Interprétation d'un programme avec multifonctions géométriques

C. Essert, P. Schreck et J.-F. Dufourd

LSIIT - UPRES-A CNRS 7005
Université Louis Pasteur
Pôle API, Bd. Sébastien Brant
67100 Strasbourg
{essert, schreck, dufourd}@dpt-info.u-strasbg.fr

Résumé : *Dans le domaine de la CAO, la possibilité de définir et manipuler des figures à l'aide d'un système de cotes est facilitée par la construction géométrique* formelle *de la figure. C'est l'approche qui a été suivie dans le logiciel YAMS. La résolution formelle d'un système de cotes produit alors un plan de construction contenant des* multifonctions *pour tenir compte des solutions multiples. L'interprétation numérique d'un plan de construction doit tenir compte de ces multifonctions et permettre à un utilisateur de parcourir efficacement l'espace des solutions. Cet article présente une spécification algébrique du cadre géométrique utilisé par YAMS pour définir clairement les notions de plan de construction et leur interprétation. Il montre ainsi qu'un tri topologique bien adapté sur le graphe de dépendance facilite le parcours de l'espace des solutions.*

Mots-clés : Constructions géométriques formelles, univers géométrique, YAMS, spécifications algébriques, parcours d'arbres.

1 Introduction

La plupart des logiciels de Dessin et Conception Assistés par Ordinateur offrent la possibilité de tracer à l'écran et d'explorer numériquement des figures géométriques définies par un système de cotes. Cette fonctionnalité de haut niveau est souvent réalisée en résolvant numériquement le système d'équations correspondant à la cotation. La résolution formelle du système de cotes symboliques offre l'avantage de pouvoir efficacement manipuler la figure définie en faisant varier les valeurs des paramètres. Malheureusement, les outils algébriques de calcul formel sont beaucoup trop généraux pour pouvoir être efficaces à cet effet. L'approche que nous privilégions au sein du LSIIT est à la fois formelle et géométrique : les outils de résolution que nous proposons sont basés sur la géométrie classique, notamment le domaine des constructions géométriques, et ils sont suffisamment spécialisés pour être efficaces dans un cadre formel. Alors, dans une première phase, le résultat d'une résolution formelle d'un système de cotes est un *plan de construction géométrique* décrivant, dans l'ordre, la suite des objets à construire et des opérations à appliquer pour obtenir une figure solution. Une deuxième phase consiste ensuite à interpréter numériquement ce plan de construction en attribuant aux paramètres les valeurs de la cotation. Cette approche a été concrétisée par le logiciel YAMS [Mat97, DMS98] associant un solveur géométrique formel et un modeleur 3D basé sur le modèle des G-cartes [Ber98, BDFL93].

Quelle que soit l'approche utilisée, numérique ou formelle, algébrique ou géométrique, un système de cotes ne définit habituellement pas une figure unique. Lorsqu'un nombre infini de figures répond à la cotation, on dit que le système est sous-contraint. Lorsqu'il y a un nombre fini non-nul de solutions, on dit que le système est bien contraint. C'est ainsi que le dessin coté à la figure 1, qui correspond à un système bien contraint (modulo les déplacements), conduit aux deux solutions qui sont données graphiquement à la figure 2.

Dans le cas où un système est bien contraint, l'exploration de l'espace des solutions n'est pas aussi facile qu'on pourrait le penser [Cha98]. En effet, l'existence d'équations polynomiales de degré supérieur ou égal à 2, d'un point de vue algébrique, ou d'intersections multiples, d'un

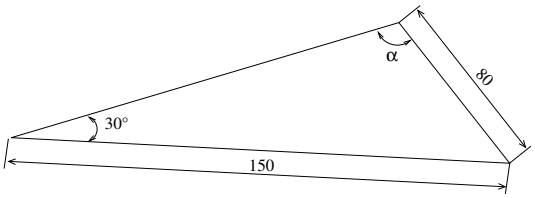

FIG. 1 – *esquisse cotée*

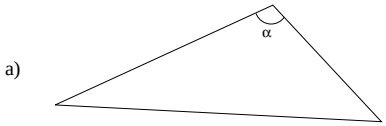

FIG. 2 – *deux solutions différentes : a) α est un angle obtus, b) α est un angle aigu*

point de vue géométrique, conduit rapidement à une explosion combinatoire du nombre des solutions. Avec notre approche, cette multiplication du nombre des solutions est le résultat de l'existence de multifonctions dans le plan de construction. L'objectif de ce premier article sur le sujet est de montrer comment l'approche géométrique formelle permet de structurer cet espace des solutions et d'introduire quelques méthodes pour tenter de le parcourir efficacement.

Une des autres préoccupations de notre équipe a trait au domaine du génie logiciel et concerne la description rigoureuse des modèles utilisés et de leur implantation [BD98]. C'est pourquoi nous utilisons dans cet article le formalisme des spécifications algébriques, et notamment le langage OBJ3 [GWM+92] pour présenter l'interprétation des plans de construction géométriques. Notons que toutes les notions présentées ont été spécifiées en OBJ3 et testées sur des jeux d'essais.

Cet article se découpe de la manière suivante. A la section 2, nous présentons rapidement le logiciel YAMS en insistant sur l'aspect solveur géométrique. A la section 3, nous spécifions la notion d'univers géométrique utilisée par YAMS. A la section 4, nous précisons ce que signifie l'interprétation d'un plan de construction en spécifiant notamment les arbres d'interprétation. A la section 5, nous indiquons comment on peut espérer réduire de manière statique la complexité de l'interprétation. Finalement, nous concluons à la section 6 en esquissant des méthodes d'élagage de l'arbre des interprétations.

2 Présentation de YAMS

YAMS est un solveur géométrique 2D formel associé au modeleur 3D TOPOFIL [BDFL93]. Une description précise de cette association pouvant être trouvée dans [Mat97, DMS97], nous ne présentons ici que l'aspect solveur de YAMS. La méthode de résolution utilisée par YAMS agit par décomposition ascendante du système de contraintes géométriques. Cette décomposition est basée sur l'invariance par déplacement des systèmes de cotes, tandis que la résolution effective des sous-systèmes est laissée à des méthodes de résolution indépendantes. Actuellement, YAMS considère deux classes de résolution locales. La première repose sur la méthode des lieux [Leb50] et la deuxième fabrique un petit système polynomial destiné à être interprété par une méthode numérique, celle de Newton-Raphson par exemple, ou par homotopie [LM95].

Cette collaboration de plusieurs méthodes locales est réalisée dans YAMS par une architecture de système multi-agents avec tableau noir. Les méthodes géométriques sont implantées par des agents experts qui sont des systèmes à base de production. La méthode analytique applique un algorithme de recherche de sous-systèmes bien formés minimaux inspiré de celui de König-Hall. Le tableau noir contient les informations communes à tous les agents, comme le système de contraintes initial et les parties déjà résolues. Ce tableau noir est géré par un agent particulier,

FIG. 3 – *plan de construction exhibé par YAMS et sortie graphique d'une solution*

le superviseur, qui planifie également le lancement des agents.

Lorsqu'une esquisse cotée est tracée par l'utilisateur, YAMS nomme les objets géométriques (points, droites, etc.) en jeu par des identificateurs, abstrait les cotes numériques en paramètres formels et engendre un système de contraintes géométriques. Puis, dans une première phase, il produit des définitions en fonction des contraintes posées. Ces définitions qui font correspondre des termes fonctionnels à des identificateurs sont regroupées en un *plan de construction*. Celui-ci indique comment et dans quel ordre doivent être tracés les différents objets géométriques constituant la figure. Dans une deuxième phase, les cotes effectives données au départ sont ensuite passées comme paramètres pour l'interprétation du plan de construction.

Reprenons l'exemple donné en introduction. L'utilisateur trace l'esquisse cotée donnée en figure 1. YAMS attribue alors des noms symboliques aux différents objets de l'esquisse, puis exhibe un plan de construction, présenté à gauche dans la figure 3. Après l'interprétation numérique du plan de construction, les figures solutions sont produites, comme montré à la figure 3, à droite.

Les fonctions utilisées lors de l'interprétation peuvent parfois donner des résultats multiples. Par exemple, l'intersection d'une droite et d'un cercle donne deux points. De telles fonctions retournant un ensemble de valeurs, sont appelées *multifonctions*. L'existence de multifonctions dans un plan de construction conduit à la création de points de choix dans le processus d'interprétation. Actuellement, YAMS parcourt l'espace de toutes les solutions possibles par *backtracking*.

3 Spécification d'un univers géométrique et des plans de construction

Nous décrivons l'univers géométrique actuel de YAMS dans le formalisme des spécifications algébriques [Wir90]. Le langage de spécification OBJ3 utilisé permet une description concise, rigoureuse et sans ambiguïté des composants logiciels tout en s'affranchissant des détails d'implantation.

3.1 Méta-spécification d'univers géométriques

Dans la perspective d'une utilisation par YAMS de différents univers géométriques, voire de leur définition par un expert, nous ne souhaitons pas décrire *un* univers géométrique particulier, mais la manière de constituer des univers. Ainsi, cette section présente une description axiomatique, sous forme de spécification algébrique, d'un cadre métathéorique dans lequel nous allons pouvoir considérer par exemple la sorte *sorte géométrique* ou la sorte *symbole fonctionnel*. De cette manière, nous pourrons traiter les symboles fonctionnels et prédicatifs comme des objets sur

lesquels nous effectuerons des opérations, comme la construction de termes abordée plus loin.

Tout d'abord, nous montrons comment, dans ce cadre, définir l'univers géométrique. La signature comprend une sorte `S` des sortes géométriques, une sorte `F` de symboles fonctionnels et une sorte `P` des symboles prédicatifs. Les symboles fonctionnels servent à décrire des constructions, tandis que les symboles prédicatifs servent à décrire des contraintes. Dans cet article, nous nous focalisons sur les symboles et termes fonctionnels qui interviennent directement dans la constitution des plans de construction. Notons à cet égard, que nous interprétons toujours les symboles fonctionnels par des multifonctions. C'est pourquoi, nous employons indifféremment les termes fonction et multifonction dans la suite.

Les sortes géométriques sont vues comme des constantes de sorte `S`. Les symboles fonctionnels et prédicatifs considérés comme des constantes, respectivement de sorte `F` et `P`, sont munis d'une arité et d'une coarité. Les symboles fonctionnels et prédicatifs sont ici spécifiés par les sortes `F` et `P`. Nous avons ainsi l'extrait de spécification suivant :

```
sort S F P .
ops point line circle long angle depl : -> S .
ops center-of medradcir : -> F .
ops perp distpp : -> P .
```

Les symboles fonctionnels `center-of` et `medradcir` sont mis respectivement pour les fonctions donnant le centre d'un cercle et le cercle passant par deux points connus et de rayon connu. Les symboles prédicatifs `perp` et `distpp` désignent respectivement les contraintes de perpendicularité entre deux droites et de distance entre deux points.

Un des intérêts du passage au niveau métathéorique est de pouvoir attacher des informations supplémentaires aux symboles fonctionnels. C'est ainsi qu'à la notion de profil d'un symbole fonctionnel comprenant l'arité et la coarité, nous adjoignons deux entiers, correspondant respectivement aux *degrés de multiplicité* et de *risque* des fonctions qu'ils représentent.

Le degré de multiplicité M est le cardinal maximal de l'ensemble calculé par une fonction. Par exemple, la fonction qui calcule les droites tangentes à deux cercles possède degré de multiplicité de 4, car elle peut produire de 0 à 4 droites selon les positions relatives des cercles. Une vraie fonction, comme `center-of`, a un degré de multiplicité de 1.

Le degré de risque R est un coefficient qui quantifie la probabilité qu'a une multifonction d'échouer, i.e. de retourner un ensemble vide. Le centre d'un cercle n'ayant aucune chance d'échouer, son coefficient de risque est 0, alors que l'intersection d'une droite et d'un cercle est bien plus hasardeuse et possède un degré de risque de 4. Toutes ces informations sont contenues dans le *profil* du symbole fonctionnel. Ce profil est spécifié en OBJ3 par la ligne suivante :

```
op _=>_M_R_ : Slist S Int Int -> Profil .
```

où `Slist` est la sorte des listes de sortes géométriques séparées par ". Ainsi, le profil du symbole `medradcir` est codé par l'équation :

```
eq profil(medradcir) = Point " Point " Long => Circle M 2 R 3 .
```

Dans notre univers géométrique, nous faisons usage de symboles de variables que nous nommons identificateurs simples. Ces symboles sont typés en leur adjoignant une sorte géométrique. Pour distinguer des objets définis dans des sous-figures différentes, nous indiçons les identificateurs typés en leur accolant le numéro de la sous-figure où ils sont définis. Finalement, nous spécifions la sorte `I` des *identificateurs indicés* à l'aide du constructeur en notation infixée suivant.

```
op _#_._ : S Id Index -> I .
```

Les symboles fonctionnels, ou prédicatifs, et les identificateurs indicés sont utilisés pour construire des *termes*, comme nous l'expliquons à la section suivante.

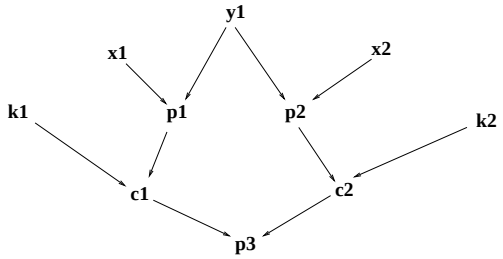

FIG. 4 – *Graphe de dépendance*

3.2 Termes et définitions

Les termes sont formés sur le modèle classique des termes du premier ordre. Un terme est donc soit un simple identificateur indicé de sorte `I`, soit un terme fonctionnel de sorte `Termf`, soit un terme prédicatif de sorte `Termp`. La notion de sous-sortage permet de hiérarchiser ces sortes de termes : `subsorts Termf Termp I < Term .` Avec le formalisme d'OBJ3, nous décrivons ainsi les termes fonctionnels et prédicatifs :

```
op _[_] : F Ilist -> Termf .
op _[_] : P Ilist -> Termp .
```

où `Ilist` est une liste d'identificateurs indicés appelée arguments. Par exemple, le terme prédicatif `perp[line#d1.1 " line#d2.1]` représente la contrainte imposant aux droites `d1` et `d2` d'être perpendiculaires, dans la sous-figure 1.

Une définition, spécifiée par la sorte `Def`, est la mise en correspondance d'un identificateur indicé et d'un terme fonctionnel bien formé. Une définition décrit la construction d'un objet géométrique désigné par son identificateur indicé. Le terme fonctionnel associé contient les objets géométriques dont dépend sa construction ainsi que la fonction qui doit leur être appliquée. La forme générale d'une définition est donnée par le constructeur `:=` dont le profil en OBJ3 est :

```
sort Def .
op _:=_ : I Termf -> Def .
```

Par exemple, `line#d1.1 := lpp[point#p1.1,point#p2.1]` définit dans la sous-figure 1 la droite `d1` passant par les points `p1` et `p2`.

La définition d'un paramètre est traitée comme un cas particulier, et possède la forme suivante : `i := param`, où `i` est un identificateur indicé.

3.3 Ensemble de définitions et graphe de dépendance

Si l'on fait abstraction de l'ordre d'apparition des définitions dans un plan de construction, on obtient un ensemble de définitions, où chaque définition se rapporte à un élément différent. Nous appelons un tel ensemble un ensemble non ambigu de définitions (NADS).

Un NADS contient une relation de dépendance entre les définitions. Cette relation se traduit par un graphe orienté sur les définitions que nous appelons le *graphe de dépendance*. Il existe un arc d'une définition à une autre si et seulement si l'identificateur défini dans la première fait partie des arguments de la deuxième. Les définitions sans argument déjà défini, ou qui sont des paramètres, sont des sources, et les définitions qui ne sont pas argument dans une autre définition sont des puits.

La figure 4 présente un exemple de graphe de dépendance associé au NADS suivant :

```
p1 := initp[x1 y1]        y1 := param            p3 := intercc[c1 c2]
x2 := param               k1 := param            x1 := param
c2 := mkcir[p2 k2]        c1 := mkcir[p1 k1]
p2 := initp[x2 y1]        k2 := param
```

Pour que le NADS corresponde effectivement à un plan de construction, il faut que le graphe associé soit *triangulaire*, ce qui signifie qu'il doit impérativement remplir les trois conditions suivantes :

(i) toutes ses sources doivent être des paramètres
(ii) les arguments des définitions doivent être tous définis
(iii) il ne doit pas contenir de circuit

On peut alors ordonner les définitions d'un NADS en faisant un *tri topologique* du graphe de dépendance pour retrouver une liste de définitions, i.e. un plan de construction. Naturellement, plusieurs tris topologiques sont possibles, et nous verrons à la section 5 des heuristiques de tris pour que le plan produit puisse être interprété efficacement.

3.4 Tri topologique paramétré par une fonction de choix

Rappelons en quelques mots le principe du tri topologique. On construit progressivement une liste de définitions en ajoutant à chaque fois un élément complètement défini choisi parmi un ensemble de successeurs d'éléments déjà triés. L'ensemble de départ contient les sources du NADS. Le choix de l'élément à un insérer dans la liste résultat est fait par la fonction `choice` de profil `op choice: Def-set Ilist ->Def .` Nous présentons ci-dessous une spécification paramétrée par la fonction `choice` pour cet algorithme.

```
op topo-sort : Def-set -> Def-list .
op topo-sort : Def-set Def-set Ilist -> Def-list .      *** surcharge
op topo-sort : Def-set Def-set Ilist Def -> Def-list .  *** surcharge

var ds cur-ds : Def-set .  var il : Ilist . var ch : Def .

eq topo-sort(ds) = topo-sort(defs(sources(ds),ds),ds,i-emptylist) .
eq topo-sort(d-emptyset,ds,il) = d-emptylist .
eq topo-sort(cur-ds,ds,il) = topo-sort(cur-ds,ds,il,choice(cur-ds,il)) .
eq topo-sort(cur-ds,ds,il,ch) =
   ch " topo-sort(union(del(ch,cur-ds),succ(ch,ds)), ds, ident(ch) " il) .
```

Pour isoler les paramètres d'un plan et leur associer ainsi plus facilement des valeurs, nous préférons utiliser la notion de *fonction de construction*. Une telle fonction est la concaténation d'une liste d'identificateurs indicés, qui décrivent des paramètres, et d'une liste de définitions, qui correspond au plan de construction amputé des définitions des paramètres.

4 Interprétation d'une fonction de construction

Intuitivement, l'interprétation d'une fonction de construction a pour but d'associer à chaque objet géométrique défini dans la fonction une valeur numérique correspondant à ses coordonnées dans le plan euclidien. Les valeurs de départ, les paramètres effectifs, sont données par l'utilisateur. En fait, l'interprétation de certains symboles fonctionnels par des multifonctions conduit à considérer des ensembles de coordonnées pour chacun des objets calculés. Le résultat d'une interprétation d'une fonction de construction est donc en réalité un arbre de valeurs numériques.

La fonction `interp`, qui réalise cette interprétation, a pour profil

```
op interp : Fun Numlist -> IT .
```

où `Numlist` désigne la sorte des listes de valeurs numériques, et `IT` la sorte des *arbres d'interprétation*. Si la sorte `Numlist` n'appelle aucun commentaire particulier, la sorte `IT` doit en revanche être expliquée plus précisément.

FIG. 5 – *solutions résultantes*

4.1 Arbre d'interprétation

Concrètement, un arbre d'interprétation est construit niveau par niveau, en interprétant successivement les définitions de la fonction de construction. A chaque définition correspond un niveau. Chaque noeud contient un système de coordonnées possibles de l'objet défini à ce niveau. La valeur d'un noeud est trouvée en interprétant la définition correspondant au niveau et en utilisant comme arguments les valeurs numériques des ancêtres. Plusieurs fils sont créés dans le cas d'une vraie multifonction.

C'est ainsi que l'interprétation du plan de construction suivant avec des valeurs effectives de l'énoncé produit l'arbre d'interprétation de la figure 6, en admettant que les cercles `c1` et `c2` sont sécants, on obtient une sortie graphique comme à la figure 5.

```
x1 := param                    p2 := initp[x2 y2]
y1 := param                    c1 := mkcir[p1 k1]
x2 := param                    c2 := mkcir[p2 k2]
y2 := param                    i1 := intercc[c1 c2]
k1 := param                    d1 := lpp[p1 p2]
k2 := param                    i2 := interlc[d1 c1]
p1 := initp[x1 y1]             d2 := lpp[i1 i2]
```

La manière de spécifier les arbres d'interprétation doit refléter leur mode de construction durant l'interprétation. C'est pourquoi nous avons choisi de spécifier l'arbre d'interprétation d'une façon particulière que nous appelons codage par le nombre de frères. C'est une variante du codage par occurrences, avec la différence que chaque noeud est désigné non pas par son numéro dans la fratrie mais par son nombre de frères.

Dans notre spécification en OBJ3, un arbre est une liste de *niveaux* (`Floor`). Chaque niveau est étiqueté par un identificateur indicé, et comporte une liste d'entiers représentant le codage par le nombre de frères (`Brothers`) ainsi que la liste des coordonnées calculées pour cet identificateur (`Coordlist`). Ceci se traduit par les lignes OBJ3 suivantes:

```
sort Floor .
including LIST[Floor]*(sort List to IT , op emptylist to emptytree)
op _|_|_ : I Brothers Coordlist -> Floor .
```

Le codage pour notre exemple est donné à la figure 6, avec `Vsi` la valeur de la solution `si` pour l'objet `s`. Ceci signifie que `Vi1` et `V'i1` sont deux frères et ont le même père, tandis que les deux `Vd1` n'ont pas le même père et ne sont pas frères, chacun est fils unique. Pourtant, on les duplique car chacun correspond à une branche initialisée par `Vi1`, `V'i1`, etc. Lorsqu'on additionne tous les frères pour un objet, on retrouve le nombre de noeuds présents à la hauteur de l'arbre qui correspond à cet objet.

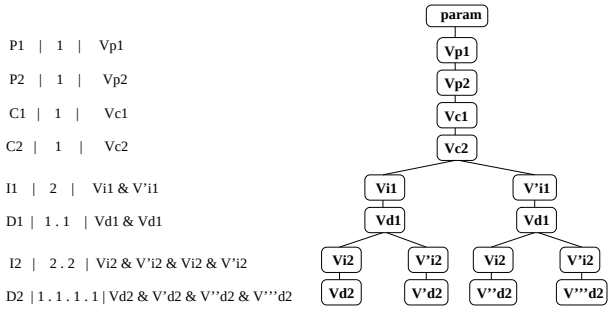

FIG. 6 – *arbre des solutions*

4.2 Interprétation

L'interprétation d'une fonction de construction consiste à examiner successivement les définitions dans l'ordre où elles apparaissent, pour produire un arbre d'interprétation.

L'interprétation d'une définition à partir d'un arbre partiel ajoute un nouveau niveau à cet arbre. Ceci correspond à la fonction `interp`, dont la spécification est donnée ci-dessous, et qui surcharge la fonction `interp` donnée en début de section. Ce nouveau niveau est formé de l'identificateur de la définition et du code (de sorte `Code`) obtenu par interprétation du terme. L'interprétation du terme, réalisée également par une surcharge de la fonction `interp`, construit le code pour le nouveau niveau en calculant une nouvelle descendance pour chaque ancienne feuille. Chaque descendance est le résultat de l'interprétation du symbole fonctionnel avec des valeurs extraites de la branche partant de la feuille. Il est créé autant de nouvelles feuilles que de valeurs résultant de l'interprétation du symbole fonctionnel.

```
op interp : Def-list IT -> IT .        *** interprete une liste de définitions
op interp : Term IT -> Code .          *** interprete un terme
op interp : F Coordlist -> Coordlist .  *** interprete un symbole fonctionnel
```

Les multifonctions utilisées pour interpréter les symboles fonctionnels peuvent, selon les valeurs passées en arguments, ne donner aucune solution, comme par exemple l'intersection de deux droites parallèles. Dans ce cas, la valeur renvoyée est `fail`.

Le nombre de résultats obtenus pour toute l'interprétation est égal au nombre de branches de l'arbre d'interprétation. Il est au plus égal au produit des degrés de multiplicité de chacune des multifonctions qui interviennent. On ne peut donc éviter une explosion exponentielle de l'espace des solutions. Actuellement, l'exploration de cet espace se fait par backtracking dans l'arbre. Pour améliorer l'efficacité de ce backtracking, on peut agir de manière statique en modifiant l'ordre des définitions dans le plan de construction tout en respectant le graphe de dépendance. Nous montrons dans la section suivante que ceci peut être obtenu en instanciant la fonction `choice` du tri topologique décrit à la section 3.4 avec des heuristiques particulières. Une autre possibilité, qui n'est pas décrite dans cet article, est de s'aider de l'esquisse pour ne conserver que les solutions les plus intéressantes. Cette approche fait actuellement l'objet de recherches dans notre équipe.

5 Modification de l'ordre d'un plan de construction

Nous présentons différentes fonctions de choix pour paramétrer le tri topologique d'un NADS.

5.1 Réduire le nombre de noeuds

L'exploration complète de l'arbre d'interprétation conduit à examiner tous les noeuds de l'arbre. En notant d_i le degré de multiplicité de la définition de niveau i, le nombre de noeuds est égal

à $\sum_{j=1}^{n} \prod_{i=1}^{j} d_i$. Cette quantité est évidemment rendue minimale lorsque les d_i sont rangés par ordre croissant. En faisant remonter vers la racine de l'arbre d'interprétation les définitions qui ont les degrés de multiplicité les plus faibles, on réduit ainsi le nombre de noeuds à parcourir par backtracking. Ceci se traduit, dans l'algorithme de tri topologique, par la fonction de choix suivante :

```
eq multi-choice(d1,il) = d1 .
eq multi-choice(d1 d2 ds,il) =
   if ((is-param(d1) or are-in-list(args-term(d1),il))
       and (multi(d1) >= multi(d2)))
   then multi-choice(d1 ds,il) else multi-choice(d2 ds,il) fi .
```

La fonction de choix précédente résulte d'une vision, disons optimiste des choses où toutes les multifonctions sont aussi productives que possibles. En tenant compte de la probabilité d'échouer d'une fonction, on veut au contraire élaguer lors d'un échec le plus gros sous-arbre possible. Il faut alors faire placer au début du plan de construction les fonctions ayant le plus grand risque d'échouer. On minimise ainsi le nombre de calculs intermédiaires qui interviennent avant ces fonctions, et si elles viennent réellement à échouer, le sous-arbre éliminé est plus volumineux. La fonction de choix résultant de cette heuristique est donnée ci-dessous.

```
eq risk-choice(d1,il) = d1 .
eq risk-choice(d1 d2 ds,il) =
   if ((is-param(d1) or are-in-list(args-term(d1),il))
       and (risk(d1) <= risk(d2)))
   then risk-choice(d1 ds,il) else risk-choice(d2 ds,il) fi .
```

5.2 Regrouper les éléments d'une même sous-figure

Ordonner les définitions en tenant compte à la fois des dépendances mais aussi de l'appartenance des éléments à des sous-figures différentes permet aussi d'avoir un parcours plus efficace de l'arbre. En effet, si l'on regroupe au maximum les définitions dont les éléments appartiennent à la même sous-figure, l'extraction des arguments parmi une branche de l'arbre est plus rapide. Les définitions et les arguments dont elles dépendent sont plus proches et donc accessibles plus rapidement.

On peut aussi permettre à l'utilisateur de visualiser les différentes sous-figures et de n'effectuer le bactracking que sur une sous-figure donnée. Ceci permet de ne sélectionner qu'une partie de la solution, et de comparer des figures.

5.3 Discussion

Les critères de tri topologique que nous avons énoncés ci-dessus ne sont pas forcément indépendants. On peut les combiner en leur imposant des priorités : il est ainsi possible de choisir dans les définitions éligibles celles qui définissent un objet faisant partie de la sous-figure courante, puis parmi celles-ci, celles qui ont le degré de risque le plus important, et enfin, parmi celles qui restent en lice, celles qui ont le degré de multiplicité le plus petit. Ces critères de tri ne sont pas les seuls possibles. On peut, par exemple, considérer la complexité (en calcul) des fonctions pour placer les définitions les plus complexes en début ou en fin de plan. On peut aussi diminuer la distance de cheminement entre une définition et les objets dont elle dépend afin de limiter les duplications de calculs inutiles.

Nous avons présenté ici une manière statique de structurer l'espace des solutions. Il existe d'autres possibilités pour améliorer l'efficacité de l'exploration ou pour la faciliter à un utilisateur. Par exemple, on se rend compte, après la spécification des arbres d'interprétation, que les valeurs (et les calculs) pour des noeuds d'un même niveau sont souvent répétées. On peut donc mettre en place une table de hachage conservant pour chaque définition, les valeurs des arguments et la valeur du résultat.

6 Conclusion

Nous avons présenté dans cet article une formalisation des univers géométriques manipulés par YAMS. Nous avons insisté sur les notions de plan et fonction de construction et sur la manière de les interpréter. Nous avons montré comment la présence de multifonctions induit la notion d'arbre d'interprétation et nous avons indiqué quelques méthodes pour transformer cet arbre afin de rendre son parcours plus efficace.

Cette première approche doit être bien sûr complétée par des méthodes de réduction de l'espace des solutions pour ne produire que les solutions les plus proches de l'esquisse. Nous étudions actuellement deux méthodes différentes. La première consiste à orienter le plus possible les objets manipulés en introduisant dans l'esquisse des contraintes de type inégalité. Ces contraintes peuvent être entrées explicitement par l'utilisateur ou déduite de l'esquisse et de situations précédentes. La deuxième méthode s'appuie sur des propriétés d'invariance locales par similitude : on essaye ainsi de mesurer la similitude entre l'esquisse et la solution. Nous espérons, dans les meilleurs cas, à n'avoir qu'à "geler" la branche de l'arbre d'interprétation correspondant à l'esquisse.

Références

[BD98] Y. Bertrand and J.-F. Dufourd. Du modèle géométrique à la programmation par les spécifications algébriques. *Technique et science informatiques*, 17(4):405–532, 1998.

[BDFL93] Y. Bertrand, J.-F. Dufourd, J. Françon, and P. Lienhardt. Algebraic specification and development in geometric modeling. In *Proceedings of the TAPSOFT Conference, LNCS 668*, pages 75–89. Springer-Verlag, 1993.

[Ber98] Y. Bertrand. Topofil : un modeleur interactif d'objets tridimensionnels à base topologique. *Technique et science informatiques*, 17(4):405–532, 1998.

[Cha98] L Champciaux. *Introduction de techniques d'apprentissage en modélisation déclarative*. PhD thesis, Ecole des Mines de Nantes, 1998.

[DMS97] J.-F. Dufourd, P. Mathis, and P. Schreck. Formal resolution of geometrical constraint systems by assembling. In *Proceding of the 4th ACM-Siggraph Solid Modeling and Applications, Atlanta*, pages 271–284. ACM Press, 1997.

[DMS98] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Journal of Artificial Intelligence*, pages 73–119, 1998.

[GWM+92] J.A. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.-P. Jouannaud. *Applications of algebraic specifications using OBJ*, chapter Introducing OBJ. Cambridge University Press, 1992.

[Leb50] H. Lebesgue. *Leçons sur les constructions géométriques*. Gauthier-Villars, Paris, 1950.

[LM95] H. Lamure and D. Michelucci. Solving constraints systems by homotopy. In *Proceedings of 3rd ACM/IEEE Symposium on Solid Modeling and Applications*, pages 263–269, 1995.

[Mat97] P. Mathis. *Un système de résolution de contraintes par assemblage en modélisation géométrique*. PhD thesis, Université de Strabourg, 1997.

[Wir90] M. Wirsing. *Handbook of Theoretical Computer Science*, chapter Algebraic specification, pages 675–788. Elsevier, North-Holland, Amsterdam, 1990.