le résultat attendu.*

Le système algébrique (S) correspondant à cet énoncé est le suivant :

1. Les mathématiciens parlent de telles dépendances sous le vocable : décorélation de variables.

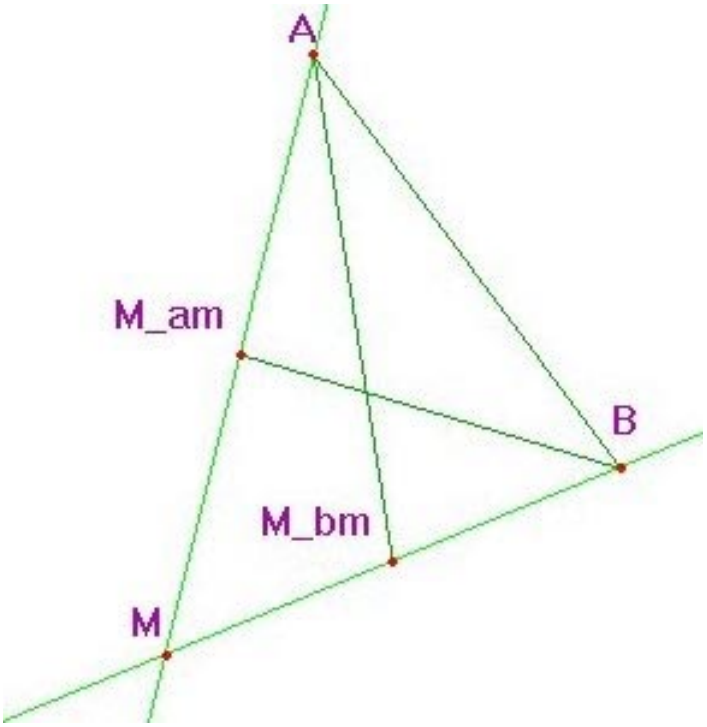

FIG. *7.2* – Triangle déterminé par un de ses angles et les longueurs de ses médianes opposées.

$$
\begin{aligned}
y_M &= 2x_M && (7.9)\\
y_M &= \frac{1}{2}x_M + \frac{3}{2} && (7.10)\\
y_A &= 2x_A && (7.11)\\
y_B &= \frac{1}{2}x_B + \frac{3}{2} && (7.12)\\
2x_{M_{AM}} &= x_M + x_A && (7.13)\\
2y_{M_{AM}} &= y_M + y_A && (7.14)\\
2x_{M_{BM}} &= x_M + x_B && (7.15)\\
2y_{M_{BM}} &= y_M + y_B && (7.16)\\
(x_A - x_{M_{BM}})^2 + (y_A - y_{M_{BM}})^2 &= 3^2 && (7.17)\\
(x_B - x_{M_{AM}})^2 + (y_B - y_{M_{AM}})^2 &= 3^2 && (7.18)\\
& && (7.19)
\end{aligned}
$$

Les résultats, obtenus quasi-instantanément (resp. en 380 millisecondes) en PrologIV sur un PC à 300 MHz à partir du système précédent sans (resp. avec) énumération, sont résumés dans le tableau 7.2.

Objets	***Variables***	***Intervalles***	
		Sans énumération	***Avec énumération***
OBJETS DONNÉS			
droite Dam	m_{AM}	*[4, 4]*	*[4, 4]*
	p_{AM}	*[-18.7, -18.7]*	*[-18.7, -18.7]*
droite Dbm	m_{BM}	*[0.4, 0.4]*	*[0.4, 0.4]*
	p_{BM}	*[3.2, 3.2]*	*[3.2, 3.2]*
distance $D_{AM_{BM}}$	$d_{AM_{BM}}$	*[4.5, 4.5]*	*[4.5, 4.5]*
distance $D_{BM_{AM}}$	$d_{BM_{AM}}$	*[3.5, 3.5]*	*[3.5, 3.5]*
OBJETS CONSTRUITS			
point A	x_A	$[-\infty, +\infty]$	*[7.395201, 7.395203]*
	y_A	$[-\infty, +\infty]$	*[10.880806, 10.88081]*
point B	x_B	$[-\infty, +\infty]$	*[10.086597, 10.086603]*
	y_B	$[-\infty, +\infty]$	*[7.234638, 7.234642]*
point M	x_M	$[-\infty, +\infty]$	*[6.083333, 6.083334]*
	y_M	$[-\infty, +\infty]$	*[5.633332, 5.633334]*
point M_{AM}	$x_{M_{AM}}$	$[-\infty, +\infty]$	*[6.739267, 6.739269]*
	$y_{M_{AM}}$	$[-\infty, +\infty]$	*[8.257068, 8.257072]*
point M_{BM}	$x_{M_{BM}}$	$[-\infty, +\infty]$	*[8.084964, 8.084968]*
	$y_{M_{BM}}$	$[-\infty, +\infty]$	*[6.433985, 6.433988]*

TAB. *7.2* – Triangle construit à partir de l'angle $\widehat{AMB}$ et de la longueur des médianes issues des sommets A et B.

7.4 Agent complétion d'objets

Nous décrivons dans les paragraphes suivants un agent dit de "complétion d'objets". Nous commençons par exposer d'une part les objectifs de cet agent et d'autre part l'intérêt d'ajouter à la spécification d'une figure des objets redondants. Puis, nous illustrons nos propos par un exemple d'ajout d'objets.

7.4.1 Objectifs de l'agent de complétion d'objets

Cet agent a pour charge d'ajouter à la spécification de la figure des objets intermédiaires redondants, facilitant la construction de cette dernière. Un des objectifs de ces ajouts est la recherche d'une construction linéaire de la figure, c'est-à-dire, que l'introduction des objets redondants doit aboutir à la détermination d'un système linéaire permettant la construction. Ainsi, il évalue les atomes de propriété appCc et distPP (cf tableau 4.1 page 59).

Les objets intermédiaires redondants ajoutés sont la corde commune à deux cercles, c'est-à-dire l'ajout systématique de la droite d'équation :

$$(2x - x_1 - x_2)(x_2 - x_1) + (2y - y_1 - y_2)(y_2 - y_1) = r_1^2 - r_2^2 \tag{7.20}$$

pour deux cercles d'équation respective :

$$(x - x_1)^2 + (y - y_1)^2 = r_1^2 \tag{7.21}$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2 \tag{7.22}$$

L'intérêt de l'introduction de la corde commune à deux cercles est double. Tout d'abord, elle permet à l'agent intervalle d'obtenir des domaines plus petits. Mais surtout, elle rend le problème quadratique en deux variables de la construction des points d'intersections de deux cercles, quadratique en une seule variable. Par conséquent, ce problème devient soluble par la coopération des agents linéaire et quadratique.

7.4.2 Exemple de complétion d'objets

Afin d'illustrer le double intérêt de l'introduction de la corde commune à deux cercles, nous considérons le problème suivant.

Exemple 7.3 COMPLÉTION D'OBJETS

Le problème consiste à construire un point d'intersection des cercles C_1 et C_2 centrés respectivement en les points O_1 et O_2 de coordonnées respectives $(0, 0)$ et $(2, 3)$ et de rayon respectif 2 et 3.

Le système d'équations (S) correspondant est :

$$x^2 + y^2 = 2^2 \tag{7.23}$$
$$(x-2)^2 + (y-3)^2 = 3^2 \tag{7.24}$$

La corde commune à ces deux cercles est la droite d'équation :

$$2(2x-2) + 3(2y-3) = 2^2 - 3^2 \tag{7.25}$$

Les intervalles obtenus instantanément sans énumération en PrologIV à partir du système précédent avec ou sans ajout de la corde commune aux deux cercles, sont résumés dans le tableau 7.3.

Variable	***Sans corde commune***	***Avec corde commune***
x	*[-0.8284273, 2]*	*[-0.7692311, 2]*
y	*[0, 2]*	*[0, 1.846154]*

TAB. *7.3* – Intersection de deux cercles connus.

Ainsi, les domaines de l'abscisse et de l'ordonnée du point d'intersection de deux cercles sont davantage réduits en introduisant la corde commune à ces derniers.

De plus, le système d'équations :

$$x^2 + y^2 = 2^2 \tag{7.26}$$
$$(x-2)^2 + (y-3)^2 = 3^2 \tag{7.27}$$
$$y = \frac{-2}{3}x + \frac{4}{3} \tag{7.28}$$

correspondant aux cercles et à la corde commune à ceux-ci peut se réécrire, en remplaçant la variable y dans la première équation, en le système suivant :

$$\frac{5}{9}x^2 + \frac{16}{9}x - \frac{52}{9} = 0 \tag{7.29}$$
$$(x-2)^2 + (y-3)^2 = 3^2 \tag{7.30}$$
$$y = \frac{-2}{3}x + \frac{4}{3} \tag{7.31}$$

Ainsi, on obtient une équation quadratique en la variable x soluble par l'agent quadratique.

7.5 Agent complétion de propriétés

Nous décrivons dans les paragraphes suivants un agent dit de "complétion de propriétés". Nous commençons par exposer les objectifs de cet agent. Puis, nous énumérons les phases de la recherche d'une propriété redondante. Ensuite, nous proposons deux heuristiques visant à déterminer une construction d'une figure non complètement construite. Enfin, nous illustrons nos propos par un exemple de recherche de propriétés redondantes.

7.5.1 Objectifs de l'agent de complétion de propriétés

Cet agent a pour charge d'ajouter à la spécification de la figure des propriétés redondantes liant les éléments de cette dernière, c'est-à-dire de rechercher et d'ajouter à la spécification de la figure des propriétés déductibles de la spécification de celle-ci. L'objectif de ces ajouts est d'aider les autres agents à déterminer une construction d'une figure lorsque celle-ci ne peut être déterminée à partir de la spécification de l'utilisateur. Une autre conséquence attendue de l'ajout de telles propriétés redondantes est l'accélération du processus de résolution.

7.5.2 Phases d'une recherche de propriétés redondantes

Une première approche consiste à chercher à prouver symboliquement [Ost97a] toutes les propriétés possibles. Même si cette première approche est correcte, elle souffre en pratique du coût d'une recherche de preuve symbolique.

Pour minimiser le nombre de ces recherches, une idée consiste à éliminer de la liste des propriétés possibles les contre-propriétés, c'est-à-dire les propriétés fausses sur un modèle de la figure. Dans la situation où la recherche de propriétés redondantes a pour objectif d'aider le processus de résolution, on doit alors faire face paradoxalement au problème d'éliminer de la liste des propriétés possibles les propriétés fausses sur une figure que le système n'arrive pas à construire. Cependant dans le cadre de la recherche de propriétés redondantes, la construction de la figure peut être menée à partir d'un ensemble quelconque d'objets de base.

Il s'ensuit qu'il s'agit de construire une figure à partir d'un ensemble quelconque d'objets de base de la manière la plus judicieuse possible, c'est-à-dire d'une manière telle qu'un minimum de racines carrées soient introduites. Des travaux récents [Bou96]

montrent que, dans de nombreux cas, il serait possible de trouver presque toujours une construction linéaire, c'est-à-dire qu'il serait presque toujours possible de transformer le système de contraintes d'une figure en un système de contraintes linéaires à partir d'un ensemble bien choisi d'objets de base. L'intérêt de déterminer de telles constructions est de pouvoir ensuite valider partiellement ou invalider rapidement de manière numérique exacte une propriété sur le modèle construit.

Les phases de la recherche de propriétés redondantes que nous proposons sont :

1. Produire la liste des propriétés redondantes possibles. Dans la situation où la recherche de propriétés redondantes a pour objectif d'aider le processus de résolution, cette phase peut se limiter à la production de la liste des propriétés redondantes possibles faisant intervenir les éléments de la figure non construits dans la construction projetée et les éléments construits.

2. Eliminer de cette liste les propriétés dont la validité ne peut pas être prouvée de manière numérique approchée correcte, c'est-à-dire à l'aide de l'arithmétique des intervalles, sur le (ou les) dessin(s) produit(s).

3. Eliminer de la liste restante les propriétés dont la validité ne peut pas être prouvée de manière numérique exacte sur le (ou les) dessin(s) produit(s).

4. Eliminer de la liste restante les propriétés dont la validité ne peut pas être prouvée symboliquement.

Ces phases sont éventuellement précédées d'une phase préliminaire qui consiste à déterminer une construction de la figure à partir d'un nouvel ensemble d'objets de base.

7.5.3 Heuristiques de construction

Comme nous l'avons signalé précédemment, si la recherche de propriétés redondantes a pour objectif d'aider à la résolution du problème de construction grâce à l'ajout de ces dernières, nous devons faire face au problème de déterminer une construction de la figure introduisant le minimim de racines carrées. Pour ce faire, il faudrait utiliser un algorithme de recherche de type "branch & bound" en énumérant sur les ensembles d'objets de base possibles. En fait pour des raisons d'efficacité, on peut se satisfaire de l'obtention de la première construction.

La question se pose de savoir s'il est possible de déterminer de manière déterministe un ensemble d'objets de base conduisant à une construction de la figure introduisant peu d'extensions algébriques. La réponse que nous apportons à cette interrogation comprend les deux heuristiques suivantes :

- Heuristique "non construit". L'idée de cette heuristique de reconstruction est de considérer comme ensemble d'objets de base un ensemble contenant au moins un objet non construit dans la figure projetée par l'utilisateur.

  L'approche consiste à instancier successivement les objets de la figure en commençant par les objets non construits dans la figure projetée par l'utilisateur jusqu'à aboutir à une construction complète.

  La justification de cette approche est subjective et consiste considérer que si on ne peut construire certains objets, il peut être intéressant de prendre ceux-ci comme objets de base.

- Heuristique "plus contraint". L'idée de cette heuristique de reconstruction est de considérer comme ensemble d'objets de base un ensemble contenant l'objet de la figure sur lequel porte le plus grand nombre de contraintes.

  L'approche consiste à instancier successivement les objets de la figure en commençant par les objets sur lesquels porte le plus grand nombre de contraintes jusqu'à aboutir à une construction complète.

  La justification de cette approche se trouver dans le fait qu'à chaque étape, l'instanciation de l'objet sur lequel porte le plus grand nombre de contraintes à pour conséquence de réduire le système d'équations sous-jacent.

Outre ces heuristiques de reconstruction, il faut aussi définir des méthodes de demi-instanciation introduisant un minimum de racines carrées. En effet, la demi-instanciation d'un point A sur un cercle C de centre le point O de coordonnées (x_O, y_O) et de rayon r peut être faite suivant une approche :

- Avec introduction d'une racine carrée. Cette méthode consiste simplement à instancier par exemple l'abscisse du point M dans l'intervalle $[x_O - r, x_O + r]$ et à calculer l'ordonnée correspondante.

- Sans introduction d'une racine carrée. Cette méthode consiste à construire un point sur un cercle en s'appuyant sur la construction d'un triangle rectangle. L'idée consiste à partir d'un point M déjà construit linéairement sur le cercle, à

prendre son symétrique N par rapport au centre O du cercle. Cette construction linéaire se résume à spécifier que le point O est le milieu des points M et N. Puis, il s'agit de considérer un point P quelconque défini d'une manière linéaire, et de construire la droite d_{MP}. La construction s'achève en considérant le projeté orthogonal A du point N sur la droite d_{MP}.

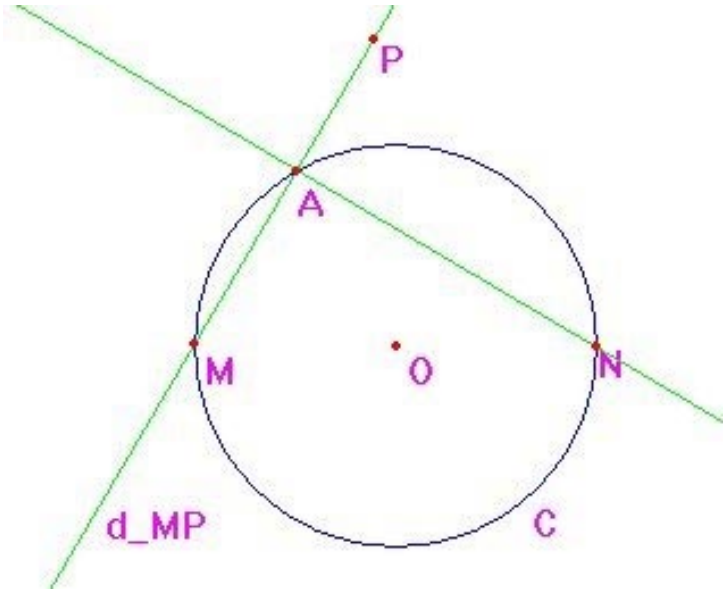

FIG. 7.4 – *Demi-instanciation d'un point sur cercle sans introduction d'une racine carrée.*

Cette méthode de demi-instanciation linéaire d'un point sur un cercle est illustrée par la figure 7.4.

Ces heuristiques de reconstruction associées à de telles approches judicieuses de la demi-instanciation ont donné des résultats très intéressant comme nous l'expliquons dans le chapitre 10.

7.5.4 Exemple de complétion de propriétés

Afin d'illustrer cette recherche de propriétés redondantes, nous considérons le problème suivant extrait de [But75].

Exemple 7.5 COMPLÉTION DE PROPRIÉTÉS

Soient les points distincts A, B, O et R. Soit le cercle C de centre O passant par le point R. Construire les droites d et d′ passant respectivement par les points A et B

telles que la droite d' soit la médiatrice du segment $[M_1, M_2]$ formé par les intersections de la droite d et du cercle C.

La figure 7.5 illustre le problème de construction posé.

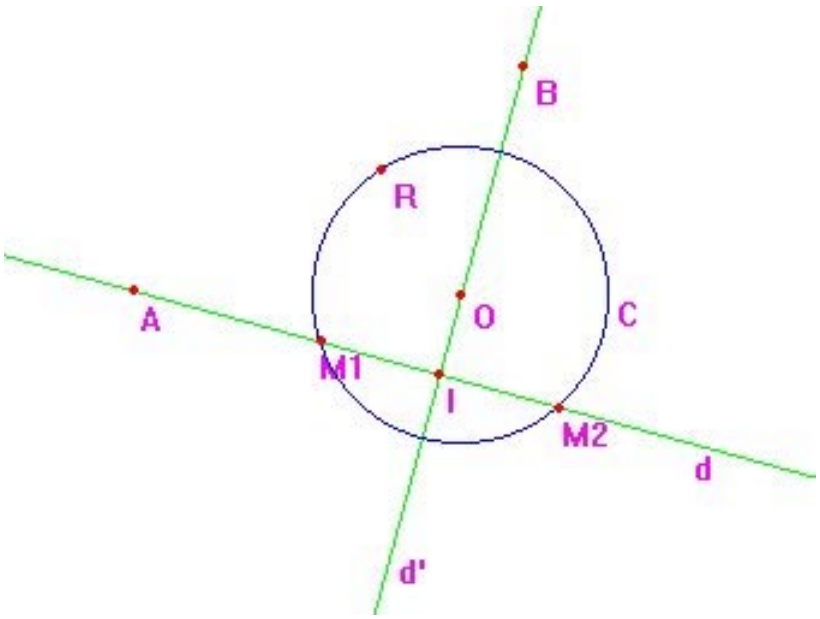

FIG. *7.5* – Ajout de propriétés redondantes.

A l'aide des agents linéaire, quadratique et complétion d'objets, la construction de la figure reste inachevée. Seul le cercle C est construit. Une étape préliminaire consiste par conséquent à construire le dessin de la figure 7.5 à partir des points instanciés distinctement R, M_1 et M_2 et des points demi-instanciés A et B en employant l'heuristique dite du "non construit". Cette étape menée à bien, la recherche de propriétés redondantes est conduite comme suit :

1. *Produire la liste suivante :*

   [appDr(A, d'), ..., appDr(O, d'), ..., milieu(A, M_1, O), ...]

   qui regroupe les propriétés des objets non construits dans la construction projetée. Ici, ces objets sont : I, M_1, M_2, d et d'.

2. *Réduit la liste précédente à la liste :*

   [appDr(O, d')]

qui contient la seule propriété non invalidé grâce à des vérifications numériques approchées correctes sur le modèle construit à l'aide de l'arithmétique des intervalles.

3. *Cette propriété restante pouvant être vérifiée de manière exacte sur la figure, il s'ensuit que la liste des propriétés possibles reste identique.*

4. *Cette propriété restante pouvant être prouvée symboliquement, on obtient comme propriété redondante l'appartenance du point O à la droite d'.*

L'ajout de la propriété redondante d'appartenance du point O à la droite d' rend ce problème de construction d'une part linéaire et d'autre part soluble par notre agent règle et compas dont la description suit.

7.6 Agent règle et compas

Nous décrivons dans les paragraphes suivants un agent dit "règle et compas". Nous commençons par exposer les objectifs de cet agent. Puis, nous détaillons une classification par ordre de difficulté croissante des problèmes de construction de figures. Nous abordons ensuite la problématique de la définition de cet agent. Enfin, nous illustrons nos propos par un exemple de recherche de plan de construction.

7.6.1 Objectifs de l'agent règle et compas

Cet agent a pour objectif d'établir un plan de construction d'une figure à l'aide d'instruments géométriques en prenant pour hypothèse que celle-ci est constructible numériquement. Un plan de construction consiste en une séquence de constructions simples où chacune utilise les objets de base de la figure, ou les objets déjà construits, pour construire un nouvel objet. Si ce plan de construction ne peut être établi dans une première approche, il s'appuie sur une résolution numérique du problème pour déterminer des lieux géométriques [Car88] permettant de poursuivre la détermination du plan de construction [CI97].

L'intérêt de cette recherche d'un plan de construction d'une figure est double. Tout d'abord, elle conduit à des animations beaucoup plus fluides grâce à l'exploitation de ce dernier. En effet, la construction d'une figure à l'aide d'un plan de construction se résume à suivre les étapes de celui-ci qui déterminent les objets de la figure les uns

après les autres. Ensuite, elle peut présenter un double intérêt d'ordre pédagogique. En effet, cette recherche peut d'une part aider un élève en difficulté face à un problème de construction et d'autre part être utilisée par un professeur afin d'étudier la faisabilité et la difficulté de problèmes de construction.

7.6.2 Classification des problèmes de construction

Nous distinguons les trois grandes classes de problèmes suivantes :

- Monde clos. Cette classe de problèmes correspond aux situations où tous les objets et toutes les propriétés nécessaires à l'établissement d'un plan de construction sont présents dans la spécification de la figure. Il s'ensuit qu'il suffit de trouver à chaque étape quel objet inconnu peut être construit à partir des objets déjà construits ou de base. Ceci se résume à un tri topologique.

  Un exemple de problème de cette classe est la construction d'un triangle ABC à partir de deux de ses sommets et de son orthocentre G.

- Monde clos étendu. Cette classe de problèmes correspond aux situations où tous les objets nécessaires à l'établissement d'un plan de construction sont présents dans la spécification de la figure, mais où toutes les propriétés nécessaire ne sont pas nécessairement présentes. Ces propriétés manquantes ne sont pas explicitement données, mais sont déductibles de la spécification de la figure. Il s'ensuit qu'il suffit de rechercher et d'ajouter à la spécification de la figure toutes les propriétés déductibles de la spécification, et non présentes dans cette dernière afin de se ramener à un problème de la classe précédente du monde clos.

  Un exemple de problème de cette classe est le problème 7.5.

- Monde non clos. Cette classe de problèmes correspond aux situations où tous les objets et toutes les propriétés nécessaires à l'établissement d'un plan de construction ne sont pas nécessairement présents dans la spécification de la figure. Pour pallier à ces insuffisances, il s'agit d'une part de déterminer des objets utiles à la construction de la figure, et d'autre part d'être capable de trouver une construction de ces objets. Pour ce faire, nous utilisons l'étude des lieux des points de la figure comme le propose Petersen J [Pet90].

  Nous divisons cette classe en les sous-classes suivantes :

  - *Monde non clos linéaire.* Cette classe de problèmes correspond aux situations du monde non clos où le système d'équations résultant de la spécifica-

tion de la figure augmenté des objets et des propriétés trouvées est linéaire, ou résolu de manière linéaire. Il s'ensuit que les lieux des points étudiés dans cette sous-classe de problèmes sont des droites.

Un exemple de problème de cette classe est le problème 7.6 suivant.

Exemple 7.6 CONSTRUCTION DU MONDE NON CLOS LINÉAIRE

Construire un triangle $a'b'c'$ semblable au triangle abc inscrit dans le triangle ABC.

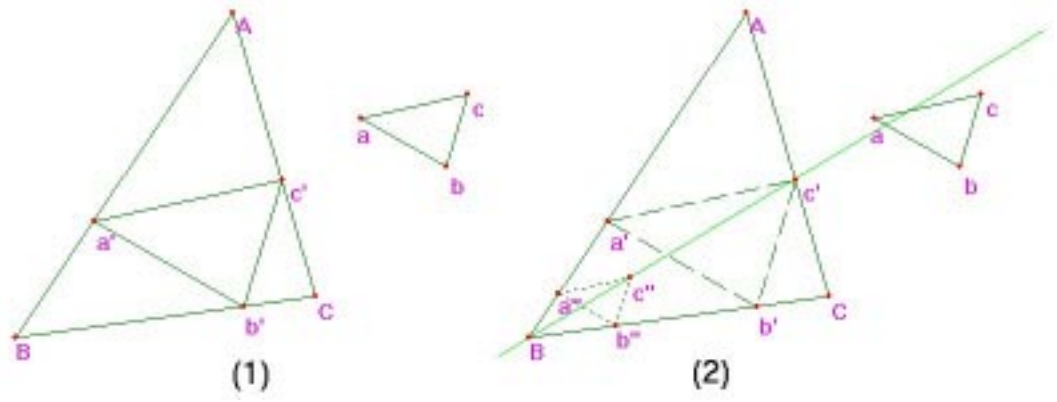

FIG. *7.6* – Triangle inscrit.

La figure 7.6(1) illustre le résultat attendu. Comme le montre la figure 7.6(2), une manière d'arriver à ce résultat consiste à déterminer le lieu du point c' si celui-ci n'a plus la propriété d'appartenir au segment $[AC]$. A l'aide de ce lieu, la construction du triangle se résume à considérer le point d'intersection du segment $[AC]$ et du lieu de c', puis à construire successivement à partir de cette intersection les parallèles au triangle de référence.

– *Monde non clos non linéaire.* Cette classe de problèmes couvre tous les autres problèmes de construction de figures géométriques. Plus particulièrement, cette classe contient tous les problèmes difficiles comme les problèmes dont la spécification est cyclique (cf exemple 1.6 page 31), et les problèmes auxquels, par réalisme, nous ne nous attaquons pas comme par exemple la construction d'un polygone régulier.

La figure 7.7 illustre cette classification des problèmes de construction des figures géométriques.

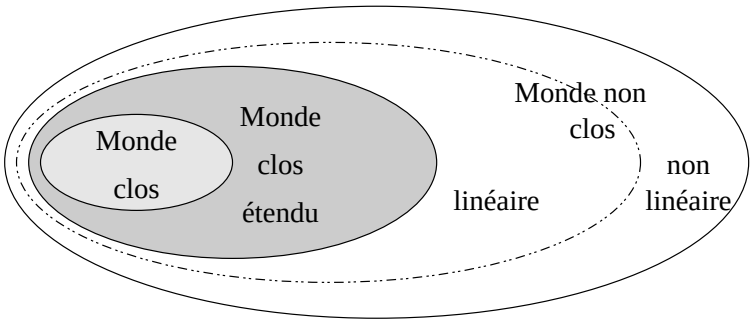

FIG. 7.7 – *Les classes des problèmes de construction.*

7.6.3 Problématique de l'agent règle et compas

Nous exposons dans les paragraphes suivants la problématique d'un agent règle et compas.

Problématique des spécifications surcontraintes

Une spécification d'un problème de construction est surcontrainte dans les cas suivants :

- elle contient des propriétés redondantes. Un exemple simple consiste à considérer la spécification d'un segment et de son point milieu dans laquelle ce dernier est spécifié comme appartenant au segment.

  Face à une telle situation, une première attitude peut consister à retirer dans un premier temps la surcontrainte et à chercher à prouver la validité de cette dernière dans un deuxième temps. Notre approche est différente. Elle consiste à ignorer les surcontraintes. En effet, la recherche d'un plan de construction s'effectue toujours avec comme précondition la présence d'un modèle de la figure respectant la spécification de cette dernière.

- elle est "rigidifiée" par l'instanciation d'un trop grand nombre de points de base. Un exemple simple consiste à considérer une droite passant par trois points distincts instanciés.

  De telles figures ne sont pas constructibles au sens de Cabri-Géomètre, et la recherche d'un plan de construction de ces dernières conduit à un échec.

Problématique des constructions multiples

A l'occasion de l'établissement d'un plan de construction, il peut arriver qu'un objet puisse être construit de plusieurs manières différentes. Il se pose alors le problème de choisir de quelle manière construire de tels objets en sachant quoi qu'il en soit que ces différentes séquences conduisent à la même figure dynamique. Pour résoudre ce problème, il nous faut définir au préalable un critère de choix. Principalement deux points de vue peuvent être considérés.

- la simplicité de construction. Ce point de vue fait référence à la relation de dépendance à privilégier pour construire un objet. Par exemple, dans le cas de l'exercice 7.5 page 131 il n'est pas évident de savoir si la construction du point I comme point d'intersection des droites d et d' est plus simple que la construction du point I comme point milieu des points M_1 et M_2. Cet exemple montre clairement que ce critère reste très subjectif et par conséquent difficile à appréhender. De plus, il peut être considéré d'une manière locale ou globale.

- la rapidité de construction. Ce point de vue fait référence au nombre d'opérations sous-jacentes nécessaires à la construction de la figure. Par exemple, comme la détermination des coordonnées d'un point milieu implique un nombre de calculs inférieur à la détermination d'un point à l'intersection de deux cercles, il s'ensuit que construire un point milieu est plus rapide que construire un point d'intersection de deux cercles.

Notre approche est de privilégier la rapidité de construction. Ce choix a été motivé par le souci d'une animation de qualité, et donc par conséquent rapide.

7.6.4 Exemple de recherche d'un plan de construction

Afin d'illustrer cette recherche d'un plan de construction, nous considérons le problème suivant.

Exemple 7.8 RECHERCHE D'UN PLAN DE CONSTRUCTION

Construire un point A (resp. un point B) sur une droite d (resp. une droite d') donnée tel qu'un point I donné soit le milieu du segment AB. De plus, nous considérons la droite d'' symétrique de la droite d' par rapport au point I.

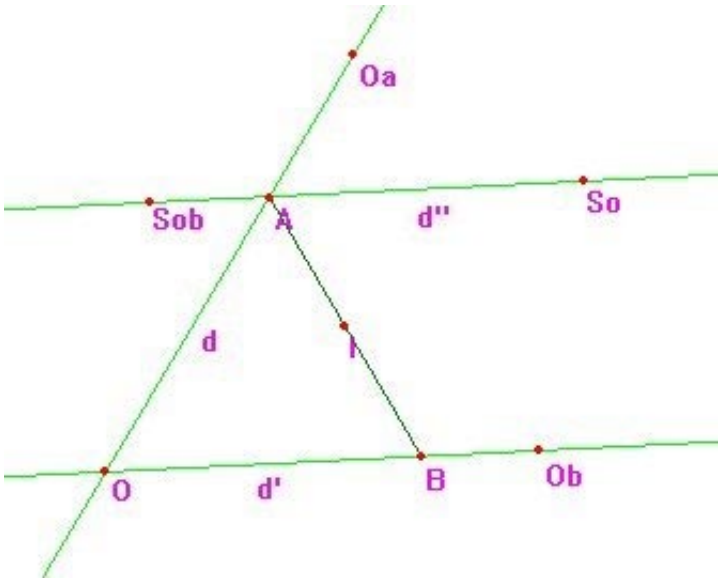

FIG. 7.8 – Recherche d'un plan de construction.

Dans le langage ELDL, la spécification logique géométrique répondant à cet énoncé est la suivante :

$$point(A) \wedge\ point(B) \wedge\ point(O) \wedge\ point(I) \wedge$$
$$point(O_a) \wedge\ point(O_b) \wedge\ point(S_{ob}) \wedge\ point(S_o) \wedge$$
$$droite(d) \wedge\ droite(d') \wedge\ droite(d'') \wedge\ segment(s_{ab}, A, B, I_{ab}) \wedge$$
$$appDr(O, d') \wedge\ appDr(O_b, d') \wedge\ appDr(O, d) \wedge\ appDr(O_a, d) \wedge$$
$$milieu(I, O, S_o) \wedge\ milieu(I, O_b, S_{ob}) \wedge\ appDr(S_o, d'') \wedge\ appDr(S_{ob}, d'') \wedge$$
$$milieu(I, A, B) \wedge\ appDr(A, d) \wedge\ appDr(B, d').$$

La figure 7.8 illustre un dessin, satisfaisant cette spécification, obtenu à partir de la donnée des points O, O_b, O_a et I.

La recherche d'un plan de construction de cette figure en la supposant appartenir au monde clos s'arrête sans avoir construit les points A et B par manque de propriétés. La recherche de propriétés redondantes met en évidence l'appartenance du point A à la droite d''. L'ajout de cette propriété permet alors de construire le point A comme le point d'intersection des droites d et d'', et enfin de construire le point B comme le symétrique du point A par rapport au point I. Ainsi, un plan de construction de cette

figure dynamique est :

1. *Créer le point O,*
2. *Créer le point O_a,*
3. *Créer le point O_b,*
4. *Créer le point I,*
5. *Construire la droite d passant par le point O et le point O_a,*
6. *Construire la droite d' passant par le point O et le point O_b,*
7. *Construire le point S_o symétrique du point O par rapport au point I,*
8. *Construire le point S_{ob} symétrique du point O_b par rapport au point I,*
9. *Construire la droite d'' passant par le point S_o et le point S_{ob},*
10. *Construire le point A à l'intersection de la droite d et de la droite d''.*
11. *Construire le point B symétrique du point A par rapport au point I,*

Grâce à l'introduction de la droite d'', nous avons pu établir un plan de construction de cet exemple dont la résolution numérique est simple. Sans cette introduction, cet exercice est caractéristique d'un problème du monde non clos linéaire. Cet droite d'', qui est le lieu du point A lorsque celui-ci est privé de sa propriété d'appartenance à la droite d, a permis de rendre cet exercice de construction caractéristique d'un problème du monde clos étendu.

Chapitre 8

Modèle de coopération

La problématique des approches coopératives réside dans la manière de faire coopérer au mieux un ensemble de composants afin d'une part de résoudre au plus vite un problème et d'autre part de permettre, par l'association de résolutions partielles, la résolution de problèmes auxquels aucun des composants ne peut s'attaquer seul. Pour ce faire, il s'agit de définir les possibles échanges d'informations de manière synchrone ou asynchrone entre les différents composants. Pour formaliser de tels problèmes de communication et de synchronisation, de nombreux paradigmes ont été proposés comme par exemple celui de la programmation concurrente avec contraintes [Sar93].

Nous présentons dans ce chapitre une formalisation des interactions des différents agents présentés dans le chapitre précédent. Le paradigme de la programmation concurrente avec contraintes nous apparaît adapté pour cette formalisation en raison d'une part de son haut niveau d'expressivité logique, et d'autre part de la méthodologie associée guidant une mise en œuvre correcte grâce à la primitive *freeze* (suspension de processus). Nous rappelons dans la section 8.1 les principes de la programmation concurrente avec contraintes. Puis dans la section 8.2, nous modélisons à l'aide de ce paradigme notre approche de la résolution de contraintes géométriques. La section 8.3 aborde le problème de la terminaison d'un calcul dans une telle approche coopérative. Enfin, nous esquissons dans la section 8.4 une définition de la classe des figures constructibles à l'aide de GDRev.

8.1 Programmation concurrente avec contraintes

Les paragraphes suivants sont consacrés à la description successive des principes de la programmation concurrente avec contraintes et de la syntaxe de tels programmes.

8.1.1 Principes de la programmation concurrente avec contraintes

Le paradigme de la programmation concurrente avec contraintes [Sar93] est né du croisement des paradigmes déclaratifs que sont la programmation logique concurrente [Sha89] et la satisfaction de contraintes. L'idée à la base de cette fusion est d'une part de généraliser les concepts de la programmation logique concurrente et d'autre part d'introduire des mécanismes de contrôle pour la satisfaction de contraintes.

L'approche concurrente avec contraintes s'articule autour d'un ensemble d'*agents* et d'un système de contraintes commun à tous les agents appelé *store*. Une propriété importante de ce dernier est que les contraintes qu'il contient s'y accumulent de manière monotone. Les agents sont des processus indépendants participant à la résolution des buts de la requête courante. Ils communiquent et se synchronisent entre eux au travers de variables partagées dans le store. Les primitives de communication et de synchronisation sont les primitives *tell* et *ask*.

La primitive *tell* est de la forme: *tell(c)* où c est une contrainte. Une requête *tell(c)* réussit et ajoute au store la contrainte c si cette dernière est consistante avec le store. Elle échoue dans le cas contraire. Cette primitive de communication possède un caractère asynchrone dans la mesure où l'exécution d'un but *tell(c)* par un agent est indépendante des autres agents.

La primitive *ask* est de la forme: $ask(X_1, \ldots, X_n)c$ où c est une contrainte, et où $X_1, \ldots, X_n$ est un ensemble de variables apparaissant dans c. Une requête $ask(X_1, \ldots, X_n)c$ réussit et ajoute au store la contrainte c, si celle-ci est impliquée par le store. Elle échoue si la contrainte c est inconsistante avec le store. Elle est suspendue jusqu'à décision si la contrainte c est consistante avec le store sans qu'elle soit impliquée par lui. Cette primitive de communication possède un caractère synchrone dans la mesure où un agent est "dégelé" dès lors qu'un autre agent contraint suffisamment le store pour que la contrainte en attente soit impliquée ou réfutée.

8.1.2 Syntaxe des programmes concurrents avec contraintes

Le tableau 8.1 détaille la syntaxe d'un programme concurrent avec contraintes. Nous indiquons ici informellement la sémantique de chacune des règles ; une description détaillée de la sémantique opérationnelle de celles-ci peut être trouvée dans l'ouvrage [Sar93].

programme	P	::=	{K. A}
clause	K	::=	K.K | g::D
corps de clause	D	::=	X^D | B ⇒ A | B → A
action	B	::=	B : B | c | tell(c) | ask(X_1, ..., X_n) c | A
agent (ou but)	A	::=	A, A | stop | fail | g | D | P

TAB. 8.1 – *Syntaxe d'un programme concurrent avec contraintes.*

Un programme P est constitué de clauses K et d'agents (de buts) A. Les agents A sont les buts initiaux à exécuter dans le contexte des déclarations K. Une clause est de la forme *g::D* où g est la tête de la clause, différente de *true* et *false*, et D est le corps de celle-ci. Un exemple de programme concurrent avec contraintes est donné par l'exemple 8.1.

Exemple 8.1 PROGRAMME CONCURRENT AVEC CONTRAINTES

Il s'agit de définir la clause min(A, B, C) qui est vrai si et seulement si C est le minimum de A et B.

min(A, B, C) :: A ≤ B : tell(C = A) → stop
min(A, B, C) :: A > B : tell(C = B) → stop

TAB. *8.2* – Programme min/3.

Le tableau 8.2 fournit une écriture de ce programme.

L'expression $X\hat{}D$ permet de définir une variable locale X au corps d'une clause D (quantifiée de manière existentielle). Pour définir un ensemble de variables locales

au corps d'une clause, il suffit d'appliquer itérativement cette règle. Par la suite, nous adoptons la notation $(X_1, \ldots, X_n)\hat{}D$ à la place de $X_1\hat{} \ldots X_n\hat{}D$. Le corps d'une clause contient toujours une opération d'engagement (*commitment operation*). Celle-ci permet de contrôler les points de choix faisant suite à l'effacement d'un agent. Principalement deux opérations d'engagement ont été définies dans le cadre de la programmation concurrente avec contraintes. Celles-ci sont d'une part l'opération $B \rightarrow A$ (*don't care commitment operation*) qui ne permet pas d'explorer le sous-arbre de recherche ayant pour racine A, mais seulement la branche de celui-ci dans laquelle il s'est engagé et ceci quelque soit le résultat de l'exploration de cette branche, et d'autre part l'opération $B \Rightarrow A$ (*parallel don't know commitment operation*) qui permet d'explorer en parallèle l'ensemble des branches liées à un point de choix. Pour la suite, nous adoptons la notation $_{\rightarrow}A$ pour $true \rightarrow A$, et de même la notation $_{\Rightarrow}A$ pour $true \Rightarrow A$. L'exemple 8.2 illustre la différence entre ces opérations d'engagement.

Exemple 8.2 OPÉRATIONS D'ENGAGEMENT

Soit la clause, illustrée par le tableau 8.3, représentant une base en données associant à chaque arbre la famille à laquelle il appartient.

$arbres(X,\ Y) :: tell(X = sapin,\ Y = conifère) \Rightarrow stop$
$arbres(X,\ Y) :: tell(X = épicéa,\ Y = conifère) \Rightarrow stop$
$\vdots$
$arbres(X,\ Y) :: tell(X = bouleau,\ Y = bétulacées) \Rightarrow stop$
$arbres(X,\ Y) :: tell(X = charme,\ Y = bétulacées) \Rightarrow stop$

TAB. *8.3* – Illustration de l'opération d'engagement.

L'exécution du programme $\{K.(X,Y)\hat{}tell(Y = conifère) : arbres(X,Y)\}$*, où* K *est le prédicat arbres(X, Y) décrit dans le tableau 8.3, conduit à l'énumération successives de l'ensemble des conifères répertoriés dans la base de données.*

Le choix de l'opération d'engagement $\rightarrow$ *(don't care commitment) dans le prédicat arbres(X, Y) à la place de l'opération d'engagement* $\Rightarrow$ *(parallel don't know commitment) conduit le même programme à établir le nom de un et de un seul conifère.*

Une action peut être une composition séquentielle d'actions grâce à l'opérateur ":". Cet opérateur permet de retarder l'exécution de l'action B_2 de la composition $B_1 : B_2$ jusqu'à ce que l'action B_1 réussisse. Elle peut aussi être un *tell(c)* ou un $ask(X_1, \ldots, X_n)c$ que nous avons précédemment décrit. Une action peut consister en

une contrainte c. L'exécution de cette action réussit si le store implique la contrainte. Elle est en attente si la contrainte est consistante avec le store sans qu'elle soit impliquée par lui. Elle échoue si elle est inconsistante avec le store. La différence qui sépare l'action réduite à la contrainte c de l'action $ask(X_1, \ldots, X_n)c$ est que cette dernière ajoute au store la contrainte c, alors que la première ne le fait pas[1]. Enfin, une action peut être l'exécution d'un agent A.

La virgule est l'opérateur de composition parallèle en programmation concurrente avec contraintes. Ainsi, l'agent A_1, A_2 se comporte comme les agents A_1 et A_2 exécutés en parallèle. L'agent *stop* conduit à un succès et l'agent *fail* à un échec. L'agent g est un appel à une clause (une procédure). Un agent peut être une composition d'agents D. Enfin, afin de supporter la notion de structure de bloc, un agent peut être un programme P.

8.2 Modèle de coopération de GDRev

L'architecture que nous adoptons est basée sur le paradigme de la programmation concurrente avec contraintes [Sar93]. Suivant cette approche, cette architecture s'articule autour des six agents présentés dans le chapitre 7 et d'un système de contraintes commun à ces derniers. Les différents agents tentent de construire tout ou partie de la figure. A cette fin, ils coopèrent entre eux, ce qui se manifeste au travers d'échanges d'informations au moyen de variables partagées contenues dans le store.

Dans les paragraphes suivants, nous commençons par décrire le système de contraintes sur lequel s'appuyent les résolveurs de GDRev. Puis, nous formalisons l'architecture générale des différents agents, ainsi que leurs interactions d'abord dans un modèle qui ne rend pas compte de priorités de déclenchement et qui est affiné ensuite. Enfin, nous détaillons successivement les communications des agents linéaire, quadratique, intervalle, complétion d'objets, complétion de propriétés, et règle et compas.

8.2.1 Système de contraintes de GDRev

Le système de contraintes commun aux agents de GDRev porte sur les domaines des rationnels, des intervalles, des arbres et des listes. Nous notons les opérateurs

1. La primitive $ask(X_1, \ldots, X_n)c$ peut être vue comme une forme abrégée de la séquence d'actions $(X_1, \ldots, X_n)^\wedge c : tell(c)$

du domaine des rationnels +, -, $\times$ et /, et les opérateurs du domaine des intervalles $\oplus, \ominus, \otimes$ et $\oslash$. La notation Edinbourg est utilisée pour représenter les listes. De plus sur le domaine des listes (resp. des arbres), nous considérons l'opérateur "." qui retourne le $n^{ième}$ élément (resp. fils) de la liste (resp. de l'arbre). Enfin sur le domaine des arbres, nous associons à l'opérateur unaire @ la fonction qui retourne l'étiquette associée à la racine de l'arbre considéré.

Pour formaliser le modèle de coopération des agents de GDRev, nous introduisons le prédicat *free(X)* qui est vérifié si et seulement si X n'est pas "connu", et le prédicat *know(X)* qui inversement est vérifié si et seulement si X est connu. Ainsi, nous considérons qu'une liste de taille non fixe est connue si elle est réduite à la liste vide, ou si elle contient au moins un élément.

Initialement, le store contient la liste d'atomes *LObjs* décrivant les objets de la figure et la liste d'atomes *LProps* décrivant les propriétés qui les lient entre eux.

Les atomes décrivant les objets de la figure sont des arbres binaires dont l'étiquette associée à la racine est à valeur dans {point, droite, demiDroite, segment, cercle et distance}. Le fils gauche est l'identificateur de l'objet, et le fils droit est une liste contenant les paramètres de l'objet. Ces paramètres sont dans l'ordre pour :

- un point : son abscisse et son ordonnée.
- une droite : les coefficients m et p de l'équation $y = m \times x + p$.
- une demi-droite : les identificateurs de l'origine et de la droite support.
- un segment : les identificateurs des extrémités et de la droite support.
- un cercle : les identificateurs du centre et du rayon.
- une distance : la mesure de celle-ci.

Ainsi, la contrainte :

- $O = LObjs.i$ associe à O le $i^{ième}$ objet décrit dans la figure.
- $T = LObjs.i@$ associe à T le type, à valeur dans {point, droite, demiDroite, segment, cercle et distance}, du $i^{ième}$ objet décrit dans la figure.
- $Id = LObjs.i.1$ associe à Id l'identificateur du $i^{ième}$ objet décrit dans la figure.

- $Def = LObjs.i.2$ associe à Def la liste des paramètres du $i^{ième}$ objet décrit dans la figure.
- $P = LObjs.i.2.j$ associe à P le $j^{ième}$ paramètre du $i^{ième}$ objet décrit dans la figure.

Les atomes décrivant les propriétes qui lient les objets entre eux sont les arbres binaires ou ternaires détaillés dans le tableau 4.1.

Les listes *LProps* et *LObjs* sont pourvues d'un suffixe libre afin d'autoriser l'ajout d'objets et de propriétés. Nous assurons ici que les atomes de propriété ajoutés à la liste *LProps* sont des termes clos et que les atomes de typage ajoutés à la liste d'objets *LObjs* sont des termes ayant deux arguments dont le premier est instancié et dont le deuxième est une liste ayant un nombre fini d'éléments.

L'exemple 8.3 illustre le contenu initial du store dans le cadre d'un exercice de construction.

Exemple 8.3 CONTENU INITIAL DU STORE

Construire un triangle rectangle et isocèle en A à partir des sommets B de coordonnées (18.5, 34.8) et C de coordonnées (-14.2, 78.3) du triangle.

Le contenu initial du système de contraintes commun aux agents de GDRev relatif à la résolution de cet exercice se résume aux listes[2] *:*

$$LProps = [appDr(a,ab),\ appDr(a,ac),\ appDr(b,ab),\ appDr(b,bc),\ appDr(c,ac),\ appDr(c,bc),\ perp(ab,ac), distPP(r,a,b),\ distPP(r,a,c) \mid \_]$$

$$LObjs = [point(a,[\_Xa,\_Ya]),\ point(b,[18.5,34.8]),\ point(c,[-14.2,78.3]),\ droite(ab,[\_Mab,\_Pab]),\ droite(ac,[\_Mac,\_Pac]),\ droite(bc,[\_Mbc,\_Pbc]),\ distance(r,[\_Dr]) \mid \_]$$

8.2.2 Architecture et interactions des différents agents

Tous les agents sont invoqués avec les listes *LProps* et *LObjs* comme arguments. Ils accèdent successivement aux atomes de la liste *LProps* afin de les évaluer.

La structure globale de l'agent linéaire (resp. quadratique, intervalle et complétion d'objets) peut être décrite par le prédicat *agentLin(LProps, LObjs)* (resp. *agentQuad(LProps, LObjs)* pour l'agent quadratique, *agentInt(LProps, LObjs)* pour l'agent

2. *La notation _Xx représente une variable.*

intervalle, *agentCptObjs(LProps, LObjs)* pour l'agent complétion d'objets). Le prédicat *agentXxx* donné dans le tableau 8.4 décrit le comportement similaire de tous les agents mentionnés ci-dessous, dans lequel *evaluerAtomeXxx(Prop, LObjs)* est propre à chaque agent.

Ces agents sont décrits dans les paragraphes 8.2.4, 8.2.5, 8.2.6 et 8.2.7.

```
agentXxx(LProps, LObjs)::
    (Prop, SProps)^ know(LProps):
      tell(LProps = [Prop | SProps]):
      (
        evaluerAtomeXxx(Prop, LObjs),
        agentXxx(SProps, LObjs)
      )
```

TAB. 8.4 – *Architecture générale d'un agent.*

Une formulation du programme dans son ensemble est donnée par le tableau 8.5, où la variable *LPlan* est une liste dont les éléments décrivent dans l'ordre les étapes d'un plan de construction de la figure. Dans cette formulation, les agents sont exécutés en parallèle en l'absence de contrôle, c'est-à-dire sans qu'aucune priorité ne soit donnée dans leur déclenchement.

A ce propos, les préférences qui ont été formulées en matière de construction dans le chapitre 1 doivent être rappelées. Celles-ci penchent en premier lieu vers les approches symboliques qui fournissent les solutions les plus propres à l'animation, puis numériques exactes, et enfin numériques approchées du moment que le dessin correspondant soit acceptable.

Il s'ensuit que la formulation 8.5 est incorrecte, car elle ne rend pas compte de ces préférences. En effet dans la mesure où les différents agents sont exécutés en parallèle sans indication de priorité, l'agent intervalle peut par exemple intervenir en tout premier lieu alors que nous souhaitons l'invoquer en dernier recours.

Ainsi, suivant les préférences établies, il s'agit d'exécuter l'agent règle et compas, puis dans un ordre quelconque les agents linéaire, quadratique, complétion d'objets et complétion de propriétés, et enfin l'agent intervalle. On doit noter que l'établissement d'un ordre d'exécution pour les agents linéaire, quadratique, complétion d'objets et complétion de propriétés peut conduire à l'obtention de figures spécifiques. Par

```
{
  ... Déclarations des clauses.
  .
  (LProps, LObjs, LPlan)^
    tell(LProps = [...Spécifications des propriétés de la figure... | _]:
    tell(LObjs = [...Spécifications des objets de la figure... | _]:
    (
      agentLin(LProps, LObjs),
      agentQuad(LProps, LObjs),
      agentInt(LProps, LObjs),
      agentCptObjs(LProps, LObjs),
      agentCptProps(LProps, LObjs),
      agentRC(LProps, LObjs, LPlan)
    )
}
```

TAB. 8.5 – *Architecture générale du programme de résolution de contraintes géométriques.*

exemple, les deux ordres d'exécution suivants produisent des figures différentes.

- L'ordre complétion d'objets, puis complétion de propriétés, puis linéaire et enfin quadratique permet d'obtenir des figures introduisant un minimum d'extensions algébriques, c'est-à-dire dont la résolution a conduit à un minimum de calculs de racines carrées.

- L'ordre linéaire, puis quadratique, puis complétion d'objets et enfin complétion de propriétés permet d'obtenir plus rapidement en générale une figure que dans les autres cas.

Dans la suite, nous ne tenons pas compte des agents complétion de propriétés et règle et compas en considérant leur exécution comme résultant à l'heure actuelle d'une demande explicite de l'utilisateur. Toutefois, il serait intéressant et, nous le pensons aisé et rapide, d'intégrer ces deux agents dans le schéma de coopération.

8.2.3 Architecture et interactions des différents agents respectant les priorités de déclenchement

Pour exprimer une notion d'ordre, une première idée consiste à utiliser l'opérateur de séquentialité ":" à la place de l'opérateur de parallélisme "," dans le corps de la requête. Malheureusement, une telle formulation conduirait très souvent à une situation de blocage. En effet, l'exécution de $B_1 : B_2$ conduit à un blocage notamment si B_1 se bloque. Ainsi, si par exemple l'évaluation de certaines propriétés par un agent sont dans un état de blocage, il s'ensuit que l'agent suivant n'est pas exécuté. Par conséquent, l'emploi de l'opérateur de séquentialité est valide seulement dans les situations où le premier agent exécuté réussit à construire entièrement et tout seul la figure.

En fait, pour exprimer l'ordre d'exécution des agents il s'agit de faire en sorte que l'exécution d'un agent soit déclenchée par son prédécesseur une fois que celui-ci a effectivement évalué l'ensemble[3] des atomes de propriété décrivant la figure. Cette approche nous conduit au schéma d'exécution illustré par la figure 8.4.

Ce schéma illustre le fait que l'agent linéaire relance ses buts en attente tant qu'il arrive à évaluer des contraintes. Une fois bloqué, il (re)lance l'agent quadratique. Celui-ci relance l'agent linéaire s'il a su évaluer une contrainte, ou l'agent complétion d'objets dans le cas contraire. D'une manière similaire, l'agent complétion d'objets relance l'agent linéaire s'il a su déterminer une corde commune ou une médiatrice, ou l'agent intervalle sinon. Enfin, l'agent intervalle relance l'agent linéaire s'il a su instancier une variable, ou conduit à un succès.

Pour formaliser un tel mécanisme, nous associons à chaque agent une liste pourvue d'un suffixe libre (*_FreesLin* pour l'agent linéaire, *_FreesQuad* pour l'agent quadratique, *_FreesCptObjs* pour l'agent complétion d'objets, *_FreesInt* pour l'agent intervalle). Ces listes sont regroupées dans une liste *_Frees*. Elles ont un double rôle. Elles permettent :

1. de gérer le processus de résolution en bloquant un agent, ou les buts d'un agent, en attente d'informations supplémentaires sur le suffixe libre de la liste lui correspondant dans l'argument *_Frees*. Pour ce faire nous avons défini le prédicat *gelerAgent(_Id, _LProps, _LObjs, _Frees)* où l'argument *_Id* est l'identificateur de l'agent à geler.

   Libérer un agent revient à spécifier que la variable (le suffixe libre) sur laquelle

3. Nous allons voir plus loin que ceci n'est pas tout à fait exact s'agissant des agents quadratique et complétion d'objets.

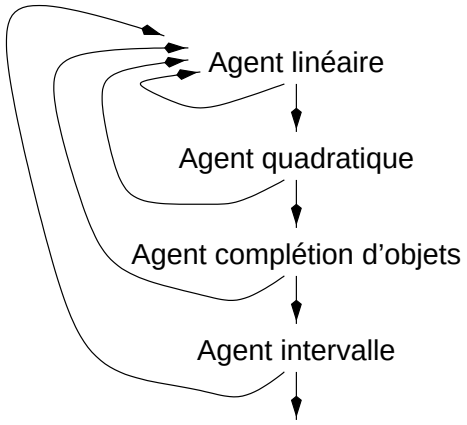

FIG. 8.4 – *Schéma de contrôle des agents.*

il est en attente d'informations supplémentaires est une liste ayant au moins un élément. Pour ce faire, nous avons défini le prédicat *libererAgent(_Id, _Frees)* où l'argument *_Id* est l'identificateur de l'agent à libérer.

2. d'"espionner" l'ajout de contraintes au système de contraintes courant par un agent en examinant la valuation du dernier élément de la liste lui correspondant. Si cet élément est une variable, cela signifie que l'agent correspondant n'a pas posé de contraintes. Si cet élément est instancié à la valeur 1, alors l'agent correspondant a réussi à poser une contrainte.

   Pour marquer qu'un agent a réussi à poser une contrainte, nous avons défini le prédicat *marquerTellContraintes(_Id, _Frees)* où l'argument *_Id* est l'identificateur de l'agent.

   Pour déterminer si un agent a (resp. n'a pas) réussi à poser une contrainte nous avons défini le prédicat *tellContraintes(_Id, _Frees)* (resp. *nonTellContraintes(_Id, _Frees)*) où l'argument *_Id* est l'identificateur de l'agent "espionné".

Il reste le problème de pouvoir déterminer si un agent a effectivement évalué tous les atomes de la liste des propriétés, dans la mesure où l'évaluation d'un atome et le parcours des atomes de la liste des propriétés se fait en parallèle. Pour résoudre ce problème, une idée consiste à comptabiliser pour chaque agent d'une part les propriétés qu'il a évaluées, et d'autre part le nombre de ses appels récursifs. Un agent peut être libéré par son prédécesseur (cf figure 8.4) si le nombre des propriétés évaluées par ce dernier correspond au nombre de ses appels récursifs avec la condition supplémentaire que la liste des propriétés à évaluer est réduite à un suffixe libre. Une manière de définir

un compteur est de construire une liste pourvue d'un suffixe libre et de considérer que le nombre d'éléments de celle-ci correspond à la valeur du compteur défini. Ainsi, pour comptabiliser d'une part le nombre des propriétés évaluées par un agent et d'autre part le nombre de ses appels récursifs, on construit deux listes de ce type appelées respectivement *In* et *Out*.

```
agentLin(LProps, LObjs, In, Out, Frees)::
    figureNonConstruite(LObjs):
    free(LProps):
    conditionLiberation(In, Out) →
      etablirBilanLin(LObjs, Frees)

agentLin(LProps, LObjs, In, Out, Frees)::
    (Prop, SProps)^ know(LProps):
      tell(LProps = [Prop | SProps]):
      marquerListe(Out):
      (
        evaluerAtomeLin(Prop, LObjs, In, Frees),
        agentLin(SProps, LObjs, In, Out, Frees)
      )

etablirBilanLin(LObjs, Frees)::
    estFigureConstruite(LObjs):
      terminerProcessus(Frees) → stop

etablirBilanLin(LObjs, Frees)::
    estFigureNonConstruite(LObjs):
    tellContraintes(lin, Frees):
    libererAgent(lin, Frees) →
      etablirBilanLin(LObjs, Frees)

etablirBilanLin(LObjs, Frees)::
    estFigureNonConstruite(LObjs):
    nonTellContraintes(lin, Frees):
    libererAgent(quad, Frees) →
      etablirBilanQuad(LObjs, Frees)
```

TAB. 8.6 – *Structure générale de l'agent linéairesous contrôle.*

La formalisation de cette idée est donnée pour l'agent linéaire dans le tableau 8.6. Le prédicat *conditionLiberation(In, Out)*, donné dans le tableau 8.7, est vrai si et seulement si les listes pourvues d'un suffixe libre *In* et *Out* ont le même nombre d'élé-

ments. Le prédicat *marquerListe(L)*, illustré par le tableau 8.8, est vrai si le suffixe libre de la liste L est une liste ayant au moins un élément instancié à 1. Le prédicat *estFigureConstruite(LObjs)* est vrai si et seulement si tous les objets de la figure contenus dans la liste *LObjs* sont construits. Inversement, le prédicat *estFigureNonConstruite(LObjs)* est vrai si et seulement si il existe au moins un objet de la figure qui ne soit pas construit. Enfin, pour achever le processus de construction, nous considérons le prédicat *terminerProcessus(Frees)* qui libère tous les processus en attente afin d'assurer une terminaison sans processus en attente.

```
conditionLiberation(In, Out) ::
    free(Out) → stop

conditionLiberation(In, Out) ::
    (SOut, SIn)^ (know(Out), know(In)) →
      tell(Out = [_ | SOut], In = [_ | SIn]) →
        conditionLiberation(SIn, SOut)
```

TAB. 8.7 – *Condition de libération de l'agent successeur.*

On peut remarquer que cette formulation a conduit à ajouter à la clause *evaluerAtomeLin* d'une part un argument *In*, c'est-à-dire un argument permettant de comptabiliser le nombre d'atomes évalués, et d'autre part l'argument *Frees*, c'est-à-dire la liste des listes de contrôle des agents afin de (re)positionner en attente l'évaluation d'une contrainte pseudo-linéaire.

```
marquerListe(L) ::
    free(L) →
      tell(L = [1 | _]) → stop

marquerListe(L) ::
    know(L) →
      (SL)^ L = [1 | SL] →
        marquerListe(SL)
```

TAB. 8.8 – *Clause marquerListe/1.*

La formalisation du contrôle de l'agent quadratique peut être faite d'une manière similaire à celle de l'agent linéaire. Toutefois, afin de réduire le nombre des racines

carrées calculées, il est bloqué une fois qu'il a évalué une contrainte quadratique. Ceci est réalisé grâce aux prédicats décrits dans le tableau 8.9.

```
agentQuad(LProps, LObjs, In, Out, Frees) ::
    free(LProps) →
      libererAgent(cptObjs, Frees)

agentQuad(LProps, LObjs, In, Out, Frees) ::
    (Prop, SProps)^ know(LProps) :
      tell(LProps = [Prop | SProps]) :
      marquerListe(Out) :
      (
        evaluerAtomeQuad(Prop, LObjs, In, Frees),
        poursuivreAgentQuad(SProps, LObjs, In, Out, Frees)
      )

poursuivreAgentQuad(LProps, LObjs, In, Out, Frees) ::
    conditionLiberation(In, Out) :
    tellContraintes(quad, Frees) :
    (
      libererAgent(lin, Frees),
      gelerAgent(quad, LProps, LObjs, In, Out, Frees)
    )

poursuivreAgentQuad(LProps, LObjs, In, Out, Frees) ::
    conditionLiberation(In, Out) :
    nonTellContraintes(quad, Frees) →
      agentQuad(LProps, LObjs, In, Out, Frees)
```

TAB. 8.9 – *Structure générale de l'agent quadratique sous contrôle.*

Les cordes communes et/ou médiatrices de la figure peuvent être recherchées et ajoutées une à une suivant un schéma de contrôle semblable à celui de l'agent quadratique ou une seule fois suivant un schéma de contrôle semblable à l'agent linéaire. Afin de privilégier la rapidité de construction, nous avons choisi de les ajouter une à une. En effet, la complexité de l'algorithme de l'agent complétion d'objets est en $\mathcal{O}(n^2)$ où n est le nombre d'atomes de propriété de la figure. Ainsi, en les ajoutant une à une, on espère limiter ce coût.

Enfin, le schéma de contrôle de l'agent intervalle est semblable à celui de l'agent linéaire à la différence qu'il ne libère aucun agent après lui.

Il s'ensuit qu'une formulation du programme dans son ensemble satisfaisant les préférences établies en matière de construction peut être celle décrite dans le tableau 8.10.

```
{
   ... Déclarations des clauses.
   .
   (LProps, LObjs)^
      tell(LProps = [...Spécifications des propriétés de la figure... | _] :
      tell(LObjs = [...Spécifications des objets de la figure... | _] →
      (FreeLin, InLin, OutLin,
      FreeQuad, InQuad, OutQuad,
      FreeCptObjs, InCptObjs, OutCptObjs,
      FreeInt, InInt, OutInt)^
         tell(Frees = [FreesLin, FreesQuad, FreesCptObjs, FreesCptProps]) :
         tell(FreesLin = [_ | _]) →
         (
            (FreesLin = [_ | _] :
            agentLin(LProps, LObjs, InLin, OutLin, Frees)
            ),
            (FreesQuad = [_ | _] :
            agentQuad(LProps, LObjs, InQuad, OutQuad, Frees)
            ),
            (FreesCptObjs = [_ | _] :
            agentCptObjs(LProps, LObjs, InCptObjs, OutCptObjs, Frees)
            ),
            (FreesInt = [_ | _] :
            agentInt(LProps, LObjs, InInt, OutInt, Frees)
            )
         )
}
```

TAB. 8.10 – *Architecture générale du programme de résolution de contraintes géométriques sous contrôle.*

On note que dans cette formulation seule l'exécution de l'agent linéaire (la clause *agentLin*) est "gardée" par une contrainte qui peut être impliquée par le store. Il s'en déduit que cet agent est le seul agent qui puisse être exécuté. Une fois son exécution terminée, celui-ci libère l'agent quadratique au travers de la variable *FreesQuad*, qui lui délivre l'agent complétion d'objets en spécifiant que la variable *FreesCptObjs* est une liste contenant au moins un élément. Enfin, ce dernier libère l'agent intervalle au travers de la variable *FreesInt*.

De la sorte, on aboutit à une formulation respectant les préférences établies, dans laquelle la rapidité de construction est privilégiée vis à vis de la complexité de la figure en terme du nombre de racines carrées calculées.

Avant de modéliser les différents agents et leurs interactions, nous introduisons le prédicat très utile *indexObjet*. Il permet de retrouver l'index d'un objet dans la liste des objets à partir de son identificateur. Sa spécification est telle que *indexObjet(Id, LObjs, Index)* est vérifié si et seulement si *Index* est l'index de l'objet identifié par *Id* dans la liste des objets *LObjs*.

8.2.4 Communications de l'agent linéaire

L'agent linéaire évalue les atomes de propriété suivants : appDr, appDD, appSeg, par, perp, invSens, memeSens et milieu. En présence d'une propriété toujours linéaire, comme par exemple les propriétés par et milieu, l'agent linéaire ajoute directement au store l'équation correspondante par un *tell*. En présence des autres propriétés, dites pseudo-linéaires, c'est-à-dire des propriétés appDr, appDD, appSeg, perp, invSens et memeSens, l'agent linéaire vérifie que celles-ci sont linéaires grâce à un *ask*, plus exactement grâce à la primitive *fix*, avant de les ajouter au store par un *tell*.

Ces dernières vérifications peuvent conduire au gel des équations pseudo-linéaires. Les ajouts éventuels peuvent linéariser une équation pseudo-linéaire et ainsi libérer l'agent linéaire. Ils peuvent aussi réduire une équation du deuxième degré en deux variables à une équation du deuxième degré en une variable, libérant par là même l'agent quadratique. Ils peuvent encore instancier un objet et libérer l'agent intervalle.

```
evaluerAtomeLin(Atome, LObjs, In)::
    estFigureConstruite(LObjs) → stop

evaluerAtomeLin(Atome, LObjs, In)::
    estFigureNonConstruite(LObjs) →
    Atome@ = par → (Ligne1, Ligne2)^
      marquerListe(In):
      indexObjet(Atome.1, LObjs, Ligne1):
      indexObjet(Atome.2, LObjs, Ligne2):
      tell(LObjs.Ligne1.2.1 = LObjs.Ligne2.2.1)
```

TAB. 8.11 – *Traitement d'une contrainte géométrique linéaire (par) par l'agent linéaire.*

Par exemple, à la propriété toujours linéaire par, on associe la clause décrite dans le tableau 8.11.

```
evaluerAtomeLin(Atome, LObjs, In)::
    estFigureNonConstruite(LObjs) →
    Atome@ = appDr → (Pt, Dr)^
      marquerListe(In):
      indexObjet(Atome.1, LObjs, Pt):
      indexObjet(Atome.2, LObjs, Dr):
      poserContrainteLin(LObjs.Pt.2.2, LObjs.Dr.2.1,
                         LObjs.Pt.2.1, LObjs.Dr.2.2, Atome, LObjs, Frees)
```

TAB. 8.12 – *Traitement d'une contrainte géométrique pseudo-linéaire (appDr) par l'agent linéaire.*

Par exemple, à la propriété pseudo linéaire appDr, on associe la clause décrite dans le tableau 8.12 où *lineaire(Id1, Id2, LObjs)* est vrai si et seulement si le produit du premier paramètre de l'objet identifié par *Id1* par le premier paramètre de l'objet identifié par *Id2* est linéaire, c'est-à-dire si l'un ou l'autre des opérandes de l'opérateur de multiplication est connue, et où inversement *pseudoLineaire(Id1, Id2, LObjs)* est vrai si et seulement si le produit du premier paramètre de l'objet identifié par *Id1* par le premier paramètre de l'objet identifié par *Id2* est pseudo-linéaire.

```
evaluerAtomeLin(Atome, LObjs, In)::
    Atome@ = appCc →
      marquerListe(In)
```

TAB. 8.13 – *Traitement d'une contrainte géométrique non linéaire par l'agent linéaire.*

Enfin, en présence d'une propriété autre que celles citées, l'agent linéaire associe une clause du type de celle décrite dans le tableau 8.13.

8.2.5 Communications de l'agent quadratique

L'agent quadratique évalue les deux atomes de propriété suivants : appCc et distPP. Il vérifie que les variables mises en jeu par l'équation du second degré correspondante sont linéairement dépendantes deux à deux par un *ask*, plus exactement à l'aide de la

primitive *fix*, avant d'ajouter au store les racines calculées par un *tell*. Si la résolution de l'équation aboutit à deux solutions, le parcours de l'ensemble des solutions est assuré par l'emploi de l'opérateur d'engagement ⇒ (*parallel don't know commitment*).

Cette vérification peut conduire certaines de ces équations à un état de gel. L'ajout de ces contraintes peut conduire aussi à la linéarisation d'une équation pseudo-linéaire, ou à la réduction d'une équation du second degré en deux variables à une équation du second degré en une variable, ou enfin à la libération de l'agent intervalle.

```
evaluerAtomeQuad(Atome, LObjs, In)::
    estFigureConstruite(LObjs) → stop.

evaluerAtomeQuad(Atome, LObjs, In)::
    estFigureNonConstruite(LObjs) →
    Atome@ = appCc → (Pt, Cc)^
      marquerListe(In):
      indexObjet(Atome.1, LObjs, Pt):
      indexObjet(Atome.2, LObjs, Cc): (Obj, Param, A, B, C, Delta)^
        dependLin(Pt, Cc, LObjs, Obj, Param, A, B, C, Delta) ⇒
          ajouterRacine(Obj, Param, LObjs, A, B, C, Delta)
```

TAB. 8.14 – *Traitement d'une contrainte géométrique non linéaire par l'agent quadratique.*

Par exemple à la propriété appCc on associe la clause décrite dans le tableau 8.14, où le prédicat *dependLin(Pt, Cc, LObjs, Obj, Param, A, B, C, Delta)* est vrai si et seulement si l'équation quadratique d'appartenance du point identifié par *Pt* au cercle identifié par *Cc* est une équation quadratique en la variable *LObjs.Obj.2.Param* aux coefficients *A*, *B* et *C*, et où le prédicat *ajouterRacine(Obj, Param, LObjs, A, B, C, Delta)* ajoute au store la racine de l'équation $A \times x^2 + B \times x + C = 0$ où x est la variable *LObjs.Obj.2.Param*.

```
evaluerAtomeQuad(Atome, LObjs, In)::
    Atome@ = appDr →
      marquerListe(In)
```

TAB. 8.15 – *Traitement d'une contrainte géométrique linéaire ou pseudo-linéaire par l'agent quadratique.*

Enfin, en présence d'une propriété autre que celles mentionnées, on associe une

clause du type de celle décrite dans le tableau 8.15.

8.2.6 Communications de l'agent intervalle

L'agent intervalle considère tous les atomes du langage. Il ajoute successivement toutes les équations correspondantes grâce à des *tell*.

L'ajout de ces contraintes peut si elles aboutissent à une instanciation linéariser une équation pseudo-linéaire et ainsi libérer l'agent linéaire, ou réduire une équation du deuxième degré en deux variables à une équation du deuxième degré en une variable, et ainsi libérer l'agent quadratique.

```
evaluerAtomeInt(Atome, LObjs)::
    Atome@ = appDr → (Pt, Dr)^
      indexObjet(Atome.1, LObjs, Pt):
      indexObjet(Atome.2, LObjs, Dr):
      tell(LObjs.Pt.2.2 = LObjs.Dr.2.1 ⊗ LObjs.Pt.2.1 ⊕ LObjs.Dr.2.2)
```

TAB. 8.16 – *Traitement d'une contrainte géométrique par l'agent intervalle.*

A chaque propriété du langage on associe une clause du type de la clause décrite dans le tableau 8.16. On peut noter que cet agent ne comptabilise pas les atomes qu'il évalue. Ceci n'a pas d'intérêt dans la mesure où ce dernier ne libère aucun agent explicitement pour des raisons d'ordonnancement.

8.2.7 Communications de l'agent complétion d'objets

L'agent complétion d'objets évalue les atomes de propriété du type appCc et distPP. En présence de deux atomes des types précédents faisant intervenir un même point, il détermine la corde commune et/ou médiatrice. Puis, il ajoute aux listes *LObjs* et *LProps* les atomes correspondant à la corde commune construite et aux propriétés la liant aux objets de la figure grâce à des *tell*.

Par ces ajouts, l'agent complétion d'objets libére tous les autres agents.

Par exemple, à la propriété appCc on associe une clause du type de la clause décrite dans le tableau 8.17. Le prédicat *rechercherAutreAppCc(Pt, Cc1, Cc2, LProps, LObjs)* est vérifié si et seulement si la liste des propriétés *LProps* contient la spécification d'appartenance du point identifié par *Pt* au cercle identifié par *Cc2*, différent

```
evaluerAtomeCptObjs(Atome, LProps, LObjs, In)::
    Atome@ = appCc → (Pt, Cc1)^
      marquerListe(In):
      indexObjet(Atome.1, LObjs, Pt):
      indexObjet(Atome.2, LObjs, Cc1): (Cc2)^
      rechercherAutreAppCc(Pt, Cc1, Cc2, LProps, LObjs):
      completerCorde(Pt, Cc1, Cc2, LProps, LObjs)
```

TAB. 8.17 – *Traitement d'une contrainte géométrique par l'agent complétion d'objets.*

du cercle identifié par *Cc1*. L'agent *completerCorde(Pt, Cc1, Cc2, LProps, LObjs)* est vrai si la liste des objets *LObjs* et la liste des propriétés *LProps* ont été complétées respectivement par la corde commune aux cercles *Cc1* et *Cc2* et par la propriété d'appartenance du point *Pt* à cette corde.

8.2.8 Communications de l'agent complétion de propriétés

Pour rechercher des propriétés redondantes, l'agent complétion de propriétés requiert une construction de la figure à partir d'objets de base différents. Les atomes correspondant aux propriétés trouvés sont ajoutées à la liste des atomes de propriétés de la figure grâce à des *tell* dans la mesure où celle-ci ne sont pas déjà présentes. Cette recherche est décrite dans le tableau 8.18.

```
agentCptProps(LProps, LObjs, Frees):: (PropsPoss, Objs, PropsRed)^
    produirePossiblesProp(LProps, LObjs, PropsPoss):
    reconstruireFigure(LProps, LObjs, Objs):
    eliminerContreProps(PropsPoss, Objs, PropsRed):
    ajouterProps(PropsRed, LProps) → stop
```

TAB. 8.18 – *Architecture de l'agent complétion de propriétés.*

Par ces ajouts, l'agent complétion de propriétés peut libérer tous les autres agents.

8.2.9 Communications de l'agent règle et compas

L'agent règle et compas cherche à établir un plan de construction d'une figure en commençant par supposer qu'il en existe un à partir du monde clos, c'est-à-dire que la spécification de la figure contient tous les objets et toutes les propriétés necessaires à la fabrication de ce plan. S'il se trouve dans une impasse, il suppose alors qu'il en existe un à partir du monde clos étendu, c'est-à-dire que la spécification de la figure contient tous les objets nécessaire, mais possiblement pas toutes les propriétés. Dans cette situation, il cherche des propriétés redondantes dans la spécification de la figure afin de ramener la recherche du plan de construction à celle d'un problème dans le monde clos. Dans le cas où il n'arrive toujours pas à établir le plan de construction de la figure, il considère alors un monde non clos. Il cherche des objets dont il arrive à déterminer une construction à la règle et au compas pouvant réduire le problème considéré à un problème de monde clos ou à un problème du monde clos étendu.

8.3 Terminaison

Il est bien sur important de vérifier la terminaison du processus de résolution [Rue94]. Pour vérifier cette terminaison, nous considérons que les résolveurs élémentaires sur lesquels il s'appuie se terminent.

Cette hypothèse étant posée il s'agit de vérifier que :

1. chacun des agents considérés termine étant donné une liste finie de propriétés décrivant les relations liant les éléments d'une figure.

2. dans son ensemble, un processus de résolution ne peut pas aboutir à un cycle sans fin, c'est-à-dire que les listes des objets et des propriétés de la figure ne peuvent pas augmenter sans cesse.

Nous commençons par examiner le premier point. Il est relativement clair que les agents linéaire, quadratique, intervalle, complétion d'objets et règle et compas terminent dans la mesure où ce sont des prédicats récursifs sur une liste finie dont la taille diminue strictement. L'agent complétion de propriété non récursif se termine aussi.

S'agissant du deuxième point, il faut montrer que les listes des objets et des propriétés de la figure n'augmentent pas sans cesse. Il est clair que si les tailles de ces listes

augmentent alors la responsabilité en incombe aux agents de complétion d'objets et de complétion de propriétés. En effet, seuls ces derniers ajoutent des éléments à la spécification de la figure. Cependant le fait que seuls des objets de type droite sont ajoutés à la spécification d'une figure par l'agent de complétion d'objets, conduit celle-ci à ne contenir qu'un nombre fini d'objets de type point, cercle et distance. Il s'ensuit que l'agent complétion de propriétés ne peut ajouter qu'un nombre fini de propriétés de type appCc et/ou distPP liant des objets de type point, cercle ou distance, et pouvant conduire à l'ajout d'une corde commune ou d'une médiatrice. Comme le nombre de propriétés de type appCc et/ou distPP est fini, le nombre des cordes communes ou médiatrice est fini. C'est-à-dire que la liste des objets de la figure ne peut pas augmenter indéfiniment. Enfin, comme la figure possède un nombre fini d'objets, elle ne possède qu'un nombre fini de propriétés les reliant entre eux. Ceci implique que la liste des propriétés de la figure ne peut pas augmenter indéfiniment.

8.4 Complétude de la résolution

Avant de définir la classe des figures pour laquelle une telle approche coopérative est complète, il convient de rappeler que celle-ci se base sur les agents linéaire, quadratique, intervalle, complétion d'objets et complétion de propriétés. En effet, souvenons-nous que l'agent règle et compas considère comme hypothèse que la figure est constructible numériquement. C'est-à-dire que si un plan de construction d'une figure dynamique a pu être établi, alors cette figure dynamique est constructible numériquement. Toutefois, la réciproque n'est pas vraie. C'est pourquoi la classe des figures constructibles par l'agent règle et compas est incluse dans celle d'une telle approche coopérative.

Il semble clair que la classe des figures constructibles par cette approche coopérative est au moins égale à l'union des classes des figures constructibles par chacun des agents considéré individuellement. Sur ce point, nous pouvons dire que la classe des figures constructibles par l'agent linéaire est celle des figures ayant une spécification linéaire ou devenant linéaire suite à la résolution d'une partie du système puisqu'il s'appuie sur l'algorithme d'élimination de Gauss augmenté [Col93] complet pour cette classe de systèmes d'équations.

En fait, il est possible de définir une autre classe de figures pour laquelle cette approche coopérative est complète. Cette classe de figures est celle des figures définies par une spécification "constructive", c'est-à-dire celle des figures définies par une spécification qui contient tous les objets et toutes les propriétés nécessaires à une construction

à la règle et au compas [Bou97].

◇ **Proposition 8.1** COMPLÉTUDE DES SPÉCIFICATIONS CONSTRUCTIVES

Notre approche coopérative est complète pour la classe des spécifications constructives.

DÉMONSTRATION : Pour montrer que notre approche est complète pour la classe des spécifications constructives, il faut montrer que les agents présents sont capables de realiser toutes les constructions élémentaires géométriques. Ces constructions élémentaires sont les suivantes :

- construction d'une droite passant par deux points. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues m et p du type :

$$y_1 = m \times x_1 + p \tag{8.1}$$
$$y_2 = m \times x_2 + p \tag{8.2}$$

  soluble par l'agent linéaire.

- construction d'un cercle de centre donné et passant par un point. Ceci se résume à résoudre un système d'une équations quadratique en l'inconnue r du type :

$$(x_1 - x_2)^2 + (y_1 - y_2)^2 = r^2 \tag{8.3}$$

  soluble par l'agent quadratique.

- construction d'une droite parallèle à une droite et passant par un point. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues m et p du type :

$$y_1 = m \times x_1 + p \tag{8.4}$$
$$m = m' \tag{8.5}$$

  soluble par l'agent linéaire.

- construction d'une droite perpendiculaire à une droite et passant par un point. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues m

et p du type :

$$y_1 = m \times x_1 + p \tag{8.6}$$
$$m = \frac{1}{m'} \tag{8.7}$$

soluble par l'agent linéaire.

– construction du point milieu de deux points. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues x et y du type :

$$2 \times x = x_1 + x_2 \tag{8.8}$$
$$2 \times y = y_1 + y_2 \tag{8.9}$$

soluble par l'agent linéaire.

– construction du symétrique d'un point par rapport à un autre point. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues x et y du type :

$$2 \times x_1 = x + x_2 \tag{8.10}$$
$$2 \times y_1 = y + y_2 \tag{8.11}$$

soluble par l'agent linéaire.

– construction de l'intersection de deux droites. Ceci se résume à résoudre un système de 2 équations linéaires en les 2 inconnues y et x du type :

$$y = m_1 \times x + p_1 \tag{8.12}$$
$$y = m_2 \times x + p_2 \tag{8.13}$$

soluble par l'agent linéaire.

– construction de l'intersection d'une droite et d'un cercle. Ceci se résume à résoudre un système d'une équation linéaires et d'une équation quadratique en les 2 inconnues y et x du type :

$$y = m \times x + p \tag{8.14}$$
$$(x - x_1)^2 + (y - y_1)^2 = r^2 \tag{8.15}$$

soluble par la coopération des agents linéaire et quadratique.

– construction de l'intersection de deux cercles. Ceci se résume à résoudre un système de 2 équations quadratiques en les 2 inconnues y et x du type :

$$(x - x_1)^2 + (y - y_1)^2 = r^2 \quad (8.16)$$
$$(x - x_2)^2 + (y - y_2)^2 = r^2 \quad (8.17)$$

soluble par la coopération des agents linéaire, quadratique et complétion d'objets. En effet, l'ajout par l'agent de complétion d'objets de la corde commune aux 2 cercles, c'est-à-dire d'une équation linéaire redondante en les variables x et y, réduit ce problème à celui de la construction de l'intersection d'une droite et d'un cercle.

□

Au delà de la classe des figures ayant une spécification constructive, nous devons nous poser le problème de la complétude de notre approche coopérative pour la classe des figures ayant une spécification "constructive par l'ajout de propriétés redondantes". Cette classe est celle des figures dont les spécifications contiennent tous les objets, mais possiblement pas toutes les propriétés nécessaires à une construction à la règle et au compas.

Pour montrer que notre approche est complète pour cette nouvelle classe de figures, il faut montrer que l'agent complétion de propriétés est en mesure de prouver la validité de toutes propriétés redondantes. Pour montrer ceci, il faut prouver d'une part que la théorie de la géométrie sur laquelle il s'appuie est cohérente et complète, et d'autre part que le démonstrateur automatique utilisé est complet. La complétude réfutationnelle dans le cadre d'une logique propositionnelle du démonstrateur que nous utilisons a été démontrée par Ostier P [Ost97b]. Malheureusement même si nous le pensons, il n'est pas évident de montrer que la théorie de la géométrie sur laquelle il s'appuie est cohérente. Celle-ci est donnée dans l'annexe A de ce manuscrit. Enfin, elle ne semble pas complète aussi.

Pour finir, nous pouvons affirmer que la classe des figures constructibles par notre approche coopérative est plus importante encore. Par exemple, les exercices 26 et 55 traités et détaillés dans l'annexe B en sont la preuve. Malheureusement, il est difficile de l'appréhender avec exactitude. La raison de cette difficulté réside dans l'agent intervalle. Il n'est pas facile en effet de déterminer a priori si les méthodes d'énumération d'intervalles conduisent à des domaines suffisamment réduits ou non.

Chapitre 9

Mise en oeuvre de la coopération

La mise en œuvre des agents de résolution de contraintes géométriques de GDRev constitue une partie centrale de notre travail. L'exposé des résultats obtenus dans le chapitre suivant, est distiné à montrer l'intérêt pratique d'une telle approche.

Nous décrivons dans ce chapitre les principaux aspects de la mise en œuvre des différents agents impliqués dans GDRev et de leurs coopérations. Nous justifions dans la section 9.1 le choix du langage de programmation utilisé. Nous descrivons dans la section 9.2 la mise en œuvre du contrôle des agents. Nous détaillons dans la section 9.3 la mise en œuvre des différents agents. Nous décrivons dans la section 9.4 les aspects de la mise en œuvre du langage ELDL. Enfin, nous indiquons dans la section 9.5 les aspects quantitatifs importants relatifs à cette mise en œuvre.

9.1 Choix du langage de développement

Pour mettre en œuvre les différents agents du processus coopératif de résolution de contraintes géométriques, il a fallu choisir un langage de programmation parmi les langages de programmation avec contraintes. En effet, de par leur caractère déclaratif, de tels langages de programmation semblent a priori particulièrement bien adaptés à l'expression de contraintes géométriques. Ils excluent directement l'usage d'un langage de programmation impératif qui auraient réclamé la reprogrammation de résolveurs déjà présents dans les langages de programmation avec contraintes.

Notre choix c'est porté sur le langage de programmation avec contraintes Prolog IV. Il a été préféré à des langages dits concurrents avec contraintes comme CC(FD) [vHSD93] ou OZ [Smo95] dans la mesure où il intègre un résolveur de contraintes linéaires exact

et aussi un résolveur non linéaire sur intervalles. Cette double intégration facile grandement la mise en œuvre en évitant des problèmes d'interopérabilités. Elle permet ainsi de définir l'ensemble du processus dans le même langage de programmation.

L'absence de caractère concurrent affiché de Prolog IV est compensée par la présence du prédicat de gel *freeze/2* qui permet de définir un contrôle de type concurrent. Rappelons que ce prédicat permet de retarder l'exécution du but contenu dans son deuxième argument tant que la valeur de l'étiquette de son premier argument n'est pas connue. Enfin, la disponibilité de Prolog IV sur les machines PC et Sun est un atout non négligeable pour une large expérimentation.

9.2 Mise en oeuvre de la coopération

Nous décrivons dans cette section la structure du programme dans son ensemble, puis la structure de coopération des différents agents.

9.2.1 Structure générale du programme

Dans la mise en œuvre actuelle, la recherche d'un plan de construction d'une figure dynamique, de même que la recherche de ses propriétés redondantes, fait suite à une demande explicite de l'utilisateur. Ceci nous a conduit à dissocier du processus de résolution les agents correspondants. Cette dissociation trouve sa justification dans la possible mise en évidence de l'intérêt de ces recherches qui en résulte. Toutefois, il serait aisé et rapide d'intégrer ces deux agents dans le schéma de coopération.

Le programme dans son ensemble est décrit par le prédicat *construireFigure(_LProps, _LObjs)* ci-dessous. Celui-ci a pour objet de mettre en attente les différents agents intervenant dans le processus de résolution, puis de libérer le premier d'entre eux, c'est-à-dire l'agent linéaire.

```
% -- construireFigure/2 ---------------------------------------- %
%                                                                 %
% construireFigure(_LProps, _LObjs)                               %
% construireFigure(liste(Propriete), liste(Objet))                %
%                                                                 %
% Le prédicat 'construireFigure' est vérifié si la figure         %
% répondant à la spécification contenue dans les arguments        %
```

```
% '_LProps' et '_LObjs' est constructible.                          %
% ------------------------------------------------------------------ %

construireFigure(_LProps, _LObjs) :-
    _Frees = [_FreesLin, _FreesQuad, _FreesCptObjs, _FreesInt],
    freeze(_FreesLin,  agentLin(_LProps, _LObjs, _Frees),
    freeze(_FreesQuad, agentQuad(_LProps, _LObjs, _Frees)),
    freeze(_FreesCptObjs,  agentCptObjs(_LProps, _LObjs, _Frees)),
    freeze(_FreesInt,  agentInt(_LProps, _LObjs, _Frees)),
    FreesLin = [_ | _].
```

Cette mise en œuvre est la transcription directe de la modélisation présentée dans le chapitre précédent.

9.2.2 Structure de coopération des agents

La structure générale de l'agent linéaire sous contrôle correspond au prédicat *agentLin(_LProps, _LObjs, _Frees)* détaillé ci-dessous.

```
% -- agentLin/3 ------------------------------------------------- %
%                                                                  %
% agentLin(_LProps, _LObjs, _Frees)                                %
% agentLin(liste(Propriete), liste(Objet), ?liste)                 %
%                                                                  %
% Le prédicat 'agentLin' est vérifié si les contraintes           %
% géométriques de la figure contenues dans les arguments          %
% '_LProps' et '_LObjs' sont satisfaites.                          %
% ------------------------------------------------------------------ %

% -- L'ensemble des propriétés a été examiné. l'agent LINEAIRE -- %
% -- se place en attente de nouvelles propriétés. --------------- %
agentLin(_LProps, _LObjs, _Frees) :-
    var(_LProps),
    freeze(_LProps, agentLin(_LProps, _LObjs, _Frees)),
    etablirBilanLin(_LObjs, _Frees).

% -- Il reste des atomes de propriété à évaluer. ---------------- %
```

```
agentLin(_LProps, _LObjs, _Frees) :-
    nonvar(_LProps),
    _LProps = [_Prop | _SProps],
    evaluerAtomeLin(_Prop, _Frees),
    agentLin(_SProps, _LObjs, _Frees).

% -- etablirBilanLin/2 ------------------------------------------ %
%                                                                  %
% etablirBilanLin(_LObjs, _Frees)                                  %
% etablirBilanLin(liste(Objet), ?liste)                            %
%                                                                  %
% Le prédicat 'etablirBilanLin' est toujours vérifié et gère le    %
% processus de construction faisant suite à l'agent linéaire       %
% ---------------------------------------------------------------- %

% -- Cas où la figure est construite. --------------------------- %
etablirBilanLin(_LObjs, _Frees) :-
    estFigureConstruite(_LObjs),
    !,
    write("\n\n== >> Figure construite << == \n").

% -- (Re)lance de l'agent QUADRATIQUE puisque l'agent LINEAIRE -- %
% -- est inefficace. ------------------------------------------- %
etablirBilanLin(_LObjs, _Frees) :-
    nonTellContraintes(lineaire, _Frees),
    libererAgent(quadratique, _Frees),
    etablirBilanQuad(_LObjs, _Frees).

% -- Relance l'agent LINEAIRE puisqu'il a posé de nouvelles ----- %
% -- contraintes. ---------------------------------------------- %
etablirBilanLin(_LObjs, _Frees) :-
    tellContraintes(lineaire, _Frees),
    libererAgent(lineaire, _Frees),
    etablirBilanLin(_LObjs, _Frees).
```

L'agent linéaire évalue successivement les atomes de propriétés de la figure. Il conduit à un succès si la figure est construite entièrement auquel cas le prédicat *est-*

FigureConstruite(_LObjs) est vérifié, ou il se relance ou (re)lance l'agent quadratique suivant les actions qu'il a ou non pu poser une contrainte linéaire.

Il convient de noter que pour des raisons d'efficacité, la terminaison "propre" du processus de construction, c'est-à-dire sans processus en attente, n'a pas été mise en œuvre. Ceci n'altère en rien la preuve de terminaison.

La mise en œuvre du contrôle de l'agent quadratique peut être faite d'une manière similaire à celle de l'agent linéaire. Toutefois, afin de réduire le nombre des racines carrées calculées, il est bloqué une fois qu'il a évalué une contrainte quadratique. Ceci est réalisé grâce aux prédicats suivants.

```
% -- agentQuad/3 ------------------------------------------------ %
%                                                                  %
% agentQuad(_LProps, _LObjs, _Frees)                               %
% agentQuad(liste(Propriete), liste(Objet), ?liste)                %
%                                                                  %
% Le prédicat 'agentQuad' est vérifié si les contraintes           %
% géométriques de la figure contenues dans les arguments           %
% '_LProps' et '_LObjs' sont satisfaites.                          %
% ---------------------------------------------------------------- %

% -- L'ensemble des propriétés a été examiné. L'agent ----------- %
% -- QUADRATIQUE se place en attente de nouvelles propriétés. --- %
% -- L'agent COMPLETION D'OBJETS est (re)lancé puisque l'agent -- %
% -- QUADRATIQUE n'a posé aucune contrainte. -------------------- %
agentQuad(_LProps, _LObjs, _Frees) :-
    var(_LProps),
    freeze(_LProps, agentQuad(_LProps, _LObjs, _Frees)),
    libererAgent(cptObjs, _Frees),
    etablirBilanCptObjs(_LObjs, _Frees).

% -- Des atomes de propriété restent à évaluer. ----------------- %
agentQuad(_LProps, _LObjs, _Frees) :-
    nonvar(_LProps),
    _LProps = [_Prop | _SProps],
    evaluerAtomeQuad(_Prop, _Frees),
    poursuivreAgentQuad(_SProps, _LObjs, _Frees).
```

```
% -- poursuivreAgentQuad/3 ------------------------------------ %
%                                                                %
% poursuivreAgentQuad(_LProps, _LObjs, _Frees)                   %
% poursuivreAgentQuad(liste(Propriete), liste(Objet), ?liste)    %
%                                                                %
% Le prédicat 'poursuivreAgentQuad' est toujours vérifié et      %
% poursuit ou non la résolution des contraintes géométriques à   %
% l'aide de l'agent quadratique suivant q'une contrainte         %
% quadratique a été résolue ou non.                              %
% -------------------------------------------------------------- %

% -- L'agent QUADRATIQUE est mis en attente - Il a résolu une --- %
% -- propriété. ----------------------------------------------- %
poursuivreAgentQuad(_LProps, _LObjs, _Frees) :-
    tellContraintes(quadratique, _Frees),
    gelerAgent(quadratique, _LProps, _LObjs, _Frees).

% -- La résolution est poursuivie par l'agent QUADRATIQUE ------- %
% -- Aucune propriété n'a été résolue. ------------------------- %
poursuivreAgentQuad(_LProps, _LObjs, _Frees) :-
    nonTellContraintes(quadratique, _Frees),
    agentQuad(_LProps, _LObjs, _Frees).

% -- etablirBilanQuad/2 --------------------------------------- %
%                                                                %
% etablirBilanQuad(_LObjs, _Frees)                               %
% etablirBilanQuad(liste(Objet), ?liste)                         %
%                                                                %
% Le prédicat 'etablirBilanQuad' est toujours vérifié et gère le %
% processus de construction faisant suite à l'agent quadratique. %
% -------------------------------------------------------------- %

% -- Cas où la figure est construite. ------------------------- %
etablirBilanQuad(_LObjs, _Frees) :-
    estFigureConstruite(_LObjs),
    !,
```

```
    write("\n\n== >> Figure construite << == \n").

% -- (Re)lance de l'agent COMPLETION D'OBJETS puisque l'agent --- %
% -- QUADRATIQUE n'a été inefficace. ---------------------------- %
etablirBilanQuad(_LObjs, _Frees) :-
    nonTellContraintes(quadratique, _Frees),
    libererAgent(cptObjs, _Frees),
    etablirBilanCptObjs(_LObjs, _Frees).

% -- Relance l'agent LINEAIRE puisqu'il a su poser une ---------- %
% -- contrainte. ------------------------------------------------ %
etablirBilanQuad(_LObjs, _Frees) :-
    tellContraintes(quadratique, _Frees),
    libererAgent(lineaire, _Frees),
    etablirBilanLin(_LObjs, _Frees).
```

Enfin, la mise en œuvre du schéma de contrôle de l'agent complétion d'objets se calque sur celle de l'agent quadratique, et celui de l'agent intervalle sur celui de l'agent linéaire à la différence qu'il ne libère aucun agent après lui.

9.3 Mise en oeuvre des différents agents

Pour une mise en œuvre plus efficace des prédicats d'évaluation des atomes de propriété propre à chaque agent, nous avons associé aux arguments de ces derniers les objets mis en jeu, et non uniquement leur identificateur. De la sorte, on évite des parcours incessant de la liste des objets *_LObjs*.

Nous décrivons successivement dans cette section la mise en œuvre des agents linéaire, quadratique, intervalle, complétion d'objets, complétion de propriétés, et règle et compas.

9.3.1 Mise en oeuvre de l'agent linéaire

La mise en œuvre du prédicat *evaluerAtomeLin* est relativement directe. Celle-ci est en partie donnée ci-dessous.

```
% -- evaluerAtomeLin/2 ------------------------------------------ %
```

```
%                                                                    %
% evaluerAtomeLin(_Prop, _Frees)                                     %
% evaluerAtomeLin(Propriete, ?liste)                                 %
%                                                                    %
% Le prédicat 'evaluerAtomeLin' est vérifié si la contrainte        %
% géométrique linéaire correspondant à '_Prop' est satisfaite.      %
% ------------------------------------------------------------------ %

% -- appDr ------------------------------------------------------- %
evaluerAtomeLin(appDr(_Pt, _Dr), _Frees) :-
    point(_Pt, _X, _Y),
    droite(_Dr, _M, _P),
    poserContrainteLin(_Y, _M, _X, _P, appDr(_Pt, _Dr), _Frees).

% -- appCc ------------------------------------------------------- %
evaluerAtomeLin(appCc(_, _), _) :-
    write("équation quadratique \n").

% -- par --------------------------------------------------------- %
evaluerAtomeLin(par(_Ligne1, _Ligne2), _Frees) :-
    mObjetLigne(_Ligne1, _M),
    mObjetLigne(_Ligne2, _M),
    marquerTellContraintes(lineaire, _Frees).

% -- On arrive ici suite à un échec lié à une propriété non ----- %
% -- consistante ------------------------------------------------- %
evaluerAtomeLin(_, _) :-
    !,
    fail.
```

La première règle est consacrée à l'évaluation de la propriété pseudo-linéaire d'appartenance d'un point à une droite. Son sous but *point(_Pt, _X, _Y)* (resp. *droite(_Dr, _M, _P)*) vérifie si les arguments $\_X$ et $\_Y$ sont l'abscisse et l'ordonnée du point *_Pt* (resp. vérifie si les arguments $\_M$ et $\_P$ sont les coefficients de la droite *_Dr.*). Le prédicat *poserContrainteLin(_Y, _M, _X, _P, _Prop, _Frees)* est vérifié si l'équation $Y = M \times X + P$ est linéaire. Il repositionne en attente la contrainte géométrique si celle-ci n'est pas linéaire. Il échoue si la contrainte n'est pas consistante avec le système

de contraintes courant.

La deuxième règle est consacrée à l'évaluation d'une propriété non linéaire. Dans ce cas, l'évaluation réussit toujours et affiche à l'écran un message indiquant que l'agent linéaire n'a pas évalué une contrainte quadratique.

La troisième règle est consacrée à l'évaluation de la contrainte géométrique toujours linéaire par. Elle consiste d'une part à poser la contrainte spécifiant que les objets mis en jeu ont le même coefficient directeur, et d'autre part à marquer à l'aide du prédicat *marquerTellContraintes(lineaire, _Frees)* l'ajout de la contrainte linéaire si celle-ci est consistante avec le système de contraintes courant.

Enfin, la dernière règle présentée correspond au cas où une contrainte n'est pas consistante avec le système courant. Elle conduit à l'échec de la construction.

9.3.2 Mise en oeuvre de l'agent quadratique

La principale difficulté rencontrée dans la mise en œuvre de l'agent quadratique réside dans la définition du prédicat *relatLin* défini dans le paragraphe 8.2.5. Celle-ci est donnée ci-dessous.

```
% -- relatLin/4 ------------------------------------------------ %
%                                                                 %
% relatLin(_A, _B, _X, _Y)                                        %
% relatLin(?rationel, ?rationel, var, var)                        %
%                                                                 %
% Le prédicat 'relatLin' est vérifié si et seulement si les       %
% variables '_X' et '_Y' sont liées linéairement selon une        %
% équation de la forme : _Y = _A * _X + _B, avec _A et _B connus. %
% ------------------------------------------------------------- %

relatLin(_A, _B, _X, _Y) :-
    record(typePremierCouple, aucun),
    record(typeDeuxiemeCouple, aucun),
    trouverPremierCoupleValeurs(_X, _Y, _X1, _Y1),
    trouverDeuxiemeCoupleValeurs(_X, _Y, _X2, _Y2),
    _Y1 = _A * _X1 + _B,
    _Y2 = _A * _X2 + _B.
```

```
% -- trouverPremierCoupleValeurs/4 ------------------------------ %
%                                                                  %
% trouverPremierCoupleValeurs(_X, _Y, _X1, _Y1)                    %
% trouverPremierCoupleValeurs(var, var, ?rationel, ?rationel)      %
%                                                                  %
% Le prédicat 'trouverPremierCoupleValeurs' est vérifié si et      %
% seulement si '_X' et '_Y' vérifient une équation linéaire et si  %
% '_X1' et '_Y1' sont des valeurs particulières de '_X' et '_Y'.   %
% ---------------------------------------------------------------- %

% -- '_X1' est une valeur minimum de '_X'. ---------------------- %
trouverPremierCoupleValeurs(_X, _Y, _, _) :-
    minimizelin(_X),
    rational(_Y),
    record(typePremierCouple, minimize),
    crecord(premierX, _X),
    crecord(premierY, _Y),
    fail.

% -- '_X' n'a pas de valeur minimum, '_X1' prend a valeur ------- %
% -- maximum de '_X'. ------------------------------------------- %
trouverPremierCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, aucun),
    maximizelin(_X),
    rational(_Y),
    record(typePremierCouple, maximize),
    crecord(premierX, _X),
    crecord(premierY, _Y),
    fail.

% -- '_X' n'a ni une valeur minimum, ni une valeur maximum. ----- %
% -- '_X1' prend une valeur quelconque. ------------------------- %
trouverPremierCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, aucun),
    _X = 4,
    rational(_Y),
    record(typePremierCouple, quelconque),
```

```
    crecord(premierX, _X),
    crecord(premierY, _Y),
    fail.

% -- Récupération du premier couple de valeurs s'il a été ------- %
% -- déterminé. ------------------------------------------------- %
trouverPremierCoupleValeurs(_X, _Y, _X1, _Y1) :-
    recorded(typePremierCouple, _TypePremier),
    dif(_TypePremier, aucun),
    crecorded(premierX, _X1),
    crecorded(premierY, _Y1).

% -- trouverDeuxiemeCoupleValeurs/4 ----------------------------- %
%                                                                 %
% trouverDeuxiemeCoupleValeurs(_X, _Y, _X2, _Y2)                  %
% trouverDeuxiemeCoupleValeurs(?réel, ?réel, ?réel, ?réel)        %
%                                                                 %
% Le prédicat 'trouverDeuxiemeCoupleValeurs' est vérifié si et    %
% seulement si '_X' et '_Y' vérifient une équation linéaire et si %
% '_X2' et '_Y2' sont des valeurs particulières de '_X' et '_Y'.  %
% --------------------------------------------------------------- %

% -- '_X' et '_Y' ne sont pas liés par une équation linéaire. --- %
trouverDeuxiemeCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, aucun),
    !,
    fail.

% -- '_X' n'a pas de valeur minimum, ni de valeur maximum. ------ %
% -- '_X2' prend une valeur quelconque (différente de '_X1'). --- %
trouverDeuxiemeCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, quelconque),
    _X = 2,
    rational(_Y),
    record(typeDeuxiemeCouple, quelconque),
    crecord(deuxiemeX, _X),
    crecord(deuxiemeY, _Y),
```

```
    fail.

% -- '_X' a une valeur minimum, '_X2' est une valeur maximum. --- %
trouverDeuxiemeCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, minimize),
    maximizelin(_X),
    rational(_Y),
    record(typeDeuxiemeCouple, maximize),
    crecord(deuxiemeX, _X),
    crecord(deuxiemeY, _Y),
    fail.

% -- '_X'  a une valeur minimum, mais n'a pas de valeur -------- %
% -- maximum, '_X2' est déterminé à partir de '_X1'. ----------- %
trouverDeuxiemeCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, minimize),
    recorded(typeDeuxiemeCouple, aucun),
    crecorded(premierX, _Xpremier),
    _X = _Xpremier + 1,
    rational(_Y),
    record(typeDeuxiemeCouple, maximize),
    crecord(deuxiemeX, _X),
    crecord(deuxiemeY, _Y),
    fail.

% -- '_X' a une valeur maximum, mais n'a pas de valeur minimum - %
% -- '_X2' est déterminé à partir de '_X1'. -------------------- %
trouverDeuxiemeCoupleValeurs(_X, _Y, _, _) :-
    recorded(typePremierCouple, maximize),
    crecorded(premierX, _Xpremier),
    _X = _Xpremier - 1,
    rational(_Y),
    record(typeDeuxiemeCouple, maximize),
    crecord(deuxiemeX, _X),
    crecord(deuxiemeY, _Y),
    fail.
```

```
% -- Récupération du deuxième couple de valeurs s'il a été ----- %
% -- déterminé. ------------------------------------------------ %
trouverDeuxiemeCoupleValeurs(_X, _Y, _X2, _Y2) :-
    recorded(typeDeuxiemeCouple, _TypeDeuxieme),
    dif(_TypeDeuxieme, aucun),
    crecorded(deuxiemeX, _X2),
    crecorded(deuxiemeY, _Y2).
```

Nous n'avons malheureusement pas trouvé de manière plus élégante d'écrire ce prédicat. Il nous semble en effet dommage d'être obligé d'avoir recours à de tels effet de bord, alors que l'information désirée est connue des résolveurs. Après avoir demandé aux concepteurs de Prolog III et Prolog IV qu'une telle primitive soit intégrée, nous espérons qu'elle fasse bientôt partie de la norme Iso Prolog.

9.3.3 Mise en oeuvre de l'agent intervalle

La mise en œuvre du prédicat *evaluerAtomeInt* est directe. Celle-ci est illustrée en partie ci-dessous.

```
% -- evaluerAtomeInt/2 ----------------------------------------- %
%                                                                 %
% evaluerAtomeInt(_Prop, _Frees)                                  %
% evaluerAtomeInt(Propriete, ?liste)                              %
%                                                                 %
% Le prédicat 'evaluerAtomeInt' est vérifié si la contrainte      %
% géométrique correspondant à '_Prop' est valide dans le système  %
% de contraintes sur intervalles courant.                         %
% --------------------------------------------------------------- %

% -- appDr ----------------------------------------------------- %
evaluerAtomeInt(appDr(_Pt, _Dr), _) :-
    point(_Pt, _X, _Y),
    droite(_Dr, _M, _P),
    _Y = _M .*. _X .+. _P.
```

Cette mise en œuvre s'appuie ainsi directement sur le résolveur sur intervalles de Prolog IV.

Afin d'obtenir des domaines de définition suffisamment réduits pour être acceptables à l'écran, nous avons recours au prédicat d'énumération d'intervalles *realsplit*. L'heuristique d'énumération d'intervalles que nous utilisons consiste à appliquer le prédicat *realsplit* sur une des coordonnées des points non construits les plus contraints.

De plus, nous prenons comme hypothèse que la figure est construite dans la partie strictement positive d'une feuille repérée suivant un axe orthonormé. Ceci nous permet de contraindre les coordonnées des points d'une figure à appartenir à l'intervalle $[1, 1600]$. Le choix de ce domaine n'est pas fortuit. Il correspond à la résolution d'un écran d'ordinateur qui nous sert à repérer les éléments d'une figure. Il résulte aussi de l'hypothèse que l'utilisateur s'arrange souvent pour que sa figure tienne dans la fenêtre d'affichage. Grâce à cette hypothèse, on peut réduire considérablement le domaine associé aux coordonnées des points d'une figure en excluant les valeurs sources de difficultés, c'est-à-dire les infinis et le zéro. Sans entrer dans les détails, cette hypothèse nous permet d'obtenir de meilleurs résultats. Toutefois, si la spécification d'une contrainte portant sur un point échoue, le résolution sur intervalles est relancée en considérant cette fois-ci que le point incriminé appartient à l'intervalle $]-\infty, +\infty[$ [Cha96].

9.3.4 Mise en oeuvre de l'agent complétion d'objets

La mise en œuvre du prédicat *evaluerAtomeCptObj* est directe. Celle-ci est illustrée en partie ci-dessous.

```
% -- evaluerAtomeCptObj/4 ------------------------------------- %
%                                                               %
% evaluerAtomeCptObj(_Atome, _LProps, _LObjs, _Frees)           %
% evaluerAtomeCptObj(Propriete, liste(Propriete), liste(Objet), %
%                    ?liste)                                    %
%                                                               %
% Le prédicat 'evaluerAtomeCptObj' est toujours vérifié et      %
% cherche à compléter la spécification de la figure par des     %
% cordes communes ou médiatrices.                               %
% ------------------------------------------------------------- %

% -- '_Atome' est une propriété de type appCc, '_LProps' -------- %
% -- contient une propriété de type appCc ou distPP compatible. - %
evaluerAtomeCptObj(_Atome, _LProps, _LObjs, _Frees) :-
```

```
    _Atome = appCc(_IdPt1, _IdCc),
    rechercherAutreAppCc(_Atome, _AutreAtome, _LProps),
    produireDroite(_Atome, _AutreAtome, _LObjs, _Droite, _PropDroite),
    marquerTellContraintes(cptObjs, _Frees),
    ajouterObjet(_Droite, _LObjs),
    ajouterProprietes(_PropDroite, _LProps).

% -- '_Atome' est une propriété non évaluable par l'agent ------- %
% -- COMPLETION D'OBJETS, ou une propriété appCc unique. -------- %
evaluerAtomeCptObj(_, _, _, _).

% -- rechercherAutreAppCc/3 ------------------------------------- %
%                                                                 %
% rechercherAutreAppCc(_Atome, _AutreAtome, _LProps)              %
% rechercherAutreAppCc(Propriete, ?Propriete, liste(Propriete))   %
%                                                                 %
% Le prédicat 'rechercherAutreAppCc' est vérifié si '_AutreAtome' %
% est propriété compatible avec l'atome '_Atome' pour former une  %
% cordes communes ou médiatrices.                                 %
% --------------------------------------------------------------- %

% -- '_AutreAtome' est une propriété compatible avec la --------- %
% -- propriété '_Atome'. ---------------------------------------- %
rechercherAutreAppCc(_Atome, _AutreAtome, _LProps) :-
    nonvar(_LProps),
    _LProps = [_AutreAtome | _SProps],
    _Atome = appCc(_Pt1, _Cc),
    _AutreAtome = appCc(_Pt2, _Cc),
    dif(_Pt1, _Pt2).

% -- '_Atome' est une propriété non évaluable par l'agent ------- %
% -- COMPLETION D'OBJETS. --------------------------------------- %
rechercherAutreAppCc(_Atome, _AutreAtome, _LProps) :-
    nonvar( _LProps),
    _LProps = [_ | _SProps],
    rechercherAutreAppCc(_Atome, _AutreAtome, _SProps).
```

La première règle correspond au cas où l'atome de propriété est une appartenance d'un point à un cercle. Il s'agit alors de trouver une propriété du même type et compatible, à l'aide du prédicat *rechercherAutreAppCc(_Atome, _AutreAtome, _LProps)*. Ce prédicat est vérifié si la liste des propriétés *_LProps* contient un atome *_AutreAtome* compatible avec l'atome *_Atome* pour former une corde commune ou une médiatrice. Ceci étant il s'agit de déterminer la corde commune ou médiatrice à l'aide du prédicat *produireDroite(_Atome, _AutreAtome, _LObjs, _Droite, _PropDroite)*, pour enfin ajouter l'objet produit ainsi que ses propriétés à la spécification de la figure.

La deuxième règle du prédicat *evaluerAtomeCptObj(_Atome, _LProps, _LObjs, _Frees)* correspond au cas où l'atome de propriété est ni un atome de type appCc, ni un atome de type distPP. Dans ce cas, l'évaluation réussit toujours.

La recherche d'une propriété du type appCc ou distPP compatible avec un atome donné est illustrée en partie. Cette recherche consiste en un simple parcours d'une liste.

9.3.5 Mise en oeuvre de l'agent complétion de propriétés

Dans la mesure où nous avons dissocié, dans la mise en œuvre actuelle, cet agent du processus de résolution, il s'ensuit que sa mise en œuvre est une application directe du schéma présenté dans le paragraphe 7.5. Celle-ci est illustrée ci-dessous.

```
% -- agentCptProps/3 ------------------------------------------ %
%                                                               %
% agentCptProps(_LProps, _LObjs, _Frees)                        %
% agentCptProps(liste(Propriete), liste(Objet), ?liste)         %
%                                                               %
% Le prédicat 'agentCptProps' est toujours vérifié et ajoute à la %
% liste des propriétés '_LProps' les propriétés redondantes liant %
% les objets de la liste '_LObjs'.                              %
% ------------------------------------------------------------- %

agentCptProps(_LProps, _LObjs, _Frees) :-
    produirePossiblesProps(_LProps, _LObjs, _PropsPoss),
    reconstruireFigure(_LProps, _LObjs, _Objs),
    validerProprietes(_Objs, _LProps, _PropsPoss, _PropsRed),
    ajouterProprietes(_PropsRed, _LProps).
```

Cette mise en œuvre se décompose en quatre sous-buts. Le prédicat *produireProssiblesProps(_LProps, _LObjs, _PropsPoss)* est vérifié si la liste *_PropsPoss* est la liste des possibles propriétés reliant des éléments de la figure contenus dans *_LObjs* privée de la liste *_LProps*. La reconstruction de la figure à partir d'un nouvel ensemble d'objets de base est réalisée à l'aide du prédicat *reconstruireFigure(_LProps, _LObjs, _Objs)* qui est vérifié si *_Objs* correspond à un dessin de la figure. Ensuite, on élimine en s'appuyant sur le dessin contruit les propriétés fausses et on valide les propriétés restantes grâce à un démonstrateur automatique [Ost97b] dans le prédicat *validerProprietes(_Objs, _LProps, _PropsPoss, _PropsRed)* qui est vérifié si la liste *_PropsRed* est la liste des propriétés redondantes de la figure. Enfin, la liste des propriétés redondantes trouvées est ajoutée à la spécification de la figure grâce au prédicat *ajouterProprietes(_PropsRed, _LProps)* qui est vérifié si la liste pourvue d'un suffixe libre *_LProps* a pour derniers éléments les éléments de la liste *_PropsRed*.

9.3.6 Mise en oeuvre de l'agent règle et compas

La mise en œuvre de l'agent règle et compas hors du processus coopératif de résolution est illustrée brièvement ci-dessous.

```
% -- agentRC/3 ------------------------------------------------ %
%                                                               %
% agentRC(_LProps, _LObjs, _Plan)                               %
% agentRC(liste(Propriete), liste(Objet), ?liste)               %
%                                                               %
% Le prédicat 'agentRC' est vérifié si une plan de construction à %
% la règle et au compas de la figure dynamique courante est     %
% établi dans l'argument '_Plan'.                               %
% ------------------------------------------------------------- %

agentRC(_LProps, _LObjs, _Plan) :-
    _Frees = [FreesClos, FreesExt, FreesNonClos],
    freeze(_FreesClos, mondeClos(_LProps, _LObjs, _Plan, _Frees)),
    freeze(_FreesClosExt, mondeClosEtendu(_LProps, _LObjs, _Frees)),
    freeze(_FreesNonClos, mondeNonClos(_LProps, _LObjs, _Frees)),
    _FreesClos = [_ | _].
```

Celle-ci consiste à chercher à établir un plan de construction de la figure dynamique en prenant pour hypothèse qu'elle appartient au monde clos (cf paragraphe 7.6.2). Pour ce faire, nous avons défini le prédicat *mondeClos(_LProps, _LObjs, _Plan, _Frees)*. Si cette recherche n'aboutit pas alors, suivant un processus de coopération semblable à celui guidant les différents agents de GDRev, une recherche sur le monde clos étendu est menée à l'aide du prédicat *mondeClosEtendu(_LProps, _LObjs, _Frees)*. Celui-ci ajoute les propriétés redondantes trouvées et relance la recherche du plan de construction sur le monde clos, ou tente une recherche partielle d'objets pouvant aider à l'établissement du plan de construction si aucune propriété redondante n'est trouvée. Pour cela, le prédicat *mondeNonClos(_LProps, _LObjs, _Frees)* est dégelé. Celle-ci consiste uniquement en l'ajout des cordes communes et des médiatrices.

Une mise en œuvre de la recherche d'un plan de construction d'une figure du monde non clos linéaire [CI97] existe, mais elle n'est toujours pas intégrée à l'ensemble.

9.4 Mise en oeuvre du langage ELDL

Nous présentons dans les paragraphes suivants les aspects liés à la mise en œuvre du langage ELDL. Ceux-ci concernent la mise en œuvre d'une part d'un interpréteur pour les clauses du langage, et d'autre part de la négation partiellement introduite.

9.4.1 Interprétation du langage ELDL

Nous avons introduit dans la paragraphe 4.2.2 page 63 la notion d'implication dans les spécifications d'une figure. De même qu'en programmation logique, nous devons pour aboutir à une mise en œuvre de notre modèle, trancher sur les deux questions du choix du but de la résolvante à réduire et du choix de la règle du programme pour effectuer la réduction. Ce dernier choix conditionne le modèle d'exécution de l'interpréteur.

Notre approche consiste comme en Prolog d'une part à considérer les règles suivant leur ordre d'apparition dans le programme avec retour arrière (*backtracking*), et d'autre part à choisir comme but, le but le plus à gauche de la résolvante. Cette approche autorise une organisation en pile. Nous dépilons le but à réduire et nous empilons les sous-buts dérivés jusqu'à obtenir un échec, ou un succès avec une pile vide.

Les raisons de ces choix se trouvent dans la facile compréhension du modèle d'exé-

cution d'un tel interpréteur. Ils se trouvent aussi dans la possible mise en œuvre efficace et facile qui en découle.

```
1   InterpreterFigure(specification : formule ELDL, liste_clauses : formules ELDL)
2
3   resolvante ← ∅
4   echec ← faux
5   Empiler(specification, resolvante)
6   Tant Que ¬ echec et ¬ Vide(resolvante) Faire
7       DepilerBut(resolvante, but)
8       ChoisirClause(but, liste_clauses, clause, reussite_choix_clause)
9       Si reussite_choix_clause Alors
10          SauverPointChoix(resolvante, but, clause)
11          Empiler(clause, resolvante)
12          ExtraireSousButs(resolvante, sous_buts)
13          ConstruireFigure(sous_buts, reussite_construction)
14          Si ¬ reussite_reconstruction Alors
15              RestaurerPointChoix
16          FinSi
17      Sinon
18          RestaurerPointChoix
19          echec ← Vide(resolvante)
20      FinSi
21  FinTantQue
```

Tab. 9.1 – *Algorithme d'interprétation de spécifications ELDL.*

Afin d'accélérer le processus de résolution, nous avons différencié notre algorithme de celui d'un interpréteur issu d'un langage de programmation logique. Celui-ci est détaillé dans le tableau 9.1. La différence réside dans la procédure *ExtraireSousButs* qui dépile de la résolvante non pas un sous-but à la fois, mais tous les premiers sous-buts gauche tant que ceux-ci sont des atomes ou la négation d'atomes du langage LDL.

Il convient de remarquer qu'avec une tel interpréteur, nous nous confrontons aux mêmes limites relatives au traitement de la négation que Prolog.

9.4.2 Mise en oeuvre de la négation du langage ELDL

La mise en œuvre de la négation partielle que nous avons introduite s'appuie sur le prédicat *dif/2* de Prolog IV. Ce prédicat établit une diséquation entre les deux valeurs quelconques qui constituent ses arguments.

Nous décrivont dans les paragraphes suivant la mise en œuvre de la négation dans les agents linéaire et intervalle.

A l'heure actuelle, l'agent quadratique ne supporte pas la négation des atomes appCc et distPP.

Mise en oeuvre de la négation dans l'agent linéaire

La mise en œuvre du prédicat *evaluerAtomeLin* pour la négation d'atome est en partie donnée ci-dessous.

```
% -- evaluerAtomeLin/2 ---------------------------------------- %
%                                                               %
% evaluerAtomeLin(_Prop, _Frees)                                %
% evaluerAtomeLin(Propriete, ?liste)                            %
%                                                               %
% Le prédicat 'evaluerAtomeLin' est vérifié si la contrainte    %
% géométrique linéaire correspondant à '_Prop' est satisfaite.  %
% ------------------------------------------------------------- %

% -- non(appSeg) ------------------------------------------------ %
evaluerAtomeLin(non(appSeg(_Pt, _Seg)), _Frees) :-
    point(_Pt, _X, _Y),
    segment(_Seg, _, _, _, _, _M, _P),
    poserProduitLin(_M, _X, _P, _MXP, non(appSeg(_Pt, _Seg)), _Frees),
    dif(_Y, _MXP).

evaluerAtomeLin(non(appSeg(_Pt, _Seg)), _Frees) :-
    point(_Pt, _X, _Y),
    segment(_Seg, _X1, _, _X2, _, _, _),
    poserMinLin(_X1, _X2, _Xmin, non(appSeg(_Pt, _Seg)), _Frees),
    ltlin(_X, _Xmin).
```

```
evaluerAtomeLin(non(appSeg(_Pt, _Seg)), _Frees) :-
    point(_Pt, _X, _Y),
    segment(_Seg, _X1, _, _X2, _, _, _),
    poserMaxLin(_X1, _X2, _Xmax, non(appSeg(_Pt, _Seg)), _Frees),
    gtlin(_X, _Xmax).

% -- non(milieu) ------------------------------------------------ %
evaluerAtomeLin(non(milieu(_Pt, _Pt1, _Pt2)), _) :-
    point(_Pt, _X, _Y),
    point(_Pt1, _X1, _Y1),
    point(_Pt2, _X2, _Y2),
    dif([2*_X, 2*_Y], [_X1+_X2, _Y1+_Y2]).
```

Cette mise en œuvre est directe si la négation peut s'exprimer sous la forme d'une diséquation, comme l'illustre la négation de la propriété milieu (quatrième règle). Elle s'appuie sur le retour arrière dans le cas contraire comme l'illustre celle de la propriété appSeg (première, deuxième et troisième règle).

Mise en oeuvre de la négation dans l'agent intervalle

La mise en œuvre du prédicat *evaluerAtomeInt* pour la négation d'atome est en partie donnée ci-dessous.

```
% -- evaluerAtomeInt/2 ------------------------------------------ %
%                                                                  %
% evaluerAtomeInt(_Prop, _Frees)                                   %
% evaluerAtomeInt(Propriete, ?liste)                               %
%                                                                  %
% Le prédicat 'evaluerAtomeInt' est vérifié si la contrainte      %
% géométrique correspondant à '_Prop' est valide dans le système  %
% de contraintes sur intervalles courant.                          %
% ---------------------------------------------------------------- %

% -- non(appSeg) ------------------------------------------------ %
evaluerAtomeInt(non(appSeg(_Pt, _Seg)), _) :-
    point(_Pt, _X, _Y),
```

```
    segment(_Seg, _, _, _, _, _M, _P),
    dif(_Y, _M .*. _X .+. _P).

evaluerAtomeInt(non(appSeg(_Pt, _Seg)), _) :-
    point(_Pt, _X, _Y),
    segment(_Seg, _X1, _, _X2, _, _, _),
    min(_Xmin, _X1, _X2),
    lt(_X, _Xmin).

evaluerAtomeInt(non(appSeg(_Pt, _Seg)), _) :-
    point(_Pt, _X, _Y),
    segment(_Seg, _X1, _, _X2, _, _, _),
    max(_Xmax, _X1, _X2),
    lt(_X, _Xmax).

% -- non(appCc) ------------------------------------------------ %
evaluerAtomeInt(appCc(_Pt, _Cc), _) :-
    point(_Pt, _X, _Y),
    cercle(_Cc, _Xo, _Yo, _R),
    dif(square(_Xo .-. _X) .+. square(_Yo .-. _Y), square(_R)).

% -- non(milieu) ----------------------------------------------- %
evaluerAtomeInt(non(milieu(_Pt, _Pt1, _Pt2)), _) :-
    point(_Pt, _X, _Y),
    point(_Pt1, _X1, _Y1),
    point(_Pt2, _X2, _Y2),
    dif([2.*._X, 2.*._Y], [_X1.+._X2, _Y1.+._Y2]).
```

Cette mise en œuvre s'appuie directement sur le résolveur sur intervalles de Prolog IV. On peut noter que l'agent intervalle prend en compte la négation des atomes de propriété appCc et distPP (quatrième règle).

9.5 Aspects quantitatifs

La mise en œuvre des différents agents intervenant dans le processus de résolution de contraintes géométriques de GDRev, et de leurs interactions représente :

- l'écriture de 8.000 lignes de Prolog IV contenus dans 45 fichiers.
- la définition de 340 prédicats.
- une mise en œuvre d'une durée de 5 mois homme.
- un code de 610 Ko.

Ces quelques chiffres confirment l'intérêt d'user d'un tel langage de programmation dont les principaux avantages sont la production d'un code d'une remarquable concision, la facile correction des erreurs de programmation, et l'aisance de la maintenance du code. De plus, l'usage de ce langage de programmation de haut niveau nous a permis d'expérimenter rapidement tout au long de la mise en œuvre de nouvelles heuristiques.

Chapitre 10

Résultats expérimentaux

OUTRE TOUS LES EXEMPLES DE CONSTRUCTION présentés dans ce manuscrit, il convient de présenter dans la mesure du possible des exercices que GDRev sait résoudre, mais aussi ceux que GDRev ne sait pas résoudre.

Nous exposons dans ce chapitre les résultats obtenus avec notre approche coopérative de la résolution de contraintes géométriques sur un ordinateur PC à 300 MHz avec 64 Mo de RAM sous Windows NT. Nous commençons par illustrer dans la section 10.1 les performances de GDRev concernant la résolution de contraintes géométriques. Puis, nous nous concentrons dans la section 10.2 sur les résultats des heuristiques de construction automatique de figures géométriques à partir d'un ensemble quelconque d'objets de base. Enfin, la section 10.3 conclut ce chapitre.

10.1 Résultats sur la résolution de contraintes géométriques

Le système GDRev a été testé sur un ensemble de 40 exercices de construction géométrique principalement extrait de [But75]. Le détail de ces résolutions peut être trouvé dans l'annexe B de ce manuscrit.

Nous présentons d'abord des exemples typiques de problèmes de construction résolus. Nous examinons ensuite des exemples typiques de problèmes de construction que GDRev n'arrive pas à résoudre pour l'instant.

10.1.1 Exemples de résolution

Chaque exercice de construction présenté dans ce paragraphe est accompagné d'un tableau détaillant le numéro de l'exercice, le temps d'exécution et les agents intervenant selon qu'il s'agit de la première construction ou de constructions suivantes (c'est-à-dire d'une construction basée sur les mêmes ensembles d'objets de base et de propriétés auxquels sont ajoutés les objets et les propriétés redondants déterminés par la première construction).

Exemple 10.1 SEGMENT MILIEU

Construire un segment porté par deux droites sécantes données connaissant son milieu.

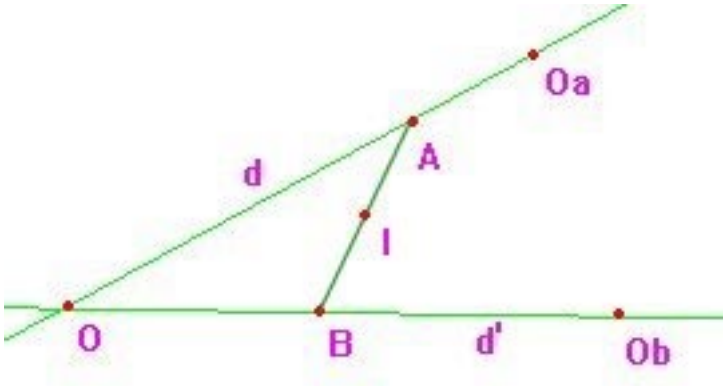

FIG. *10.1* – Segment milieu.

La spécification ELDL de cette figure est :

$$point(A) \wedge\ point(B) \wedge\ point(O) \wedge\ point(I) \wedge$$
$$point(Oa) \wedge\ point(Ob) \wedge$$
$$droite(d) \wedge\ droite(d') \wedge\ segment(s_{AB}, A, B, I_{AB}) \wedge$$
$$appDr(O, d) \wedge\ appDr(O, d') \wedge\ appDr(Oa, d) \wedge\ appDr(Ob, d') \wedge$$
$$milieu(I, A, B) \wedge\ appDr(A, d) \wedge\ appDr(B, d').$$

La résolution de cette spécification géométrique à partir de la donnée des points O, Oa, Ob et I conduit au dessin illustré par la figure 10.1.

On remarque sur le tableau récapitulatif 10.1 que cette construction requiert uniquement l'intervention de l'agent linéaire.

N°	Cons-truc-tion	Agents intervenant Lin	Quad	Cpt obj	Cpt prop	Int	Tps (sec)
10.1	1	X					0
	suiv.	X					0

TAB. *10.1* – Agents intervenant dans la construction du problème appelé segment milieu.

Cet exemple illustre la classe des problèmes de construction auxquels l'agent linéaire peut s'attaquer seul. Il est aussi typique d'un scénario ayant pour objectif la découverte de la géométrie dynamique déclarative. En effet, un moyen de construire cette figure dynamique consiste dans un premier temps à construire la figure dynamique ayant la même spécification géométrique, mais ayant une spécification dynamique différente, puis dans un deuxième temps à modifier cette dernière afin d'aboutir à la figure dynamique désirée. Un exemple d'une telle spécification dynamique différente est de considérer les points O, A et B instanciés et Oa et Ob demi-instanciés.

Exemple 10.2 TRIANGLE MÉDIANE

Construire un triangle ABC connaissant B, le milieu de AC, la longueur de la médiane issue de C et la direction de celle-ci.

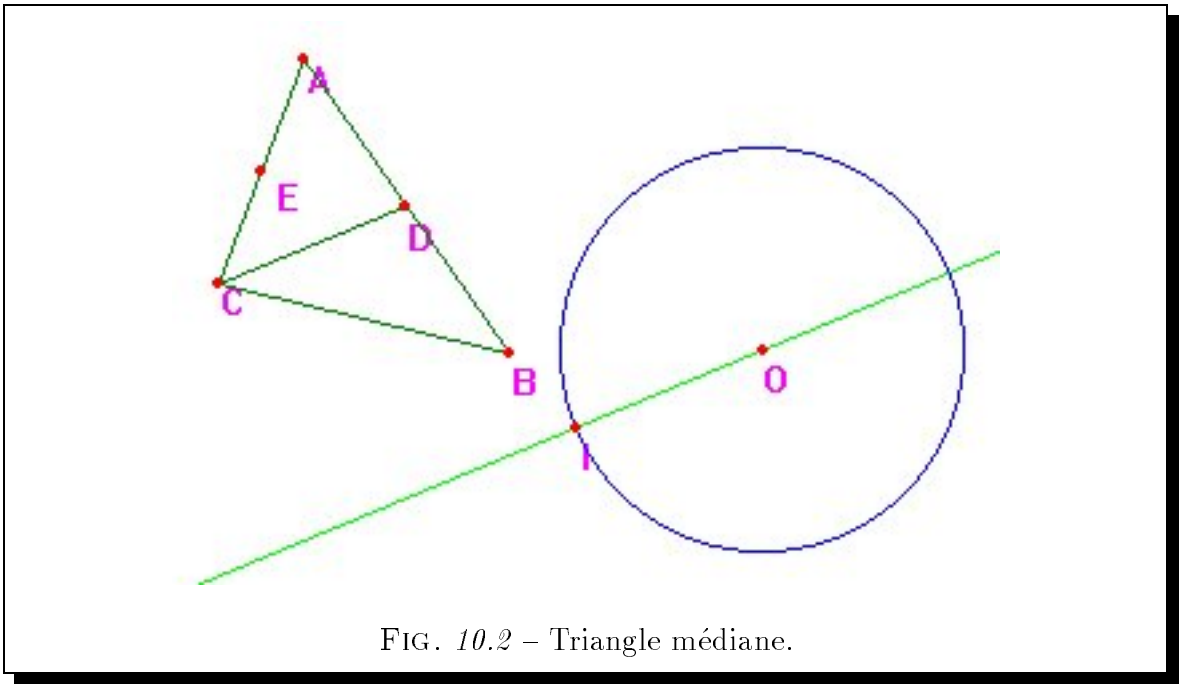

FIG. *10.2* – Triangle médiane.

La spécification ELDL de cette figure est :

$point(A) \wedge\ point(B) \wedge\ point(C) \wedge$

$point(D) \wedge\ point(E) \wedge\ point(O) \wedge\ point(I) \wedge$

$segment(S_{AB}, A, B, I_{AB}) \wedge\ segment(S_{AC}, A, C, I_{AC}) \wedge$

$segment(S_{BC}, B, C, I_{BC}) \wedge\ segment(S_{CD}, C, D, I_{CD}) \wedge$

$droite(D_{OI}) \wedge\ distance(d_{CD}) \wedge\ cercle(c, O, d_{CD}) \wedge$

$appDr(O, D_{OI}) \wedge\ appDr(I, D_{OI}) \wedge$

$milieu(D, A, B) \wedge\ milieu(E, A, C) \wedge$

$distPP(d_{CD}, C, D) \wedge\ appCc(I, c) \wedge\ par(D_{OI}, S_{CD}).$

La résolution de cette spécification géométrique à partir de la donnée des points O, I, B et E conduit au dessin illustré par la figure 10.2.

N°	**Construction**	**Agents intervenant**					**Tps (sec)**
		Lin	**Quad**	**Cpt obj**	**Cpt prop**	**Int**	
10.2	*1*	*X*	*X*				*0.16*
	suiv.	*X*	*X*				*0.16*

TAB. *10.2* – Agents intervenant dans la construction du problème appelé triangle médiane.

On remarque sur le tableau récapitulatif 10.2 que cette construction requiert l'intervention des agents linéaire et quadratique.

Cet exemple illustre le pouvoir de résolution de la combinaison des agents linéaire et quadratique. Il montre que la coopération de ces derniers permet de résoudre cet exercice difficile, dont la construction à l'aide de la règle et du compas est encore moins évidente [ACT98].

Exemple 10.3 CERCLE CIRCONSCRIT

Construire le cercle circonscrit à un triangle quelconque donné.

La spécification ELDL de cette figure est :

$point(A) \wedge\ point(B) \wedge\ point(C) \wedge\ segment(S_{AB}, A, B, I_{AB}) \wedge$

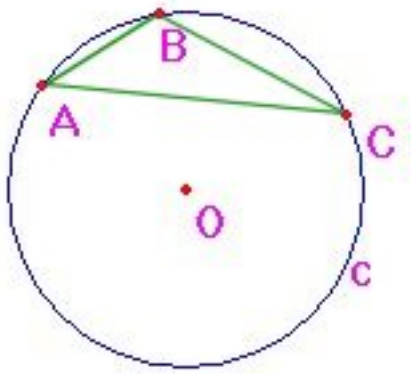

FIG. *10.3* – Cercle circonscrit.

$$point(O) \wedge segment(S_{AC}, A, C, I_{AC}) \wedge segment(S_{BC}, B, C, I_{BC}) \wedge$$
$$distance(r) \wedge cercle(c, O, r) \wedge$$
$$appCc(A, c) \wedge appCc(B, c) \wedge appCc(C, c) \wedge distPP(r, O, A).$$

La résolution de cette spécification géométrique à partir de la donnée des points A, B et C conduit au dessin illustré par la figure 10.3.

N°	***Construction***	***Agents intervenant***					***Tps (sec)***
		Lin	***Quad***	***Cpt obj***	***Cpt prop***	***Int***	
10.3	*1*	*X*	*X*	*X*			*0.17*
	suiv.	*X*	*X*				*0.09*

TAB. *10.3* – Agents intervenant dans la construction du problème appelé cercle circonscrit.

On remarque sur le tableau récapitulatif 10.3 que cette construction requiert l'intervention des agents linéaire, quadratique et complétion d'objets.

Cet exemple illustre l'intérêt de la coopération de résolveurs pour la résolution d'un problème de construction. En effet, l'ajout des médiatrices des côtés du triangle réduit ce problème à la construction de l'intersection de deux droites. Il est aussi typique de la "réversibilité" des constructions puisque suivant la spécification dynamique l'énoncé du problème correspond à construire le cercle circonscrit à un triangle donné (si les

points A, B et C sont instanciés), ou à inscrire un triangle dans un cercle donné (si par exemple les points O et A sont instanciés, et les points B et C demi-instanciés).

Exemple 10.4 BUTHION N° 45

Soient les points distincts A, B, O et R. Soit le cercle C de centre O passant par le point R. Construire les droites d et d′ passant respectivement par les points A et B telles que la droite d′ soit la médiatrice du segment [M_1, M_2] formé par les intersections de la droite d et du cercle C.

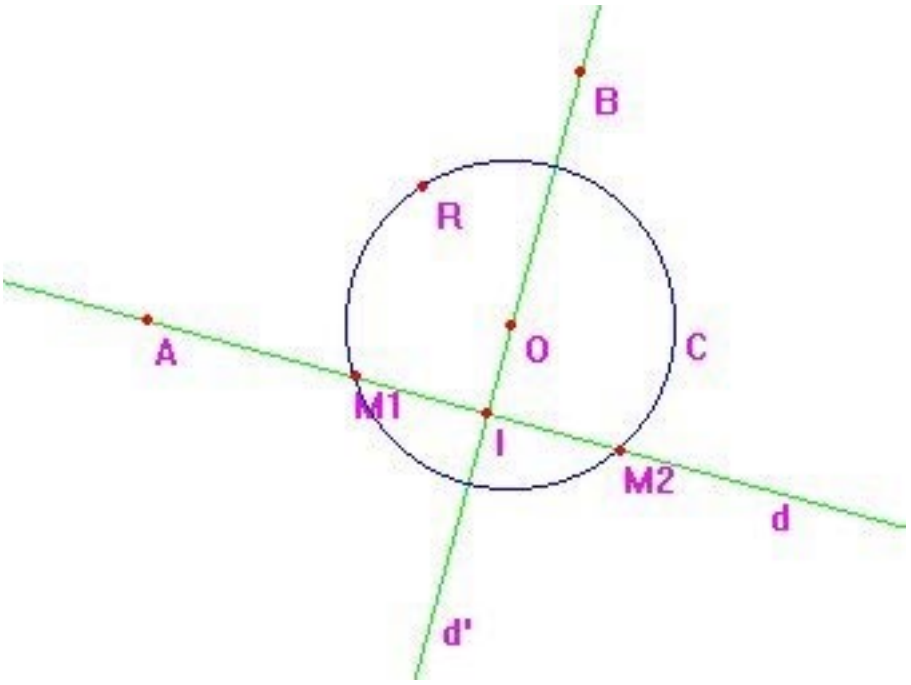

FIG. *10.4* – Buthion N° 45.

La spécification ELDL de cette figure est :

$point(A) \wedge\ point(B) \wedge\ point(O) \wedge\ point(R) \wedge$
$droite(d) \wedge\ droite(d') \wedge\ cercle(c, O, r) \wedge\ distance(r) \wedge$
$point(I) \wedge\ point(M_1) \wedge\ point(M_2) \wedge$
$appDr(A, d) \wedge\ appDr(B, d') \wedge$
$appDr(I, d) \wedge\ appDr(I, d') \wedge\ appDr(M_1, d) \wedge\ appDr(M_2, d) \wedge$
$appCc(M_1, c) \wedge\ appCc(M_2, c) \wedge\ appCc(R, c) \wedge\ distPP(r, O, R) \wedge$
$milieu(I, M_1, M_2) \wedge\ perp(d, d').$

La résolution de cette spécification géométrique à partir de la donnée des points A, B, O et R conduit au dessin illustré par la figure 10.4.

N°	Cons-truc-tion	Agents intervenant					Tps (sec)
		Lin	Quad	Cpt obj	Cpt prop	Int	
10.4	1	X	X	X	X		73.24
	suiv.	X	X				0.45

TAB. *10.4* – Agents intervenant dans la construction du problème appelé Buthion N° 45.

On remarque sur le tableau récapitulatif 10.4 que cette construction requiert l'intervention des agents linéaire, quadratique, complétion d'objets et complétion de propriétés.

Sur cet exemple, l'intérêt des agents de complétion d'objets et de propriétés ressort particulièrement puisque leurs actions aboutissent à la linéarisation, ou quasi-linéarisation, de la figure. Plus précisément, l'ajout de la propriété redondante d'appartenance du point O à la droite d′ réduit respectivement les problèmes de la construction des droites d′ et d, aux problèmes de la construction d'une droite passant par deux points et au problème de la construction d'une droite perpendiculaire à une droite passant par un point. Cet ajout rend de plus cet exercice soluble par notre agent règle et compas ce qui est d'autant plus intéressant pour les performances d'animation.

Exemple 10.5 TRIANGLE ISOCÈLE RECTANGLE

Construire un triangle isocèle rectangle en A connaissant C, la direction BC et la longueur BC.

La spécification ELDL de cette figure est :

$$\begin{aligned}
&\mathit{point}(A) \wedge \mathit{point}(B) \wedge \mathit{point}(C) \wedge \mathit{distance}(d) \wedge \mathit{point}(I) \wedge \mathit{point}(J) \wedge \\
&\mathit{droite}(D_{AB}) \wedge \mathit{droite}(D_{AC}) \wedge \mathit{droite}(D_{BC}) \wedge \mathit{droite}(D_{IJ}) \wedge \\
&\mathit{distance}(d_{BC}) \wedge \mathit{distPP}(d_{BC}, I, J) \wedge \mathit{distPP}(d_{BC}, B, C) \wedge \\
&\mathit{appDr}(A, D_{AB}) \wedge \mathit{appDr}(A, D_{AC}) \wedge \mathit{appDr}(B, D_{AB}) \wedge \\
&\mathit{appDr}(B, D_{BC}) \wedge \mathit{appDr}(C, D_{AC}) \wedge \mathit{appDr}(C, D_{BC}) \wedge \\
&\mathit{appDr}(I, D_{IJ}) \wedge \mathit{appDr}(J, D_{IJ}) \wedge \mathit{par}(D_{IJ}, D_{BC}) \wedge \\
&\mathit{distPP}(d, A, B) \wedge \mathit{distPP}(d, A, C) \wedge \mathit{perp}(D_{AB}, D_{AC}).
\end{aligned}$$

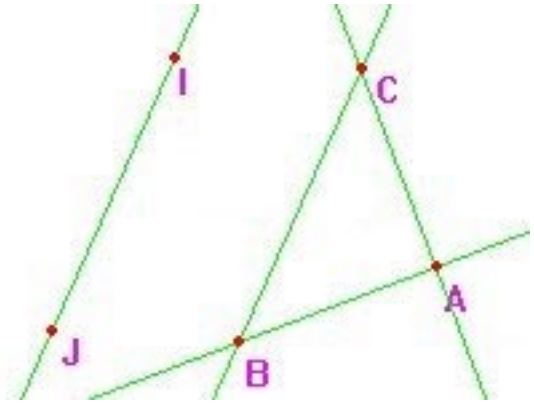

FIG. *10.5* – Triangle isocèle rectangle en A.

La résolution de cette spécification géométrique à partir de la donnée des points I, J et C conduit au dessin illustré par la figure 10.5.

N°	***Construction***	***Agents intervenant***					***Tps (sec)***
		Lin	***Quad***	***Cpt obj***	***Cpt prop***	***Int***	
10.5	*1*	*X*	*X*	*X*		*X*	*0.8*
	suiv.	*X*	*X*			*X*	*0.8*

TAB. *10.5* – Agents intervenant dans la construction d'un triangle isocèle rectangle.

On remarque sur le tableau récapitulatif 10.5 que cette construction requiert l'intervention de quatre agents.

- *l'agent linéaire construit la droite BC.*
- *l'agent quadratique détermine le point B.*
- *l'agent complétion d'objets rajoute la médiatrice de BC.*
- *l'agent intervalle approxime le point A et les droites AB et AC.*

L'intérêt de la complétion d'objets et de propriétés est moins flagrant sur cet exemple. En effet dans une telle situation, l'ajout d'objets et/ou de propriétés redondantes ne permettent pas d'aboutir à la linéarisation, ou quasi-linéarisation, de la figure. On note alors que la coopération des résolveurs ne permet pas d'améliorations sensibles des temps des constructions succédant à la première dans le cas où il y a

intervention des agents complétion d'objets et intervalle. Ceci est lié au coût des énumérations d'intervalles nécessaires à l'obtention d'une figure acceptable. Néanmoins, la coopération des différents résolveurs permet au moins de résoudre cet exercice de construction difficile.

10.1.2 Exemples de non résolution

Comme ou pourait s'y attendre, GDRev ne résout pas tous les problèmes de construction que nous lui avons proposés. Par exemple, il ne peut, étant donné un triangle ABC, construire trois cercles inscrits dans les angles $\widehat{A}$, $\widehat{B}$ et $\widehat{C}$ du triangle et tels que chacun d'eux soit tangent aux deux autres. Ce problème est très difficile.

Nous pouvons dire que ces problèmes dépassent en fait les capacités du résolveur sur intervalles et que dans cette mesure, il s'agit de problèmes difficiles. Appartiennent à cette catégorie, des problèmes "cycliques" tels que les suivants :

- Inscrire un triangle rectangle dans un cercle dont les côtés passent chacun par un point donné.
- Construire un rectangle dont les 4 côtés passent chacun par un point donné et dont les diagonales aient une longueur donnée.

10.2 Résultats concernant la recherche automatique d'une construction "linéaire"

Afin d'apprécier l'intérêt des heuristiques de construction automatique de figures géométriques à partir d'un ensemble quelconque d'objets de base (cf paragraphe 7.5.3 page 129), nous avons sélectionné un ensemble de 193 exercices parmi les 512 exercices que compte l'ouvrage de Chou S-C [Cho88]. Le choix de ces exercices a été influencé par les résultats obtenus manuellement par Bouhineau D [Bou97] sur la même problématique. En particulier, nous avons testé tous les problèmes de construction nécessitant des extensions algébriques même si les constructions ont subi des modifications locales et/ou globales, et tous les exercices dont les spécifications algébriques correspondantes comportent plus de 3 équations du second degré.

Le tableau 10.6 résume les résultats obtenus, mais les détails de ces derniers sont donnés dans l'annexe C.

	Echec	0	1	2	3	4	5	6	7	8	9	10	11	12
Complexité des problèmes étudiés														
Nb		12	12	13	29	56	41	10	1	8	5	5		1
Résultats de Bouhineau D														
Loc		122	57	11	3									
Glob		173	20											
Résultats de nos heuristiques														
Con	66	55	27	26	16	2		1						
Plus	29	67	40	29	19	9								

Tab. 10.6 – *Résumé des résultats obtenus pour une reconstruction quelconque.*

Dans le tableau 10.6 nous donnons à la ligne intitulée *Nb* la complexité des problèmes étudiés en terme du nombre de situations dont les spécifications algébriques correspondantes comportent n équations du second degré. Les résultats obtenus manuellement par Bouhineau D sont donnés grâce à la ligne intitulée *Loc* (resp. *Glob*) qui fournit le nombre de situations nécessitant n extensions algébriques pour une construction modifiée localement (resp. localement et globalement). Enfin, nous donnons les résultats des heuristiques décrites dans le paragraphe 7.5.3 page 129. La ligne intitulée *Con* (resp. *Plus*) résume le nombre de situations nécessitant n extensions algébriques pour des constructions effectuées à l'aide de l'heuristique dite du *non construit* (resp. du *plus contraint*). La colonne *Echec* comptabilise le nombre de situations qui n'ont pu être reconstruites aléatoirement à partir de l'ensemble d'objets de base déterminé par l'heuristique. Un exemple typique et fréquent de tels échecs est la situation qui consiste à placer aléatoirement 3 points sur un cercle pour former un triangle équilatéral.

Dans [Bou97], Bouhineau D propose un critère d'acceptabilité d'une construction. Selon ce dernier, une figure est de complexité acceptable si sa construction modifiée localement et/ou globalement introduit au plus 4 extensions algébriques. Ce critère est lié à la complexité exponentielle en temps et en espace d'une arithmétique exacte prenant en compte les racines carrées. Il correspond à un coût supplémentaire d'un facteur 1000 par rapport à une arithmétique exacte sur les rationnels.

Avec ce critère d'acceptabilité, toutes les situations traitées à l'aide de l'heuristique dite du *plus contraint* sont acceptables. Seulement 1 situation conduit à un excès à l'aide de l'heuristique dite du *non construit.* Ces résultats sont d'autant plus importants que l'échantillon des situations traitées sur les 512 situations répertoriées comprend au moins toutes celles susceptibles d'introduire plus de 4 extensions algé-

briques, c'est-à-dire toutes celles dont les spécifications algébriques correspondantes comportent plus de 3 équations du second degré, c'est-à-dire toutes celles susceptibles de ne pas être acceptées. Plus précisément, les constructions effectuées à l'aide de l'heuristique dite du *plus contraint* (resp. du *non construit*) conduisent à des constructions rationnelles dans près des 2/5 (resp. dans près des 4/9) des cas lorsqu'elles aboutissent.

Ces résultats montre la faisabilité de l'automatisation de la construction exacte de figures géométriques à partir d'un ensemble quelconque d'objets de base. Plus particulièrement, ils montrent qu'il est pratiquement possible de déterminer de manière déterministe dans bon nombre de cas un ensemble d'objets de base conduisant à une construction introduisant un nombre acceptable d'extensions algébriques. L'intérêt de ceci est qu'il devient ainsi possible de vérifier numériquement mais exactement sur une figure la véracité d'une ou d'un ensemble de propriétés.

Ecart	**0**	**+1**	**+2**	**+3**	**+4**	**+5**	**+6**	**Total**
HEURISTIQUES								
non construit	61	24	24	15	2		1	127
plus contraint	70	43	26	17	8			164

TAB. 10.7 – *Heuristiques de construction quelconque Vs Construction modifiée globalement.*

Le tableau 10.7 compare les résultats des heuristiques vis à vis de la solution optimale. Cette comparaison se fait au travers du nombre de situations introduisant plus d'extensions algébriques que nécessaire lors de la reconstruction. Il nous laisse penser que sur les expérimentations menées l'heuristique dite du *plus contraint* fournit de meilleurs résultats que l'heuristique dite du *non construit*. Tout d'abord, elle s'écarte moins de la solution optimale puisqu'elle aboutit à celle-ci ou à une solution introduisant 1 extension de plus dans 69% des cas contre 67% des cas. Enfin mais surtout, elle réussit plus souvent à construire la figure.

10.3 Conclusion

Les résultats présentés dans ce chapitre apparaissent très encourageants.

Tout d'abord sur le problème de résolution de contraintes géométriques, la coopération d'agents spécifiques a permis d'une part de résoudre des problèmes réputés difficiles comme les exemples 10.5 et 10.4 et d'autre part de réduire considérablement

les temps de résolution comme par exemple pour le problème 10.4. Cette réduction appréciable permet de supporter l'animation de telles figures dynamiques.

Concernant le problème de la construction automatique de figures géométriques à partir d'un ensemble quelconque d'objets de base avec pour objectif d'introduire un minimum d'extensions algébriques, c'est-à-dire d'aboutir à un minimum de calcul de racines carrées, les heuristiques définies nous ont permis d'obtenir de manière déterministe la solution optimale ou une solution proche de cette dernière, c'est-à-dire une solution introduisant 1 extension de plus que nécessaire, dans près de 69% des exercices considérés. De plus, si la construction a abouti, elle a toujours (pour l'heuristique dite du *plus contraint*) ou presque toujours (pour l'heuristique dite du *non construit*) introduit moins d'extensions algébriques que le seuil acceptable. Ceci nous permet d'envisager la définition d'une fonctionnalité de vérification numérique mais exacte de propriétés sur une figure.

Quatrième partie

Autre application

Chapitre 11

Système préceptoriel de géométrie : le prototype SPhinx

Le point de départ de cette étude a été le constat que les environnements informatisés d'apprentissage humain sont principalement centrés sur l'apprenant. L'enseignant a dans de tels systèmes un rôle secondaire. Si ce rôle secondaire laissé à l'enseignant n'a que peu d'importance dans le cadre d'un enseignement de la géométrie en classe, ce manque étant comblé par la présence physique des deux intervenants, il n'en est pas de même dans le cadre d'un enseignement à distance. En effet, dans une telle situation, la disponibilité du professeur est primordiale afin que l'élève ne s'interroge pas sur la fiabilité et l'efficacité de tels systèmes distribués. De plus pour des raisons économiques, la réduction du temps de télé-présence du professeur est d'un intérêt non négligeable. C'est pourquoi, notre démarche consiste à apporter une aide à un enseignant au cours de l'expérimentation d'une situation pédagogique grâce à l'utilisation d'un système de diagnostic et de mise en évidence d'erreurs de construction [BL96]. Plus spécialement, nous nous intéressons aux situations d'enseignement de géométrie. L'aide proposée repose sur la construction automatique d'exemples et de contre-exemples géométriques à partir de la figure de l'apprenant lorsque l'analyse de celle-ci, conformément à l'énoncé du problème donné par le professeur, a révélé une erreur.

Nous étudions dans ce chapitre la notion de "préceptoriel" comme système d'aide à l'enseignant en prenant comme exemple les logiciels de construction de figures géométriques. Nous analysons dans la section 11.1 le cadre de travail. L'analyse d'un tel cadre de travail nous conduit à définir dans la section 11.2 les concepts de contre-exemples et d'exemples géométriques face à une construction erronée. Nous intégrons ensuite

ces concepts dans un schéma de processus d'aide et nous étudions la problématique de cette intégration dans la section 11.3. Nous présentons dans la section 11.4 le prototype d'aide que nous avons développé. Nous illustrons ces concepts dans la section 11.5 au travers un exemple de scénario. Enfin, la section 11.6 conclue ce chapitre.

Le travail présenté ici a été présenté aux journées EIAO'97 de Cachan [Cha97] et à ITS'96 [CB96].

11.1 Analyse du cadre de travail

Le cadre de cette étude est la construction de figures géométriques au moyen d'environnements informatisés d'apprentissage humain. Cette activité de construction peut être une activité en soi afin de vérifier que l'élève a bien compris les hypothèses et les conclusions du problème [Des94]. Mais, elle peut aussi être le support d'une activité de découverte et de démonstration de propriétés géométriques [ANT90].

En nous remémorant le principe énoncé par Laborde C dans [Lab93] et présenté dans le paragraphe 1.1, il s'ensuit que la production d'un élève est double. Elle est d'une part visuelle avec le dessin rapporté à l'écran, et d'autre part géométrique avec le procédé de construction suivi.

On peut classer ces doubles productions d'un élève dans l'une des trois catégories suivantes :

- les dessins visuellement justes et construits correctement.
- les dessins visuellement justes et construits incorrectement.
- les dessins visuellement non justes et donc construits incorrectement.

Comme énoncé précédemment par Laborde C, la figure de l'étudiant doit respecter la classe d'animation du problème posé par le professeur. Elle doit naturellement contenir toutes les propriétés demandées, mais aussi seulement les propriétés demandées. En effet, les enseignants de géométrie ne se satisfont pas de figures particulières. Ce rejet est motivé par la volonté d'éviter les fausses conjectures des élèves induites par des particularismes. Ainsi, les deux types d'erreurs retrouvées sont les :

- sous-contraintes : lorsqu'une propriété demandée n'est pas satisfaite.

– sur-contraintes : lorsqu'une propriété est satisfaite alors qu'elle n'est pas demandée.

Les moyens d'investigation mis à la disposition du professeur pour vérifier les productions des apprenants sont au nombre de trois. Bonnet J-F les mentionne dans [Bon93]. Il s'agit de l'utilisation de l'"historique", de la manipulation de tous les objets de base de la figure, et de l'affichage de tous les éléments de la figure. Comme moyen de test, Laborde C [Lab93] propose l'utilisation d'une macro construction. Le test consiste à superposer la figure de l'élève avec le résultat de l'application de la macro construction du professeur avec comme arguments les données du problème.

Notre objectif est de compléter ces moyens d'investigation ou de test par un système de production automatique de contre-exemples ou d'exemples géométriques dans les cas de figures sous et/ou sur-contraintes.

Exemple 11.1 CONTRE-EXEMPLE ET EXEMPLE GÉOMÉTRIQUE

Le problème posé à l'élève consiste à construire un triangle ABC rectangle en A. Il répond à cet exercice par la construction :

1. *Création du cercle G de centre A passant par le point B,*
2. *Construction d'un point C sur le cercle G,*
3. *Construction des segment [A, B], [B, C] et [A, C].*

Cette construction est erronée même si visuellement elle semble répondre à l'attente du professeur. L'élève a commis ici deux erreurs : l'absence d'orthogonalité en A et la présence d'égalité de longueur des segments [A, B] et [A, C].

Pour mettre en évidence la première erreur, on peut proposer un contre-exemple obtenu par rotation du point C autour de A. Afin de montrer le particularisme des segments [A, B] et [B, C], on peut afficher un exemple de triangle rectangle quelconque, comme illustré sur la figure 11.1.

Il convient de signaler qu'un tel système d'aide peut tout aussi bien faire partie d'un système tutoriel intelligent pour y être utilisé directement par un apprenant au cours de l'expérimentation d'une situation pédagogique. Il ne nous appartient pas de juger du caractère didactique d'un tel système. On trouve des exemples de systèmes de diagnostic et de génération de contre-exemples pour l'algèbre dans Aplusix [NJNG93] et PG [Eve89].

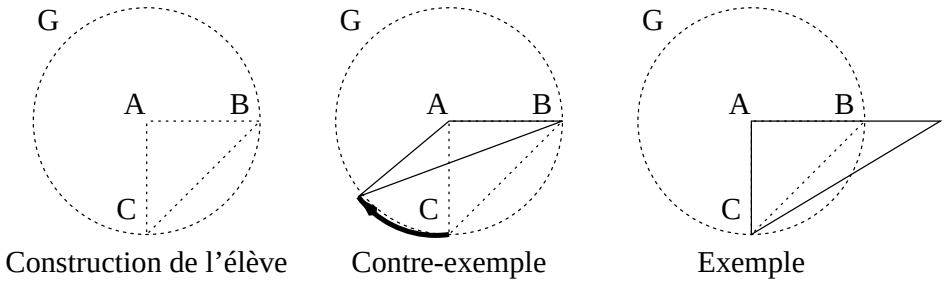

FIG. *11.1* – Exemple de construction erronée, contre-exemple et exemple.

11.2 Exemples et contre-exemples géométriques

L'analyse du cadre de travail, dans lequel nous nous plaçons, nous conduit à définir dans les paragraphes qui suivent les concepts de contre-exemples et d'exemples géométriques face à une construction erronée.

11.2.1 Qu'est-ce qu'un contre-exemple ?

Un contre-exemple est une figure mettant clairement en évidence les insuffisances d'une figure sous-contrainte ou sous et sur-contrainte. Au regard du principe énoncé par Laborde J-M :

> *"... une figure n'est acceptée comme contre-exemple que si l'utilisateur perçoit le chemin qui conduit de "sa" configuration à celle proposée comme contre-exemple par l'environnement."*
> [Lab95] (page 37)

il apparaît qu'un contre-exemple est obtenu par ajout à la spécification issue de la construction de l'élève, d'une propriété antagoniste à la propriété manquante. De plus, afin de lier la figure de l'élève au contre-exemple, un préceptoriel doit proposer des animations, des déformations, menant de la première à la deuxième. Nous présentons un exemple d'une telle obtention et de telles déformations dans le paragraphe 11.5.

11.2.2 Qu'est-ce qu'un exemple ?

Un exemple est une figure montrant clairement les particularismes d'une figure sur-contrainte ou sur et sous-contrainte. De même que précédemment, pour être ac-

cepté l'exemple doit être bâti à partir de la figure de l'élève par libération de la ou des contraintes supplémentaires. Toutefois, il n'est pas possible dans ce cas de proposer une animation menant de la figure de l'élève à l'exemple puisque sur-contrainte. C'est pourquoi, afin de les lier, un préceptoriel doit inviter le professeur à se rendre compte par lui même de l'impossibilité de déformer la figure de l'élève jusqu'à obtenir l'exemple.

11.3 Processus d'aide en géométrie

Afin de poursuivre l'étude qui précède, nous examinons quel peut être le schéma d'un processus d'aide d'un préceptoriel de géométrie. Ensuite, nous discutons de la problématique engendrée par celui-ci.

11.3.1 Schéma d'un processus d'aide en géométrie

La figure 11.2 illustre le schéma général que nous proposons d'un processus d'aide en géométrie au vue de l'analyse précédente. Dans une première étape, le préceptoriel analyse la figure de l'élève conformément à la spécification du problème de construction donné par le professeur. Suite à cette analyse, le système propose par défaut la mise en évidence d'une erreur. Eventuellement, le professeur peut demander interactivement à ce qu'une autre erreur soit mise en évidence. Une fois l'erreur choisie, le préceptoriel construit l'exemple ou le contre-exemple associé. Dans le cas de la construction d'un exemple, le système invite ensuite le professeur à se rendre compte par lui même de l'impossibilité de déformer la figure de l'élève jusqu'à obtenir l'exemple construit. Dans le cas de la construction d'un contre-exemple, le système choisit et exécute par défaut des animations menant de la figure de l'élève au contre-exemple construit. Eventuellement, le professeur peut demander à ce que d'autres animations soient opérées.

11.3.2 Problématique d'un processus d'aide en géométrie

Un tel processus d'aide pose de multiples questions relatives au diagnostic des erreurs et à l'obtention d'exemples ou de contre-exemples.

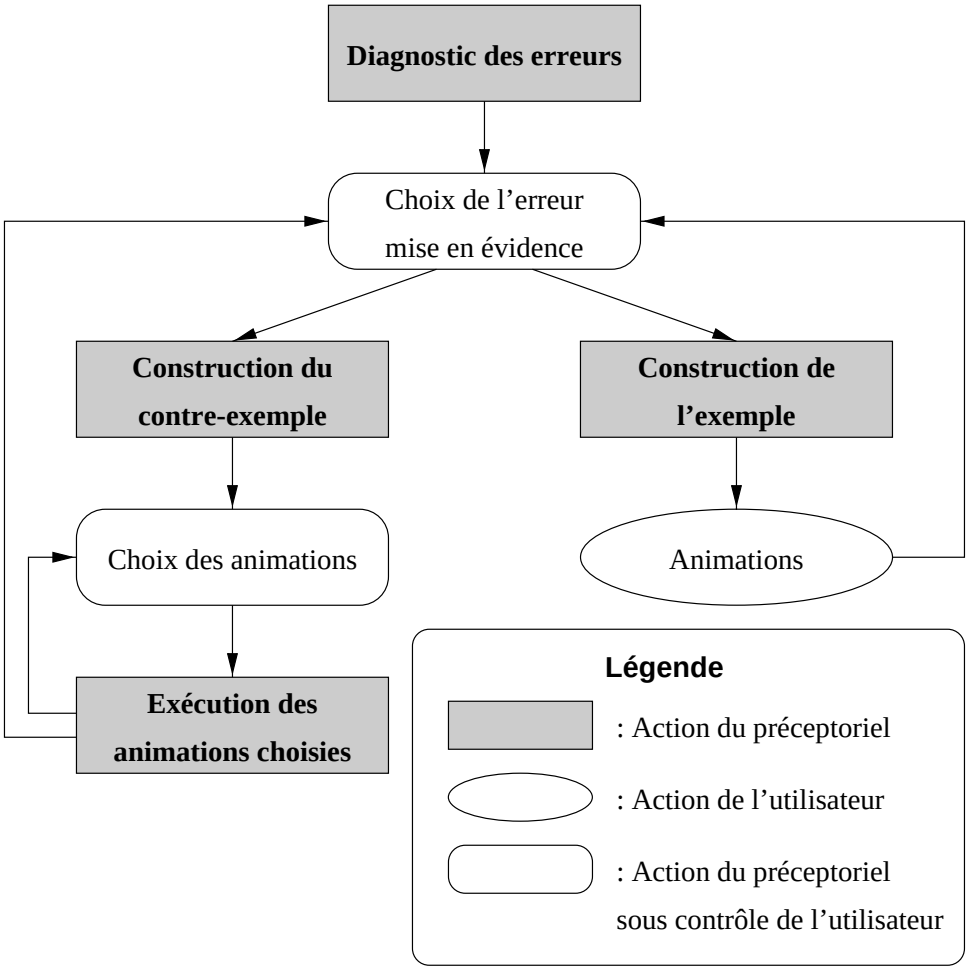

FIG. 11.2 – *Schéma d'un processus d'aide en géométrie dynamique.*

Problématique du diagnostic

Le problème de la validité d'une construction géométrique vis à vis d'une spécification soulève deux questions.

– Quel type de diagnostic opérer?

En percevant les erreurs des apprenants non plus seulement comme des fautes mais comme significatives des connaissances de ces derniers, il s'ensuit une augmentation conséquente de l'importance des systèmes de diagnostic. Les premières méthodes utilisées ont consisté en la construction d'une base d'erreurs. Toutefois, dans les systèmes ouvert comme les micro-mondes tels que Cabri-Géomètre, cette constitution n'est pas envisageable. Les auteurs de [NP93] apportent une

solution en prenant en compte le contexte composé de l'ensemble des solutions du problème et de l'état de la résolution. Une autre solution consiste à identifier et décrire des classes de comportements corrects et incorrects afin d'opérer des diagnostics de type classification heuristique [Cla85]. Cette solution a déjà été mise en œuvre dans le système Schnaps [BTS95].

Notre approche consiste à vérifier sur la construction le respect des propriétés du problème. On trouve répertoriées dans le tableau 11.1 les erreurs possibles dans les cas de propriétés manquantes.

– Quel est le schéma du processus de diagnostic de solutions particulières?

Pour résoudre le problème de la détection de solutions particulières, on se heurte à la recherche des propriétés redondantes d'une figure. Une figure contient des propriétés redondantes si des éléments de cette dernière sont liés entre eux non pas par des propriétés présentes dans la spécification, mais par des propriétés issues de la construction[1]. Ainsi, pour ne pas opérer de faux diagnostics en considérant particulière une propriété normalement issue de la construction, il convient de déterminer toutes les propriétés redondantes du problème avant de rechercher les particularités d'une figure d'un apprenant. De plus, on peut remarquer qu'un apprenant peut faire intervenir dans sa construction des objets non spécifiés. Il en résulte que la recherche des particularismes d'une figure d'un élève doit faire intervenir seulement les objets du problème contenus dans celle-ci.

Ces problèmes peuvent être abordés suivants des méthodes géométriques [Des95], algébriques [Wu94], numériques exactes [Bou96] ou enfin numériques approchées correctes [Ben94]. Notre préférence actuelle va en faveur des méthodes numériques approchées correctes dans la mesure où ces méthodes s'avèrent rapides, correctes et prennent en compte le dessin.

Problématique de l'obtention d'exemples ou de contre-exemples

Une fois les erreurs commises détectées, il s'agit, en gardant à l'esprit le principe de Laborde J-M précédemment énoncé dans le paragraphe 11.2.1 de :

– choisir parmi les erreurs commises par l'élève la ou les erreurs à mettre en évidence.

1. Par exemple, la figure constituée d'une droite AB passant par les points A et B et de deux droites D et D' perpendiculaires à la droite AB passant respectivement par les points A et B, a comme propriété redondante le parallélisme entre les droites D et D'.

Au regard d'expérimentations, il apparaît préférable de montrer une seule erreur à la fois. En effet, dans le cas contraire il n'est pas facile de comprendre le cheminement qui conduit de la figure de l'élève au contre-exemple ou à l'exemple. Il se pose alors le problème du choix du type de l'erreur à mettre en évidence. Le choix le plus immédiat est en faveur des propriétés manquantes d'une part parce qu'il est moins probable que l'apprenant construise une figure particulière et d'autre part parce que l'apprenant confond souvent deux notions ce qui lie les sur-contraintes à des sous-contraintes. Enfin, il se pose le problème du choix proprement dit de l'erreur à exhiber.

Tous ces choix sont effectués par défaut par le système sous le contrôle toutefois du professeur par l'intermédiaire des préférences qu'il peut exprimer. L'intérêt de tels comportements par défaut est double. Ils permettent au professeur d'une part de se décharger de ces contraintes de choix et d'autre part de définir par là même ses préférences stylistiques et didactiques.

– choisir la manière de mettre en évidence une erreur.

Ce choix doit consister dans le cas d'une sous-contrainte en une correspondance entre les classes d'erreurs et les propriétés antagonistes ajoutées. Un exemple d'une telle correspondance est donné par le tableau 11.1.

Propriété manquante	**Propriété antagoniste ajoutée**
Un point n'appartient pas à une droite	Le point est placé à une distance supérieure à un seuil fixé de la droite
Un point n'appartient pas à un segment	Le point est placé à une distance supérieure à un seuil fixé du segment
Un point n'appartient pas à un cercle	Le point est placé à une distance supérieure au diamètre du cercle, du centre de ce dernier
Un objet n'est pas parallèles à un autre objet	Les objets sont spécifiés perpendiculaires
Un objet n'est pas perpendiculaires à un autre objet	Les objets sont spécifiés parallèles
Un point n'est pas le milieu d'un segment	Le point est placé à une distance d'un quart de la longueur du segment, d'une extrémité de celui-ci

TAB. 11.1 – *Correspondance entre propriété manquante et propriété antagoniste ajoutée.*

Dans le cas d'une sur-contrainte, deux situations se présentent :

- Si la figure contient uniquement des sur-contraintes, il faut libérer l'erreur sélectionnée.
- Sinon, il faut d'une part libérer l'erreur sélectionnée et d'autre part rajouter les propriétés manquantes afin de montrer à l'élève un exemple correct.

11.4 Présentation du prototype

Les difficultés rencontrées dans l'implémentation d'une première maquette [CB96] nous ont conduit à définir une nouvelle architecture et par conséquent à revoir les phases d'analyse d'une figure.

11.4.1 Architecture

Pour concevoir l'architecture[2] de la seconde version de notre maquette illustrée par la figure 11.3, nous avons cherché à minimiser les problèmes d'interopérabilité. C'est pourquoi cette architecture s'articule autour d'une part de Cabri-Géomètre pour l'acquisition et l'animation de la figure, et d'autre part d'un système, appelé SPhinx, entièrement développé en BNR-Prolog [OV90] pour le diagnostic, la génération de la spécification d'exemples ou de contre-exemples et la construction de ces derniers.

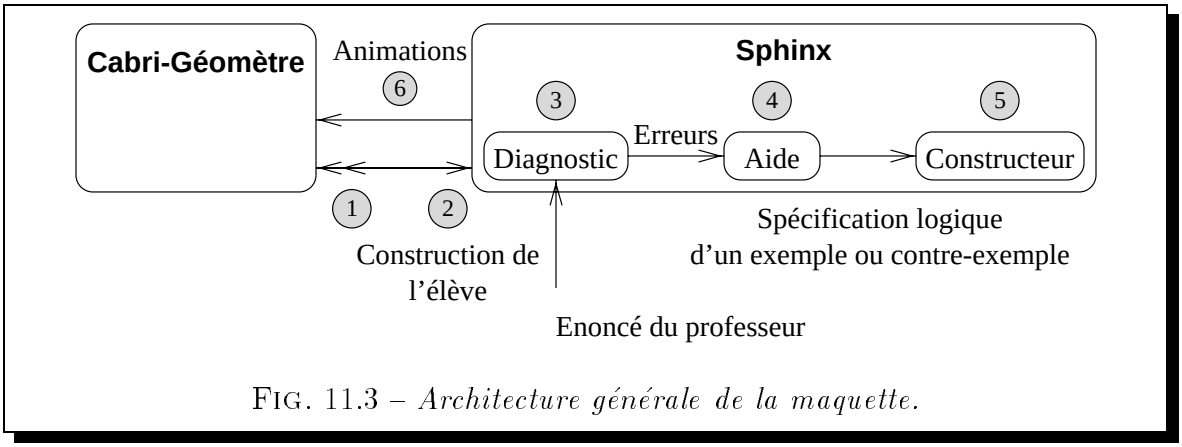

FIG. 11.3 – *Architecture générale de la maquette.*

2. Cette nouvelle architecture a été rendue possible par l'utilisation de la dernière version de BNR-Prolog autorisant la définition de prédicats externes, écrits notamment dans le langage C. Ces prédicats externes permettent l'établissement d'un dialogue entre Cabri-Géomètre et le préceptoriel au travers d'évènements du système Apple (*AppleEvents*).

11.4.2 Phases d'analyse

On retrouve numérotées sur la figure 11.3 les phases de l'analyse d'une construction. Dans une phase préliminaire le professeur donne une spécification logique du problème. Les phases suivantes s'articulent ainsi :

1. L'élève construit à l'aide de Cabri-Géomètre, une figure supposée répondre au problème énoncé par le professeur.
2. La figure construite est récupérée par le système SPhinx afin de déterminer la spécification logique géométrique de cette figure.
3. Le diagnostic de la figure est opéré conformément aux spécifications du problème données par le professeur.
4. Les erreurs détectées sont transmises à un module d'aide qui sélectionne l'erreur à mettre en évidence et détermine, à partir de la spécification issue de la construction de l'élève et de l'erreur à exhiber choisie, la spécification logique géométrique de l'exemple ou du contre-exemple associé.
5. L'exemple ou le contre-exemple est construit par le module de construction automatique de figures géométriques du système SPhinx.
6. Le professeur commande les animations déterminées à Cabri-Géomètre.

11.4.3 Description des composants

Les choix conceptuels retenus sont les suivants pour le composant :

- de diagnostic :

  L'approche que nous avons choisie pour effectuer le diagnostic est l'approche numérique approchée pour sa simplicité de mise en œuvre et sa rapidité. Il s'effectue en deux étapes. Après une vérification du respect sur le dessin des propriétés du problème, une recherche par saturation des particularismes de la figure est engagée.

- d'aide à la spécification d'exemple et de contre-exemples :

  La contrainte antagoniste imposée à un élément sous-spécifié pour la construction d'un contre-exemple est celle illustrée par le tableau 11.1.

Parmi les différents types d'animations imaginables (translation, rotation), le système produit des translations horizontales et verticales. Ceci nous est apparu être la meilleure façon de percevoir les transformations. La mise en œuvre en est ainsi simplifiée.

– de construction automatique de figure géométrique :

La construction automatique des exemples et contre-exemples est basée sur une approche numérique approchée correcte à l'aide de la programmation logique avec contraintes sur intervalles.

11.4.4 Aspects quantitatifs

Dans son ensemble, la mise en œuvre de la maquette SPhinx représente :

– l'écriture de 13.000 lignes de BNR-Prolog contenues dans 42 fichiers,

– l'écriture de 4.000 lignes de C contenues dans 12 fichiers,

– la définition de 680 prédicats,

– une mise en œuvre d'une durée de 4 mois homme,

– un code de 1.3 Mo,

Ces quelques chiffres confirment l'intérêt d'user d'un tel langage de programmation dont les principaux avantages sont la production d'un code d'une remarquable concision, la facile correction des erreurs de programmation, et l'aisance de la maintenance du code. De plus, l'usage de ce langage de programmation de haut niveau nous a permis d'expérimenter rapidement tout au long de la mise en œuvre de nouvelles heuristiques.

11.5 Exemple de scénario d'aide

Afin d'illustrer le processus d'aide combinant la génération d'exemples et de contre-exemples, nous considérons le scénario typque suivant.

Exemple 11.4 Scénario d'aide

Le problème donné consiste à construire la médiane du triangle ABC issue du sommet A.

La spécification logique de ce problème est :

$$\begin{aligned}
&\mathit{point}(A) \wedge \ \mathit{point}(B) \wedge \ \mathit{point}(C) \wedge \ \mathit{point}(H) \wedge \\
&\mathit{droite}(D_{AB}) \wedge \ \mathit{droite}(D_{AC}) \wedge \ \mathit{droite}(D_{BC}) \wedge \ \mathit{droite}(D_{AH}) \wedge \\
&\mathit{appDr}(A,\ D_{AB}) \wedge \ \mathit{appDr}(B,\ D_{AB}) \wedge \\
&\mathit{appDr}(A,\ D_{AC}) \wedge \ \mathit{appDr}(C,\ D_{AC}) \wedge \\
&\mathit{appDr}(B,\ D_{BC}) \wedge \ \mathit{appDr}(C,\ D_{BC}) \wedge \\
&\mathit{appDr}(A,\ D_{AH}) \wedge \ \mathit{appDr}(H,\ D_{AH}) \wedge \\
&\mathit{appDr}(H,\ D_{BC}) \wedge \ \mathit{milieu}(H,\ B,\ C).
\end{aligned}$$

1. *La construction fournie par l'élève, illustrée en trait plein sur la figure 11.4, est la suivante :*

   1 Créer les points A, B et C quelconques,

   2 Construire les segments $[A,\ B]$, $[A,\ C]$ et $[B,\ C]$,

   3 Construire la droite perpendiculaire au segment $[B,\ C]$ passant par le point A,

   4 Construire le point d'intersection H de la droite perpendiculaire au segment $[B,\ C]$ passant par le point A et du segment $[B,\ C]$.

   L'erreur commise par l'élève réside dans la confusion entre la notion de hauteur et la notion de médiane dans un triangle.

2. *Si le dessin semble répondre au problème, le point H étant proche du milieu du segment $[B,\ C]$, c'est parce qu'il s'agit d'un dessin où le triangle ABC semble isocèle en A. La construction de l'élève correspond à la spécification logique suivante :*

$$\begin{aligned}
&\mathit{point}(A) \wedge \ \mathit{point}(B) \wedge \ \mathit{point}(C) \wedge \ \mathit{point}(H) \wedge \\
&\mathit{droite}(D_{AB}) \wedge \ \mathit{droite}(D_{AC}) \wedge \ \mathit{droite}(D_{BC}) \wedge \ \mathit{droite}(D_{AH}) \wedge \\
&\mathit{appDr}(A,\ D_{AB}) \wedge \ \mathit{appDr}(B,\ D_{AB}) \wedge \\
&\mathit{appDr}(A,\ D_{AC}) \wedge \ \mathit{appDr}(C,\ D_{AC}) \wedge \\
&\mathit{appDr}(B,\ D_{BC}) \wedge \ \mathit{appDr}(C,\ D_{BC}) \wedge
\end{aligned}$$

$appDr(A, D_{AH}) \wedge\ appDr(H, D_{AH}) \wedge$
$appDr(H, D_{BC}) \wedge\ perp(D_{AH}, D_{BC}).$

Cette spécification ne répond pas au problème posé. En effet, cette figure est incorrecte d'une part parce qu'elle n'implique pas que le point H soit le milieu du segment [B, C], et d'autre part parce qu'elle implique que la droite D_{AH} est perpendiculaire à la droite D_{BC}.

3. *L'étape de diagnostic consiste à déterminer que le point H n'est pas le milieu du segment [B, C] et que la droite D_{AH} est perpendiculaire à la droite D_{BC}.*

4. *Une fois le diagnostic effectué, le système choisit en fonction des préférences du professeur l'erreur à mettre en évidence. Deux cas se présentent :*

   - *Si le système choisit de mettre en évidence la sous-contrainte alors afin de montrer clairement que le point H n'est pas le milieu du segment [B, C], il choisit de le contraindre à être au quart de la distance du segment [B, C], comme cela est explicité dans le tableau 11.1. La spécification logique obtenue du contre-exemple est la suivante :*

     $point(A) \wedge\ point(B) \wedge\ point(C) \wedge\ point(H) \wedge$
     $droite(D_{AB}) \wedge\ droite(D_{AC}) \wedge\ droite(D_{BC}) \wedge\ droite(D_{AH}) \wedge$
     $appDr(A, D_{AB}) \wedge\ appDr(B, D_{AB}) \wedge$
     $appDr(A, D_{AC}) \wedge\ appDr(C, D_{AC}) \wedge$
     $appDr(B, D_{BC}) \wedge\ appDr(C, D_{BC}) \wedge$
     $appDr(A, D_{AH}) \wedge\ appDr(H, D_{AH}) \wedge$
     $appDr(H, D_{BC}) \wedge\ perp(D_{AH}, D_{BC}) \wedge$
     $point(i) \wedge\ milieu(i, B, C) \wedge\ milieu(H, B, i).$

   - *Si le système choisit de mettre en évidence la sur-contrainte alors d'une part il retire de la spécification la perpendicularité des droites D_{AH} et D_{BC} et d'autre part, il ajoute l'égalité des distances $|B, H|$ et $|H, C|$. La spécification logique obtenue de l'exemple est la suivante :*

     $point(A) \wedge\ point(B) \wedge\ point(C) \wedge\ point(H) \wedge$
     $droite(D_{AB}) \wedge\ droite(D_{AC}) \wedge\ droite(D_{BC}) \wedge\ droite(D_{AH}) \wedge$
     $appDr(A, D_{AB}) \wedge\ appDr(B, D_{AB}) \wedge$
     $appDr(A, D_{AC}) \wedge\ appDr(C, D_{AC}) \wedge$
     $appDr(B, D_{BC}) \wedge\ appDr(C, D_{BC}) \wedge$
     $appDr(A, D_{AH}) \wedge\ appDr(H, D_{AH}) \wedge$
     $appDr(H, D_{BC}) \wedge\ milieu(H, B, C).$

5-6. Suivant le type de l'erreur sélectionnée :

- *Si le système a choisi de mettre en évidence la sous-contrainte, alors l'ajout d'une contrainte à un élément de la figure impliquant la libération d'une autre, le système libère successivement les points de base de la figure afin de permettre la construction des contre-exemples. Ainsi, le point* A*, ou le point* B*, ou encore le point* C *sont déplacés suivant l'axe des abscisses ou l'axe des ordonnées, donnant lieu aux constructions en pointillées rapportées sur la figure 11.4.*
- *Si le système a choisi de mettre en évidence la sur-contrainte, alors il fixe arbitrairement successivement les points de base de la figure afin de construire des exemples, donnant lieu aux constructions rapportées sur la figure 11.4.*

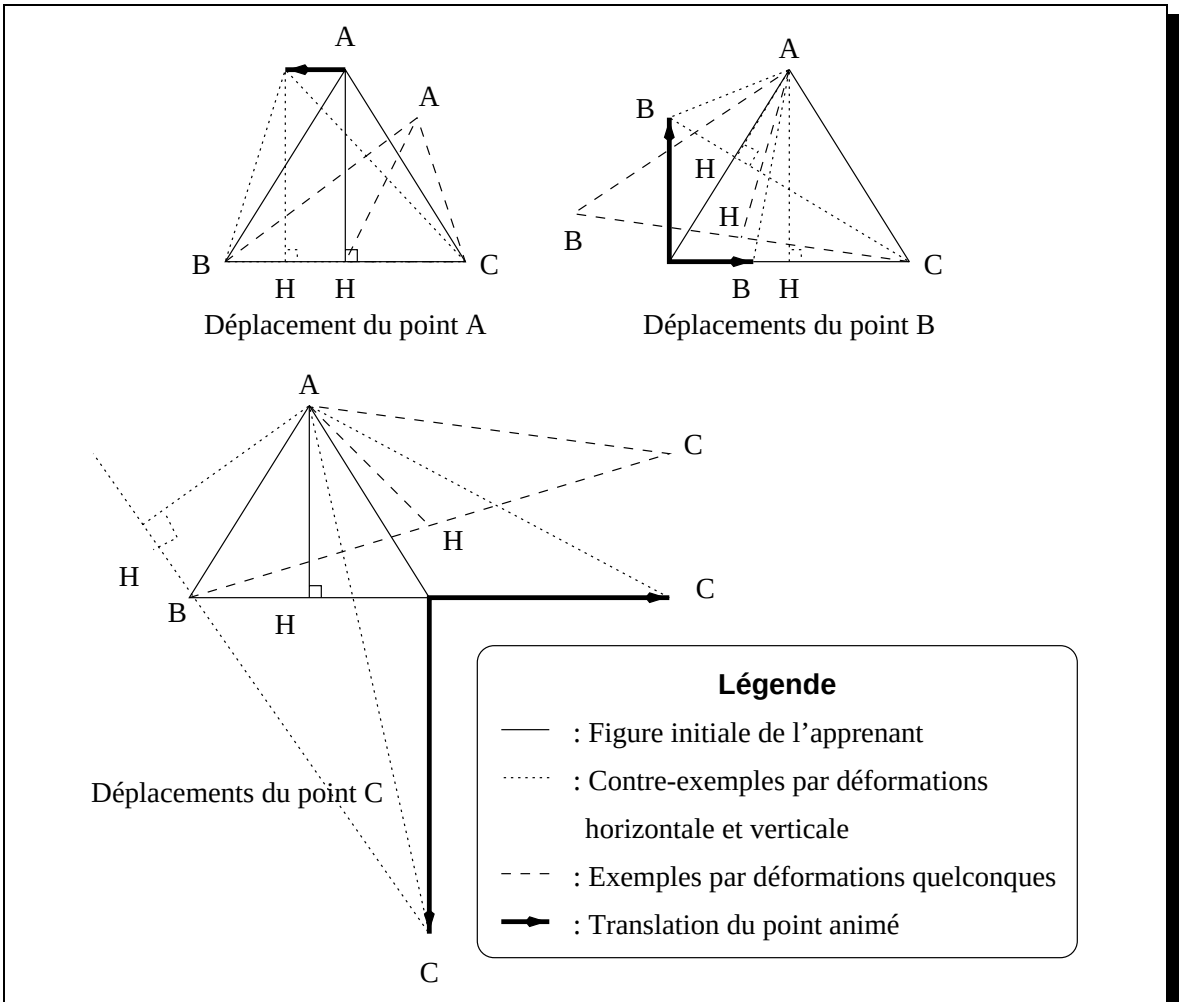

FIG. *11.4* – Animations menant aux exemples et aux contre-exemples.

11.6 Conclusion

Les situations que nous avons considérées avec un tel système d'aide à un professeur qui face à la construction d'un élève, cherche à diagnostiquer et mettre en évidence les erreurs commises dans la résolution de problèmes géométriques ont montré que les notions d'exemple et de contre-exemples enrichissent grandement le diagnostic textuel.

Afin de repousser les limites de notre système, il serait intéressant d'utiliser la version scriptable de Cabri-Géomètre II. Cette intégration permettrait d'accroître le caractère cognitif du système en supportant par exemple de nouveaux types d'animations. Aussi, la définition d'un langage permettant au professeur de spécifier le type d'animations qu'il désire serait souhaitable.

Chapitre 12

Conclusion et perspectives

Pour dresser le bilan de ce travail, nous devons rappeler l'objectif que nous poursuivons. Il concerne les systèmes de géométrie dynamique, et la distinction, dans ces derniers, des aspects géométriques des aspects dynamiques d'une figure. Nous avons expliqué en introduction que notre problématique se décline en trois volets. Il convient par conséquent d'examiner les apports de ce travail sur chacun des ces volets.

– La définition d'un langage logique géométrique.

  Le système GDRev est bâti autour du langage logique géométrique LDL (*Logical Description Language)*. Afin de supporter la définition d'objets complexes, nous avons étendu ce langage de base, en introduisant l'implication dans les définitions, c'est-à-dire que nous avons permis de définir des objets sous forme de clauses. De plus afin par exemple d'être en mesure de différencier les solutions d'un problème, nous avons partiellement introduit la négation. Ces extensions ont conduit au langage logique géométrique ELDL (*Extented LDL*) dans lequel le concept de clause est vue comme le pendant déclaratif du concept de macro construction dans Cabri-Géomètre.

  On peut noter qu'outre la forme "modulaire" apportée à la spécification des figures dynamiques, ces extensions permettent de définir simplement des figures récursives auxquelles il est aisé d'associer une "sous-figure" modélisant d'une manière purement géométrique le niveau de récurrence de ces dernières. La spécification de cette "sous-figure" est d'autant plus intéressante qu'elle correspond à un système d'équations facile à résoudre puisque linéaire (cf paragraphe 4.2.2). Nous avons ainsi abouti à des figures ayant un double caractère dynamique, dont le nouveau caractère dynamique correspond à la déformation d'un objet de base

de la "sous-figure" modélisant le niveau de récurrence.

– L'intégration des concepts de manipulation directe dans la définition du système.

L'originalité des langages d'interface de GDRev réside dans la prise en compte des concepts de manipulation directe lors de leur définition. Cette prise en compte s'est fait au niveau du dessin, mais aussi au niveau des spécifications géométriques et dynamiques. Ainsi, l'utilisateur peut à loisir construire la spécification d'une figure en employant soit des moyens graphiques soit des moyens textuels. Pour assurer le passage de l'un à l'autre de ces moyens, GDRev fournit la possibilité d'établir une correspondance entre le même objet figurant sur le dessin et dans la spécification géométrique.

Garantir graphiquement par manipulation directe sur le dessin la même capacité d'expression que textuellement sur la spécification a nécessité, comme on l'a vu, l'introduction des fonctionnalités de fixation (libération) de points et d'ajout (suppression) de propriétés. L'intérêt et l'originalité de ces fonctionnalités est de pouvoir ainsi aborder un problème de construction de figure géométrique suivant une approche "compositionnelle". Il devient possible à partir de deux figures indépendantes d'en composer une troisième en ajoutant une propriété liant un objet de l'une à un objet de l'autre. Le problème de ces ajouts est qu'ils conduisent parfois à une figure surcontrainte. Pour résoudre ce problème, nous avons proposé deux schémas d'ajout dont le choix revient en définitive à l'utilisateur au travers des préférences du système (cf paragraphe 5.3.3).

Une autre obligation du même type concerne la saisie graphique d'une clause. Dans un cadre déclaratif, il n'existe pas d'une part de structure orientée de laquelle extraire à partir de l'ensemble des arguments un sous-graphe correspondant à la clause désirée, et d'autre part d'ensemble d'objets initiaux et finaux. Dans la mesure où pour résoudre ce problème, il n'y a pas de règle reconnue, nous avons proposé de nous appuyer sur une vision arborescente d'une figure dynamique déclarative. Celle-ci nous a conduit à ramener ce problème à déterminer dans une telle structure la plus petite longueur de chemin telle que tous les arguments peuvent être construit à partir de la donnée de $n-1$ de ces derniers. Bien que cette proposition apporte satisfaction sur l'ensemble des situations que nous avons considérées, notre sentiment est que ce problème demande une étude en soi (cf paragraphe 5.3.2).

– La définition de méthodes de résolution de contraintes géométriques.

Pour construire les figures géométriques spécifiées logiquement, GDRev s'appuie sur une approche coopérative de résolveurs. Celle-ci est basée sur une architecture concurrente avec contraintes. Elle s'organise autour d'un système de contraintes commun à un ensemble de 6 d'agents appelés linéaire, quadratique, intervalle, complétion d'objets, complétion de propriétés, et règle et compas. Ceux-ci coopèrent afin de constuire la figure demandée en partageant des informations de manière synchrone et asynchrone.

L'intérêt et l'originalité de cette approche réside d'une part dans la définition des agents intervalle, complétion d'objets, complétion de propriétés, et règle et compas, et d'autre part dans leur architecture coopérative de l'ensemble.

Plus particulièrement, l'intérêt d'user d'un agent intervalle est triple. Il permet de s'attaquer rapidement et d'une manière correcte à des problèmes de construction difficiles qualitativement, c'est-à-dire de s'attaquer à des figures contenant peu d'objets reliés entre eux principalement par des contraintes géométriques se traduisant par des équations quadratiques.

L'usage des agents quadratique, complétion d'objets et complétion de propriétés est symptomatique de la prise en compte du domaine d'application. L'intérêt des ajouts de contraintes redondantes ou d'objets intermédiaires est manisfeste lorsqu'ils conduisent à la linéarisation ou quasi-linéarisation du système d'équations résultant de la spécification géométrique.

Pour rechercher de telles contraintes redondante, l'agent complétion de propriétés s'appuie sur une construction de la figure afin d'éliminer grâce à cette dernière les possibles propriétés fausses. Dans l'optique de déterminer un ensemble d'objet de base d'une figure, tel que la construction de cette dernière à partir de ceux-ci introduise un minimum d'extensions algébriques, nous avons défini deux heuristiques de choix. Les résultats obtenus à l'aide de ces heuristiques sur un ensemble de 193 exercices choisis parmis les 512 situations décrites dans l'ouvrage [Cho88] sont très positifs. Toutes les situations traitées à l'aide de l'heuristique dite du *plus contraint* introduisent un nombre acceptable d'extensions. Et seulement 1 situation conduit à un excès à l'aide de l'heuristique dite du *non construit*. Ces résultats sont d'autant plus importants que l'échantillon de situations traité sur les 512 situations répertoriées comprend au moins toutes celles susceptibles d'introduire un nombre non acceptable d'extensions. Ils montrent ainsi la faisabilité de notre approche.

Outre ces résultats, cette approche coopérative nous a permis de nous attaquer à un ensemble d'une quarantaine d'exercices de construction extrait principalement de [But75].

Enfin, nous montrons, dans le chapitre portant sur un préceptoriel de géométrie un autre usage de la résolution de contraintes géométriques.

Il nous apparaît aussi important de signaler que dans ce domaine, comme dans beaucoup d'autres domaines, conception se conjugue obligatoirement avec réalisation. Nos objectifs comprennent évidemment la faisabilité des propositions. Une réalisation telle que celle de GDRev a nécessité la mise en œuvre d'une part de l'interface (langages en C++) et d'autre part des résolveurs (agents en Prolog IV), avec des technologies logicielles robustes (architecture PAC, contraintes concurrentes).

Les perspectives que nous envisageons à ce travail concernent :

– les mécanismes de résolution.

A court terme, nous pensons à l'intégration d'un agent linéaire exact et d'un agent quadratique exact. L'objectif de ces intégrations est d'être en mesure d'une part de construire des figures conduisant à des échec en raison d'erreurs d'arrondi, et d'autre part de vérifier la véracité d'une propriété de manière exacte sur un dessin, beaucoup plus rapidement que peut le faire exactement sur la figure le démonstrateur automatique dont nous disposons.

A moyen terme, nous envisageons l'intégration d'un agent algébrique basé sur le calcul de bases de Gröbner [Buc96] ou l'algorithme d'élimination de Ritt-Wu [Wu94]. L'objectif de cette intégration est de pallier à la très grande dépendance des algorithmes de propagation d'intervalles en réécrivant le système de contraintes sous une forme proche d'une forme triangulaire.

A long terme, nous envisageons les axes de recherches suivants :

- *La combinaison de plans de construction partiels et de résolutions numériques par contraintes pour l'animation de figures.* L'objectif de cette combinaison est d'accélérer le processus de construction. L'idée est d'exploiter autant que possible des plans de construction partiels lorsque le plan de construction de la figure n'a pas pu être établi dans son ensemble.
- *L'applicabilité d'une approche différentielle au problème de l'animation de figures géométriques.* L'objectif de cette approche est de minimiser le temps

nécessaire à la reconstruction d'une figure lors d'une animation. L'idée est de se baser sur un première construction afin de déterminer par le calcul ou l'approximation d'équations différentielles une nouvelle figure obtenue par un petit déplacement d'un objet de la figure [Old94].

– les langages de spécification.

A court terme, nous pensons étendre les langages de spécification géométrique à l'aide de clauses. Par exemple, nous souhaitons définir les objets demi-plan et arc de cercle. De plus, nous souhaitons étendre les langages de spécification d'affichage et d'animation.

A moyen terme, nous envisageons d'étendre l'ensemble des objets instanciables. Actuellement, celui-ci est réduit aux objets de type point. Il faudrait considérer les droites et les distances comme des objets de base.

Enfin à long terme, nous envisageons étendre l'ensemble des objets au domaine de la géométrie dans l'espace.

– les langages d'interface.

A court terme, nous pensons achever les fonctionnalités prévues mais non encore mise en œuvre comme la construction automatique du lieu d'un objet ou l'impression des figures.

A moyen terme, nous envisageons les projets suivants :

- *L'extension des fonctionnalités d'affichage.* Une limite de la mise en œuvre actuelle est qu'en présence de solutions multiples, seule la première solution est présentée à l'utilisateur. Nous pensons qu'une fonctionnalité intéressante consisterait à mettre directement à disposition de l'utilisateur, par exemple dans une barre d'outils, des fonctions de navigations entre les différentes solutions répondant à une spécification.
- *L'extension des fonctionnalités dynamiques.* Une idée intéressante consisterait à définir une notion de configuration dynamique, définissable au travers des langages d'interface par manipulation directe, et telle que l'utilisateur puisse passer d'une configuration à une autre beaucoup plus rapidement et simplement qu'il est possible de le faire actuellement grâce aux opérations *Fixer point* et *Libérer point.*

A long terme, nous envisageons les axes de recherches suivants :

- *L'inférence de spécifications géométriques.* Cet axe de recherche s'intéresse à la phase d'acquisition d'une spécification logique de figure géométrique. L'idée de ce mode d'acquisition de spécification logique de figure géométrique est de confier à l'utilisateur la réalisation interactive d'un dessin exprimant de manière implicite et approximative les propriétés géométriques désirées. Il s'agit pour le système d'inférer automatiquement une spécification logique à partir du dessin proposé par l'utilisateur. Le dessin peut être considéré comme un exemple approché de configuration de la figure géométrique à spécifier. Grâce à un processus d'inférence automatique, nous visons un moyen plus aisé d'acquisition d'une spécification logique de figure. Le système se charge de l'exploitation automatique du dessin pour en extraire interactivement un modèle géométrique.

  Typiquement, le système fournit la spécification obtenue à chaque étape d'interaction. Pour apprécier la correction des propriétés inférées, l'utilisateur peut consulter la spécification. Une consultation textuelle peut être inconfortable. Ainsi, le système doit par conséquent permettre d'animer le dessin en accord avec la spécification inférée pour percevoir plus facilement cette spécification.

  Cet axe de recherches correspond à ce que nous avons appelé le langage DDL dans le paragraphe 4.1.2. Un premier prototype opérationnel, réalisé par Nebon F [Neb96] en Prolog III, a donné des résultats encourageants.

- *L'induction de spécifications récursives.* Cet axe de recherche est dans le prolongement naturel du précédent. Il s'intéresse aussi à la phase d'acquisition d'une spécification logique de figure géométrique, à la différence que les spécifications logiques attendues sont cette fois des spécifications récursives. L'induction de telles spécifications se fait à partir de la donnée d'exemples et de contre-exemples sous la forme de dessins réalisés à main levée.

  Cet axe de recherches correspond à ce que nous avons appelé le langage EDDL dans le paragraphe 4.1.3. Une première étude à été réalisée par Coste P [Cos97]. Cet étude a montré l'intérêt de l'approche, mais aussi sa très grande difficulté.

- *L'édition électronique de figures géométriques dynamiques.* Cet recherche, à caractère pratique, consiste à étudier dans quelle mesure les figures géométriques dynamiques produites par GDRev peuvent être intégrées dans

des documents multimédias. En effet, nous pensons que les documents à vocation éducative édités jusqu'à présent gagneraient à contenir de telle figures dynamiques. Une telle intégration peut être mise en œuvre grâce aux technologies OLE/DCOM [Pet96]. GDRev dispose actuellement de fonctions clientes, c'est-à-dire qu'il supporte l'intégration de documents externes dans ses documents propres. Il s'agit maintenant qu'il dispose de fonctions serveur, c'est-à-dire qu'il soit capable d'exporter les figures qu'il construit.

Pour finir, il nous paraît intéressant d'étudier d'une façon plus approfondie les nouvelles activités pédagogiques qui pourraient être mise en place autour de GDRev dans le cadre de l'enseignement de la géométrie au collège et au lycée.

Bibliographie

[AAJM93] Ait-Aoudia S, Jegou R et Michelucci D : *Reduction of constraint systems,* Proceedings of the Compugraphic Conference, pages 83-92, Alvor, Portugal, 1993.

[ACT98] Allen R, Channac S et Trilling L : *Parallels between Constructing Dynamic Figures and Constructing Computer Programs,* Journal of Computer in Mathematics and Science Teaching, AACE, Submitted 1998.

[AIT93] Allen R, Idt-Paumier J et Trilling L : *Constraint Based Automatic Construction and Manipulation of Geometric Figures,* Proceedings of the XIII^{ière} International Joint Conference on Artificial Intelligence, pages 453-458, Morgan Kaufmann Publishers Inc, Chambery, August 1993.

[Ald88] Aldefeld B : *Variation of geometries based on a geometric-reasoning method,* Computer-Aided Design, Vol 20(3), pages 117-126, Avril 1988.

[ANT90] Allen R, Nicolas P et Trilling L : *Figure Correctness in an Expert System for Teaching Geometry,* Proceedings of the 8^{th} Biennal Conference of the Canadian Society for Computational Studies of Intelligence, pages 154-160, Ottawa, Canada, May 1990.

[Bau90] Baulac Y : *Un micro-monde de géométrie - Cabri-Géomètre,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Février 1990.

[BdB95] Beringer H et De Backer B : *Combinatorial Problem Solving in Constraint Logic Programming with Cooperating Solvers,* Beierle C et Plümer L (Eds), Logic Programming : Formal Methods and Practical Applications chapter, Elsevier Science, 1995.

[Bel92] Bellemain F : *Conception, réalisation et expérimentation d'un logiciel d'aide à l'enseignement de la géométrie : Cabri-Géomètre,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Octobre 1992.

[Ben94] BENHAMOU F : *Interval Constraint Logic Programming,* Proceedings of the Châtillon Spring School, pages 108-129, LNCS 910, Châtillon-sur-Seine, France, May 1994.

[Ben96] BENHAMOU F : *Heterogeneous Constraint Solving,* Proceedings of the 5^{th} International Conference Algebraic and Logic Programming, LNCS 1139, Aachen, Germany, September 1996.

[Ber94] BERNAT P : *Conception et réalisation d'un environnement interactif d'aide à la résolution de problèmes, Chypre : un exemple pour l'enseignement de la géométrie,* Thèse de Doctorat de l'Université Henri Poicaré, Nancy, Novembre 1994.

[BFH+95] BOUMA W, FUDOS I, HOFFMANN C, CAI J et PAIGE R : *Geometric constraint solver,* Computer-Aided Design, Vol 27(6), pages 487-501, Juin 1995.

[BG96] BENHAMOU F et GRANVILLIER L : *Combining Local Consistency, Symbolic Rewriting and Interval Methods,* Proceedings of the International Conference in Artificial Intelligence and Symbolic Mathematical Computation, pages 144-159, LNCS 1138, Steyr, Austria, September 1996.

[BL96] BALACHEFF N et SOURY-LAVERGNE S : *Explication et Préceptorat, à propos d'une étude de cas dans TéléCabri,* Actes des III^{èmes} Journées Explication, ed INRIA, 1996.

[BMvH94] BENHAMOU F, MC ALLESTER D et VAN HENTENRYCK P : *CLP(Intervals) Revisited,* Proceedings of the International Logic Programming Symposium, MIT Press, Ithaca, USA, 1994.

[BO97] BENHAMOU F et OLDER W-J : *Applying interval arithmetic to real, integer, and boolean constraints,* Journal of Logic Programming, Vol 32(1), pages 1-24, Juillet 1997.

[Bon93] BONNET J-F : *Témoignage : j'ai rencontré des Cabri-élèves*, Université d'été, Apprentissage et enseignement de la géométrie avec ordinateur : Utilisation du logiciel Cabri-Géomètre en classe, ed LSD2-IMAG, IREM/IUFM de Grenoble, 1993.

[Bor81] BORNING A : *The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory,* ACM Transactions on Programming Languages and Systems, Vol 3(4), pages 353-387, Octobre 1981.

[Bou95] BOUHINEAU D : *Vers une approche déclarative pour les logiciels de dessins géométriques,* Actes des IV^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 55-66, EYROLLES, Cachan, Mars 1995.

[Bou96] BOUHINEAU D : *Solving Geometrical Constraint Systems Using CLP Based on Linear Constraint Solver,* Proceedings of the International Conference on Artificial Intelligence and Symbolic Mathematical Computation, pages 274-288, LNCS 1138, Steyr, Austria, September 1996.

[Bou97] BOUHINEAU D : *Construction automatique de figures géométriques & Programmation logique avec Contraintes,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Juin 1997.

[BR98] BRÜDERLIN B et ROLLER D : *Geometric Constraint Solving and Applications,* Springer-Verlag, 1998.

[BS95] BAADER F et SCHULZ K-U : *On the Combination of Symbolic Constraints, Solution Domains, and Constraint Solvers,* Proceedings of the First International Conference on Principles and Practice of Constraint Programming, pages 380-397, LNCS 976, Cassis, France, September 1995.

[BTS95] BLONDEL F-M, TARIZZO M et SCHWOB M : *Diagnostic des actions de l'élève en vue d'une interaction dans un environnement ouvert en chimie,* Actes des IV^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 149-160, EYROLLES, Cachan, Mars 1995.

[Buc96] BUCHBERGER B : *Symbolic Computation : Computer Algebra and Logic,* Proceedings of the International Conference on Frontiers of Combining Systems, pages 193-220, Applied Logic Series, Kluwer Academic Publishers, Munich, Germany, March 1996.

[But75] BUTHION M : *Un programme qui résoud formellement des problèmes de constructions géométriques,* Thèse de Doctorat de l'Université Pierre et Marie Curie, Paris VI, Juin 1975.

[But79] BUTHION M : *Un Programme qui résout formellement des problèmes de constructions géométriques,* RAIRO Informatique/Computer Science, Vol 13(1), pages 73-106, 1979.

[Cab94] CABRI-GÉOMÈTRE COLLÈGE : *Cabri-classe, Apprendre la géométrie avec un logiciel*, DidaTech, IMAG, UJF, CNRS, Editions Archimède, 1994.

[Cap93] CAPPONI B : *Utilisation du logiciel Cabri-Géomètre en classe,* Actes de l'Université d'été Apprentissage et enseignement de la géométrie avec ordinateur : Utilisation du logiciel Cabri-Géomètre en classe, pages 49-59, Grenoble, Juillet 1993.

[Car88] CARREGA J-C : *Théorie des corps, la règle et le compas,* Hermann, 1989.

[CB96] CHANNAC S et BOUHINEAU D : *La programmation logique par contraintes pour l'aide à l'enseignant,* Proceedings of the 3^{rd} International Conference on Intelligent Tutoring Systems, pages 333-342, LNCS 1086, Montréal, Juin 1996.

[CDR98] COLLAVIZZA H, DELOBEL F et RUEHER M : *A Note on Partial Consistencies over Continuous Domains,* Proceedings of the 4^{th} International Conference on Principles and Practice of Contraint Programming, pages 147-161, LNCS 1520, Pisa, Italie, Octobre 1998.

[CG95] CUOCO A-A et GOLDENBERG E-P : *Dynamic Geometry as a Bridge from Euclidean Geometry to Analysis,* Geometry Turned On, Dynamic software in learning, teaching, and research chapter, pages 33-47, The Mathematical Association of America, 1995.

[CG96] CUOCO A-A et GOLDENBERG E-P : *What is Dynamic Geometry ?,* Designing Learning Environments for Developing Understanding of Geometry and Space chapter, Hillsdale, NJ Erlbaum, 1996.

[CGZ97] CHOU S-C, GAO X-S et ZHANG J-Z : *Automated Generation of Construction Steps for Geometric Constraint Problems,* Automated Reasoning and Its Applications : Essays in Honor of Larry Wos chapter, The MIT Press, 1997.

[Cha96] CHANNAC S : *Techniques d'Intelligence Artificielle pour l'Execution de Programmes Logiques Géométriques,* Actes des $II^{ième}$ Journées Infographie Interactive et Intelligence Artificielle, pages 187-212, Limoges, Avril 1996.

[Cha97] CHANNAC S : *Sur la génération d'exemples et de contre-exemples géométriques dans un préceptoriel,* Actes des $V^{ième}$ Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 109-120, Hermès, Cachan, Mai 1997.

[Cho88] CHOU S-C : *Mechanical Geometry Theorem Proving,* Mathematics and Its Applications, D Reidel Publishing Company, 1988.

[CI97] COSTE P et IDT-PAUMIER J : *Aide à la construction d'une figure géométrique donnée par sa spécification,* Actes des V^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, Hermès, Cachan, Mai 1997.

[CL94] CHIU C-K et LEE J-H-M : *Towards Practical Interval Constraint Solving in Logic Programming,* Proceedings of the International Logic Programming Symposium, pages 109-123, The MIT Press, 1994.

[Cla85] CLANCEY W-J : *Heuristic Classification,* In Artificial Intelligence, Vol 27, pages 289-350, 1985.

[Cle87] CLEARY J-G : *Logical Arithmetic,* Future Computing Systems, Vol 2(2), pages 125-149, 1987.

[Col90] COLMERAUER A : *An Introduction to PrologIII,* In Communication of the ACM, Vol 33(7), pages 69-90, July 1990.

[Col93] COLMERAUER A : *Naive Solving of Non-linear Constraints,* Benhamou F et Colmerauer A (Eds), Constraint Logic Programming : Selected Research chapter, The MIT Press, 1993.

[Col96] COLMERAUER A : *Prolog IV - Constaints Inside,* Manuel de Prolog IV, Parc Technologique de Luminy, Case 919, 13288 Marseille cedex 09, France, 1996.

[Cos97] COSTE P : *Acquisition automatique de figures géométriques dynamiques récursives,* Rapport de DEA de l'Université Joseph Fourier, Grenoble, Juin 1997.

[Cou90] COUTAZ J : *Interface homme-ordinateur : Conception et Réalisation,* Dunod, Paris, 1990.

[Cou91] COUTAZ J : *Interface Homme-Machine : un regard critique,* Technique et science informatiques, Vol 10(1), pages 53-64, Dunod, 1991.

[Cup96] CUPPENS R : *Faire de la géométrie en jouant avec Cabri-Géomètre,* A.P.M.E.P., Paris, 1996.

[Cza89] CZAPOR S-R : *Solving Algebraic Equations : Combining Buchberger's Algorithm with Multivariate Factorization,* In Journal of Symbolic Computation, Vol 7(1), pages 49-53, January 1989.

[Dav87] DAVIS E : *Constraint Propogation with Interval Labels,* Artificial Intelligence, Vol 32(3), pages 281-331, Juillet 1987.

[Des94] DESMOULINS C : *Etude et réalisation d'un système tuteur pour la construction de figures géométriques,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Février 1994.

[Des95] DESMOULINS C : *La détection de solutions particulières dans TALC: une approche logique basée sur des extensions de l'énoncé du problème,* Actes des IV^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 161-172, EYROLLES, Cachan, Mars 1995.

[DMS96] DUFOURD J-F, MATHIS P et SCHRECK P : *Constructions géométriques sous contraintes par assemblage de figures résolues,* Technical Report 96/04, Université Louis Pasteur, Strasbourg, Mars 1996.

[Eve89] EVERTSZ R : *The Generation of 'Critical Problems' By Abstract Interpretations of Student Models,* Proceedings of the International Joint Conference on Artificial Intelligence, pages 483-488, Detroit, USA, August 1989.

[FG96] FRANKLIN S et GRAESSER A : *Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents,* Proceedings of the European Conference on Artificial Intelligence Workshop on Agent Theories, Architectures and Languages, pages 21-35, LNCS 1193, Budapest, Hongrie, Août 1996.

[FvDFH90] FOLEY J-D, VAN DAM A, FEINER S-K et HUGHES J-F : *Computer Graphics, Principles and Practice,* The Systems Programming Series, Second Edition, Addison-Wesley Publishing Company, 1990.

[Gar95] GARCIA C : *Conception et réalisation d'une interface homme ordinateur interactive pour la construction et l'animation de figures géométriques,* Mémoire d'ingénieur du Conservatoire National des Arts et Métiers, Grenoble, Mars 1995.

[GC98a] GAO X-S et CHOU S-C : *Solving geometric constraint systems. I. A global propagation approach,* In Computer-Aided Design, Vol 30(1), pages 47-54, Janvier 1998.

[GC98b] GAO X-S et CHOU S-C : *Solving geometric constraint systems. II. A symbolic approach and decision of Rc-constructibility,* Computer-Aided Design, Vol 30(2), pages 115-122, Fevrier 1998.

[Gra97] GRANVILLIERS L : *Transformations symboliques et consistance de bloc de CSP continus,* Actes des VI^{ième} Journées Francophones de Programmation en Logique et Programmation par Contraintes, pages 195-209, Hermes, Orléans , Mai 1997.

[HG95] HOWER W et GRAF W-H : *Research in Constraint-Based Layout, Visualization, CAD, and Related Topics : A Bibliographical Survey,* Proceedings of the International Workshop on Constraints for Graphics and Visualization, pages 3-25, Cassis, Septembre 1995.

[HL93] HOLLMAN J et LANGEMYR L : *Algorithms for Non-linear Algebraic Constraints,* Benhamou F et Colmerauer A (Eds), Constraint Logic Programming : Selected Research chapter, The MIT Press, 1993.

[HN94] HEYDON A et NELSON G : *The Juno-2 Constraint-Based Drawing Editor,* Research Report 131a, Digital Systems Research Center, Palo Alto, Decembre 1994.

[Hon94] HONG H : *Confluency of Cooperative Constraint Solvers,* Technical Report, Research Institute for Symbolic Computation, Johannes Kepler University, Linz, January 1994.

[vHSD93] VAN HENTENRYCK P, SARASWAT V et DEVILLE Y : *Design, Implementations and Evaluation of the Constraint Language cc(FD),* Technical Report CS-93-02, January 1993.

[Jac95] JACKIW R-N : *Drawing Worlds : Scripted Exploration Environments in The Geometer's Sketchpad,* Geometry Turned On, Dynamic software in learning, teaching, and research chapter, pages 179-185, The Mathematical Association of America, 1995.

[Kea95] KEYTON M : *Students Discovering Geometry Using Dynamic Geometry Software,* Geometry Turned On, Dynamic software in learning, teaching, and research chapter, pages 63-68, The Mathematical Association of America, 1995.

[KMPR98] KETTANI N, MIGNET D, PARÉ P et ROSENTHAL-SABROUX C : *De Merise à UML,* Eyrolles, 1998.

[KR94] KIRCHNER H et RINGEISSEN C : *Combining Symbolic Constraint Solvers on Algebraic Domains,* Journal of Symbolic Computation, Vol 18(2), pages 113-155, August 1994.

[Kra92a] KRAMER G-A : *A geometric constraint engine,* In Artificial Intelligence, Vol 58, pages 327-360, 1992.

[Kra92b] KRAMER G-A : *Solving Geometric Constraint Systems : A Case Study in Kinematic,* Artificial Intelligence, The MIT Press, 1992.

[Lab93] LABORDE C : *Apprendre à voir et manier l'objet géométrique au delà du tracé dans Cabri-Géomètree,* Actes de l'Université d'été Apprentissage et enseignement de la géométrie avec ordinateur : Utilisation du logiciel Cabri-Géomètre en classe, pages 87-101, Grenoble, Juillet 1993.

[Lab95] LABORDE J-M : *Des connaissances abstraites aux réalités artificielles, le concept de micromonde Cabri,* Actes des IV^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 29-41, EYROLLES, Cachan, Mars 1995.

[LD95] LUCAS M et DESMONTILS E : *Les modeleurs déclaratifs,* Revue internationale de CFAO et d'informatique graphique, Vol 10(6), pages 559-585, 1995.

[LGRT96] LHOMME O, GOTLIEB A, RUEHER M et TAILLIBERT P : *Boosting the Interval Narrowing Algorithme,* Proceedings of the Joint International Conference and Symposium on Logic Programming, pages 378-392, MIT Press, Septembre 1996.

[Lho93] LHOMME O : *Consistency Techniques for Numeric CSPs,* Proceedings of the XIII^{ière} International Joint Conference on Artificial Intelligence, pages 232-238, Morgan Kaufmann Publishers Inc, Chambery, France, August 1993.

[LM94] LAMURE H et MICHELUCCI D : *Résolution de contraintes géométriques par homotopie,* Actes des II^{ième} Journées de l'Association Française d'Informatique Graphique, pages 95-104, Toulouse, 1994.

[LR97] LHOMME O et RUEHER M : *Application des techniques CSP au raisonnement sur les intervalles,* Revue d'intelligence artificielle, Vol 11(3), pages 283-311, Septembre 1997.

[Lue97] LUENGO V : *Un micromonde de preuve intégrant la réfutation : Cabri-Euclide,* Actes des V^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 85-97, Hermes, Cachan, Mai 1997.

[Mac77] MACKWORTH A-K : *Consistency in Networks of Relations,* Artificial Intelligence, Vol 8, pages 99-118, 1992.

[Mar96] MARTI P : *SDRC: Un système de coopération entre solveurs pour la résolution de contraintes non-linéaires sur les réels,* Thèse de Doctorat de l'Université de Nice - Sophia Antipolis, Décembre 1996.

[Mat97] MATHIS P : *Constructions Géométriques sous Contraintes en Modélisation à base Topologique,* Thèse de Doctorat de l'Université Louis Pasteur, Strasbourg, Octobre 1997.

[MBB89] MALONEY J-H, BORNING A et FREEMAN-BENSON B-N : *Constraint Technology for User-Interface Construction in ThingLab II,* Technical Report 89-05-02, University of Washington, Seattle, Mai 1989.

[Mon96] MONFROY E : *Collaboration de Solveurs pour la Programmation Logique à Contraintes,* Thèse de Doctorat de l'Université Henri Poincaré, Nancy, Novembre 1996.

[MR95] MARTI P et RUEHER M : *A Distributed Cooperating Constraints Solving System,* International Journal on Artificial Intelligence Tools, 4(1-2), pages 93-113, June 1995.

[MRS96] MONFROY E, RUSINOWITCH M et SCHOTT R : *Implementing non-linear constraints with cooperative solvers,* Proceedings of ACM Symposium on Applied Computing, pages 63-72, February 1996.

[MS98] MARRIOTT K et STUCKEY P-J : *Programming with Constraints : An Introduction,* The MIT Press, 1998.

[Nan90] NANARD J : *La manipulation directe en interface homme-machine,* Thèse d'état, Université de Sciences et Techniques du Languedoc, Montpellier, 1990.

[Neb96] NEBON F : *Inférence Automatique de Spécifications Logiques à partir de dessins,* Mémoire d'ingénieur du Conservatoire National des Arts et Métiers, Grenoble, Mars 1996.

[Nic89] NICOLAS P : *Construction et vérification de figures géométriques dans le système MENTONIEZH,* Thèse de Doctorat de l'Université de Rennes, Février 1989.

[NJNG93] NGUYEN-XUAN A, JOLY F, NICAUD J-F et GÉLIS J-M : *Une méthode de diagnostic des connaissances en algèbre pour un module de modélisation de l'élève,* Actes des III^{ièmes} Journées Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 217-228, EYROLLES, Cachan, Février 1993.

[NO79] NELSON G et OPPEN D-C : *Simplification by Cooperating Decision Procedures,* ACM Transactions on Programming Languages and Systems, Vol 1(2), pages 245-257, October 1979.

[NP93] NOEL C et PY D : *Diagnostic des erreurs de l'élève basé sur le contexte,* Actes des III^{ième} Journées Environnements Interactifs d'Apprentissage avec Ordinateur, pages 229-240, EYROLLES, Cachan, Février 1993.

[Old94] OLDER W-J : *Application of Relational Interval Arithmetic to Ordinary Differential Equations,* Technical Report, Bell Northern Research, Ottawa, August 1994.

[Ost97a] OSTIER P : *A Complete Deduction System for Reasoning with Temporary Assumptions,* Proceedings of the International Workshop First Order Theorem Proving, RISC-Linz Report Series, Linz, Autriche, 1997.

[Ost97b] OSTIER P : *Hypotheses Domains System : un système d'inférence pour la construction de preuves naturelles et la production d'hypothèses,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Novembre 1997.

[OV90] OLDER W-J et VELLINO A : *Extending Prolog with Constraint Arithmetic on Real Intervals,* Proceedings of the Canadian Conference on Electrical and Computer Engineering, 1990.

[Owe91] OWEN J-C : *Algebraic Solution for Geometry from Dimensional Constraints,* Proceedings of the First ACM Symposium of Solid Modeling and CAD/CAM Applications, pages 397-407, ACM Press, 1991.

[Pay93] PAYAN C : *Cabri-Géomètre : du continu au discret,* Rapport Technique, 1993.

[PB94] PESANT G et BOYER M : *Quad-CLP($\mathcal{R}$) : Adding the power of quadratic constraints,* Proceedings of the Second Principles and Pratice of Constraint Programming Workshop, Rosario, USA, May 1994.

[Pet90] PETERSEN J : *Problémes de Constructions GÉOMÉTRIQUES,* Les Grands Classiques Gauthier-Villars, Editions Jacques Gabay, 1990.

[Pet96] PETZOLD C et YAO P : *Programmer sous Windows 95,* Microsoft Press, 1996.

[Pes95] PESANT G : *Une approche géométrique aux contraintes arithmétiques quadratiques en programmation logique avec contraintes,* Thèse de Doctorat de l'Université de Montréal, Québec, Canada, Décembre 1995.

[Py96] PY D : *Aide à la démonstration en géométrie : le projet Mentoniezh,* Sciences et techniques éducatives, Vol 3(2), pages 227-256, 1996.

[Qas97] QASEM S : *Conception et Réalisation d'une interface 3D pour Cabri-Géomètre,* Thèse de Doctorat de l'Université Joseph Fourier, Grenoble, Décembre 1997.

[RS97] RUEHER M et SOLNON C : *Concurrent Cooperating Solvers over Reals,* Reliable Computing, Vol 3(3), pages 325-333, 1997.

[Rue94] RUEHER M : *An Architecture for Cooperating Constraint Solvers on Reals,* Proceedings of the Châtillon Spring School, pages 231-250, LNCS 910, Châtillon-sur-Seine, France, May 1994.

[Sap91] SAPOSSNEK M : *Research on Contraint-Based Design Systems,* Technical Report EDRC 18-24-91, Engineering Design Research Center, Pittsburgh, 1991.

[Sar93] SARASWAT V-A : *Concurrent Constraint Programming,* The MIT Press, 1993.

[Sch93] SCHRECK P : *Automatisation des constructions géométriques à la règle et au compas,* Thèse de Doctorat de l'Université Louis Pasteur, Strasbourg, Janvier 1993.

[Sch94] SCHRECK P : *Modélisation et implantation d'un système à base de connaissances pour les constructions géométriques,* Revue d'intelligence artificielle, Vol 8(3), pages 223-247, 1994.

[SG94] SCHUMANN H et GREEN D : *Discovering Geometry with a Computer - using Cabri-Géomètre,* Chartwell-Bratt, 1994.

[SG95] SCHUMANN H et GREEN D : *Producing and Using Loci with Dynamic Geometry Software,* Geometry Turned On, Dynamic software in learning, teaching, and research chapter, pages 79-87, The Mathematical Association of America, 1995.

[Sha89] SHAPIRO E : *The Family of Concurrent Logic Programming Languages,* ACM Computing Surveys, Vol 21(3), pages 413-510, September 1989.

[Shn83] SHNEIDERMAN B : *Direct Manipulation : A Step Beyond Programming Languages,* IEEE Computer, Vol 16(8), pages 57-69, August 1983.

[Sho93] SHOHAM Y : *Agent-oriented programming,* Artificial Intelligence, Vol 60(1), pages 51-92, March 1993.

[Smo95] SMOLKA G : *The Oz Programming Model,* Proceedings of the International Conference on Computer Science Today, LNCS 1000, Berlin, Germany, 1996.

[SS90] STERLING L et SHAPIRO E : *L'art de Prolog,* Masson, 1990.

[TDVP88] TERRACHER P, DELORD R, VINRICH G et PRIVAT B : *Mathématiques 4^e,* Hachette Collèges, 1988.

[Trâ'n96] TRÂ'N Q-N : *A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization,* Technical Report, Université de Johannes Kepler, Linz, Austria, Octobre 1996.

[Tri96a] TRILLING L : *Programmation géométrique impérative et logique,* Actes des Journées Francophones des Langages Applicatifs, pages 147-164, INRIA Collection Didactique, Val-Morin, Canada, Janvier 1996.

[Tri96b] TRILLING L : *Rétrospective sur le projet Mentoniezh,* Sciences et techniques éducatives, Vol 3(2), pages 157-162, 1996.

[Wil83] WILLIAMS G : *The Lisa Computer System,* Byte, Vol 8(2), pages 33-50, Février 1983.

[Wil84] WILLIAMS G : *The Apple Macintosh Computer,* Byte, Vol 9(2), pages 30-54, Février 1984.

[WJ95] WOOLDRIDGE M-J et JENNINGS N-R : *Intelligent Agents : Theory and Practice,* Knowledge Engineering Review, Vol 10(2), 1995.

[Wu94] WU W : *Mechanical Theorem Proving in Geometries,* Springer-Verlag Wien New York, 1994.

Annexe A

Théorie Géométrique

La connaissance du démonstrateur automatique est représentée par un ensemble de 32 théorèmes. Ces théorèmes de géométrie sont extraits de résumés de cours de collège et de lycée [TDVP88, Des94, Ost97b].

Les théorèmes et les faits correspondant à la figure sont représentés par un paquet de faits prolog. Chaque fait prolog est de la forme $gdrevHypothese(\_, L', [L'_1, \ldots, L'n])$ dans lequel le premier argument est un identificateur à valeur dans {tig, enonce} indiquant si le fait est un théorème de la théorie géomètrique modélisée ou un fait de l'utilisateur, et où $L', L'_1, \ldots, L'_n$ désignent les traductions des littéraux $L, L_1, \ldots, L_n$ du théorème de la forme $L_1, \ldots, L_n \Rightarrow L$ représenté. La traduction :

- d'un littéral positif A est A.
- d'un littéral négatif $\neg$ A est *non(A)*.
- du littéral particulier $\perp$ est *false*.

Afin d'obtenir un démonstrateur raisonnablement efficace sur les problèmes de géométrie considérés, une aide lui est apportée dans le choix des théorèmes utilisés. Cette aide consiste à vérifier sur un modèle de la figure si une propriété est fausse afin d'élaguer des branches de l'arbre de recherche qui s'avéreraient vaines à explorer. Pour cela, le prédicat *contreModele(_Prop* est ajouté. Il a pour fonction de vérifier sur le modèle de la figure si la propriété *_Prop* est vérifiée ou non.

```
% -- Alignement de trois points ------------------------------ %
```

```
% Trois points situés sur une même droite sont alignés

gdrevHypothese(tig,
               alignes(_Pt1, _Pt2, _Pt3),
               [point(_Pt1),
                point(_Pt2),
                droite(_D),
                contreModele(appDr(_Pt1, _D)),
                contreModele(appDr(_Pt2, _D)),
                appDr(_Pt1, _D),
                appDr(_Pt2, _D),
                appDr(_Pt3, _D),
                non(egalPt(_Pt1, _Pt2)),
                non(egalPt(_Pt1, _Pt3)),
                non(egalPt(_Pt2, _Pt3))]).

% -- Milieu ------------------------------------------------- %

% O est le milieu de [AB] si OA = OB et O, A, B sont alignés

gdrevHypothese(tig,
               milieu(_O, _A, _B),
               [point(_O),
                point(_A),
                distance(_L),
                contreModele(distPP(_L, _O, _A)),
                distPP(_L, _O, _A),
                distPP(_L, _O, _B),
                alignes(_O, _A, _B),
                non(egalPt(_A, _B))]).

% -- définition par le cercle

% O est le milieu de [AB} si [AB] est un diamètre d'un cercle
% de centre O

gdrevHypothese(tig,
```

```
                milieu(_O, _A, _B),
                [point(_O),
                 point(_A),
                 distance(_L),
                 cercle(_C, _O, _L),
                 contreModele(distPP(_L, _O, _A)),
                 appCc(_A, _C),
                 appCc(_B, _C),
                 alignes(_O, _A, _B)]).

% -- définition par la médiatrice

% O est le milieu de [AB} si c'est le pied de la médiatrice de [AB]

gdrevHypothese(tig,
                milieu(_O, _A, _B),
                [point(_A),
                 point(_B),
                 point(_O),
                 droite(_D),
                 contreModele(appDr(_O, _D)),
                 appDr(_O, _D),
                 mediatrice(_D, _A, _B),
                 alignes(_O, _A, _B)]).

% -- Médiatrice ------------------------------------------------ %

% -- définition par les distances

% D est la médiatrice de [AB] si deux points distincts de D sont à
% égales distances de A et de B

gdrevHypothese(tig,
                mediatrice(_D, _A, _B),
                [droite(_D),
                 point(_A),
                 point(_O),
```

```
                contreModele(appDr(_O, _D)),
                point(Q),
                contreModele(non(egalPt(_O, _Q))),
                contreModele(appDr(_Q, _D)),
                distance(_L1),
                contreModele(distPP(_L1, _O, _A)),
                distance(_L2),
                contreModele(distPP(_L2, _Q, _A)),
                appDr(_O, _D),
                appDr(_Q, _D),
                distPP(_L1, _O, _A),
                distPP(_L1, _O, _B),
                distPP(_L2, _Q, _A),
                distPP(_L2, _Q, _B),
                non(egalPt(_A, _B)),
                non(egalPt(_O, _Q))]).

% Nota: sans egalPt(_O, _Q) on pourrait avoir _O = _Q.
% à insérer dans les contraintes car _O et _Q sont construits.

% -- définition par la perpendiculaire et milieu

% D est la médiatrice de [AB] si elle passe par le
% milieu de [AB] et est perpendiculaire à (AB)

gdrevHypothese(tig,
               mediatrice(_D, _A, _B),
               [droite(_D),
                point(_A),
                point(_I),
                droite(_E),
                contreModele(milieu(_I, _A, _B)),
                contreModele(perp(_D, _E)),
                contreModele(appDr(_A, _E)),
                contreModele(appDr(_I, _D)),
                perp(_E, _D),
                appDr(_A, _E),
```

```
                appDr(_B, _E),
                appDr(_I, _D),
                milieu(_I, _A, _B),
                non(egalPt(_A, _B))]).

% -- Distance ------------------------------------------------ %

% R est la distance séparant A de B si B est le milieu de AC
% et si R est la distance séparant B de C

gdrevHypothese(tig,
               distPP(_R, _A, _B),
               [point(_A),
                point(_B),
                point(_C),
                contreModele(milieu(_B, _A, _C)),
                distPP(_R, _B, _C),
                milieu(_B, _A, _C)])).

% 2 hypothèses à cause de la réécriture de distance
% la distance entre le centre et un point d'un cercle
% est égale au rayon du cercle

gdrevHypothese(tig,
               distPP(_R, _O, _A),
               [cercle(_C, _O, _R),
                appCc(_A, _C)]).

gdrevHypothese(tig,
               distPP(_R, _O, _A),
               [cercle(_C, _A, _R),
                appCc(_O, _C)]).

% -- Parallélisme -------------------------------------------- %

% Deux droites sont parallèles si elles sont toutes deux
% perpendiculaires à une même droite
```

```
gdrevHypothese(tig,
               par(_D1, _D2),
               [droite(_D1),
                droite(_D3),
                contreModele(perp(_D1, _D3)),
                perp(_D1, _D3),
                perp(_D3, _D2)]).

% -- définition par transitivité

% Deux droites sont parallèles si elles sont parallèles
% à une même droite

gdrevHypothese(tig,
               par(_D1, _D2),
               [droite(_D1),
                droite(_D3),
                contreModele(par(_D1, _D3)),
                par(_D1, _D3),
                par(_D3, _D2)]).

% -- Perpendicularité ----------------------------------------- %

% Deux droites sont perpendiculaires si elles sont toutes deux
% parallèle à une même droite

gdrevHypothese(tig,
               perp(_D1, _D2),
               [droite(_D1),
                droite(_D3),
                contreModele(perp(_D1, _D3)),
                perp(_D1, _D3),
                par(_D3, _D2)]).

% -- Appartenance à une droite -------------------------------- %
```

```
% -- définition par médiatrice

% Un point situé à égale distance de deux points
% appartient à la médiatrice de ces deux points

gdrevHypothese(tig,
               appDr(_M, _D),
               [point(_M),
                droite(_D),
                point(_A),
                point(_B),
                distance(_L),
                contreModele(distPP(_L, _M, _A)),
                contreModele(distPP(_L, _M, _B)),
                distPP(_L, _M, _A),
                distPP(_L, _M, _B),
                mediatrice(_D, _A, _B)]).

% -- définition par demi-droite

% Un point appartient à une droite s'il appartient à
% une demi-droite ayant la droite pour support

gdrevHypothese(tig,
               appDr(_M, _D),
               [point(_M),
                droite(_D),
                demiDroite(_L, _N, _D),
                point(_N),
                contreModele(appDD(_M, _L)),
                appDD(_M, _L)]).

% -- définition par segment

% Un point appartient à une droite s'il appartient à
% un segment ayant la droite pour support
```

```
gdrevHypothese(tig,
               appDr(_M, _D),
               [point(_M),
                droite(_D),
                segment(_S, _A, _B, _D),
                point(_A),
                point(_B),
                contreModele(appSeg(_M, _S)),
                appSeg(_M, _S)]).

% Un point appartient à une droite s'il est l'extremite d'un
% segment ayant la droite pour support

gdrevHypothese(tig,
               appDr(_M, _D),
               [point(_M),
                droite(_D),
                point(_A),
                contreModele(appDr(_A, _D)),
                segment(_S, _A, _M, _D)]).

% -- Appartenance à un cercle ---------------------------------- %

% Un point situé à une distance R du centre d'un
% cercle de rayon R appartient à ce cercle

gdrevHypothese(tig,
               appCc(_M, _C),
               [cercle(_C, _O, _R),
                distPP(_R, _O, _M)]).

% -- Egalite de points ----------------------------------------- %

% Deux droites distinctes se coupent en un seul point

gdrevHypothese(tig,
               egalPt(_Pt1, _Pt2),
```

```
               [point(_Pt1),
                droite(_D1),
                contreModele(appDr(_Pt1, _D1)),
                droite(_D2),
                contreModele(non(egalDr(_D1, _D2))),
                contreModele(appDr(_Pt2, _D2)),
                non(egalDr(_D1, _D2)),
                appDr(_Pt1, _D1),
                appDr(_Pt1, _D2),
                appDr(_Pt2, _D1),
                appDr(_Pt2, _D2)]).

% Il est faut que 2 points soient égaux s'ils sont première et
% deuxième extremité d'un segment

gdrevHypothese(tig,
               false,
               [point(_A),
                point(_B),
                contreModele(non(egalPt(_A, _B))),
                segment(_S, _A, _B, _D),
                egalPt(_A, _B)]).

% Deux points sont égaux s'ils sont les centres d'un même cercle

gdrevHypothese(tig,
               egalPt(_Pt1, _Pt2),
               [cercle(_C, _Pt1, _R),
                cercle(_C, _Pt2, _R)]).

% Il est faux que 2 points soient égaux s'il existe une
% distance les séparant

gdrevHypothese(tig,
               false,
               [point(_A),
                point(_B),
```

```
                contreModele(non(egalPt(_A, _B))),
                distance(_L),
                contreModele(distPP(_L, _A, _B)),
                distPP(_L, _A, _B),
                egalPt(_A, _B)]).

% Deux points P1 et P2 sont égaux s'il existe une droite D et
% un point P3 distinct de P1 et P2 tels que P1, P2 et P3
% appartiennent à D, et tels qu'ils soient tous les trois séparés
% d'un point O par une distance L

gdrevHypothese(tig,
               egalPt(_Pt1, _Pt2),
               [point(_Pt1),
                droite(_D),
                point(_Pt2),
                point(_Pt3),
                contreModele(non(egalPt(_Pt3, _Pt2))),
                contreModele(appDr(_Pt1, _D)),
                contreModele(appDr(_Pt2, _D)),
                contreModele(appDr(_Pt3, _D)),
                appDr(_Pt1, _D),
                appDr(_Pt2, _D),
                appDr(_Pt3, _D),
                contreModele(non(egalPt(_Pt1, _Pt3))),
                distance(_L),
                point(O),
                contreModele(distPP(_L, _O, _Pt1)),
                contreModele(distPP(_L, _O, _Pt2)),
                contreModele(distPP(_L, _O, _Pt3)),
                distPP(_L, _Pt1, _O),
                distPP(_L, _Pt2, _O),
                non(egalPt(_Pt2, _Pt3)),
                distPP(_L, _Pt3, _O),
                non(egalPt(_Pt1, _Pt3))]).

% Si L est la distance d'un point P3 à une droite D1, deux points
```

```
% de D1 sont à une distance L du point P3 s'ils sont égaux

gdrevHypothese(tig,
               egalPt(_Pt1, _Pt2),
               [point(_Pt1),
                droite(_D1),
                contreModele(appDr(_Pt1, _D1)),
                point(_Pt2),
                contreModele(appDr(_Pt2, _D1)),
                droite(_D2),
                contreModele(appDr(_Pt2, _D2)),
                contreModele(perp(_D1, _D2)),
                appDr(_Pt1, _D1),
                appDr(_Pt2, _D1),
                appDr(_Pt2, _D2),
                point(_Pt3),
                contreModele(appDr(_Pt3, _D2)),
                appDr(_Pt3, _D2),
                perp(_D1, _D2),
                distance(_L),
                distPP(_L, _Pt3, _Pt1),
                distPP(_L, _Pt3, _Pt2)]).

% Deux points sont égaux s'ils sont à l'intersection
% de deux cercles distincts, et s'ils ne sont pas égaux
% à un troisième point appartenant aux deux cercles

gdrevHypothese(tig,
               egalPt(_Pt1, _Pt2),
               [point(_Pt1),
                point(_O1),
                distance(_L1),
                cercle(_C1, _O1, _L1),
                contreModele(distPP(_L1, _Pt1, _O1)),
                point(_O2),
                distance(_L2),
                cercle(_C2, _O2, _L2),
```

```
                contreModele(non(egalPt(_O1, _O2))),
                contreModele(distPP(_L2, _O2, _Pt1)),
                point(_Pt3),
                contreModele(distPP(_L1, _O1, _Pt3)),
                contreModele(distPP(_L2, _O2, _Pt3)),
                contreModele(non(egalPt(_Pt1, _Pt3))),
                appCc(_Pt1, _C1),
                appCc(_Pt2, _C1),
                appCc(_Pt3, _C1),
                appCc(_Pt1, _C2),
                appCc(_Pt2, _C2),
                appCc(_Pt3, _C2),
                non(egalPt(_O1, _O2)),
                non(egalPt(_Pt1, _Pt3)),
                non(egalPt(_Pt2, _Pt3))]).

% -- Egalite de droites ------------------------------------- %

% -- définition par points

% Deux droites sont égales s'il existe deux points distincts
% qui appartiennent à toutes les deux

gdrevHypothese(tig,
               egalDr(_D1, _D2),
               [point(_Pt1),
                droite(_D1),
                contreModele(appDr(_Pt1, _D1)),
                point(_Pt2),
                contreModele(appDr(_Pt2, _D1)),
                contreModele(non(egalPt(_Pt1, _Pt2))),
                appDr(_Pt1, _D1),
                appDr(_Pt1, _D2),
                appDr(_Pt2, _D1),
                appDr(_Pt2, _D2),
                non(egalPt(_Pt1, _Pt2))]).
```

```
% -- définition par parallelisme

% Deux droites sont égales si elles ont un point commun et
% si elles sont toutes les deux parallèles à une troisième droite

gdrevHypothese(tig,
               egalDr(_D1, _D2),
               [droite(_D1),
                point(_Pt),
                contreModele(appDr(_Pt, _D1)),
                droite(_D),
                contreModele(par(_D, _D1)),
                appDr(_Pt, _D1),
                appDr(_Pt, _D2),
                par(_D1, _D),
                par(_D2, _D)]).

% -- définition par perpendicularité

% Deux droites sont égales si elles ont un point commun et
% si elles sont toutes les deux perpendiculaires à une troisième droite

gdrevHypothese(tig,
               egalDr(_D1, _D2),
               [droite(_D1),
                point(_Pt),
                contreModele(appDr(_Pt, _D1)),
                droite(_D),
                contreModele(perp(_D, _D1)),
                appDr(_Pt, _D1),
                appDr(_Pt, _D2),
                perp(_D1, _D),
                perp(_D2, _D)]).

% Il est faux que deux droites égales soient perpendiculaires

gdrevHypothese(tig,
```

```
               false,
               [droite(_D1),
                droite(_D2),
                contreModele(perp(_D1, _D2)),
                perp(_D1, _D2),
                egalDr(_D1, _D2)]).

% -- Egalité de demi-droites ---------------------------------- %



% -- Egalité de segments -------------------------------------- %

% Deux segments sont égaux s'ils ont les mêmes extrémités

gdrevHypothese(tig,
               egalSeg(_S1, _S2),
               [segment(_S1, _A, _B, _D1),
                segment(_S2, _A, _B, _D2)]).

% -- Egalité de cercles --------------------------------------- %

% Deux cercles sont égaux s'ils ont même centre et même rayon

gdrevHypothese(tig,
               egalCc(_C1, _C2),
               [cercle(_C1, _O, _L),
                cercle(_C2, _O, _L)]).

% -- Egalité de distances ------------------------------------- %

% -- définition par points

% Deux distances sont égales si elles sont toutes les deux la
% distance séparant les deux mêmes points

gdrevHypothese(tig,
```

```
                egalDist(_L1, _L2),
                [point(_A),
                 point(_B),
                 distance(_L1),
                 contreModele(distPP(_L, _A, _B)),
                 distPP(_L1, _A, _B),
                 distPP(_L2, _A, _B)]).

% -- définition par tansitivité

% Deux distances sont égales si elles sont toutes les deux égales
% à une troisième distance

gdrevHypothese(tig,
                egalDist(_L1, _L3),
                [transitivite(decrementerTransitivite),
                 distance(_L1),
                 distance(_L2),
                 contreModele(egalDist(_L1, _L2)),
                 distance(_L3),
                 egalDist(_L1, _L2),
                 egalDist(_L2, _L3)]).
```

Annexe B

Exercices résolus par GDRev

Les tableaux B.1, B.2, B.3, B.4 et B.5 cataloguent des exercices traités avec GDRev sur un ordinateur PC à 300 MHz avec 64 Mo de RAM sous Windows NT. Pour chacun, nous spécifions son numéro dans [But75] ou [Pet90], son énoncé, le temps d'exécution et les agents intervenant selon qu'il s'agisse de la première construction ou de constructions "suivantes". On entend par construction suivante, une construction basée sur les mêmes ensembles d'objets de base et de propriétés auxquels sont ajoutés les objets et les propriétés redondants déterminés par la première construction. Ces constructions sont celles qui doivent être réalisées pour l'animation de la figure.

N°	Enoncé	Cons-truc-tion	Agents intervenant					Tps (sec)
			Lin	Qua	Cpt obj	Cpt pro	Int	
1	Construire un triangle rectangle isocèle en A, connaissant B et C	1	X		X		X	0.8
		suiv.	X				X	0.8
2	Construire un triangle rectangle et isocèle en B, connaissant B et C	1	X	X				0.06
		suiv.	X	X				0.06
4	Construire un triangle connaissant B, le milieu de AC, la longueur BC et la direction de BC	1	X	X				0.17
		suiv.	X	X				0.17
5	Construire un triangle connaissant B et C, et les longueurs AB et AC	1	X	X	X			0.05
		suiv.	X	X				0.03
6	Construire un triangle connaissant B et C, les directions des hauteurs issues de A et B	1	X					0
		suiv.	X					0
7	Construire un triangle connaissant B et C, les angles $\widehat{ABC}$ et $\widehat{ACB}$	1	X					0
		suiv.	X					0
8	Construire un triangle isocèle en A, connaissant B et C et la direction de BA	1	X	X	X			0.06
		suiv.	X	X				0
9	Construire un triangle isocèle en A, connaissant B et C, et la longueur AC	1	X	X	X			0.05
		suiv.	X	X				0
10	Construire un triangle isocèle en A, connaissant B et C,et la hauteur issue de A	1	X	X				0.05
		suiv.	X	X				0.05

TAB. B.1 – *Exemples de résultats obtenus.*

N°	Enoncé	Cons-truc-tion	Agents intervenant					Tps (sec)
			Lin	Qua	Cpt obj	Cpt pro	Int	
17	Construire un triangle rectangle en A tel que BC = 2AB, connaissant B et C	1	X				X	0.6
		suiv.	X				X	0.6
18	Construire un triangle connaissant A, les directions de AB et BC, l'angle $\widehat{A}$ et le rayon du cercle inscrit	1	X				X	1.98
		suiv.	X				X	1.98
19	Construire le cercle circonscrit à un triangle	1	X	X	X			0.17
		suiv.	X	X				0.09
20	Construire un cercle de centre donné et tangent à une droite	1	X	X				0
		suiv.	X	X				0
23	Construire un cercle passant par un point et tangent en un point à une droite	1	X	X	X			0.11
		suiv.	X	X				0.05
24	Construire un cercle passant par deux points et centré sur une droite	1	X	X	X			0.5
		suiv.	X	X				0
26	Construire un cercle de rayon donné et tangent à deux droites	1	X				X	2.58
		suiv.	X				X	2.58
29	Construire un cercle centré sur une droite et tangent en un point à une droite	1	X	X				0.06
		suiv.	X	X				0.06
44	On donne deux droites d et d' et un point A sur d. Construire une droite qui coupe d et d' en B et B' telle que AB = BB'	1						échec
		suiv.						échec

TAB. B.2 – *Exemples de résultats obtenus.*

N°	Enoncé	Cons-truc-tion	Agents intervenant Lin	Qua	Cpt obj	Cpt pro	Int	Tps (sec)
45	On donne un cercle c et deux points A et B hors du cercle. Construire une droite d contenant A et dont les intersections C et D avec le cercle soient équidistantes de B	1	X	X	X	X		73.24
		suiv.	X	X				0.45
55	Construire la tangent à un cercle issue d'un point donné	1	X				X	0.05
		suiv.	X				X	0.05
56	Construire la tangente à un cercle de direction donnée	1	X	X				0
		suiv.	X	X				0
58	On donne trois points A, B et C. Tracer le cercle passant par ces points, et tracer la corde AD issue de A et dont le milieu se trouve sur BC	1	X	X	X			0.77
		suiv.	X	X				0.11
65	Construire un cercle de rayon donné qui passe par deux points	1		X	X			0.17
		suiv.		X				0.06
69	Déterminer sur une circonférence de cercle un point qui soit à une distance donnée d'une droite	1	X				X	0
		suiv.	X				X	0
96	Construire un triangle connaissant B et C, la hauteur issue de B et le rayon du cercle circonscrit	1	X	X	X			0.88
		suiv.	X	X				0.06

TAB. B.3 – *Exemples de résultats obtenus.*

N°	Enoncé	Cons-truc-tion	Agents intervenant					Tps (sec)
			Lin	Qua	Cpt obj	Cpt pro	Int	
100	Construire un triangle connaissant B et C, la hauteur issue de B et la médiane issue de A	1	X					0
		suiv.	X					0
101	Construire un triangle connaissant A, le milieu de BC, la hauteur issue de A et la longueur AC	1	X	X				0.06
		suiv.	X	X				0.06
126	Dans un triangle, inscrire un triangle dont les directions des trois côtés sont données	1	X					0.05
		suiv.	X					0.05
127	Inscrire un carré dans un triangle	1	X				X	2.22
		suiv.	X				X	2.22
134	Sur une droite donnée, trouver un point qui soit à égale distance d'un point et d'une droite	1	X				X	0.49
		suiv.	X				X	0.49
P 53	Inscrire un triangle rectangle dont les côtés passent chacun par un point donné dans un cercle	1						échec
		suiv.						échec
P 368	Construire un rectangle dont les 4 côtés passent chacun par un point donné et dont les diagonales aient une longueur donnée	1						échec
		suiv.						échec
P 404	Dans un triangle ABC, inscrire 3 cercles tels que chacun d'eux soit tangent aux deux autres et à deux côtés du triangle	1						échec
		suiv.						échec

TAB. B.4 – *Exemples de résultats obtenus.*

N°	Enoncé	Cons-truc-tion	Agents intervenant Lin	Qua	Cpt obj	Cpt pro	Int	Tps (sec)
.	Construire un segment porté par deux droites sécantes étant donné son milieu	1	X					0
		suiv.	X					0
.	Construire un triangle connaissant A, les directions de AB et AC, et les longueurs des médianes issues de B et C	1					X	0.6
		suiv.					X	0.6
.	Construire un triangle connaissant B, le milieu de AC, et la longueur et la direction de la médianes issues de C	1	X	X				0.16
		suiv.	X	X				0.16
.	Construire un triangle connaissant ses trois médianes et tel que le sommet A soit sur une droite donnée	1	X					0
		suiv.	X					0
.	Inscrire un cercle dans un triangle donné	1					X	5.65
		suiv.					X	5.65
.	Construire une droite tangente à deux cercles	1					X	1.06
		suiv.					X	1.06

TAB. B.5 – *Exemples de résultats obtenus.*

Annexe C

Résultats des heuristiques de construction "linéaires" de figures

Les tableaux C.1, C.2, C.3, C.4, C.5 et C.6 donnent les exercices traités avec GDRev ayant pour objet la construction d'une figure géométrique avec choix libre de l'ensemble des objets de base. L'objectif est d'obtenir une construction possedant le minimum d'extensions algébriques. Pour chacun de ces exercices, nous spécifions son numéro dans [Cho88], le nombre d'équations quadratiques qu'il contient, les résultats obtenus manuellement par Bouhineau D [Bou97] (la colonne locales correspond au nombre d'extensions introduites si on s'autorise des modifications locales de la construction, la colonne globale correspond au nombre d'extensions introduites si on s'autorise des modifications locales et globales de la construction), ainsi que le nombre d'extensions introduites par nos heuristiques.

Nous rappelons que le problème aborder ici est d'examiner dans quelle mesure des heuristiques permettent de déterminer de manière déterministe un ensemble d'objets de base conduisant à une construction de la figure introduisant peu d'extensions algébriques. L'objectif visé est de se passer d'un algorithme d'énumération sur l'ensemble d'objets de base.

Ces résultats ont été obtenus sur un ordinateur PC à 300 MHz avec 64 Mo de RAM sous Windows NT. Les énoncés de ces exercices sont accessibles à l'URL :*http://www-lsr.imag.fr/pliage/GDRev.html*.

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
1	0	0	0	0	5.71	0	4.34
2	0	0	0	0	11.21	0	9.11
3	0	0	0	0	9.54	0	10.21
4	0	0	0	0	8.27	0	7.86
6	6	0	0	3	81.64	0	26.74
7	5	0	0	3	43.51	0	10.27
8	0	0	0	0	11.02	0	2.25
10	5	0	0	échec		3	33.01
11	5	0	0	échec		3	28.13
12	5	0	0	échec		échec	
13	5	0	0	échec		0	24.01
14	5	0	0	0	22.08	1	41.19
19	5	0	0	1	157.92	1	230.53
23	0	0	0	0	28.78	0	34.17
25	3	0	0	échec		0	71.08
29	4	0	0	2	11.48	0	7.03
31	2	0	0	0	12.79	0	16.53
35	5	1	1	2	12.63	2	25.60
39	1	1	1	échec		échec	
40	1	1	1	échec		échec	
43	10	1	1	échec		échec	
44	12	1	1	échec		échec	
45	10	1	1	échec		échec	
48	4	0	0	2	27.74	4	19.59
65	9	0	0	échec		3	110.62
66	6	0	0	échec		1	42.08
70	4	2	0	0	3.18	0	2.75
72	6	0	0	1	85.25	1	28.01
77	5	0	0	1	70.36	1	36.42
79	6	2	0	0	2.37	0	1.75
80	8	0	0	3	47.46	0	146
87	4	1	1	2	21.97	1	14.45
88	1	0	0	0	8.17	1	52.18
89	1	1	1	échec		1	23.35
90	4	0	0	échec		2	10.71

TAB. C.1 – *Exemples de résultats obtenus.*

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
94	3	1	0	0	25.08	1	28.24
95	1	1	0	1	24.6	0	7.03
96	5	0	0	2	12.24	3	9.61
97	2	1	0	échec		1	7.58
99	4	0	0	2	45.37	3	10.60
104	5	1	1	1	12.79	2	4.67
106	5	1	1	3	14.45	3	52.84
107	4	0	0	2	4.6	0	32.13
108	3	1	0	0	3.91	0	1.86
109	4	0	0	0	62.56	0	69.26
110	1	1	0	1	4.11	0	2.69
111	4	2	0	échec		3	15.32
112	2	1	0	échec		1	4.77
113	3	2	0	échec		0	6.53
115	4	0	0	0	17.68	3	30.87
116	5	0	0	1	29.11	3	33.18
117	3	1	0	1	11.01	0	10.33
118	4	0	0	2	12.69	2	21.69
119	5	1	1	0	49.38	0	31.58
123	4	0	0	0	15.93	0	35.37
125	4	2	1	1	10.63	3	12.14
128	3	1	1	1	9.56	2	20.43
129	2	1	1	1	8.46	2	11.86
130	4	0	0	échec		2	13.08
131	4	1	0	échec		échec	
132	2	1	1	1	11.86	2	12.79
135	3	3	0	échec		échec	
136	1	1	0	0	18.29	0	44.87
137	3	1	0	1	13.17	2	22.3
138	3	1	0	1	12.71	1	7.36
139	3	1	0	0	31.54	2	30.1
143	4	0	0	échec		échec	
144	3	0	0	échec		0	90.18
152	2	2	0	0	66.02	0	41.03
153	1	1	0	échec		0	11.64

TAB. C.2 – *Exemples de résultats obtenus.*

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
162	5	3	0	0	43.15	échec	
163	5	3	0	0	67.51	0	57.23
172	2	2	0	0	44.89	0	17.96
173	0	0	0	0	0.77	0	1.65
175	4	0	0	2	13.23	0	56.19
176	1	1	0	0	5.51	1	2.76
180	4	0	0	2	14.11	1	50.32
184	5	0	0	1	11.86	3	28.67
185	0	0	0	0	4.67	0	6.76
190	2	0	0	0	22.47	0	18.51
195	3	0	0	échec		0	28.18
196	2	0	0	échec		0	21.69
197	2	0	0	0	5.83	0	27.19
198	4	0	0	échec		2	48.72
199	4	0	0	échec		1	71.29
200	4	0	0	0	39.49	1	55.86
201	4	0	0	échec		échec	
202	4	0	0	échec		échec	
203	3	1	0	0	15.26	0	20.93
208	6	1	1	2	33.77	2	38.44
212	8	0	0	3	35.32	0	46.63
215	6	0	0	2	35.75	0	92.27
216	4	0	0	0	4.67	0	15.77
217	8	0	0	3	24.33	0	165.71
219	4	1	0	0	16.59	1	7.25
220	3	1	0	0	19.72	0	8.57
222	1	1	0	échec		0	5.49
223	5	0	0	3	5.49	3	6.31
224	0	0	0	0	2.53	0	2.58
225	5	0	0	1	11.31	1	13.49
236	5	0	0	échec		3	71.12
237	4	0	0	1	36.09	2	35.04
239	4	0	0	0	20.38	0	52.12
240	10	0	0	4	24.33	1	39.11
241	4	0	0	échec		échec	

TAB. C.3 – *Exemples de résultats obtenus.*

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
243	10	1	1	1	45.59	1	47.77
244	6	0	0	0	17.35	0	18.51
250	3	0	0	0	26.3	0	24.5
258	1	1	0	0	12.74	1	5.33
266	3	0	0	0	8.67	0	13.07
269	5	0	0	échec		échec	
277	4	0	0	0	43.07	0	49.22
278	9	0	0	4	45.86	0	86.18
281	3	0	0	0	7.09	2	9.78
282	3	0	0	3	19.86	2	40.15
283	3	0	0	échec		1	32.02
287	4	0	0	0	14.94	2	8.46
289	9	0	0	0	49.88	1	183.18
290	9	0	0	1	88.1	0	183.45
291	6	0	0	2	15.16	1	42.79
292	4	0	0	2	10.88	échec	
294	5	0	0	échec		0	60.58
298	5	1	0	3	38.61	2	50.37
300	3	1	0	échec		2	13.45
302	4	0	0	échec		2	87.55
303	4	0	0	échec		2	10.22
304	5	0	0	échec		0	22.74
305	4	0	0	2	21.42	1	12.69
306	5	0	0	2	44.76	0	32.07
307	5	0	0	3	26.59	3	18.23
308	4	0	0	2	27.68	2	9.29
309	8	0	0	2	43.01	0	92.32
310	6	0	0	2	127.15	2	39.82
315	9	0	0	échec		4	13.85
321	0	0	0	0	6.31	0	7.14
331	4	0	0	0	37.62	0	14.28
333	1	1	0	0	10.22	1	10.93
339	4	0	0	échec		échec	
349	5	0	0	1	79.31	1	39.5
357	1	1	0	0	34.99	0	5.05

TAB. C.4 – *Exemples de résultats obtenus.*

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
363	1	1	0	0	31.42	1	15.27
364	4	0	0	échec		0	8.74
372	4	2	0	2	44.55	1	26.14
376	7	0	0	0	31.86	1	6.32
378	3	1	0	0	48	0	17.96
379	3	1	0	2	35.26	1	11.09
391	4	0	0	échec		0	8.79
393	5	1	1	échec		3	46.42
394	3	1	0	1	60.31	2	86.61
395	3	1	0	1	50.48	1	27.74
396	5	0	0	1	36.42	2	11.64
398	3	1	0	0	15.82	0	12.09
400	4	0	0	échec		0	8.35
401	4	0	0	1	65.63	3	9.01
406	4	0	0	1	14.77	0	5.77
407	2	1	0	0	44.22	1	9.94
409	3	1	0	0	16.04	0	19.45
417	4	0	0	3	18.92	3	17.59
424	3	1	0	0	26.42	1	5.66
428	5	0	0	échec		4	16.76
429	4	0	0	échec		échec	
432	3	1	0	échec		échec	
445	4	0	0	échec		4	31.32
449	4	0	0	échec		4	35.97
450	4	0	0	échec		4	46.72
451	4	0	0	échec		4	56.63
452	5	1	0	2	176.14	échec	
453	2	1	1	échec		échec	
456	5	1	0	échec		échec	
457	4	1	0	échec		échec	
458	5	2	0	échec		échec	
459	4	0	0	échec		échec	
461	5	1	0	échec		échec	
462	4	0	0	1	123.98	3	45.06
470	5	0	0	échec		échec	

TAB. C.5 – *Exemples de résultats obtenus.*

N°	Nb d'équations du second degré	Résultats de Bouhineau D		Heuristique non construit		Heuristique plus contraint	
		Nb ext. locales	Nb ext. globales	Nb ext.	Tps (sec)	Nb ext.	Tps (sec)
476	5	0	0	2	38.01	1	17.58
478	2	2	0	2	346.03	2	113.7
483	5	0	0	2	47.13	1	19.89
484	5	0	0	3	23.62	1	6.1
485	4	0	0	3	18.35	1	8.45
486	8	0	0	3	108.7	2	49.32
487	8	0	0	3	81.78	3	34.16
488	10	0	0	3	127.81	1	115.84
489	4	0	0	0	66.9	2	19.55
490	8	0	0	échec		échec	
491	6	0	0	échec		4	131.39
492	5	2	1	échec		4	152.92
495	5	0	0	1	36.52	1	22.08
497	3	0	0	0	13.79	0	20.87
501	8	0	0	6	57.18	0	98.43
504	4	1	0	échec		échec	
505	4	1	0	2	53.44	2	61.51
510	5	0	0	échec		2	42.73

TAB. C.6 – *Exemples de résultats obtenus.*

Résumé : Cette thèse a pour objet de montrer la faisabilité d'un système de "géométrie dynamique déclarative". Un tel système, GDRev (pour *Géométrie Déclarative Réversible*) a été conçu et réalisé, dans l'optique de l'enseignement de la géométrie.

D'un point de vue conceptuel, GDRev repose sur la définition d'un langage logique, ELDL (pour *Extented Logical Description Language*), pour l'expression de spécifications de "figures" (l'objet mathématique sous-jacent à un dessin) : il intègre la possibilité de spécifications modulaires et récursives, via l'usage de "clauses". Au niveau dessin, GDRev est pourvu d'un langage de construction et d'animation dont la sémantique est définie à l'aide de ELDL. l'interface, qui peut être vu comme une extension déclarative de celle de Cabri-Géomètre, doit assurer, d'une façon originale, d'une part des fonctionnalités équivalentes par manipulation directe sur la figure et sur le dessin, d'autre part un invariant imposant la cohérence temporelle entre figure et dessin.

D'un point de vue algorithmique, GDRev résout les contraintes géométriques par "coopération de solveurs" reposant sur un schéma de "programmation concurrente avec contraintes". Trois résolveurs généraux (*linéaires, quadratiques, intervalle*) coopèrent avec trois résolveurs spécifiques et originaux : *complétion d'objets* (créant automatiquement des objets), *complétion de propriétés* (ajoutant automatiquement des propriétés redondantes à la figure), *règle et compas* (calculant une construction optimisée de la figure pour l'animation du dessin).

D'un point de vue pratique, GDRev est réalisé par interopérabilité entre les interfaces écrites en Visual C++ et le résolveur de contraintes géométriques écrit en Prolog IV. Les expérimentations réalisées ont donné des résultats encourageants en particulier en ce qui concerne le choix des heuristiques utilisées.

Mots clés : Géométrie dynamique déclarative, Programmation concurrente avec contraintes, Manipulation directe, Prolog IV.

Abstract : The objective of this thesis is to demonstrate the feasibility of a declarative dynamic geometry system. GDRev, which stands for Reversible Declarative Geometry, is such a system and has been design and implemented with the teaching of geometry in mind.

From the logical point of view, GDRev is based on the definition of a logic language ELDL, which stands for Extented Logical Description Language. LDL is used to express specifications of geometric figures which form the mathematical objects underlying drawings of figures. LDL provides clauses which allow specifications to be recursive and allow specifications to be expressed in modular format. GDRev possesses a construction and animation language whose semantics are defined using ELDL. The GDRev interface can be viewed as a declarative extension of that of Cabri-Géomètre. On the one hand, the interface must provide equivalent direct manipulation operations carried out on the figure and on the drawing, and, on the other hand, it must maintain at all times a coherence between the figure and drawing.

From the algorithmic point of view, GDRev solves geometric constraints using cooperation of solvers based on the concurrent constraint programming paradigm. Three general solvers, one linear, one quadratic, one interval, cooperate with three specific solvers that are original to this work. One of the specific solvers is the object completion solver which creates automatically geometric objects necessary for constructing the figure. The second is the property completion solver which adds automatically redondant properties to obtain a construction of the figure. The ruler and compass solver calculates an optimized construction with which to drag the figure rapidly.

GDRev is implemented using interaction between the interfaces which are written in Visual C++ and the geometric constraint solver which is written in Prolog IV. The tests carried out on the system have given encouraging results, especially those concerning the choice of heuristics used.

Keywords : Declarative dynamic geometry, Concurrent constraint programming, Direct manipulation, Prolog IV.