UNIVERSITE DE NANCY 1
CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

MEPHISTO : UN OUTIL DE VALIDATION DE MODELES TRIDIMENSIONNELS

THESE

présentée et soutenue publiquement le 20 juillet 1987

A L'UNIVERSITE DE NANCY 1
pour l'obtention du titre de
DOCTEUR DE 3ième cycle en INFORMATIQUE

par

MARTINE CAMONIN

devant la Commission d'Examen

Président :	Jean-Pierre FINANCE
Examinateurs :	Roger MOHR
	Pierre MARCHAND
	Yvon GARDAN

UNIVERSITE DE NANCY 1
CENTRE DE RECHERCHE EN INFORMATIQUE DE NANCY

MEPHISTO : UN OUTIL DE VALIDATION DE MODELES TRIDIMENSIONNELS

THESE

présentée et soutenue publiquement le 20 juillet 1987

A L'UNIVERSITE DE NANCY 1
pour l'obtention du titre de
DOCTEUR DE 3ième cycle en INFORMATIQUE

par

MARTINE CAMONIN

devant la Commission d'Examen

Président :	Jean-Pierre FINANCE
Examinateurs :	Roger MOHR
	Pierre MARCHAND
	Yvon GARDAN

Que cette thèse qui se termine soit l'occasion pour moi d'exprimer mes remerciements à l'égard de :

Roger MOHR, professeur à l'Institut National Polytechnique de Lorraine, qui m'a accueillie dans son équipe et qui a accepté de diriger mon travail. Qu'il trouve ici ma reconnaissance sincère.

Jean-Pierre FINANCE, directeur du CRIN et professeur à l'Université de Nancy 1, qui me fait l'honneur de présider ce jury. Ses remarques et suggestions m'ont permis d'améliorer la clarté et la présentation de cette thèse.

Pierre MARCHAND, professeur à l'Université de Nancy 1, qui s'est intéressé à mon travail et qui, tout au long de ces derniers mois, a su me prodiguer les encouragements nécessaires à l'aboutissement de cette thèse. Qu'il se voit profondément remercié du temps passé à la lecture et à la correction de cet ouvrage.

Yvon GARDAN, professeur à l'Université de Metz, qui a accepté d'être membre de ce jury.

Tous mes collègues du CRIN pour leur soutien amical.

Tous les *Vaxomanes* qui ont du supporter trop souvent les débordements en tous genres de mes programmes.

GUEST STAR : Mac Intosh Plus

Les costumes sont de Mac Write et les décors de Mac Draw.

A DIDIER

A MES PARENTS

INTRODUCTION

CHAPITRE 1 : LA MODELISATION GEOMETRIQUE

CHAPITRE 2 : TRIDENT : UN SYSTEME D'INTERPRETATION DE SCENES

CHAPITRE 3 : MEPHISTO : UN OUTIL DE VALIDATION DE MODELES 3D

INTRODUCTION

La **modélisation géométrique** est une technique permettant de décrire des objets, ceci dans un but de traitement automatique par ordinateur. Depuis son apparition en 1963 dans le système SKETCHPAD développé par Y. SUTHERLAND au M.I.T., son importance croît dans de nombreux domaines comme la *vision* par ordinateur, la *conception assistée par ordinateur*, la *simulation*, la *robotique* et *l'animation*... Même s'il existe différents types de modélisations - fil de fer, surfacique, volumique -, chaque application développe souvent son propre modèle, adapté aux problèmes à résoudre, dans un contexte bien spécifique.

Quelque soit le modèle utilisé, la description et la manipulation d'objets complexes pose, dans certains cas, des problèmes de **validité** : lorsque l'on donne une description, il faut s'assurer que l'objet décrit représente effectivement un objet réel.

Le **système MEPHISTO**, qui fait l'objet de cette thèse, résoud les problèmes de validité posés par le choix de représentation effectué dans le cadre du système de vision tridimensionnelle TRIDENT, développé au CRIN [ZARO83] [MAZA84] [MASI84]. Pour interpréter une scène, TRIDENT applique le principe de la vision humaine. Il s'appuie sur deux types de connaissances : les informations contenues dans une image de la scène qu'il doit interpréter et une connaissance du monde dans lequel il évolue, appelée **modèle**. Le modèle choisi permet de décrire des familles d'objets génériques construits par unions de primitives. La description de scènes est donnée par un ensemble de règles donnant une hiérarchie de composition des formes; l'assemblage des formes entre elles ainsi que leurs positions et dimensions respectives sont précisées par des ensembles de contraintes associées à chacune des règles. Le modèle obtenu est complexe et ne peut être utilisé par TRIDENT que s'il est cohérent. Cette **vérification de cohérence** est effectuée au préalable par le système MEPHISTO.

Le *premier chapitre* de cette thèse donne une étude des propriétés des modélisations et un éventail des diverses représentations actuellement mises en oeuvre dans tous les domaines d'application.

Le *second chapitre* décrit la structure du système d'interprétation de scènes TRIDENT, et plus particulièrement le choix de représentation effectué dans ce projet pour modéliser les objets tridimensionnels.

Le *chapitre 3* décrit en détail la structure du système MEPHISTO qui fait l'objet de ce travail. Sa tâche est de décider de la cohérence - ou de la validité - d'un modèle fourni par l'utilisateur, avant qu'il ne soit utilisé par TRIDENT. Dans le contexte de la représentation choisie , un modèle peut être vu comme un graphe ET/OU avec contraintes. Comme nous le verrons, le problème de la vérification de validité se ramène à celui de la construction d'une instance d'un graphe ET/OU avec contraintes. Ceci nous a conduit à l'étude des techniques de résolution de graphes ET/OU et de résolution de systèmes de contraintes. Une solution sera apportée à ces différents problèmes.

CHAPITRE II

LA MODELISATION DU SOLIDE

1 - INTRODUCTION

Dans le terme **modélisation des objets solides**, on regroupe habituellement un ensemble de *théories*, de *techniques* et de *systèmes* permettant d'exprimer et de manipuler toutes les informations que l'on possède sur des solides quelconques. Ces informations, encore appelées **représentations**, permettent le calcul automatique des propriétés géométriques et physiques des solides, ainsi que la visualisation des objets correspondants.

Le *domaine d'application* le plus courant de la modélisation du solide est encore aujourd'hui la conception assistée par ordinateur, en lien étroit avec la fabrication assistée par ordinateur. Son rôle est important en architecture pour la construction de batiments ou l'aménagement d'intérieurs, en mécanique pour la construction et l'assemblage de pièces de moteurs, en automobile et en aéronautique pour la conception des carrosseries de voitures et des fuselages d'avions, ou même encore dans le domaine du prêt-à-porter. La CAO intervient donc partout, pour remplacer le travail du concepteur devant sa planche à dessin ou sa maquette.

Toutefois, les applications de la modélisation ne s'arrêtent pas à la CAO/CFAO, mais elles s'étendent à tous les domaines faisant appel à l'informatique graphique : la chimie, la médecine, l'animation, le cinéma, la publicité ou la simulation ...etc.

Toutes ces applications de la modélisation du solide ont un point commun : elles recquièrent une *visualisation* des objets sur un écran graphique. Cependant, il existe de nombreuses modélisations possibles, dépendant du type de l'application envisagé. Par exemple, dans le domaine de l'animation, du cinéma ou de la publicité, le facteur de réalisme de l'image obtenue su écran est prépondérant, au détriment parfois du temps de calcul nécessaire à la construction de l'image. A l'inverse, pour les simulateurs de vol, le facteur temps est le plus important. Pour les applications de type CAO/CFAO, il est indispensable, non seulement de visualiser les objets, mais également de pouvoir calculer des propriétes géométriques et physiques (centre d'inertie, masse ...). Pour les applications de type analyse de scènes (médecine par exemple), le modèle construit doit pouvoir être comparé à un modèle de référence. Le choix d'un type de modélisation ne peut donc se faire que dans le contexe d'une application particulière.

Au paragraphe 2, nous allons définir ce que l'on entend habituellement par **représentation** et en préciser quelques **propriétés** essentielles définies par REQUICHA [REQU80]. Dans le paragraphe 3, nous décrivons les diverses **possibilités de représentation**, sans toutefois avoir la prétention d'être exhaustifs.

2 - LES DIFFERENTES PROPRIETES DES REPRESENTATIONS

2.1 - INTRODUCTION

Les divers algorithmes et applications de géométrie du solide ne manipulent pas directement des objets physiques, mais plutôt *les données qui représentent ces objets*. Comment de telles données sont-elles choisies ? Certains choix sont-ils meilleurs que d'autres ?

Prenons, par exemple, le cas de la représentation d'un *polyhèdre à faces planes*. Une approche possible est de définir ce solide par ses arêtes: chaque arête est représentée par ses deux points extrêmes. Le solide est alors lui-même représenté par une collection d'arêtes, donc par un ensemble de 6-uplets de la forme

$$(X_i , Y_i , Z_i , X_j , Y_j , Z_j)$$

Un tel 6-uplet donne la syntaxe de la représentation, c'est-à-dire la **notation** que l'on doit utiliser pour représenter un polyhèdre; la **sémantique**, c'est-à-dire le sens géométrique d'un 6-uplet, nous permet d'interpréter

$$(X_i , Y_i , Z_i) \quad \text{et} \quad (X_j , Y_j , Z_j)$$

comme étant les coordonnées des deux points extrêmes d'un segment de droite, dans un repère absolu.

Pour une telle représentation, une **théorie de la modélisation géométrique** se doit d'aborder les questions fondamentales suivantes :

* la représentation d'un polyèdre par une collection d'arêtes contient-elle *suffisamment* d'informations pour permettre le calcul automatique de l'apparence extérieure de l'objet, de son volume ainsi que d'autres propriétés géométriques ?

* une collection arbitraire d'arêtes représente-t-elle un polyèdre ? autrement dit, comment être sûr qu'une représentation est *valide* ? qu'elle correspond bien à un objet réel ?

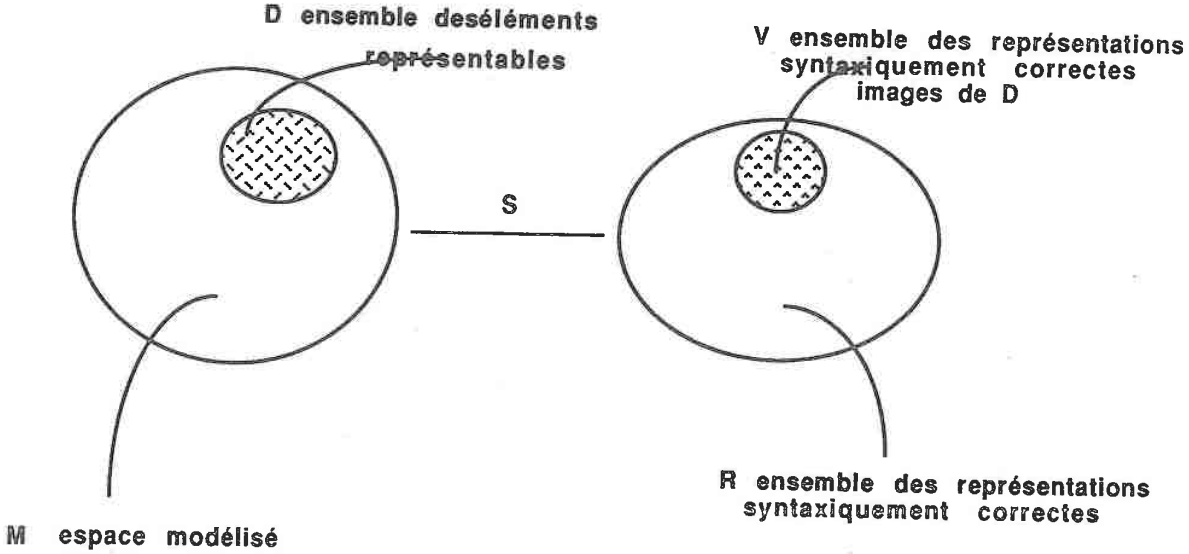
* existe-t-il plusieurs représentations possibles d'un même objet ? autrement dit, existe-t-il un problème *d'ambiguïté* ?

* existe-t-il une *meilleure* représentation?

Pour répondre à ces diverses questions, la théorie de la modélisation définit une relation appelée **schéma de représentation :**

$$\mathbb{S} : \mathcal{M} \rightarrow \mathbb{R}$$

L'ensemble $\mathcal{M}$ est l'espace que l'on veut modéliser : ses éléments sont des solides. La relation $\mathbb{S}$ définit donc une correspondance entre un solide de $\mathcal{M}$ et un élément de $\mathbb{R}$ qui représente l'espace des représentations syntaxiquement correctes (sur un alphabet donné).



En fait, le domaine de $\mathbb{S}$ est réduit à $\mathcal{D}$, l'ensemble des éléments de $\mathcal{M}$ que l'on peut effectivement représenter. Selon les représentations, on ne peut pas assurer que les ensembles $\mathcal{M}$ et $\mathcal{D}$ soient confondus, c'est-à-dire que tous les solides soient représentables.

L'ensemble des images par $\mathbb{S}$ des éléments de $\mathcal{D}$ est appelé $\mathcal{V}$: ses éléments sont donc des représentations qui sont syntaxiquement et sémantiquement correctes : nous dirons que ce sont des représentations **valides**. De même, on ne peut assurer que les ensembles $\mathcal{R}$ et $\mathcal{V}$ soient égaux, c'est-à-dire que toutes les représentations syntaxiquement correctes soient également sémantiquement correctes.

A partir de ces quelques définitions simples, le paragraphe 2.2 se propose de préciser les diverses *propriétés* d'un schéma de représentation. Au paragraphe 2.3, nous donnerons les caractéristiques principales de 7 *schémas* possibles : cette liste n'est en aucun cas exhaustive, mais donne les schémas principaux.

2.2 - PROPRIETES DES SCHEMAS DE REPRESENTATION

Le domaine

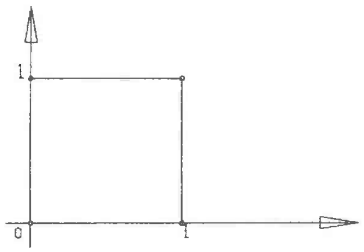
Le **domaine** d'un schéma de représentation est l'ensemble des solides représentables; il caractérise bien évidemment la puissance de description du schéma.

La validité

Le problème de la **validité** des représentations est essentiel afin d'assurer l'intégrité de la base de connaissances manipulées par un système de modélisation géométrique. Il est en effet inconcevable que de telles bases de données contiennent des représentations invalides, c'est-à-dire des objets irréels.

Ceci s'illustre facilement sur un exemple simple, en deux dimensions : si on représente un polygône par la collection de ses arêtes, voici une représentation valide :

(0, 0, 0, 1) (0, 1, 1, 1) (1, 1, 1, 0) (1, 0, 0, 0)

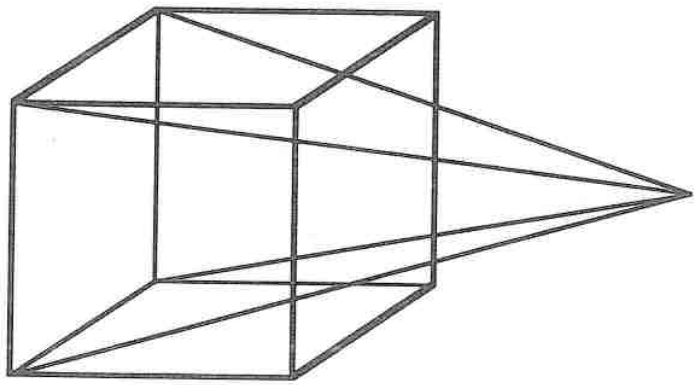


alors que la représentation suivante .

(0, 0, 0, 1) (0, 1, 1, 1) (1, 1, 2, 2) (2, 2, 3, 3)

est syntaxiquement correcte, mais n'a aucun sens géométrique dans la mesure où la collection d'arêtes ne définit pas un polygône.

Voici un exemple en dimension 3 : l'objet est défini par une collection d'arêtes définissant un cube plus 4 arêtes reliant un point extérieur à 4 sommets du cube. La représentation est invalide, puisqu'elle ne définit pas un solide.



Il est bien évident que l'utilisation d'algorithmes géométriques sur des représentations invalides peut provoquer des erreurs, ou plus gênant, produire des résultats apparemment crédibles, mais dénués de sens.

La plupart du temps, la responsabilité de cette validation est laissée à l'utilisateur, qui doit donc disposer d'outils de visualisation adaptés. Mais lorsque les représentations sont produites automatiquement et non manuellement, une telle vérification visuelle est quasi-impossible.

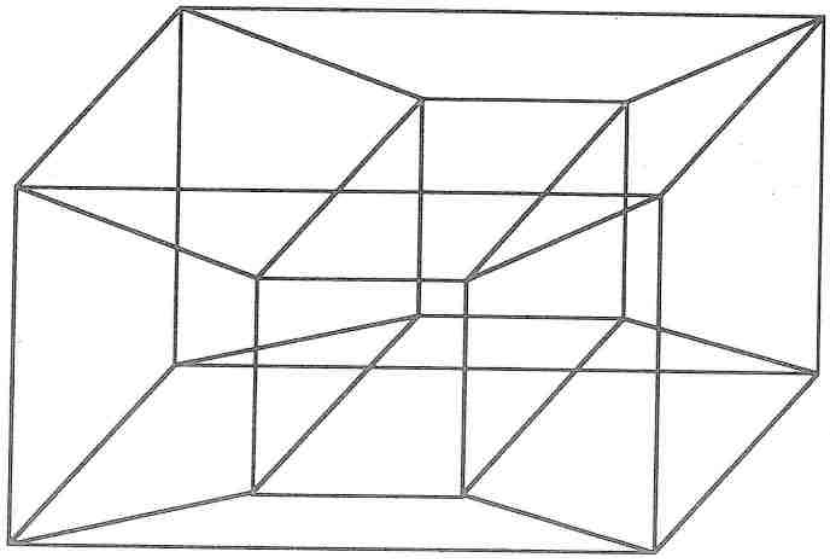
A titre d'exemple, prenons le cas d'un système de CFAO: sa tâche principale est de concevoir un objet. La première phase concerne la description de l'objet en termes de représentation. Cette description est ensuite utilisée dans une seconde phase pour fabriquer le produit. Dans ce cas, il est essentiel d'assurer de façon automatique la validité de la représentation engendrée lors de la première phase.

Donc, de plus en plus, les systèmes de modélisation géométrique sont amenés à *automatiser la phase de validation*; il existe plusieurs façons de résoudre ce problème. La plus simple consiste à assurer la validité en utilisant des schémas de représentation tels que toute représentation syntaxiquement correcte soit valide. Nous en verrons un exemple dans un paragraphe ultérieur. Si une telle représentation ne peut pas être utilisée, la validité peut être vérifiée soit une fois que la représentation est construite, soit en insérant des contraintes de validité dans les procédures de construction de la représentation. Cette dernière solution ne peut s'appliquer que dans le cadre de systèmes de conceptions interactifs.

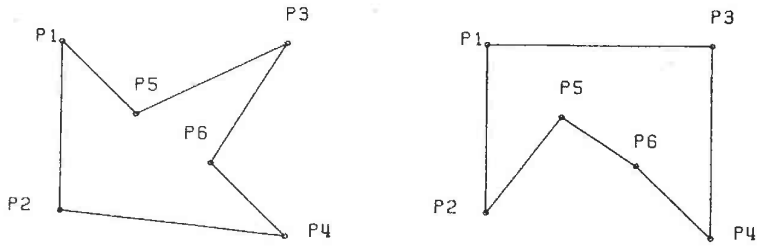
La complétude

La **complétude** est probablement l'une des plus importantes propriétés d'un schéma de représentation. En effet, dans un schéma incomplet ou **ambigu**, une même représentation peut correspondre à plusieurs entités physiques différentes.

Par exemple, la représentation d'un polyhèdre par une collection d'arêtes peut parfois être ambiguë, comme le montre la figure suivante. L'ensemble des arêtes ne permet pas de déterminer où se trouvent les faces pleines et les faces creuses.



Voici un autre exemple plus parlant, en deux dimensions : la représentation d'un polygône par l'ensemble de ses sommets est aussi ambiguë, puisque, sans apporter de restriction à la sémantique, l'ensemble de points {P1 P2 P3 P4 P5 P6 } représente l'un ou l'autre de ces deux polygônes :



Pour qu'une telle représentation soit complète, il est indispensable de préciser que les points doivent être donnés dans l'ordre du tracé du polygône. Ainsi, aucune ambiguïté n'est plus possible.

Malgré tout, il ne faut pas conclure hâtivement que les représentations ambiguës sont inutiles : elle peuvent être adéquates pour certains types d'applications. Par exemple, nous avons vu que la représentation d'un volume par l'ensemble de ses arêtes est ambiguë, mais elle peut être adéquate pour le calcul de l'enveloppe convexe du volume.

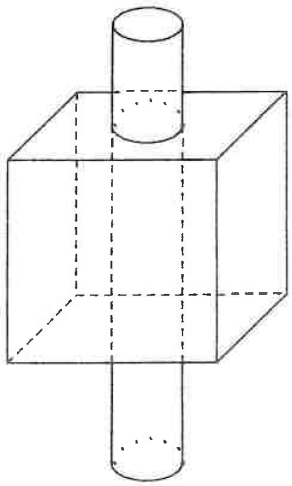
L'unicité

Le problème de l'unicité d'une représentation est symétrique à celui de l'ambiguïté. Un schéma de représentation est dit **unique** si, à tout objet du domaine correspond une seule et unique représentation.

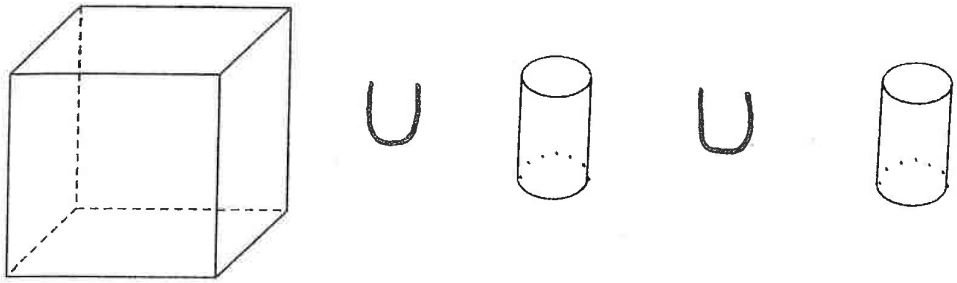
Comme les propriétés précédentes, l'unicité peut n'être indispensable que dans certaines applications, lorsque par exemple, on a besoin de tests d'égalité entre objets.

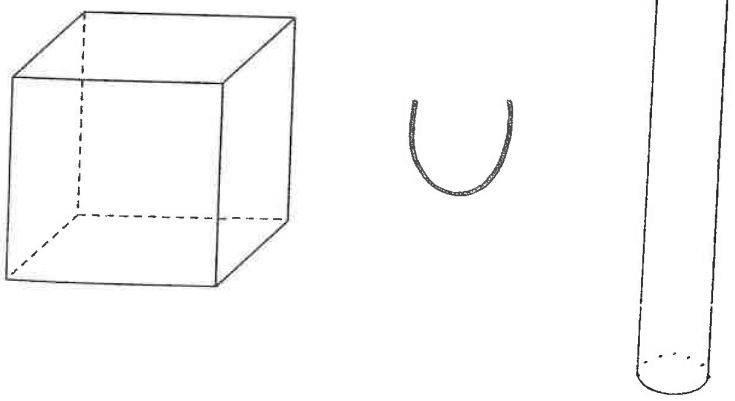
Les schémas de représentation *non ambigus* et *uniques* sont bien évidemment très intéressants et recherchés. En effet, cette double propriété implique que deux représentations distinctes correspondent à deux objets distincts; ainsi, le test d'égalité de deux représentations se réduit simplement à un test sur la syntaxe.

Prenons l'exemple d'un schéma de représentation que l'on verra en détail ultérieurement qui définit un solide par des compositions de volumes simples, appelés *primitives*. On peut représenter l'objet suivant



par l'une ou l'autre de ces deux descriptions :





La concision

La **concision** d'une représentation se rapporte plus particulièrement à sa taille, c'est-à-dire à la place mémoire nécessaire à son stockage. Il est difficile de définir cette propriété précisément.

Un schéma de représentation *concis* a bien évidemment l'avantage de ne pas contenir d'informations redondantes, ce qui, dans une certaine mesure, facilite la validation. Cependant, la concision est une propriété dont l'appréciation est totalement dépendante de l'application concernée : en effet, la redondance facilite parfois les calculs, ou peut même les éviter. Il est donc essentiel de trouver un compromis concision - efficacité.

La facilité de création

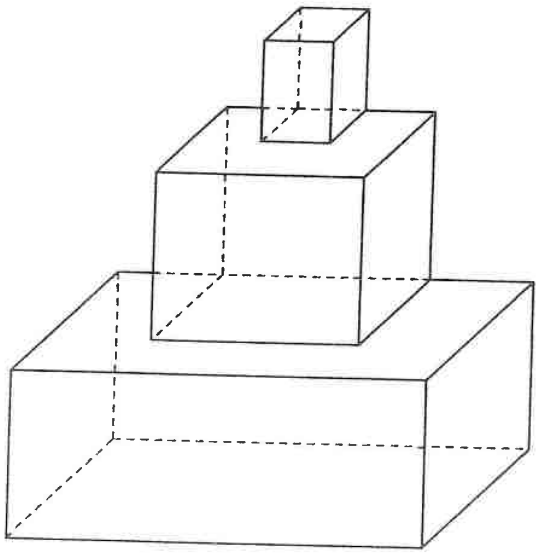
Cette propriété n'a de sens que si l'utilisateur concoit lui-même ses objets (dans le cas de conception automatique, cette caractéristique est négligeable). Elle est encore plus subjective que la propriété de concision, dans la mesure où l'utilisateur est ici directement concerné.

De plus, même si la représentation est difficile à construire, il est encore possible de mettre à la disposition du concepteur un outil de conception adapté. On retrouve ici tous les problèmes de dialogue homme - machine, propres à tous les types d'application.

De façon générale, nous pouvons dire qu'une représentation concise doit être plus **facile à créer**, puisque le concepteur a moins de données à spécifier, et que celles-ci ont toutes les chances d'être indépendantes.

L'efficacité

Cette dernière propriété se mesure dans le contexte d'applications spécifiques. Dans ce cas, les représentations doivent être vues comme les données des algorithmes géométriques. **L'efficacité** est alors une mesure de la complexité des algorithmes que l'on est obligé d'écrire. Par exemple, on a tout interêt, au point de vue complexité algorithmique, à représenter l'objet suivant par l'union de trois primitives solides - trois parallélépipèdes - que par la collection de ses arêtes, dans le cas où l'on désire en calculer le volume. Ce sera l'inverse si on désire donner un dessin fil de fer de cet objet sur un écran.



Dans le cas général, le choix d'une représentation efficace est complexe : en effet, une même application peut donner lieu à des calculs géométriques différents pour lesquels il n'existe pas une seule et unique représentation adéquate; par exemple pour une application regroupant les dessins fil de fer et les calculs de volume.

2.3 - CONCLUSION

L'expérience accumulée depuis une vingtaine d'années dans le domaine de la modélisation géométrique, quelle que soit l'application visée, montre qu'il *n'existe pas de représentation générale* permettant de représenter, manipuler ou visualiser n'importe quel type d'objet tridimensionnel.

Ainsi, pour que la complexité de certains algorithmes ne soit pas prohibitive, les systèmes existants sont souvent amenés à introduire de la redondance, au détriment de la concision. Nous verrons ultérieurement que, de plus en plus, les systèmes de modélisation géométrique se disant généraux tendent à manipuler non pas un seul schéma mais plutôt plusieurs schémas simultanément, afin d'éviter les inconvénients de chacun d'entre eux.

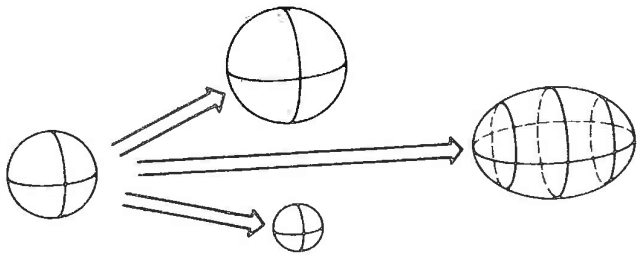
3 - QUELQUES REPRESENTATIONS USUELLES

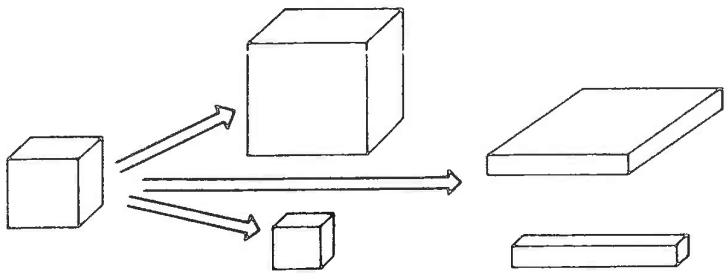
Nous nous limiterons ici à la description de schémas de représentation non ambigus, puisque, nous avons vu que l'utilisation de schémas ambigus pose quelques problèmes.

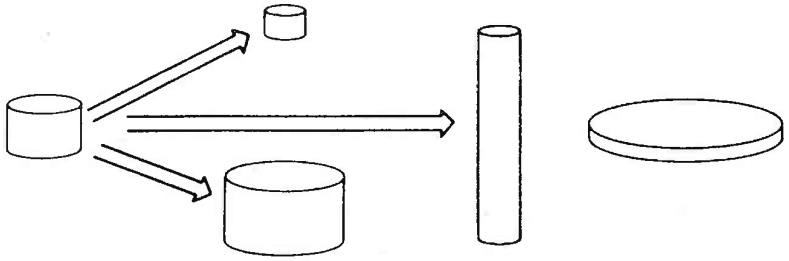
Les huit schémas que nous allons décrire dans les paragraphes suivants sont les plus utilisés : il en existe bien sûr plusieurs variantes, chacun des systèmes de modélisation géométrique définissant son propre schéma. Néanmoins, ces différents types de modélisation sont les bases de tout système.

Instanciation de primitives

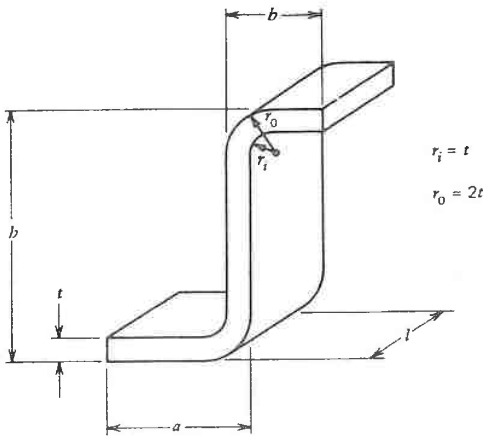
Ce schéma est basé sur la notion de **familles d'objets**, chacun d'eux étant différencié par un ensemble de paramètres. La figure ci-dessous donne un exemple : on dispose de trois familles d'objets : des sphères, des cubes et des cylindres. Chacun de ces trois types d'objets peut être instancié pour donner des objets de tailles variables. Le seul paramètre de la première famille est le rayon de la sphère, que l'on peut instancier de façon uniforme ou non. Trois paramètres doivent être instanciés pour les objets de la seconde famille : il s'agit des longueurs des 3 arêtes. Pour la troisième famille d'objets, on instancie le rayon et la hauteur.







Voici un exemple plus complet, pour lequel il faut instancier 5 paramètres.



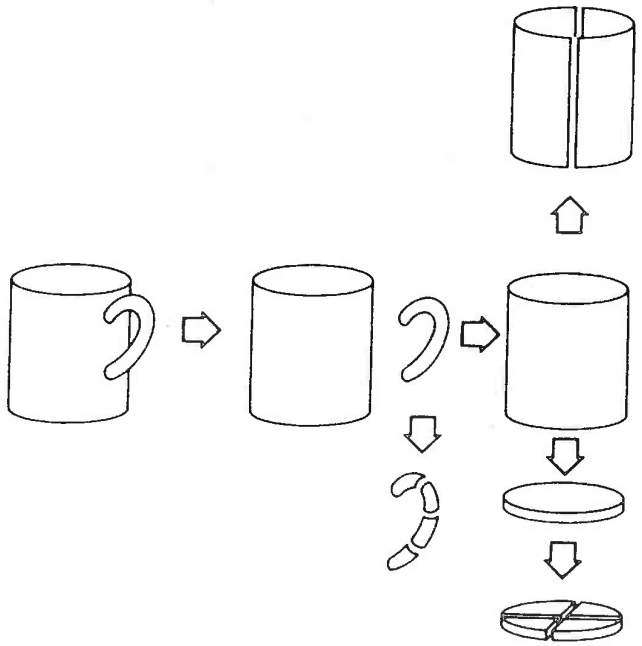
Un système de modélisation géométrique utilisant un tel schéma ne pourra donc manipuler que ces *trois types d'objets* pour lesquels sont définis un *nombre fixe de paramètres*. Si d'autres paramètres ne sont pas associés aux objets, cela signifie soit que ces valeurs sont constantes pour tous les objets d'une famille, comme par exemple la hauteur totale pour les objets de la première famille, soit qu'elles peuvent être déterminées en fonction des paramètres existants, comme par exemple la taille de chaque dent.

Le principal inconvénient de cette représentation réside dans l'*absence de combinaisons* possibles entre des instances de primitives, ce qui, dans certaines applications peut ne pas être restrictif. C'est le cas dans le domaine de la fabrication d'objets que cette technique est le plus couramment utilisée : par exemple, pour fabriquer des écrous de tailles variées, on a seulement besoin d'un modèle général d'écrou. Il suffit ensuite de faire varier certains paramètres, pour obtenir des écrous plus ou moins grands.

La validité d'une telle représentation est triviale, tout comme il est évident que le schéma est non ambigu, unique, concis et de création facile. Mais, vis à vis de telles représentations, les algorithmes de calcul de propriétés des solides rencontrent des difficultés. Ce type de modélisation n'a d'intérêt que dans le cadre d'une application très pointue.

Décomposition en cellules et énumération de l'espace occupé

Un objet est **décomposé en cellules,** dont la description est plus simple à faire que celle de l'objet original. L'exemple ci-dessous (extrait de [MORT84]) montre la décomposition d'une tasse en cellules *simples*. La représentation d'un objet complexe se ramène donc à la représentation des cellules.



Tout objet peut être représenté par l'union de cellules. Toutefois, aucune représentation n'est unique, puisque la décomposition s'effectue de façon totalement arbitraire.

L'**énumération de l'occupation spatiale** est un cas particulier de cette représentation : les cellules sont des éléments de volumes, appelés **voxels** (par analogie aux pixels des images numérisées). La modélisation d'un objet se réduit à la liste des voxels occupés par l'objet, ces voxels étant caractérisés par leurs positions dans l'espace à trois dimensions. La précision du modèle est d'autant plus grande que les voxels sont de petit volume.

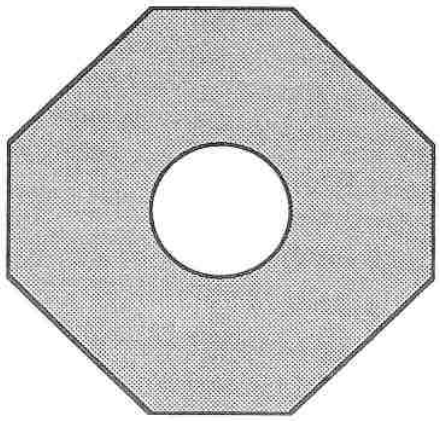
Ce type de représentation est non ambigu et unique. En revanche, il n'est pas concis : imaginez la place occupée par la représentation d'un cylindre de rayon 10 et de hauteur 100, lorsque les voxels sont de taille 2*2*2 ! C'est pourquoi, cette représentation est utilisée dans le cas de petits objets très irréguliers, comme on en trouve dans les applications biologiques.

Cette représentation favorise certains calculs géométriques comme l'union et l'intersection de volumes; de plus, les algorithmes opérant sur de telles données sont parallélélisables, ce qui n'est pas négligeable étant donné le volume de données à manipuler.

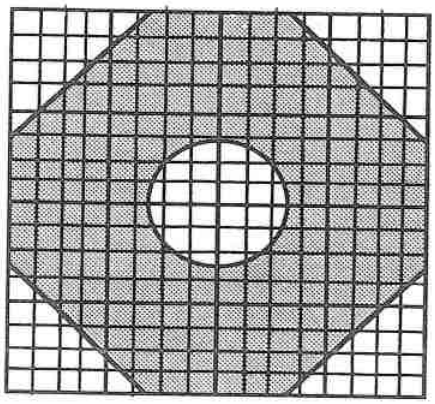
Les octrees

La modélisation par *éléments de volume* décrite au paragraphe précédent présente 2 inconvénients majeurs : la quantité d'informations à manipuler est énorme et le modèle est totalement dépourvu de structuration. C'est ce à quoi Meagher [MEAG82] a voulu remédier en définissant la notion d'**octree**, par extension de la notion de **quadtree**, en 2 dimensions. Cette notion généralise la représentation par énumération spatiale décrite dans le paragraphe précédent. Une telle représentation supprime la redondance et permet de regrouper aisément les informations uniformes dans l'espace.

Un rappel du principe des *quadtrees* nous permettra d'introduire plus facilement la notion d'octrees. Le schéma ci-dessous représente un écrou vu du dessus.



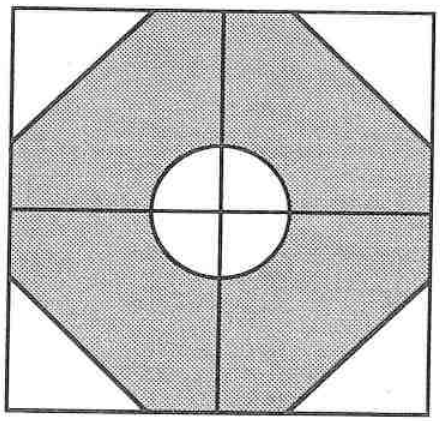
Pour numériser le dessin, on effectue un quadrillage par des carrés de tailles identiques, appelés **pixels**. A chacun d'eux, on associe la valeur 0 ou 1 selon le degré de remplissage du carré. Sur notre dessin, la numérisation donne le résultat suivant :



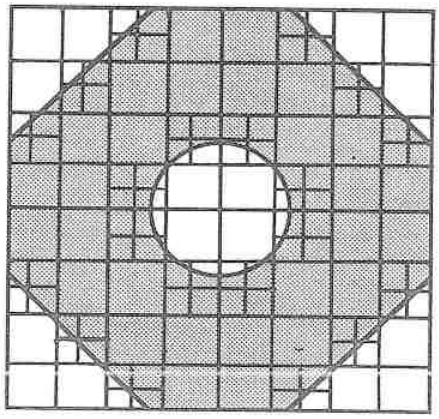
Mémoriser le dessin revient à stocker la valeur binaire associée à chaque pixel. La représentation obtenue est donc de taille proportionnelle à celle de l'objet. De plus, elle est redondante. En effet, les pixels codés 1 ou 0 ne sont pas répartis au hasard dans l'espace. Il y a une grande probabilité pour qu'un pixel codé 1 soit d'un voisin d'un autre pixel codé 1 aussi (et de même pour les 0).

Pour pallier ces inconvénients, on remet en cause le découpage initial en pixels pour appliquer un quadrillage plus grossier. On structure la représentation, sous forme *arborescente*. On utilise un quadtree, autrement dit un arbre dont tout noeud a quatre fils. La racine de cet arbre désigne le dessin tout entier. S'il ne comporte que des pixels de même valeur, le processus de décomposition s'arrête. Sinon, le dessin est découpé en 4 parties, pour donner lieu à quatre fils.

Dans notre exemple, on obtient



Le processus de découpage s'applique récursivement sur chaque fils crée et à chaque niveau. Il s'arrête sur une branche de l'arbre dès que le fils obtenu ne contient plus que des pixels de même valeur. Voici les résultats obtenus après décomposition de l'image initiale :



Si à l'origine, la représentation de l'image nécessitait 16*16 pixels, elle ne recquiert plus que 84 feuilles. Outre une minimisation de l'espace requis pour la mémorisation, une telle modélisation permet également une *structuration spatiale* de l'image, ce qui facilite les traitements locaux.

La description du principe des quadtrees permet d'étendre cette technique à la *représentation de solides*. Le volume à décrire est composé de **voxels**, auxquels on associe de la même manière, une valeur binaire codant la présence ou l'absence de matière. Si tous les voxels du solide à coder ont la même valeur, le codage s'arrête là. Dans le cas contraire, on découpe le volume en huit sous-volumes. Le même principe s'applique ensuite à chacun d'eux, jusqu'à obtention de volumes dont tous les voxels ont la même valeur.

Outre ses qualités de concision et de structuration du modèle, une telle représentation est aussi non ambiguë, unique et de plus facile à créer. Toutefois, montrer sa validité peut s'avérer être une tache délicate.

Pour finir, il est intéressant de signaler que cette modélisation permet de résoudre simplement la visualisation d'un modèle, quelque soit la direction d'observation. En effet, il suffit d'associer à chacune des directions un ordre de visualisation des 8 sous-volumes formant un volume.

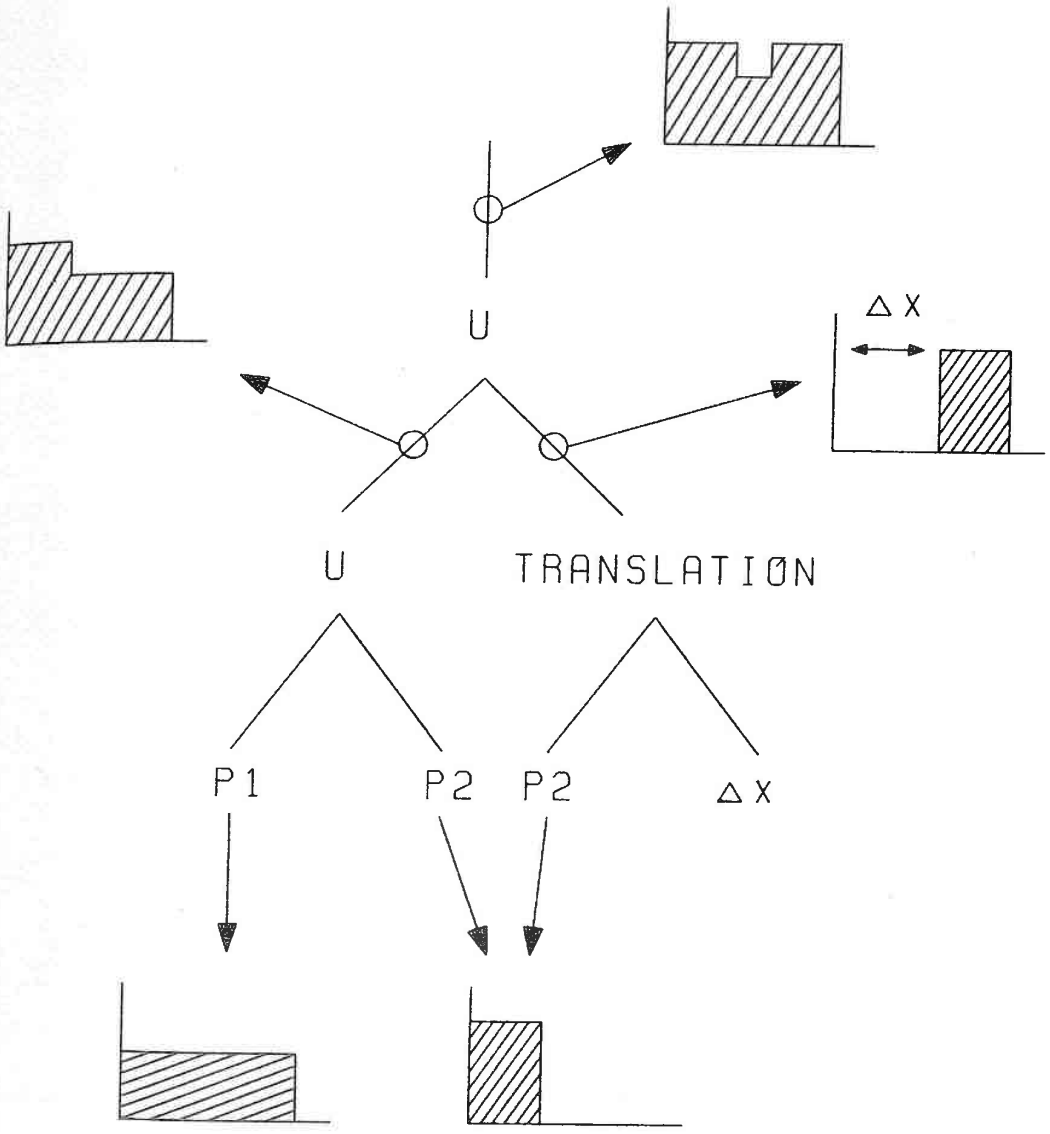
Géométrie constructive du solide (CSG)

Cette technique de modélisation regroupe un ensemble de représentations où les solides complexes sont définis par **compositions de solides** plus simples, par application des opérateurs booléens réguliers. A.A.G. REQUICHA définit la CSG comme une généralisation de la décomposition en cellules, autorisant des cellules complexes et des opérateurs quelconques. Etant donné ses propriétés intéressantes, c'est l'une des méthodes les plus couramment utilisées, en particulier en CAO. Son intérêt réside dans la *structuration* du modèle obtenu, permettant ainsi de conserver un historique de création des objets.

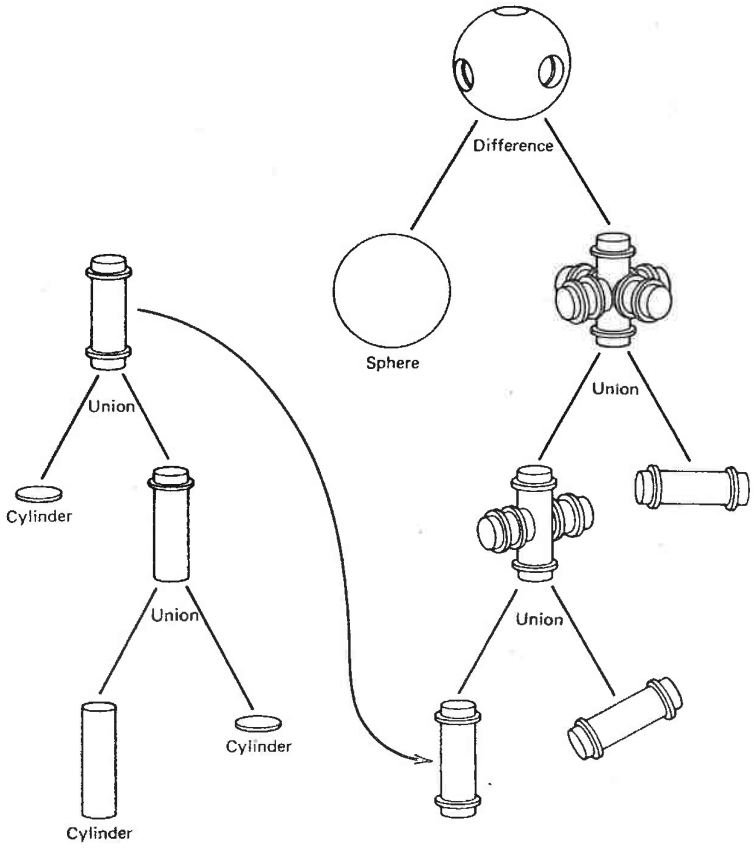
La construction d'un modèle d'objet se fait à partir de **primitives volumiques** et **d'opérateurs logiques** sur ces primitives. Parmi les opérateurs les plus utilisés, on retrouve les opérateurs de *composition* comme l'union, l'intersection et la différence, ainsi que des opérateurs de *mouvement* ne déformant pas les solides comme la translation ou la rotation. Les primitives élémentaires sont les *sphères*, les *cônes*, les *cylindres*, les *parallélépipèdes*, ... etc. Elles sont paramétrées, donc d'utilisation souple.

La modélisation d'un objet en CSG fournit un arbre binaire ordonné : à un noeud est attaché un opérateur, ses deux fils indiquent les 2 objets sur lesquels appliquer l'opération; à une feuille est attachée une primitive, ou bien une donnée utilisée par un opérateur de mouvement (facteur de translation ou angles de rotation).

Voici un exemple, limité à deux dimensions pour la clarté du dessin : le polygône à 8 cotés est l'union de deux polygônes disjoints, le premier étant la réunion des deux primitives P1 et P2, et le second une translation de la primitive P2.

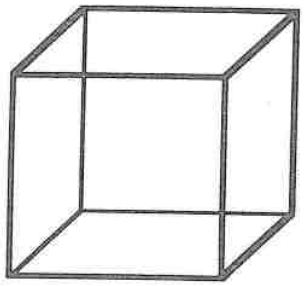

U
ΔX
U
TRANSLATION
P1
P2
P2
ΔX

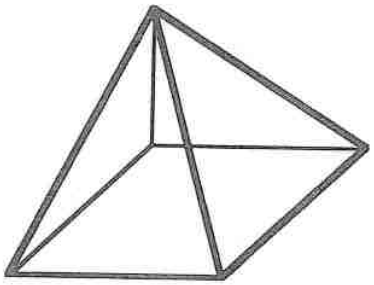
Voici un autre exemple, extrait de [MORT84], illustrant le cas à trois dimensions.



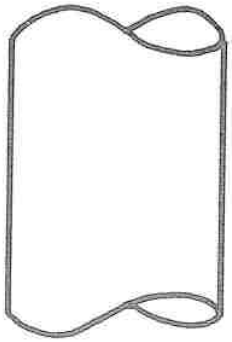
On distingue habituellement deux types de primitives :

- les primitives **bornées**, comme le cube ou la pyramide



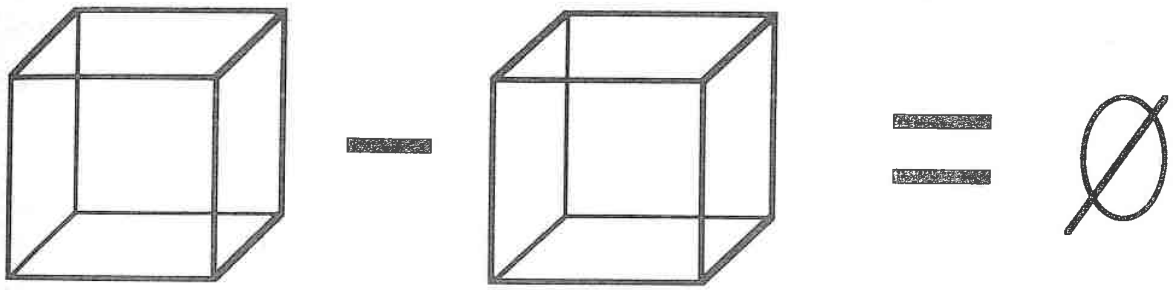


- les primitives **non** bornées, comme cet élément de volume cylindrique infini

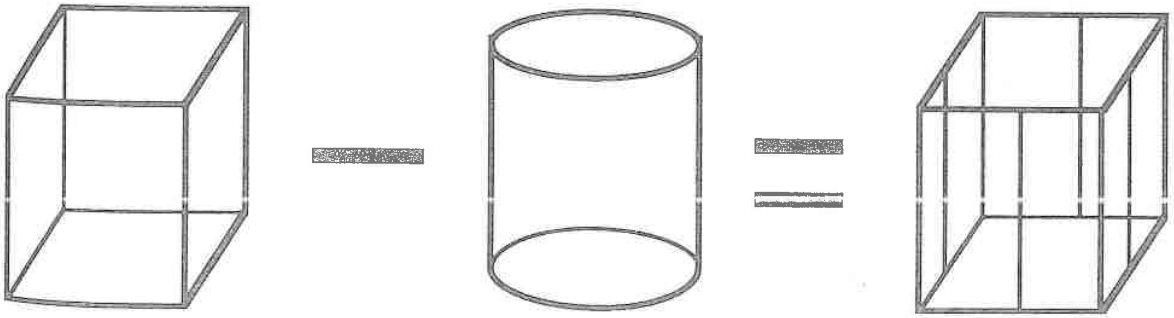


Par construction, on garantit la *validité* d'une représentation utilisant des primitives bornées, si on se limite à l'opérateur d'**union** : de cette façon, toute représentation syntaxiquement correcte l'est aussi sémantiquement, ce qui est une particularité intéressante. Par contre, la validité d'une CSG utilisant des primitives et des opérateurs quelconques pose des problèmes : il est en effet difficile d'établir si, par composition de telles primitives, on obtient ou non un objet connexe; les algorithmes de test de validité sont complexes et coûteux.

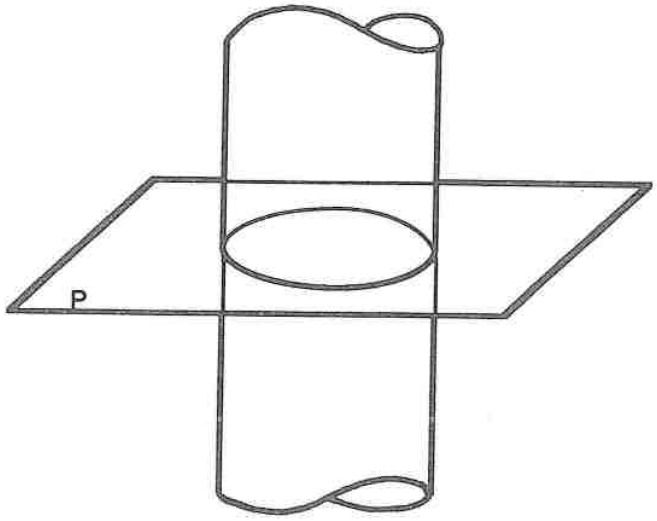
Ce premier exemple indique une opération de différence entre deux cubes identiques : le résultat donne un polyhèdre vide.



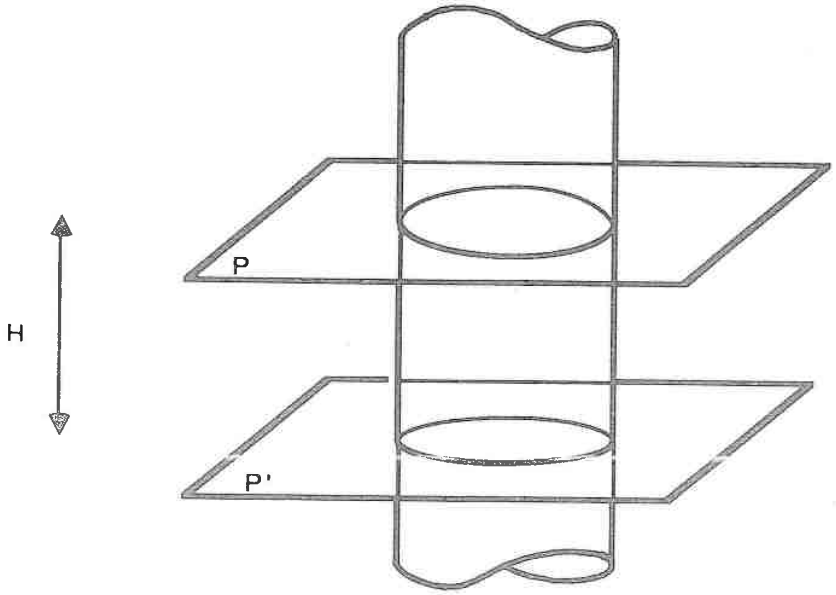
Le second exemple montre une opération de différence entre un cube et un cylindre dont le rayon de la section est égal à la longueur de l'arête d'un cube : on obtient non plus un polyhèdre, mais quatre.



Dans ce dernier exemple, l'intersection de l'élément cylindrique et de la portion d'espace infinie située au-dessous du plan P donne un *objet infini* :

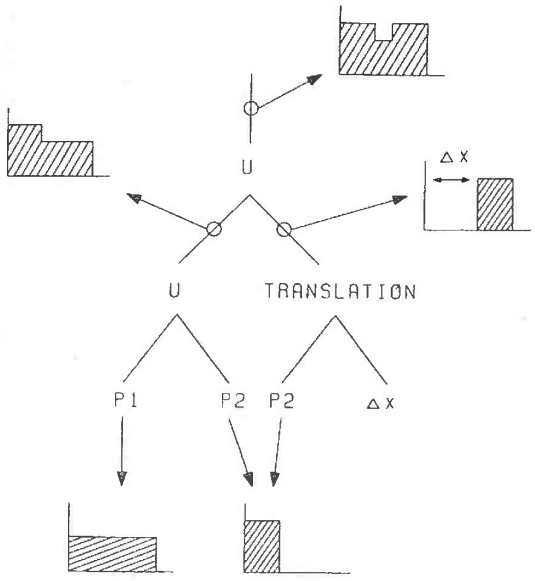


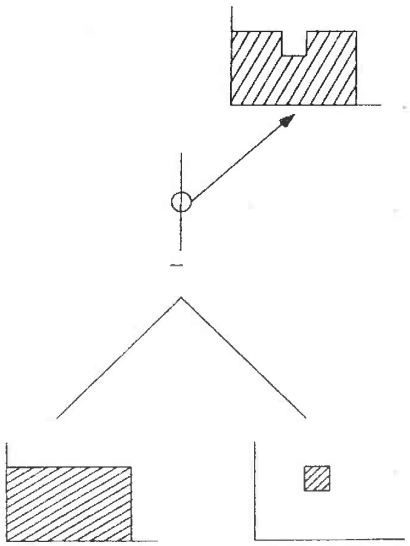
alors que l'intersection de cet élément de cylindre et de deux portions d'espace respectivement au dessous et au dessus des plans P et P' définit un cylindre, de hauteur H



Le domaine des objets représentables est limité : il est impossible de décrire par ce seul mécanisme un objet n'ayant pas d'arêtes saillantes, c'est-à-dire un objet dont les faces se raccordent par l'intermédiaire de surfaces.

Les représentations par CSG sont *non ambiguës* : en effet, un même opérateur, de construction ou de mouvement, appliqué à deux objets donnés, ne peut fournir qu'un seul résultat. En revanche, une représentation par CSG n'est *pas unique* : pour un même objet, il peut exister plusieurs décompositions possibles : par exemple, le polygône vu précédemment peut être représenté par l'un des deux arbres suivants :





Bien que cela paraisse évident, signalons tout de même qu'une telle représentation n'est *pas redondante*. La propriété de concision ne peut pas être établie de façon générale : elle dépend des primitives et des opérateurs choisis. Dans la figure précédente, la représentation du polygône nécessite selon le cas, une ou trois opérations. Si on utilise des primitives non bornées comme des portions d'espace limitées par des plans, huit plans seront nécessaires à sa représentation.

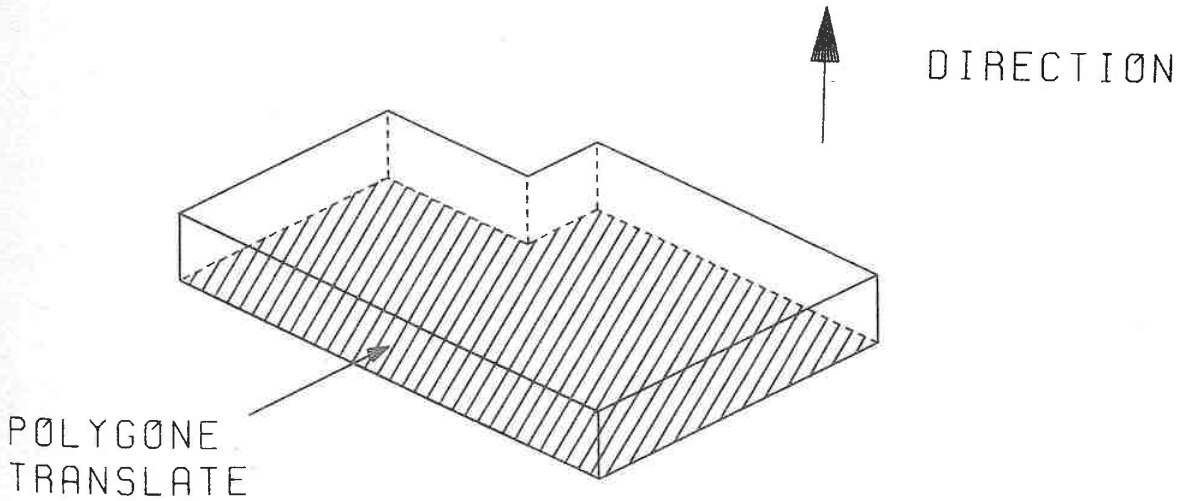
L'expérience dans ce domaine semble montrer que l'homme conçoit facilement ce type de représentation, surtout pour des pièces mécaniques. L'efficacité d'une telle représentation dépend des algorithmes utilisés : sous cette forme, les données ne sont pas utilisables aisément pour la visualisation *fil de fer* des objets. Le calcul de propriétes géométriques et physiques ainsi que la génération d'images sont plus simples. Certains de ces algorithmes sont lents mais fondamentalement parallèles : ils sont donc destinés à être directement implantés sur des matériels spécialisés.

Représentation par balayage

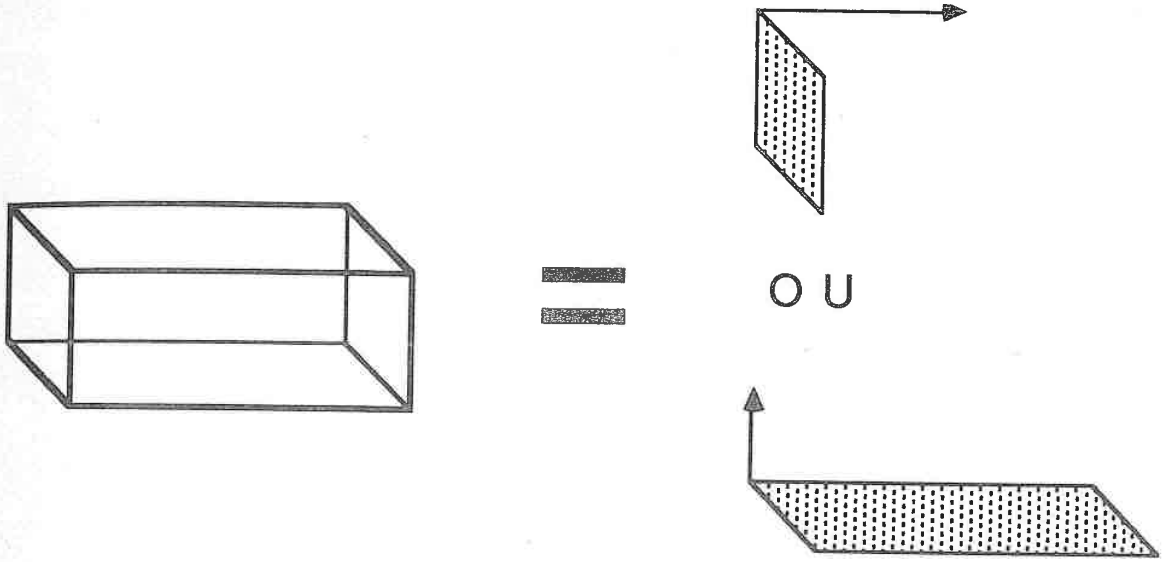
Le principe de base de cette représentation est simple : un **ensemble de points qui se déplace dans l'espace** selon une direction donnée matérialise un volume : c'est l'ensemble des points de l'espace qui, à un moment donné, ont été occupés par un des points de la figure initiale. On modélise un objet par un *ensemble de points* et la *direction* associée.

Cette technique récente est encore mal cernée - les problèmes de validité ne sont pas encore entièrement résolus - mais elle est néanmoins prometteuse. Nous allons citer essentiellement des cas particuliers de balayage.

Citons tout d'abord le **balayage par translation** : on définit un volume en déplaçant un polygône selon une direction donnée. En général, on choisit une direction perpendiculaire au plan du polygône :



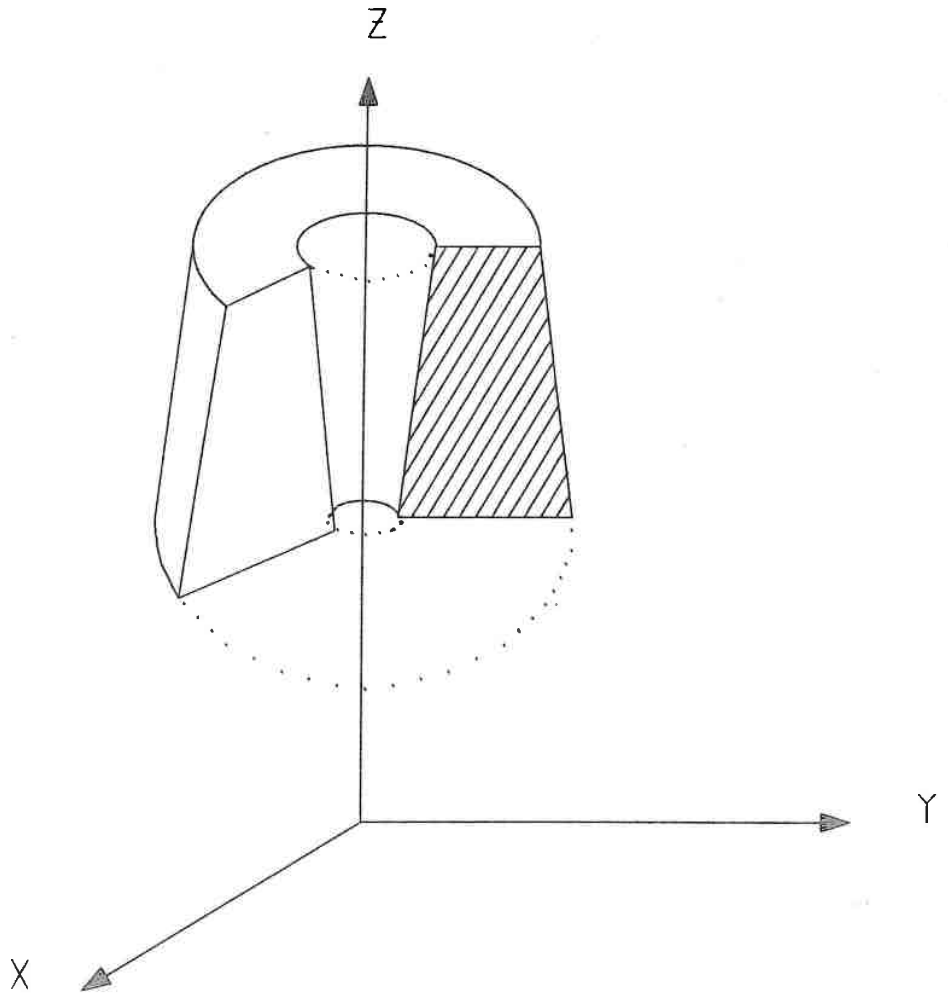
De cette façon, le problème de représentation d'un objet se réduit à la représentation d'une direction et d'un polygône, ce que l'on fait généralement en représentant sa frontière. Cette technique de représentation simple est malheureusement restrictive puisque limitée aux objets possèdant une certaine symétrie. Elle est non ambiguë, à condition que la représentation du polygône le soit; elle est non unique, comme le montre la figure suivante :



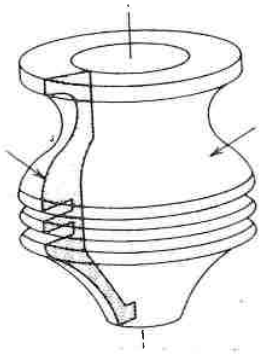
On peut lever en grande partie l'ambiguïté en limitant a priori les directions possibles. Malgré tout, il peut subsister des cas limites ambigus.

Par construction, la démonstration de la validité de cette représentation se réduit à celle des primitives utilisées. Cette représentation est donc un exemple de modélisation pour laquelle l'ensemble des représentations syntaxiquement correctes sont automatiquement sémantiquement correctes. Toutes les représentations construites sont donc valides.

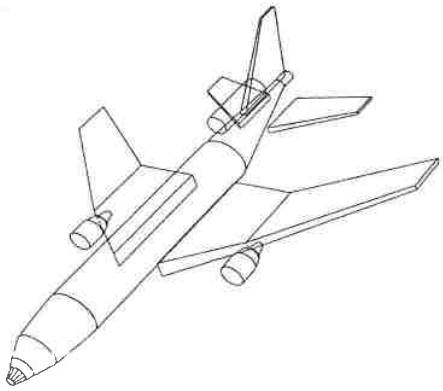
On définit le **balayage par rotation** de façon similaire : un polygône se déplace autour d'un axe; ses propriétés sont similaires au balayage par translation. En voici un exemple :



Il existe également des représentations par balayage plus générales, comme par exemple les **cylindres** et les **cônes généralisés**, introduits par Binford, Brooks et Agin [AGIN81][BROO81]. On peut voir un cylindre comme le volume décrit par une section ronde se déplacant le long d'un axe rectiligne perpendiculaire au plan ce cette section : ce principe peut être généralisé en utilisant une section et un axe quelconques; par exemple :



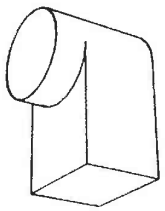
Cette représentation est très attrayante, surtout lorsque l'axe varie de façon régulière : c'est le cas pour les formes allongées comme les avions [BROO81] ou les animaux jouets [NEVA77] et tous les autres objets dont les composants ressemblent à des tubes. Toutefois, la description des objets est complexe et demande de la part du concepteur, des connaissances géométriques importantes.



Frontières

La représentation des objets par leurs **frontières externes** est utilisée couramment en architecture. Elle consiste à segmenter la surface de l'objet en faces, chacune d'elle étant définie par un ensemble d'arêtes et de points.

Cette décomposition en faces n'est pas toujours évidente, surtout lorsqu'il s'agit d'objets courbes. La figure suivante montre un exemple d'objet où la notion de face est difficile à définir.

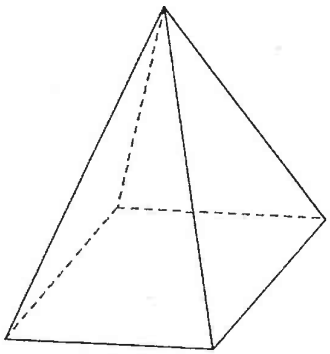


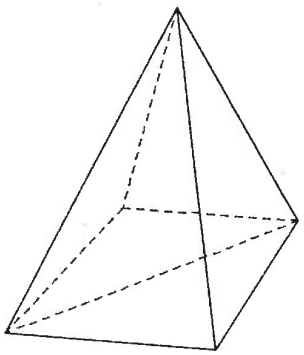
On est donc amené à préciser la notion de face : elle devra satisfaire aux conditions suivantes :

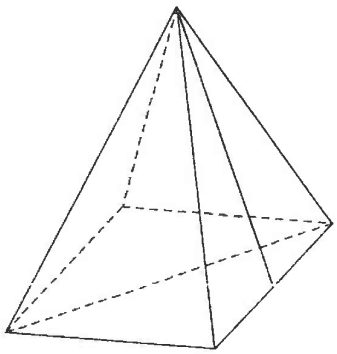
- une face d'un objet est l'une de ses frontières
- l'union de toutes les faces donne la surface entière de l'objet
- une face a une représentation 2D homogène (elle ne contient pas de point isolé)

Souvent, on impose également que toutes les faces soient disjointes ou communes par un point ou une arête, ou bien qu'elles ne comportent pas de trou. De plus, la frontière d'un solide doit être définie par un nombre fini de faces.

Une telle représentation couvre un domaine plus vaste que les représentations par CSG ou par décompositions en cellules. Si les faces ont une représentation non ambiguë, alors on démontre que la représentation est également non ambiguë. Elle n'est pas unique, puisqu'il est possible de trouver un nombre infini de décompositions de la surface d'un objet. Par exemple, une pyramide peut être décomposée en un nombre quelconques de faces, comme l'indiquent les dessins çi-dessous :







La vérification de la validité d'une représentation d'un objet par ses frontières se montre de façon ascendante : on commence par vérifier certaines conditions sur les points, puis sur les arêtes et enfin sur les faces.

Regardons rapidement le principe de la preuve de la validité dans le cas de la triangulation, c'est-à-dire lorsque chaque face est un polygône à 3 cotés. Ici les conditions à vérifier seraient :

- chaque face doit avoir 3 arêtes
- chaque arête doit avoir 2 sommets
- chaque arête doit appartenir à un nombre pair de faces
- chaque sommet doit appartenir à deux des arêtes d'une face
- les arêtes sont disjointes ou s'intersectent en un point
- les faces sont disjointes ou s'intersectent en un point ou une arête

Les deux premières conditions sont évidentes; les deux suivantes assurent globalement que les arêtes forment une *boucle fermée*. Les deux dernières conditions assurent que l'ensemble des données - faces, arêtes et sommets - forme un solide. Si les quatre premières sont faciles à vérifier, il n'en est pas de même pour les deux dernières qui requièrent des comparaisons de faces et d'arêtes.

Le volume des informations à préciser, même pour un objet simple comme la sphère, ainsi que les problèmes de validité rendent une telle représentation difficile à créer. Dans la plupart des cas, la modélisation de surfaces courbes nécessite l'utilisation d'outils d'approximation polyhédrale et de lissage [GOUR71]. Son intérêt réside dans l'accès immédiat aux informations concernant les faces et les arêtes : c'est la raison pour laquelle elle est utilisée couramment en graphique. Dans sa forme initiale, elle n'est pas redondante. Mais parfois, dans le but de diminuer la complexité des algorithmes, on est amené à introduire des informations supplémentaires, comme la normale à chaque face ou des renseignements sur l'adjacence des faces et des arêtes.

Surfaces

La représentation des objets par leurs **surfaces externes** généralise la représentation par frontières des objets courbes. En effet, les approximations polygônales nécessaires ne sont pas réalistes, puisque pour modéliser une tasse de café par exemple, 1000 faces doivent être décrites. Ce type de représentation a été introduit principalement dans une optique de génération d'images réalistes, ou plus généralement de CAO/CFAO. De plus, pour la visualisation, les algorithmes de lissage [GOUR71] deviennent inutiles.

Le principe de cette représentation est basé sur la représentation des courbes en dimension 2. Dans le plan, une courbe peut être définie par sa forme explicite

$$y = f(x)$$

ou sa forme implicite

$$F(x, y) = 0$$

La forme explicite ne permet pas de représenter de courbes fermées, ni de pentes infinies; la seconde forme entraine la résolution de systèmes d'équations. On leur préfère habituellement une forme paramétrique. Les points de la courbe sont alors définis par

$$P(t) = [\ x(t)\ \ y(t)\ ]$$

où t est un paramètre.

Cette représentation permet de décrire n'importe quel type de courbe. Elle s'étend aisément à la représentation des courbes et des surfaces dans l'espace. Leurs représentations paramétriques respectives seront

$$P(t) = [\ x(t)\ \ y(t)\ \ z(t)\ ]$$

et

$$P(u, v) = [\ x(u,v)\ \ y(u,v)\ \ z(u,v)\ ]$$

Cette modélisation n'est pas unique, puisqu'une même courbe et par extension une même surface, peut être définie de plusieurs façons. Par exemple, dans le plan, un cercle centré à l'origine et de rayon 1 est décrit indifféremment par l'une des 2 formes suivantes :

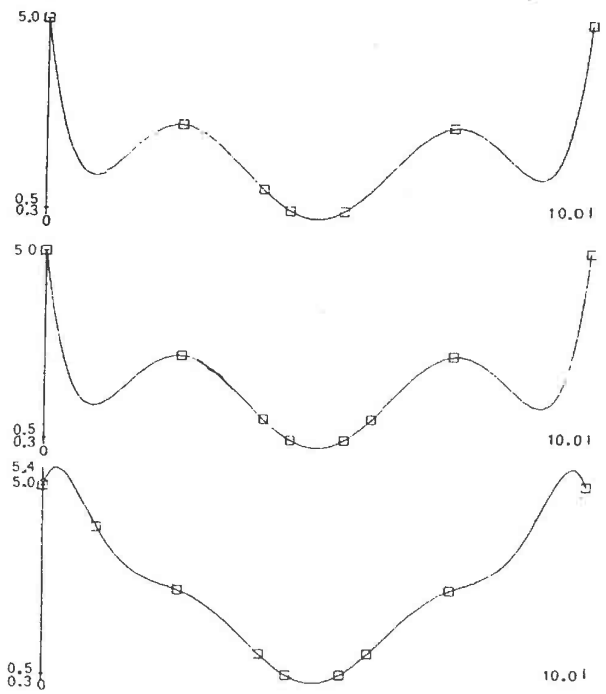
$$P(t) = [\ \cos t \quad \sin t\ ]$$

$$P(t) = [\ t \quad \sqrt{(1-t^2)}\ ]$$

L'utilisation des surfaces paramétriques peut se faire dans deux contextes différents. Dans les applications essentiellement graphiques, on cherche plutôt à interpoler une surface (ou une courbe) à partir d'un **échantillonage de points**, c'est-à-dire à restituer la forme de la courbe à partir d'un nombre fini de points. Par contre, en CAO, la conception interactive d'un objet conduit à provoquer des déformations locales et à utiliser plutôt des **points de contrôle** des surfaces.

Interpolation

Il existe de nombreuses méthodes d'interpolation de courbes et de surfaces. On peut définir un critère de qualité en essayant de minimiser l'écart entre la fonction réelle et la fonction d'approximation. On retrouve cette caractéristique dans les approximations polynômiales de type Newton ou Lagrange. Toutefois, si ce critère est numériquement important, du point de vue "graphique", il n'a plus aucun sens. La figure ci-dessous montre un exemple d'approximation polynômiale de type Lagrange, de la fonction | X-5| en 7, 8 ou 9 points : la courbe d'interpolation comporte parfois des rebonds qui ne sont pas toujours souhaitables.



L'introduction des **fonctions Splines** permet d'éviter le caractère oscillatoire illustré. Elles ont été définies par Schoenberg en 1946, dans le but de modéliser mathématiquement un instrument de dessin, utilisé en particulier dans l'industrie automobile (General Motors), pour tracer des courbes avec une grande régularité (carrosseries de voiture).

Les courbes et les surfaces sont représentées par des *fonctions polynômiales par morceaux* : on applique donc un polynôme entre chaque paire de points. Les Splines les plus couramment utilisées sont de degré trois. Si on désire interpoler une courbe passant par n points, entre deux points consécutifs, la courbe sera représentée par :

$$x(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0$$

$$y(t) = b_3 t^3 + b_2 t^2 + b_1 t + b_0$$

$$z(t) = c_3 t^3 + c_2 t^2 + c_1 t + b_0$$

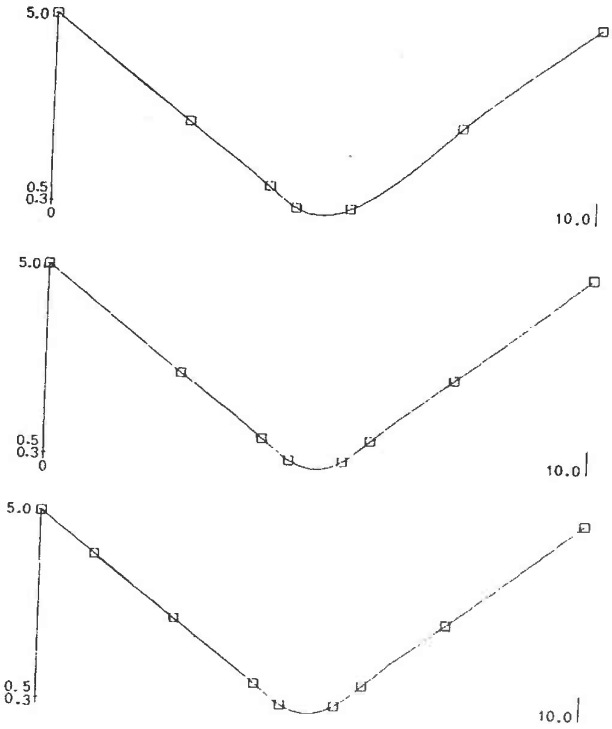
ce qu'on écrit en général sous la forme

$$\vec{P}(u) = \sum_{i=0}^{3} \vec{A_i} \, t_i$$

Comme il y a n-1 intervalles, on obtient 12(n-1) inconnues. Pour les fixer, on pose les conditions de passage par les points de l'échantillon qui fournissent 6(n-1) équations. Pour les 6(n-1) inconnues restantes, on peut soit minimiser la courbure de la courbe, soit utiliser un critère de régularité qui impose la continuité des dérivées première et seconde.

Cette technique simple d'interpolation présente toutefois deux inconvénients : elle recquiert la résolution d'un système linéaire de 12(n-1) équations; de plus, le déplacement d'un seul des points de l'échantillon provoque une déformation locale propagée sur l'ensemble de la courbe.

Le schéma ci-dessous montre une interpolation par une fonction Spline, de la fonction |X-5| vue précédemment, en 7, 8 et 9 points. La courbe obtenue est plus lissée que celle obtenue par une interpolation simple de Lagrange.



points de contrôles

Vous venons de voir au paragraphe précédent que l'utilisation des fonctions Splines pour approximer une courbe (ou une surface) ne permet pas d'effectuer des *déformations locales* sur un objet. De nouvelles modélisations sont donc apparues, visant une meilleure définition des objets courbes ainsi qu'un pouvoir accru du concepteur sur le modèle. Parmi celles le plus couramment mises en application, citons les surfaces de Coons, de Bézier, les B-Splines, les Beta-Splines et les Splines rationnelles. A titre d'exemple, nous ne nous intéresserons ici qu'aux B-Splines.

Les courbes (et les surfaces) B-Splines ont été introduites par Riesenfeld. Elles ont l'avantage de pouvoir être déformées localement, sans propagation globale des perturbations. Une courbe B-Spline est représentée paramétriquement par :

$$P(u) = \sum_{i=0}^{n} P_i \, N_{i,k}(u)$$

Les P_i représentent les points de contrôle de la courbe et les $N_{i,k}$ les fonctions de base (k est l'ordre des B-Splines). Une surface étant le produit cartésien de deux courbes, on obtient facilement la représentation paramétrique d'une surface :

$$P(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{n} P_{i,j} \; N_{i,k}(u) \; N_{j,k}(v)$$

Le domaine des fonctions $N_{i,k}$ s'étend sur un ensemble de segments $[\, t_i , t_{i+1} \,]$. La définition des fonctions de base se fait récursivement sur leur ordre :

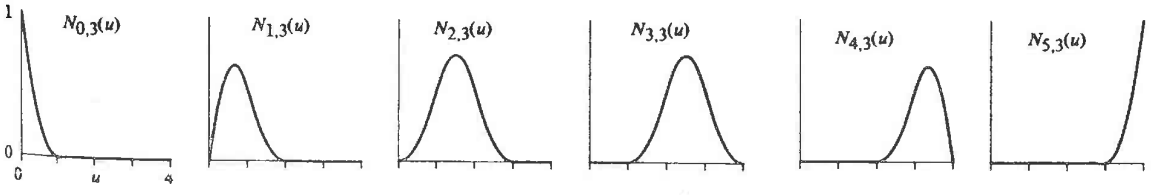
$$N_{i,k}(u) = \frac{(u - t_i)\ N_{i,k-1}(u)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - u)\ N_{i+1,k-1}(u)}{t_{i+k} - t_{i+1}}$$

en prenant comme convention $0 / 0 = 0$

On retrouve le plus fréquemment deux définitions particulières des t_i. Dans les **B-Splines uniformes non périodiques**, ils sont définis par :

$$t_i = \begin{cases} 0 & \text{si } i < k \\ i-k+2 & \text{si } k \leq i \leq n \\ n-k+2 & \text{si } i > n \end{cases}$$

La figure suivante montre, dans ce cas, les fonctions de base $N_{i,3}$, pour i variant de 0 à 5.

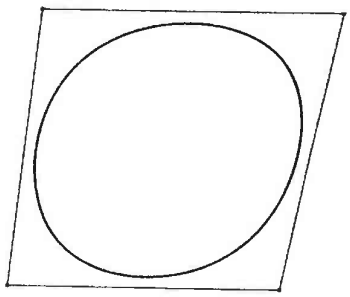


Pour les B-splines **uniformes périodiques**, $t_i = i$. Elles sont utilisées pour décrire des courbes ou des surfaces *fermées*. La forme des fonctions de base s'en trouve simplifiée :

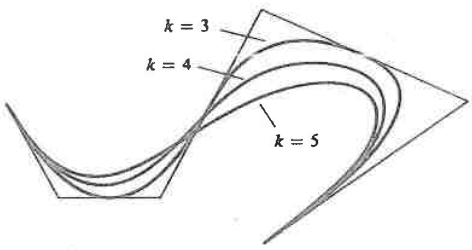
$$N_{i,k}(u) = N_{0,k}\left((u-i+n+1) \bmod (n+1)\right)$$

avec $u \in [0 .. n+1]$

La figure suivante donne un exemple de courbe obtenue à partir de 4 points de contrôles, avec k=4.

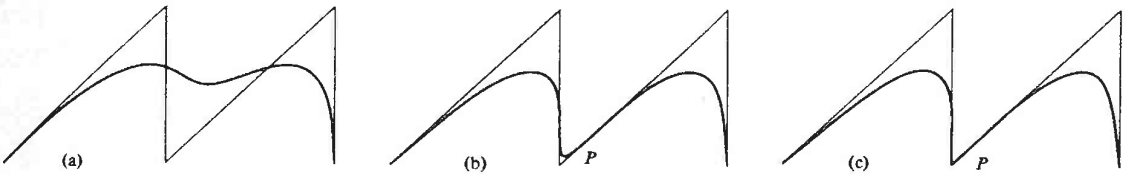


Il est important de remarquer que les fonctions de base sont nulles partout sauf sur quelques segments voisins de ceux où elles sont définies. Ce qui implique que le déplacement d'un point de contrôle n'apporte que des *déformations locales* à ce point. La portée de la déformation dépend bien sûr de l'ordre des B-splines choisi. La figure ci-dessous montre les différentes courbes obtenues, à partir de 6 points de contrôle, en utilisant des B-Splines d'ordres différents.



Comme on peut le voir sur le schéma de la page 45, les courbes et les surfaces obtenues avec cette définition ne passent en général pas par leurs points de contrôle, sauf peut-être pour le premier et le dernier points. L'ordre k permet de contrôler la continuité de la courbe ou de la surface désirée.

Il est également possible d'introduire des *points de rebroussement* dans les courbes en dupliquant les points de contrôle. La figure (a) donne la courbe obtenue à partir de 5 points de contrôle, en utilisant une B-Spline d'ordre 4. La figure (b) montre le résultat obtenu , en appliquant la même fonction, mais en dupliquant une fois le point P et la figure (c), en dupliquant 2 fois le point P.



La modélisation des objets par surfaces fournit des représentations *non ambiguës*. En effet, la surface est définie par un ensemble de points unique et une interpolation unique entre ces points. La *concision* dépend entièrement du nombre de points à fournir pour obtenir la représentation voulue. Pour finir, il faut noter que la construction d'une représentation n'est simple que si elle est interactive.

4 - CONCLUSION

Ce paragraphe était consacré à l'étude des représentation les plus courantes, ainsi que de leurs propriétés. En complément, il faut signaler que certains systèmes de modélisation géométrique utilisent parfois des **modélisations hybrides** comme CSG/frontières ou CSG/balayage : ceci signifie que les feuilles de l'arbre CSG sont, dans le premier cas des objets simples - comme le cube - ou des objets représentés par leurs frontières, et dans le second cas, des objets obtenus par balayage. Ces représentations hybrides sont cependant plus lourdes à manipuler. Le gain de place n'est appréciable que si le polygône est régulier.

Face à la diversité des différentes représentations possibles, il est souvent difficile d'effectuer le bon choix, surtout lorsqu'un même schéma est destiné à plusieurs applications. Beaucoup de systèmes de modélisation géométrique résolvent ce problème en utilisant différentes représentations. Ils mémorisent des représentations multiples d'un même objet, chacune d'elles correspondant à une utilisation différente : dans ce cas, il n'est évidemment plus question de concision.

L'intérêt de cette technique est évident : elle permet par exemple d'effectuer l'acquisition des données sous une forme simple, et de stocker diverses représentations d'un objet, chacune d'elles étant utilisée pour la résolution d'un problème bien précis. Citons par exemple le système PADL [BROW82] . La représentation principale utilisée par ce système est le graphe CSG. Les solides primitives sont des objets génériques, possèdant des dimensions et des positions par défaut. Ce type de représentation facilite la saisie et le stockage des objets. Le système convertit le graphe CSG en une représentation par frontières, pour effectuer des calculs géométriques : propriétés des volumes, intersections des objets, calcul de faces cachées ...

L'inconvénient majeur de l'existence de représentations multiples provient de la **redondance** introduite qui, d'une part, implique l'utilisation d'une place mémoire importante et d'autre part, pose des problèmes de **consistance** : comment être sûr que les différentes représentations d'un objet sont bien équivalentes ? qu'elles ne contiennent pas d'informations contradictoires ? comment effectuer automatiquement les conversions entre les représentations ? Ces problèmes théoriques sont actuellement en cours d'étude; malgré tout, bien qu'il soit montré que certaines conversions soient possibles, il n'existe pas forcément pour autant d'algorithmes qui les effectuent.

CHAPITRE II

TRIDENT : UN SYSTEME DE RECONNAISSANCE DES FORMES

1 - INTRODUCTION

Analyser et comprendre une scène tridimensionnelle est une tâche élémentaire que le système de vision humain effectue avec précision et rapidité. Ce système est d'une grande complexité car il intègre toutes sortes d'informations qui proviennent soit de la scène elle-même, soit d'une connaissance a priori de l'univers dans lequel il évolue. Il est donc impossible de construire un système de vision par ordinateur aussi performant que le système visuel humain.

Les **paramètres** différenciant les divers systèmes de vision sont nombreux. On peut citer par exemple la **nature de l'image** à traiter qui, comme on le verra plus tard, peut-être binaire, spectrale, de niveaux de gris ou de distances. Le problème de la **visibilité des objets** est également essentiel : les objets à traiter peuvent être toujours complètement visibles ou parfois partiellement cachés. Dans certains cas, ces objets peuvent être de **forme** unique - pour une pièce mécanique particulière, par exemple -, ou de forme variable mais modélisable - comme une chaise -, ou de forme difficile à modélisable - comme un bouquet de fleurs.

Ces trois facteurs de la complexité d'un système de vision (nature de l'image, visibilité et forme des objets) sont importants. Néanmoins, NEVATIA [NEVA78] estime que la position et la composition des objets restent les paramètres essentiels de cette complexité. En effet, lorsque la position et la description des objets sont connues précisément, la tâche d'un système de vision peut être par exemple de mesurer l'une des dimensions d'un objet. Parfois, seule la description est connue : il s'agit alors de localiser l'objet; si on ne connait que sa position, le problème est de déterminer la structure de l'objet. Plus généralement, lorsque position et description des objets sont inconnues, le système de vision est alors un **système de reconnaissances de scènes**. Le système TRIDENT travaille dans cette optique générale et difficile.

Classiquement, un système d'interprétation de scènes est constitué de deux couches :

* la couche appelée **BAS-NIVEAU** inclue des outils de perception sa tâche est d'extraire de la scène un ensemble d'informations appelées **indices visuels**.

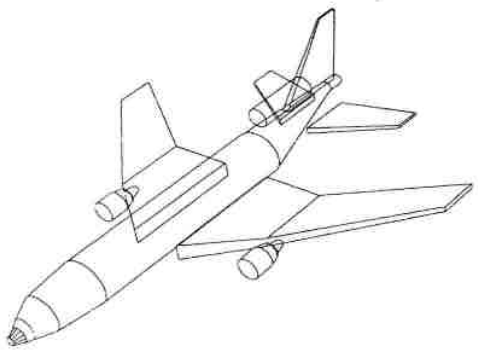
* la couche appelée **HAUT-NIVEAU** ou encore **NOYAU INTELLIGENT** effectue l'interprétation proprement dite, c'est- à-dire fait la correspondance entre les indices visuels fournis par le bas-niveau et les **indices objets** [LUX84] extraits d'une description formelle, appelée **modèle**, de l'univers dans lequel le système évolue.

Les différences entre les systèmes d'interprétation de scènes existants actuellement se situent à tous les niveaux, c'est-à-dire aussi bien dans les indices visuels et objets, que dans la structure même des deux couches et des interactions entre celles-çi.

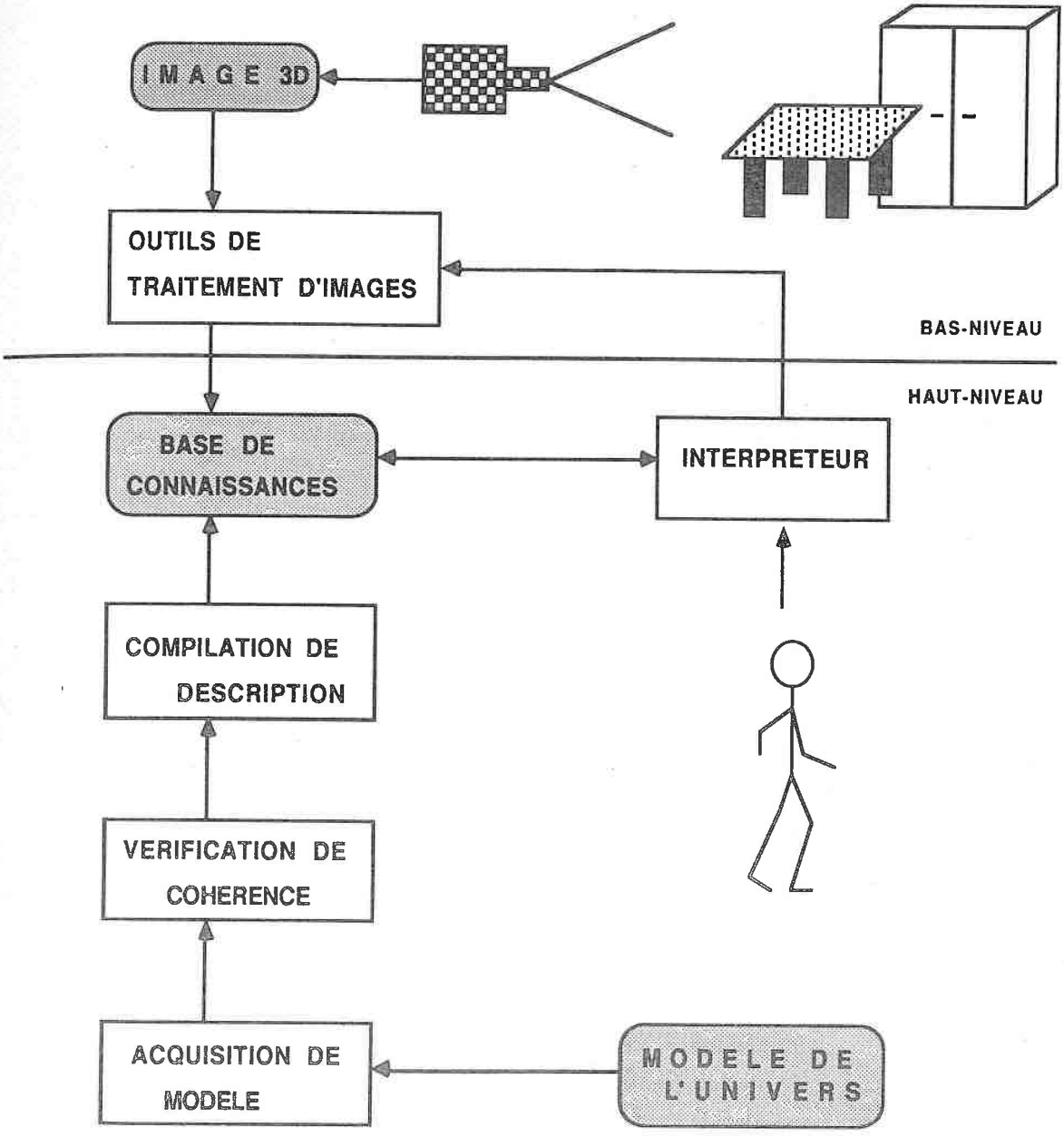
L'exemple le plus simple que l'on puisse citer est pris dans le domaine de la robotique, pour la prise d'objets sur un tapis roulant, par exemple [DODD79] : dans de tels systèmes, les images traitées sont souvent *binaires* et les indices extraits, des *contours*. Et dans ce cas, la tâche de la couche haut-niveau se réduit à la détermination de la *position* et de l'*orientation* des objets : ceci se fait par la mise en correspondance des contours extraits de l'image et de ceux du modèle.

Dans le même ordre d'idée, citons également le système développé par OSHIMA et SHIRAI [OSHI79], qui se limite à l'extraction d'indices visuels comme des plans et des régions courbes; les indices objets proviennent d'un modèle acquis dans une phase d'apprentissage durant laquelle des objets connus sont présentés au système sous différents angles; celui-çi en fait une description en termes de surfaces et de relations entre ces surfaces.

Par ailleurs, il existe d'autres systèmes de vision plus généraux, l'univers dans lequel ils évoluent n'étant pas réduit à une petite catégorie d'objets. Dans de tels systèmes, la tâche du haut-niveau est d'étiqueter les objets de la scène. Parmi ceux-çi, citons le système ACRONYM [BROO78] : le modèle qu'il utilise définit les objets par composition de primitives de type **éléments de cônes généralisés**. Un cône généralisé est décrit par un axe qui n'est pas forcément rectiligne et par une section qui n'est pas nécessairement constante se déplaçant le long de cet axe. Un tel outil est puissant car il permet la description d'une grande catégorie d'objets. En voici un exemple simple :



Le système TRIDENT, développé au CRIN [ZARO83] [MASI84] [MAZA84], s'inscrit dans cette catégorie de systèmes d'interprétation de scènes quasi-naturelles : le modèle utilisé est inspiré de celui d'Acronym. L'intérêt de ce système est d'utiliser pleinement l'information contenue implicitement dans le modèle, afin de guider et d'accélérer la reconnaissance des objets. Sa structure générale est donnée par le schéma de la page suivante. La description précise de la couche bas-niveau, appelée **système de perception**, est donnée au paragraphe 2. Le modèle utilisé par TRIDENT fera ensuite l'objet du paragraphe 3. Au paragraphe 4, nous donnerons un aperçu du fonctionnement du **système de déductions** qui constitue la couche haut-niveau.

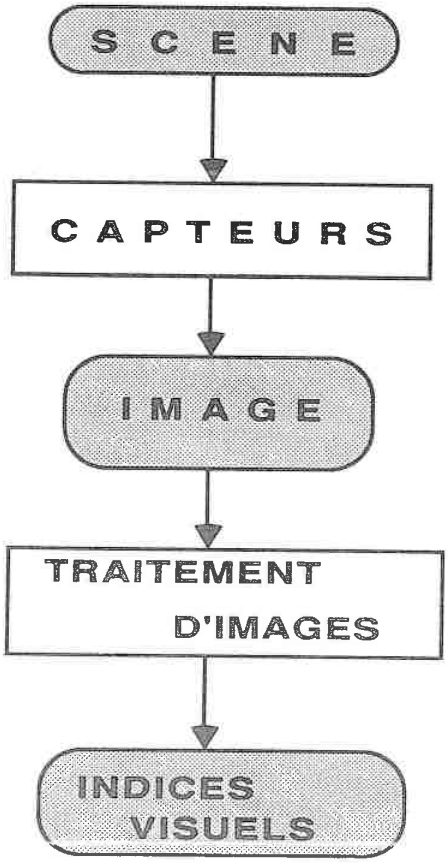


SCHEMA GENERAL DE TRIDENT

2 - LE SYSTEME DE PERCEPTION OU BAS-NIVEAU

De façon générale, le rôle du système de perception d'un interpréteur de scènes est d'extraire un **ensemble d'indices visuels** dans les images de la scène.

Le système de perception de TRIDENT a la structure suivante :



Le rôle des capteurs est l'acquisition d'une ou de plusieurs image(s) de la scène à analyser. On distingue habituellement quatre sortes d'images :

* **l'image binaire**, qui, à chaque point de l'image, associe une information binaire

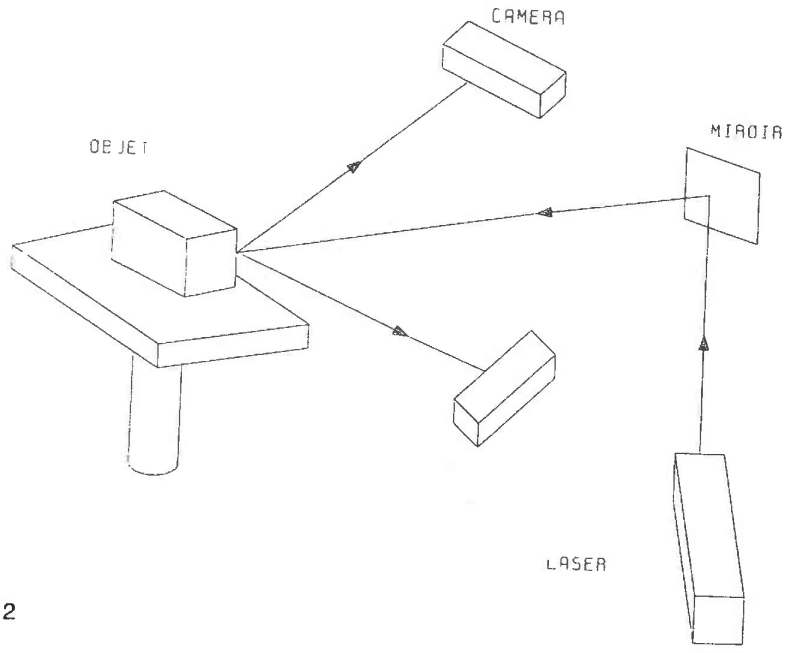
* **l'image de couleurs**, qui à chaque point lui associe sa couleur définie dans un système de coordonnées comme par exemple les systèmes RGB (Red-Green-Blue) ou CMY (Cian-Magenta-Yellow) ou HLS (Hue-Ligthness-Saturation)

* **l'image de distances**, appelée encore **image tridimensionnelle**, qui associe à chaque point la distance du point au capteur

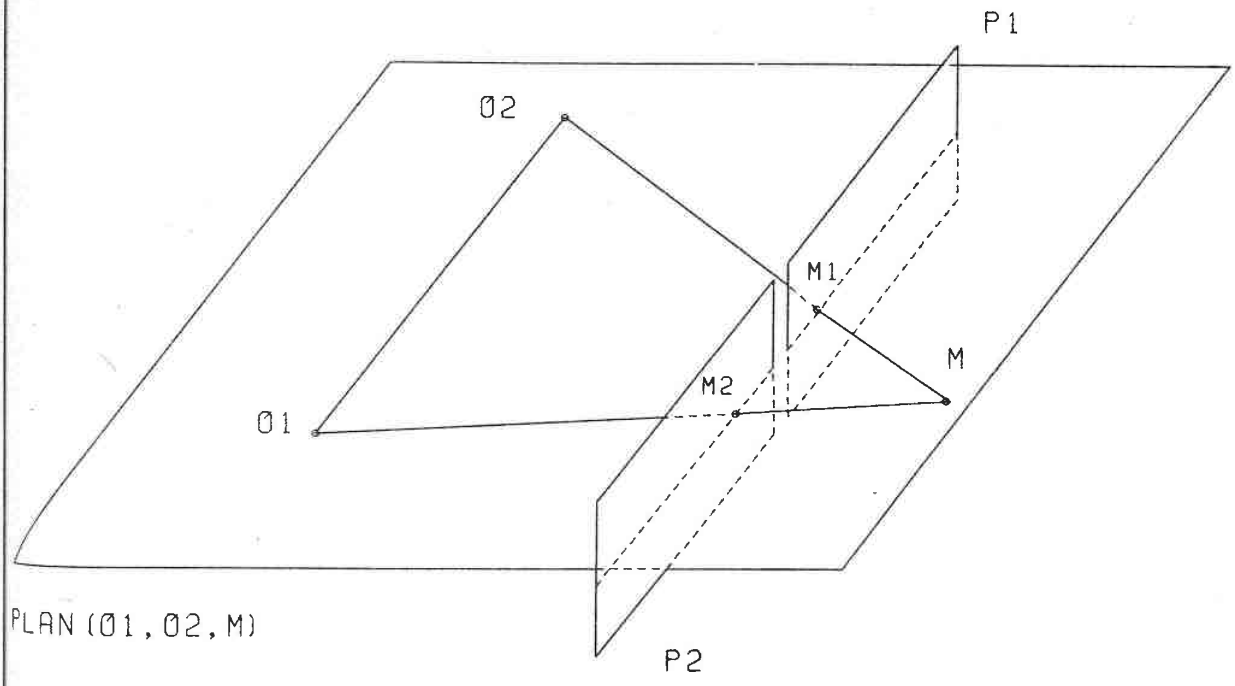
* **l'image de niveaux de gris**, qui à chaque point lui associe l'intensité lumineuse réfléchie par l'objet en ce point.

Dans TRIDENT, nous avons choisi de traiter des images de distances, car, dans ce cas seulement, la prise en compte des parties cachées est simplifiée. En effet, dans une image de distance, une discontinuité de la valeur de deux points voisins signifie qu'ils appartiennent à deux objets différents : ce résultat ne peut en aucun cas être appliqué aux images binaires, de niveaux de gris ou de couleurs, mais il existe toutefois d'autres techniques pour y parvenir. Pour capter la troisième dimension, il existe différentes solutions. Parmi celles-çi, citons les deux méthodes utilisées dans TRIDENT :

* Dans une première maquette de TRIDENT [ZARO83], l'acquisition du relief se faisait par une **technique de suréclairage** développée à l'INRIA [AYAC82] : un laser réfléchi par un miroir balaie la scène; deux caméras récupèrent le signal lumineux et permettent ainsi d'obtenir la distance de chaque point au miroir. Les résultats obtenus sont très précis : ils sont de l'ordre du millimètre, pour des objets de l'ordre du mètre.



* L'autre solution, actuellement en cours d'étude et de réalisation [WROB87] utilise les principes de la **stéréovision**. Cette technique, qui se rapproche plus du système visuel humain, consiste à déduire une information sur le relief à partir de deux ou plusieurs images bidimensionnelles d'une même scène, prises depuis des endroits différents. Il s'agit donc de mettre en correspondance des points images qui désignent le même point physique sur la scène, puis de reconstruire les coordonnées tridimensionnelles de ces points physiques par triangulation.



La seconde tâche du système de perception est d'extraire de l'image fournie par le capteur - ici, une image de distances - un ensemble d'**indices visuels**. En effet, l'image brute telle qu'elle est fournie par le capteur contient trop d'informations et n'est pas suffisamment structurée pour être directement utilisable par l'interpréteur. Le module de traitement d'images a donc pour objet d'extraire d'une image tridimensionnelle un ensemble d'indices visuels (directement exploitables).

Là encore, le choix de ces indices est varié : **contours, régions, surfaces** et **volumes**. Une des optiques de TRIDENT est de regrouper ces différentes possibilités. Un algorithme d'extraction de surfaces et de volumes simples est déjà opérationnel : il utilise la transformée de HOUGH [MULL84]. D'autre part, un algorithme de segmentation développé recemment permet de rechercher les différents contours et régions d'une image [WROM87].

L'extraction d'indices visuels à partir d'une image de distances peut se faire avant toute tentative d'analyse de la scène : on recherche ainsi tous les indices possibles, dans toute l'image. Face à cette utilisation systématique, nous étudions actuellement une utilisation **dynamique** [MASI85]. Ainsi, à tout moment de l'interprétation, devrait s'établir un *dialogue* entre haut et bas niveaux. L'interpréteur pourrait alors demander la recherche d'un type particulier d'indices sur toute l'image ou une partie de celle-çi en cours d'analyse : les informations fournies par le système de perception lui permettrait ainsi d'entamer son interprétation et d'étayer ou de réfuter ses hypothèses.

Voici quelques exemples de demandes que peut formuler l'interpréteur:

* quelles sont les surfaces évidentes de l'image (surfaces très larges) ?
* quelles sont les régions de telle portion d'image ?
* existe-t-il dans telle région, un cylindre de rayon approximativement égal à 5 ?
* existe-t-il une surface plane verticale dans telle région de l'image?

etc ...

Une telle stratégie a trois avantages essentiels : elle permet de *renforcer* la puissance de l'interpréteur, de *réduire* le temps consacré à l'extraction d'indices visuels (puisque seuls sont extraits les indices indispensables) et enfin de *diminuer* le nombre d'erreurs de segmentation (puisque celle-ci peut être guidée par des résultats antérieurs).

3 - LE MODELE DE L'UNIVERS

3.1 - INTRODUCTION

Pour interpréter une scène, le système TRIDENT s'appuie sur des connaissances a priori de l'univers dans lequel il évolue. L'ensemble de ces informations est appelé **modèle**. Comme nous l'avons vu dans le paragraphe précédent, TRIDENT est conçu pour interpréter des scènes quasi-réelles : le modèle doit donc contenir la description précise des familles de scènes que l'on aura à analyser.

Le modèle de TRIDENT est inspiré de celui proposé par ACRONYM [BROO78], sans toutefois en retenir la généralité. En effet, dans ACRONYM, les objets sont définis par composition de cylindres généralisés : un tel formalisme permet, en théorie, de décrire n'importe quel type d'objet. Ce modèle est donc général.

Dans sa forme initiale, il nous semble inutilisable par un système d'interprétation de scènes, car trop complexe et donc trop lourd à manipuler. Cette remarque s'appuie sur le fait que l'ensemble des formes effectivement traitées par ACRONYM - des avions - est plus restreint que l'ensemble des objets que le formalisme permet de décrire. Dans sa forme d'origine, ce formalisme présente le gros inconvénient d'être ambigü, puisque par exemple, un objet simple comme un cube peut être décrit de plusieurs façons.

Soulignons également que la conception du modèle d'une scène demande de la part de l'utilisateur des connaissances géométriques importantes, même pour la description de scènes ne contenant que des objets simples.

Le modèle de TRIDENT est essentiellement volumique. La création d'objets se fait par composition d'objets solides simples, appelés **primitives**. Les caractéristiques propres des objets, notamment les **dimensions**, ainsi que les **positions** relatives des objets et de leurs composants sont données sous forme de contraintes numériques. La structure précise de notre modèle sera définie au paragraphe 3.2.

Un tel modèle atteignant rapidement un haut degré de complexité, nous sommes confrontés à deux problèmes : d'une part *faciliter* la tâche de l'utilisateur dans la conception du modèle, et d'autre part permettre au mécanisme d'interprétation *d'exploiter efficacement* les informations contenues dans le modèle. La phase d'aide à la conception fait l'objet du chapitre III : elle ne sera pas plus amplement détaillée ici. Le paragraphe 3.3 précisera quelles sont les informations extraites du modèle, dans la phase de compilation.

3.2 - DESCRIPTION DU MODELE.

3.2.1 - Les règles .

Dans TRIDENT, la description d'une scène se fait par l'intermédiaire d'un **système de règles**, chacune d'elles précisant la structure logique d'un objet. Une règle associe à un objet (ou une forme) les **sous-formes** qui le composent. Ainsi, la règle

/1/ SCENE → BUREAU ARMOIRE TELEPHONE

définit la structure de l'objet complexe SCENE comme étant la composition de trois sous-formes plus simples BUREAU, ARMOIRE et TELEPHONE.

Une même forme peut être décrite de diverses façons : la règle /1/ associée à la règle

/2/ SCENE → FAUTEUIL BUREAU TELEPHONE CORBEILLE

indique les deux alternatives possibles sur la structure de l'objet SCENE. Ceci permet de définir des familles d'objets génériques.

Toute description récursive d'objet est interdite : la récursivité rend le modèle **ambigü**, puisque plusieurs descriptions peuvent être données pour le même objet. De plus, le problème de la validation de modèles se complique, puisque les formes peuvent être décomposées à l'infini.

Chaque sous-forme apparaissant en partie droite d'une règle est elle-même, soit décrite par une autre règle, soit une forme élémentaire appelée **primitive**. Aux règles /1/ et /2/, on ajoute donc les règles de description des sous-formes BUREAU, ARMOIRE, TELEPHONE, FAUTEUIL, TABLE et CORBEILLE. Par exemple :

/3/	BUREAU	→	PLATEAU-ROND PIED
/4/	BUREAU	→	PLATEAU-RECT PIED
/5/	PLATEAU-ROND	→	**CYLINDRE**
/6/	PLATEAU-RECT	→	**PARALLELEPIPEDE**
/7/	TABLE	→	PLATEAU-RECT PIED PIED PIED PIED
/8/	PIED	→	**CYLINDRE**
/9/	PIED	→	**PARALLELEPIPEDE**
/10/	CORBEILLE	→	**CYLINDRE**

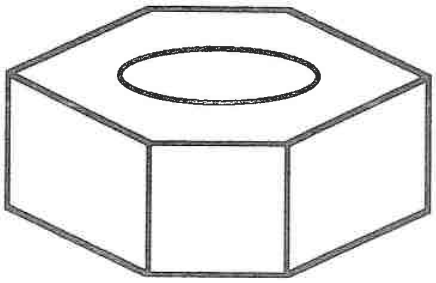
ETC ...

On pourra remarquer l'analogie de cette description avec les *grammaires à contexte libre* : ici cependant l'ordre des sous-formes en partie droite des règles n'a aucune importance.

La définition précise des primitives utilisées sera donnée ultérieurement. Ce sont des **surfaces** telles que des polygônes plans, ou bien des **volumes simples** tels que des sphères, des cylindres ou des cônes. Par la suite, nous les noterons respectivement FACETTE, SPHERE, CYLINDRE et CONE. Elles sont assemblées uniquement par unions : cette modélisation est donc un cas particulier de la modélisation par construction telle que REQUICHA l'a définie [REQU80].

Le domaine des objets représentables est restreint du fait de l'abscence d'opérateurs comme l'intersection ou la différence. Par exemple, la représentation d'objets avec des trous, tels que des écrous, est impossible. Cette importante restriction s'impose si on remarque que l'utilisation de tels opérateurs rend impossible la prédiction de l'existence de formes sur l'image : en effet, dans notre modèle, le résultat d'opérations comme l'intersection ou la différence de formes serait imprévisible, les dimensions des objets n'étant pas toujours précisées.

Par exemple, dans ce formalisme , cet écrou est impossible à décrire, puisque nous n'avons d'opération de soustraction entre objets à notre disposition :



3.2.2 - Les attributs.

Comme nous venons de le voir, une règle décrit la structure d'un objet. Elle précise uniquement les noms des sous-formes qui le composent. La définition précise de ces sous-formes, de leurs dimensions, de leurs positions relatives et absolues se fait par l'intermédiaire de caractéristiques particulières attachées aux objets, appelées **attributs**. Par exemple, la hauteur de l'objet BUREAU est une caractéristique fondamentale : elle est représentée par un attribut HAUTEUR, que l'on notera

H (BUREAU)

Nous avons classé les attributs en huit catégories, appelées **types**:

- une **distance**
- une **surface**
- un **volume**
- un **angle**
- une **réflectivité de surface**
- un **point**
- une **direction**
- une **couleur**

Le concepteur devra donc préciser le type de chaque attribut qu'il introduit. Par exemple, il devra signaler que l'attribut H (BUREAU) est de type *distance*. Cette différenciation des types d'attributs manipulés permettra d'effectuer des vérifications sémantiques lors de l'utilisation ultérieure des attributs : ce point sera détaillé dans le paragraphe suivant. Elle permet également de faire correspondre à chaque attribut du modèle, le domaine de ses valeurs possibles indépendamment de tout contexte.

Ainsi :

* les attributs de type **distance, surface, volume** prennent leurs valeurs dans $\mathbb{R}$

* un attribut de type **angle** prend des valeurs réelles dans $[0 .. 2\pi]$

* un attribut de type **réflectivité de surface** prend des valeurs réelles dans [0 .. 1]

* un attribut de type **point** prend ses valeurs dans $\mathbb{R}^3$; les éléments du triplet sont donc de type *distance* et représentent respectivement les coordonnées du point dans les axes Ox, Oy et Oz du référentiel absolu : chaque attribut de ce type sera donc décomposé en trois attributs de type *distance* que l'on notera X, Y et Z

* un attribut de type **direction** prend ses valeurs dans $[0 .. 2\pi] \times [0 .. 2\pi]$, les éléments du couple sont de type *angle*, et représentent respectivement les angles que fait la direction avec les axes Ox et Oy : chaque attribut de ce type sera décomposé en deux attributs de type *angle* que l'on notera PHI et TETA

* la définition du domaine des valeurs d'un attribut de type **couleur** pose plus de problèmes : en effet, une couleur est définie par ses coordonnées dans un système comme RGB ou CMY. Dans ce cas, un attribut de type couleur prend ses valeurs dans $\mathbb{R}^3$. L'utilisation de tels attributs nécessite, de la part du concepteur, une connaissance approfondie de la composition des couleurs dans le système utilisé, ce qui n'est pas concevable. Aussi, le système d'acquisition de modèles *discrétise* l'espace des coordonnées et propose au concepteur un ensemble de couleurs usuelles, comme par exemple :

  { ROUGE, VERT, BLANC, BLEU, JAUNE, NOIR, GRIS, BRUN, ROSE }

  L'utilisateur n'a donc pas besoin de connaitre la composition de ces couleurs car le système d'acquisition effectue lui-même la conversion dans le système choisi. Cette interface, quoique simplifiant le travail du concepteur, présente toutefois l'inconvénient de limiter le nombre de couleurs possibles.

En théorie, l'ensemble des types prédéfinis est suffisant pour définir les attributs des objets puisque, dans le cadre d'un système d'interprétation de scènes comme TRIDENT, les caractéristiques essentielles des objets sont principalement *géométriques* ou *spectrales*, puisque ce sont les seules informations que nous sommes capables d'extraire des images.

Cependant, il est possible d'imaginer l'utilisation d'attributs définissant d'autres caractéristiques des objets : par exemple pour répondre à des questions comme : *existe-t-il un endroit pour s'asseoir dans la scène ?* ou *la scène contient-elle des objets inflammables ?*. Pour cela, il faudrait bien sûr fournir au système les informations concernant d'une part la fonction des objets et d'autre part, leur matière.

Dans cette éventualité, qui rappelons-le n'est pas le but de TRIDENT, le système d'acquisition laisse donc à l'utilisateur la possibilité d'introduire de nouveaux types. A chacun d'eux sera associé l'ensemble des valeurs que pourront prendre les attributs de ce type. Dans notre exemple, il sera nécessaire d'introduire le type FONCTION avec comme valeurs autorisées

{ DECORER ASSEOIR MANGER DORMIR BOIRE RANGER }

et le type MATIERE dont l'ensemble des valeurs possibles serait

{ VERRE ACIER BOIS PLASTIQUE BETON TISSU PAPIER }

En fonction de ses besoins, le concepteur du modèle peut donc associer à chaque forme n'importe quelle caractéristique - géométrique ou non - et de type quelconque.

Toujours dans l'optique de faciliter le travail du concepteur, en ne lui imposant pas le travail fastidieux de définir des attributs apparaissant systématiquement dans chaque forme, le système d'acquisition associe de façon systématique à chaque forme du modèle, un ensemble d'attributs dits **prédéfinis**, indispensables à l'interpréteur :

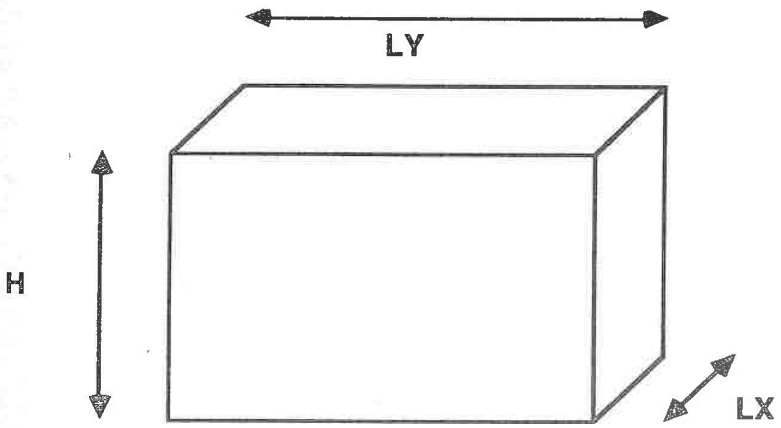
* **XMIN, XMAX, YMIN, YMAX, ZMIN,** et **ZMAX** qui déterminent les coordonnées extrêmes de l'objet sur les axes Ox, Oy et Oz du référentiel absolu dans lequel est décrit l'objet : ces attributs sont de type *distance*.

* **H** = ZMAX - ZMIN, de type *distance*, représentant la hauteur de l'objet

* **LY** = YMAX - YMIN, de type *distance* représentant la longueur de l'objet

* **LX** = XMIN - XMAX, de type *distance* représentant la largeur de l'objet

* **COUL** de type *couleur*, représentant la couleur de l'objet.



Le concepteur peut employer ces attributs comme il l'entend, sans se préoccuper de leurs définitions, puisque le système sait ce qu'ils représentent.

Pour les traitements ultérieurs, on distinguera plus généralement deux sortes d'attributs :

* les attributs dits **continus** prennent leurs valeurs dans un domaine continu : ce sont donc les attributs de type distance, surface, volume, coefficient de réflectivité et point

* les attributs dits **discrets** prennent leurs valeurs dans un domaine discret, donc essentiellement les attributs de type couleur avec la discrétisation que nous avons choisie.

3.2.3 - Les primitives et leurs attributs

Nous avons déjà précisé que les objets sont assemblés par unions de primitives comme des polygônes plans, des sphères, des cylindres, des cônes et des ellipsoides. Outre les attributs prédéfinis vus précédemment - XMAX, XMIN, YMAX, YMIN, ZMAX, ZMIN, H, L, LG et COUL - , il est nécessaire d'associer à chacune de ces primitives un ensemble d'attributs représentant leurs **caractéristiques géométriques essentielles**. Ces attributs seront également prédéfinis.

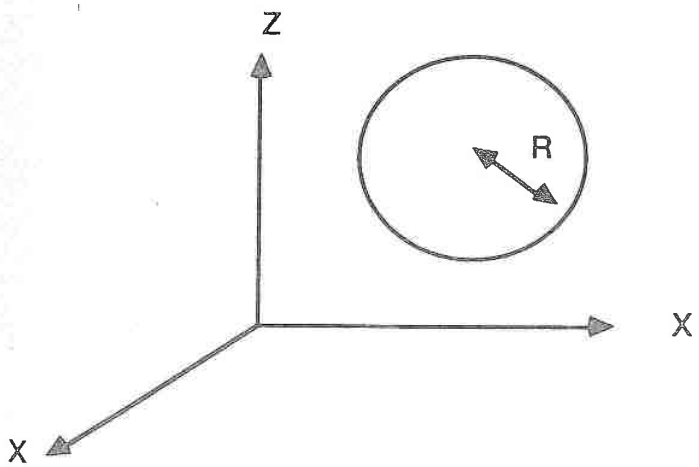
L'ensemble de tous les attributs associés à une primitive est redondant. Le concepteur de modèle peut les utiliser à son gré, car le système MEPHISTO connait les contraintes reliant les différents attributs entre eux.

A la primitive **SPHERE** sont associés les attributs :

≈ le rayon de la sphère, appelé **RAYON**, de type *distance*

≈ le centre de la sphère, appelé **CENTRE**, de type *point*; ses coordonnées sont donc notées

X (CENTRE)
Y (CENTRE)
Z (CENTRE)



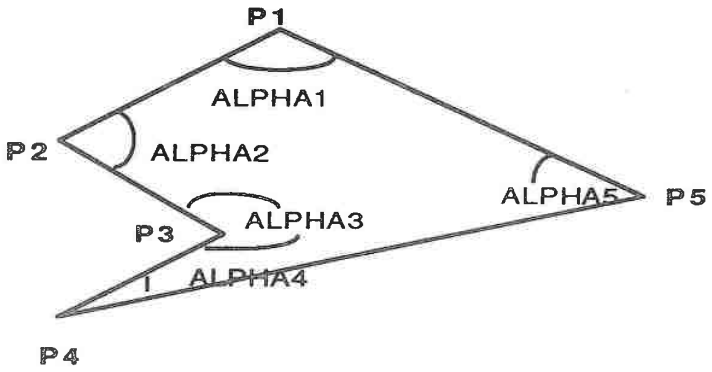
A la primitive **FACETTE$_n$** sont associés les attributs :

≈ la liste des sommets, appelés **P$_1$**, ..., **P$_n$**, de type *point* et dont les trois coordonnées sont notées respectivement

X (P$_1$)		**X (P$_n$)**
Y (P$_1$)	.....	**Y (P$_n$)**
Z (P$_1$)		**Z (P$_n$)**

≈ la liste des angles aux sommets, appelés **ALPHA$_1$**, ..., **ALPHA$_n$**, de type *angle*; chaque ALPHA$_i$ est l'angle que font les deux segments de droite [P_{i-1} P_i] et [P_i P_{i+1}].

≈ les trois coefficients de l'équation du plan de la facette, notés respectivement **PLANA**, **PLANB** et **PLANC**.



A la primitive **CONE** sont associés les attributs

≈ le point au sommet, appelé **POINT_SOMMET**, de type *point;* ses coordonnées sont donc notées

X (POINT_SOMMET)
Y (POINT_SOMMET)
Z (POINT_SOMMET)

≈ le point de la base, appelé **POINT_BASE**, de type *point*; ses coordonnées sont donc notées

X (POINT_BASE)
Y (POINT_BASE)
Z (POINT_BASE)

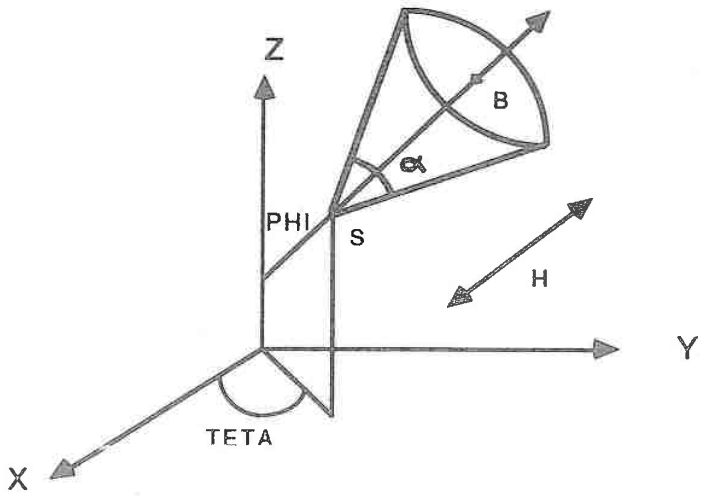
≈ l'angle au sommet, appelé **ALPHA**, de type *angle*

≈ la direction de l'axe de révolution, appelée **DELTA**, de type *direction*; ses deux composantes de type *angle* sont donc notées

PHI (DELTA) l'angle fait avec Oz
TETA (DELTA) l'angle fait avec Ox

≈ le rayon de la base du cône, appelé **RAYON**, de type *distance*

≈ la hauteur du cône, appelée **H**, de type *distance* .



A la primitive **CYLINDRE** sont associés les attributs

≈ les deux points extrêmes de l'axe, appelés **POINT_SOMMET** et **POINT_BASE**, de type *point*, et dont les coordonnées sont notées respectivement

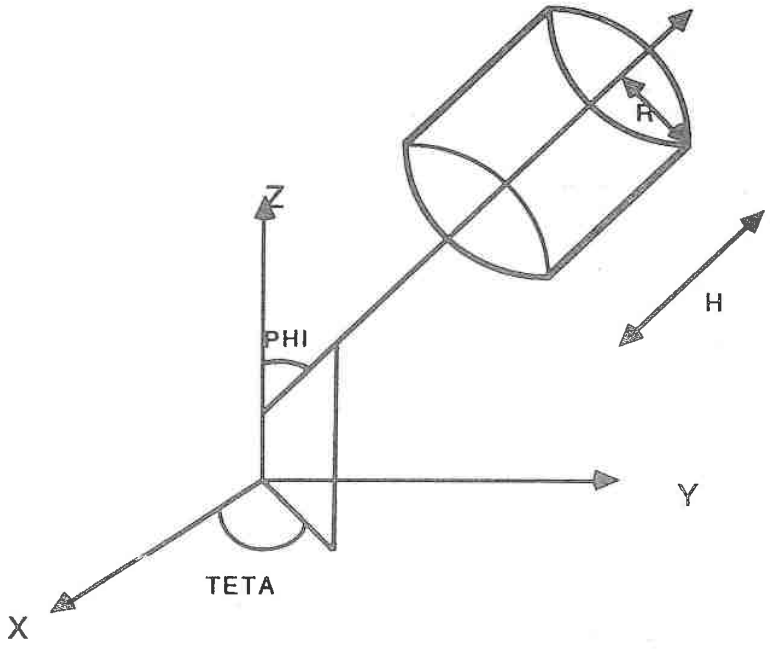
X(POINT_SOMMET)		**X (POINT_BASE)**
Y (POINT_SOMMET)	et	**Y (POINT_BASE)**
Z (POINT_SOMMET)		**Z (POINT_BASE)**

≈ le rayon de la base, appelé **RAYON**, de type *distance*

≈ la hauteur, appelée **H**, de type *distance*

≈ la direction de l'axe de révolution, appelée **DELTA**, de type *distance*; ses deux composantes de type *angle* sont notées

PHI (DELTA)
TETA (DELTA)



3.2.4 - Les contraintes

Nous avons vu au paragraphe 3.2.1. qu'une règle d'un modèle définit la composition d'une forme par unions de primitives. Aux paragraphes 3.2.2 et 3.2.3, nous associons à chaque forme - primitive ou non - un ensemble d'attributs définissant ses caractéristiques - géométriques ou non. Ces attributs vont être utilisés pour préciser en particulier les *modalités de construction et d'assemblage des formes et sous-formes* de chaque règle : ceci se fait par l'intermédiaire de **contraintes** associées à chaque règle et portant sur les attributs propres à chaque objet apparaissant dans celle-ci. Pour l'instant, nous imposons que ces contraintes soient **linéaires**. Ces contraintes peuvent être vues comme des attributs à valeur boolénne associés aux règles.

Par extension de la différence faite auparavant entre attributs dits continus et discrets, nous distinguerons deux types de contraintes : les **contraintes continues** et les **contraintes discrètes**.

De même qu'il existe des attributs prédéfinis sur les primitives, il existe également un ensemble de contraintes continues prédéfinies associées aux primitives et aux règles elles-même. Celles-çi seront détaillées dans l'un des paragraphes suivants.

3.2.4.1 - Les contraintes continues

Les contraintes continues portent uniquement sur des attributs continus. Une telle contrainte peut imposer un domaine de valeurs à un attribut soit de façon **absolue**, soit de façon **relative**. Prenons l'exemple de la règle suivante :

/1/ SCENE → BUREAU ARMOIRE TELEPHONE

Dans cette règle, on impose un domaine de valeurs à l'attribut H (BUREAU) de façon absolue par :

H (BUREAU) = 90

ou **80 ≤ H (BUREAU) ≤ 100**

et de façon relative par :

H (BUREAU) ≤ H (ARMOIRE)

ou **XMAX (BUREAU) = XMAX (ARMOIRE) + 40**

ou **H (SCENE) = max (H (BUREAU), H (ARMOIRE), H (TABLE))**

L'utilisateur peut donc à son gré, écrire des contraintes quelconques sur des attributs continus, à la seule condition que ces contraintes soient linéaires. Elles permettent ainsi de fixer les dimensions des objets. Pour déterminer la composition des formes, on procèdera généralement par coincidences de points particuliers des objets.

Signalons toutefois que, dans le cas de contraintes portant sur des attributs de types *point* et *direction*, le système d'acquisition de modèle est dans l'obligation de traduire les contraintes sous une forme directement exploitable : une contrainte portant sur des attributs de type *point* est traduite par trois contraintes portant sur les coordonnées des points, et une contrainte portant sur des attributs de type *direction* par deux contraintes portant sur les angles définissant la direction. Par exemple, la contrainte

P1 (BUREAU) = PT1 (TELEPHONE)

où P1 (BUREAU) et PT1 (TELEPHONE) sont de type *point*, est traduite par

X (P1 (BUREAU)) = X (PT1 (TELEPHONE))
Y (P1 (BUREAU)) = Y (PT1 (TELEPHONE))
Z (P1 (BUREAU)) = Z (PT1 (TELEPHONE))

En d'autres termes, on écrit que la coordonnée X du point P1, de l'objet BUREAU est identique à la coordonnée X du point PT1, de l'objet TELEPHONE.

Plus formellement, une contrainte linéaire continue est de la forme :

expression_linéaire **opérateur_de_relation** **expression_linéaire**

où

opérateur_de_relation $\in \{ =, \leq, \geq \}$
expression_linéaire est une somme de monômes (linéaires ou constants)

Dans le paragraphe 3.2.2., nous avons vu que chaque attribut est défini par un type. Ainsi, on peut effectuer des **vérifications sémantiques simples** sur les types des attributs apparaissant dans les contraintes continues introduites par l'utilisateur. Dans la mesure où elles sont exclusivement linéaires, la règle de compatibilité de types est simple : les deux expressions linéaires composant une contrainte doivent obligatoirement porter sur des attributs de même type. Par exemple, une erreur sémantique est signalée à l'utilisateur, lors de l'écriture d'une contrainte comme :

H (FAUTEUIL) = SURF (BUREAU)

si l'attribut SURF (BUREAU) est défini comme étant de type *surface*.

Ces contrôles sémantiques seraient d'une aide plus efficace dans une extension du modèle de TRIDENT prenant en compte des contraintes quelconques - linéaires ou non. En effet, la règle de compatibilité de types restera valide, mais devra tenir compte du fait que le produit de deux attributs de type *distance* donne un attribut de type *surface*, et que le produit d'un attribut de type *distance* et d'un attribut de type *surface* donne un attribut de type *volume*. La version actuelle de TRIDENT ne tient pas compte de ces possibilités.

3.2.4.2 - Les contraintes discrètes

Les contraintes discrètes portent sur des attributs discrets. De façon symétrique aux contraintes continues, on peut imposer un domaine de valeurs à un attribut de façon **relative** ou **absolue.**

Par exemple, dans la règle /1/, le concepteur peut préciser la couleur de l'objet BUREAU

≈ de **façon relative**, comme par exemple

COUL (BUREAU) = COUL (ARMOIRE)
et COUL (BUREAU) ≠ COUL (TELEPHONE)

≈ de **façon absolue**, comme par exemple

COUL (BUREAU) ∈ { GRIS, NOIR, BRUN, ROUGE }

ou bien COUL (BUREAU) ∉ { JAUNE, BLEU }

Plus formellement, une contrainte discrète a la forme suivante :

attribut_discret = attribut_discret
ou **attribut_discret ≠ attribut_discret**
ou **attribut_discret ∈ ensemble_de_valeurs**
ou **attribut_discret ∉ ensemble_de_valeurs**

Des vérifications sémantiques sont également nécessaires lors de l'écriture d'une contrainte discrète : pour une contrainte faisant intervenir les connecteurs = ou ≠, il faut s'assurer que les deux attributs sont de même type discret; pour une contrainte faisant intervenir les connecteurs ∈ ou ∉, on vérifiera que l'ensemble de valeurs apparaissant en partie droite de la contrainte est inclus dans l'ensemble des valeurs possibles du type de l'attribut apparaissant en partie gauche.

3.2.5 - Les relations

Le formalisme des **relations** a été introduit pour permettre au concepteur de décrire simplement les positions relatives des objets. Par exemple, pour préciser dans la règle /1/ que *le TELEPHONE est posé sur le BUREAU*, le système d'acquisition offre la possibilité à l'utilisateur d'écrire la relation suivante, propre à la règle /1/ :

POSE_SUR (TELEPHONE, BUREAU)

Les relations sont simples d'emploi et évitent ainsi au concepteur l'écriture de contraintes continues traduisant cette information topologique. Cependant, dans un souci d'homogénéité des informations associées à une règle, une telle relation topologique ne sera pas mémorisée comme telle : elle est traduite, par le système d'acquisition, par l'ensemble de contraintes linéaires suivant :

XMAX (TELEPHONE) ≤ XMAX (BUREAU)
XMIN (TELEPHONE) ≤ XMIN (BUREAU)
YMAX (TELEPHONE) ≤ YMAX (BUREAU)
XMIN (TELEPHONE) ≤ YMIN (BUREAU)
ZMIN (TELEPHONE) = ZMAX (BUREAU)

Cet ensemble de contraintes n'est ni nécessaire, ni suffisant pour définir la relation *est posé sur* dans l'absolu. Il correspond aux cas rencontrés le plus fréquemment. Le concepteur est libre de compléter cette définition à son gré.

Il existe **onze** relations prédéfinies à la disposition du concepteur : elles sont décrites dans le tableau ci-après. Pour chacune d'entre elles, il indique le *nom* de la relation, le *nombre de paramètres* et la *nature* de chacun d'eux, dans l'ordre dans lequel ils doivent être fournis. La seconde colonne donne un exemple d'utilisation, explicité en colonne trois. La dernière colonne fournit l'ensemble des contraintes générées lors de l'écriture de l'exemple. Il est important de noter que l'ensemble de contraintes associées à une relation n'est qu'une des solutions possibles, choisie pour sa simplicité.Tout comme dans l'exemple de la relation POSE_SUR, ces ensembles ne sont parfois ni nécéssaires ni suffisant pour décrire les relations précisément.

De plus, on remarquera que, si la fonction première des relations était de décrire les positions des objets, leur utilisation a été étendue à des paramètres quelconques de type *point* ou *direction*. Dans ce cas, on parlera plutôt de *macro-contraintes* que de relations.

TABLEAU DES RELATIONS

POSE_SUR (F1, F2)	*F1 est posée sur F2*
	zmin (F1) = zmax (F2) xmax (F1) ≥ xmin (F2) + 1 xmin (F1) ≤ xmax (F2) + 1 ymax (F1) ≥ ymin (F2) + 1 ymin (F1) ≤ ymax (F2) + 1

Cet ensemble de contraintes n'est ni nécessaire, ni suffisant pour définir la relation dans l'absolu. Il est exact lorsque les deux formes sont convexes, et que leur intersection se limite à des arêtes ou à des faces.

A_DROITE (S, D, F1, F2)	*S est l'un des symboles = ≤ ≥* *F1 est située à droite de F2 à une distance =, ≤ ou ≥ à D*
	cas = ymin (F2) - ymax (F1) = D cas ≤ ymin (F2) - ymax (F1) ≥ 0 ymin (F2) - ymax (F1) ≤ D cas ≥ ymin (F2) - ymax (F1) ≥ D

A_GAUCHE (S, D, F1, F2)	*S est l'un des symboles = ≤ ≥* *F1 est située à gauche de F2 à une distance =, ≤ ou ≥ à D*
	cas = ymin (F2) - ymax (F1) = D cas ≤ ymin (F2) - ymax (F1) ≥ 0 ymin (F2) - ymax (F1) ≤ D cas ≥ ymin (F2) - ymax (F1) ≥ D

DEVANT (S, D, F1, F2)	*S est l'un des symboles = ≤ ≥* *F1 est située devant F2 à une distance =, ≤ ou ≥ à D* cas = xmin (F2) - xmax (F1) = D cas ≤ xmin (F2) - xmax (F1) ≥ 0 xmin (F2) - xmax (F1) ≤ D cas ≥ xmin (F2) - xmax (F1) ≥ D
DERRIERE (S, D, F1, F2)	*S est l'un des symboles = ≤ ≥* *F1 est située derrière F2 à une distance =, ≤ ou ≥ à D* cas = xmin (F2) - xmax (F1) = D cas ≤ xmin (F2) - xmax (F1) ≥ 0 xmin (F2) - xmax (F1) ≤ D cas ≥ xmin (F2) - xmax (F1) ≥ D
POINTS_EGAUX (P1, ... Pn)	*les points Pi sont confondus* x(P1) = x(P2) = ... = x(Pn) y(P1) = y(P2) = ... = y(Pn) z(P1) = z(P2) = ... = z(Pn)
VERTICAL (F)	*l'axe de F est vertical* teta (delta (F)) = 0
MEME_PLAN (F1, ..., Fn)	*les facettes F1, ..., Fn sont dans le même plan* plana (F1) = ... = plana (Fn) planb (F1) = ... = planb (Fn) planc (F1) = ... = planc (Fn)

COLLE_SUR (F1, F2)	*la facette F1 est collée sur la facette F2 : F1 et F2 sont dans le même plan et F2 est à l'intérieur de F1* MEME_PLAN (F1, F2) xmax (F1) ≤ xmax (F2) ymax (F1) ≤ ymax (F2) zmax (F1) ≤ zmax (F2) xmin (F1) ≥ xmin (F2) ymin (F1) ≥ ymin (F2) zmin (F1) ≥ zmin (F2)
PARALLELE(PLAN, F1, ..., Fn)	*les facettes Fi sont parallèles au plan XY, XZ ou YZ* cas XY plana (F1) = ... = plana (Fn) = 0 planb (F1) = ... = planb (Fn) = 0 cas YZ planb (F1) = ... = planb (Fn) = 0 planc (F1) = ... = planc (Fn) = 0 cas XZ plana (F1) = ... = plana (Fn) = 0 planc (F1) = ... = planc (Fn) = 0
POSE (V, F1, ..., Fn)	*les formes Fi sont à V unités de mesure du sol* zmin (F1) = ... = zmin (Fn) = V
ASSEMBLER (F1, F2, ..., Fn)	*la facette Fi est raccordée à la droite de la facette Fi-1* POINTS_EGAUX (P2 (Fi), P1 (Fi+1)) $1 \leq i \leq n-1$ POINTS_EGAUX (P3 (Fi), P4 (Fi+1)) POINTS_EGAUX (P2 (Fn), P1 (F1)) POINTS_EGAUX (P3 (Fn), P4 (F1))

Bien que relativement complet, cet ensemble de relations peut s'avérer insuffisant. Ces dernières ont en effet servi à construire les exemples traités. Nous n'avons pas cherché à recenser toutes les relations qui pourraient être indispensables pour la construction d'une grande variété de scènes. C'est la raison pour laquelle le système d'acquisition offre donc à l'utilisateur la possibilité d'introduire de nouvelles relations. Pour chaque relation introduite, le concepteur devra préciser :

- le **nom** de la relation (différent d'un nom de relation prédéfinie)
- le **nombre de paramètres** sur lesquels porte la relation
- la **nature** de chacun des paramètres : forme, primitive, attribut de type particulier
- les **contraintes** nécessaires à la traduction de la relation.

Par exemple, le concepteur peut introduire la relation suivante :

- nom de la relation : **ELOIGNE**
- deux paramètres *forme*
- les contraintes relatives à la relation sont :

$$\begin{aligned} xmax\_1 &\leq xmin\_2 + 40 \\ ymax\_1 &\leq ymin\_2 + 40 \\ zmax\_1 &\leq zmin\_2 + 40 \end{aligned}$$

où les attributs xmax_1 et xmin_2 représentent respectivement les coordonnées extrêmes sur Ox du premier et du second paramètre.

3.2.6 - Les super-primitives

Dans le but de simplifier le travail du concepteur, nous avons défini des **super-primitives**, qui sont, en fait, des assemblages de primitives. Ce sont des volumes simples comme la pyramide - notée **PYRAMIDE** - et le parallélépipède - noté **PARALLEPIPEDE**. La description de tels objets est faite par l'intermédiaire de règles, comme les autres objets. Ces règles sont connues du système MEPHISTO : l'utilisateur peut donc utiliser ces super-primitives comme les primitives de base.

Voici par exemple la règle de description de la pyramide :

PYRAMIDE → FACETTE3 FACETTE3 FACETTE3 FACETTE3 FACETTE4

(POINTS_EGAUX P1_FACETTE3_1 P1_FACETTE3_2)
(POINTS_EGAUX P1_FACETTE3_1 P1_FACETTE3_3)
(POINTS_EGAUX P1_FACETTE3_1 P1_FACETTE3_4)
(POINTS_EGAUX P2_FACETTE3_1 P3_FACETTE3_2)
(POINTS_EGAUX P2_FACETTE3_1 P1_FACETTE4)
(POINTS_EGAUX P3_FACETTE3_1 P2_FACETTE3_4)
(POINTS_EGAUX P3_FACETTE3_1 P4_FACETTE4)
(POINTS_EGAUX P2_FACETTE3_2 P3_FACETTE3_3)
(POINTS_EGAUX P2_FACETTE3_2 P2_FACETTE4)
(POINTS_EGAUX P2_FACETTE3_3 P3_FACETTE3_4)
(POINTS_EGAUX P2_FACETTE3_3 P3_FACETTE4)

La première relation indique que le point 1 de la première occurrence de FACETTE3 dans la règle est confondu avec le point 1 de la seconde occurrence de FACETTE3 dans la partie droite de la règle.

3.2.7 - Les contraintes prédéfinies

L'existence d'attributs prédéfinis nous a conduit à prédéfinir un ensemble de contraintes attachées aux primitives utilisées et aux règles du modèle. En effet, nous avons remarqué la *redondance* qui apparait dans la définition des attributs prédéfinis des primitives.

Par exemple, pour décrire un cylindre sans ambigüité, un point, une direction et un rayon suffisent. Donner à l'utilisateur la possibilité d'utiliser un autre point du cylindre, ainsi que les coordonnées extrêmes de celui-ci offre une plus grande souplesse pour la description d'un objet. La redondance apportée doit être **contrôlée** pour ne pas introduire d'incohérence.

Laisser cette vérification aux bons soins du concepteur enlève tout intérêt à l'existence des attributs prédéfinis, d'autant plus que la vérification doit être répétée pour chacune des primitives utilisées dans le modèle. C'est la raison pour laquelle nous avons introduit des **contraintes prédéfinies** qui explicitent le lien entre les différents attributs des primitives. Ces contraintes sont précisées çi-dessous.

Le cylindre

$$XMIN \leq X(POINT\_BASE) \leq XMAX$$
$$YMIN \leq Y(POINT\_BASE) \leq YMAX$$
$$ZMIN \leq Z(POINT\_BASE) \leq ZMAX$$
$$H = distance(POINT\_BASE, POINT\_SOMMET)$$

De plus, les coordonnées extrêmes du cylindre représentent les coordonnées des points intersections du plan de la base d'équation

$$A*X + B*Y + C*Z + D = 0 \quad \text{avec} \quad A = \sin(TETA) * \cos(PHI)$$
$$B = \sin(TETA) * \sin(PHI)$$
$$C = \cos(TETA)$$

et la sphère d'équation

$$X^2 + Y^2 + Z^2 = R^2$$

La sphère

XMIN = X (CENTRE) - R
YMIN = Y (CENTRE) - R
ZMIN = Z (CENTRE) - R

XMAX = X (CENTRE) + R
YMAX = Y (CENTRE) + R
ZMAX = Z (CENTRE) + R

La facette

$$PLANA^2 + PLANB^2 + PLANC^2 = 1$$

$XMIN = \text{Min } X(P_i)$ $\quad XMAX = \text{Max } X(P_i)$ $\quad \forall i \in [1..n]$
$YMIN = \text{Min } Y(P_i)$ $\quad YMAX = \text{Max } Y(P_i)$
$ZMIN = \text{Min } Z(P_i)$ $\quad ZMAX = \text{Max } Z(P_i)$

Le cône

$XMIN \leq X(POINT\_BASE) \leq XMAX$
$YMIN \leq Y(POINT\_BASE) \leq YMAX$
$ZMIN \leq Z(POINT\_BASE) \leq ZMAX$
H = distance (POINT_BASE, POINT_SOMMET)
RAYON = tangente (ALPHA) / H

De plus, les coordonnées extrêmes du cône représentent les coordonnées des points intersections du plan de la base d'équation

$A*X + B*Y + C*Z + D = 0$ avec $A = \sin(TETA) * \cos(PHI)$
$B = \sin(TETA) * \sin(PHI)$
$C = \cos(TETA)$

et la sphère d'équation

$$X^2 + Y^2 + Z^2 = R^2$$

3.2.8 - Définition formelle d'un modèle

Pour résumer, un modèle est défini par :

$\mathcal{P}$: l'ensemble des formes primitives

$\mathcal{N}$: l'ensemble des formes non primitives

$\mathcal{F}$: $\mathcal{N} \cup \mathcal{P}$, l'ensemble de toutes les formes

$\mathcal{R}$: l'ensemble des règles, chaque règle $r \in R$ étant un quadruplet (F_r, SF_r, RL_r, CT_r) où

F_r est la forme décrite par la règle r

SF_r est l'ensemble des sous-formes composant F_r, $SF_r \subset \mathcal{F}$

RL_r est l'ensemble des relations entre les formes {Fr} $\cup$ SFr

CT_r est l'ensemble des contraintes portant sur les attributs de la forme F_r et des sous-formes de SF_r.

3.2.9- Un exemple de modèle

Les ensembles de primitives et de formes non primitives sont donnés par :

$\mathcal{P}$ = { FACET3, FACET4, CONE, CYLINDRE, SPHERE, PYRAMIDE}

$\mathcal{N}$= { SCENE, MAISON, GARAGE, ARBRE1, BOUQUET, ARBRE2, BALLON, RTOIT, MURS, GMURS, VOLET, PORTE, FENETRE, TRONC, FEUILLES, SAPIN }

/1/ SCENE → MAISON GARAGE ARBRE1 BOUQUET

lg_MAISON ≤ 100
h_GARAGE ≤ h_MAISON
h_ARBRE1 ≥ h_MAISON

(DERRIERE ≥ 1 BOUQUET MAISON)
(POSE 0 MAISON ARBRE1 BOUQUET GARAGE)
(DERRIERE ≥ 50 MAISON GARAGE)

/2/ SCENE → MAISON BOUQUET ARBRE2

(POSE 0 MAISON BOUQUET ARBRE2)
(A_DROITE ≥ 60 MAISON ARBRE2)

/3/ SCENE → GARAGE BOUQUET BALLON

h_BOUQUET ≤ h_GARAGE
l_BOUQUET ≥ 55

(POSE 0 GARAGE BOUQUET BALLON)
(POSE_SUR BALLON GARAGE)
(A_DROITE ≥ 0 BOUQUET GARAGE)

/4/ SCENE → MAISON ARBRE1 BALLON

(POSE 0 MAISON ARBRE1 BALLON)
(DERRIERE ≤ 100 ARBRE1 MAISON)
(EGAL h_MAISON h_ARBRE1)

/5/ **MAISON → RTOIT MURS**

(POSE_SUR RTOIT MURS)

/6/ **MAISON → FACET4_TOIT MURS**

(POSE_SUR FACET4_TOIT MURS)
(PARALLELE XY FACET4_TOIT)

/7/ **RTOIT → PYRAMIDE**

/8/ **MURS → FACET4_DEV FACET4_DER FACET4_GCH FACET4_DT**
PORTE_1 PORTE_2 FENETRE_1 FENETRE_2

zmax_PORTE2 ≤ zmax_FENETRE1

(PARALLELE XZ FACET4_GCH FACET4_DT)
(PARALLELE YZ FACET4_DEV FACET4_DER)
(ASSEMBLER FACET4_DEV FACET4_DT FACET4_DER FACET4_GCH)
(COLLE_SUR PORTE_1 FACET4_DEV)
(COLLE_SUR PORTE_2 FACET4_DER)
(COLLE_SUR FENETRE_1 FACET4_DER)
(COLLE_SUR FENETRE_2 FACET4_GCH)
(EGAL zmax_FACET4_DEV zmax_FACET4_DER zmax_FACET4_DT
zmax_FACET4_GCH)
(EGAL zmax_FENETRE_1 zmax_FENETRE_2)
(EGAL zmin_PORTE_1 zmin_PORTE_2 zmin_MURS)

/9/ **MURS** → **FACET4_DEV FACET4_DER FACET4_DT FACET4_GCH PORTE FENETRE_1 FENETRE_2**

ymax_FENETRE_2 ≤ ymin_FENETRE_1 + 1

(PARALLELE XZ FACET4_GCH FACET4_DT)
(PARALLELE YZ FACET4_DEV FACET4_DER)
(COLLE_SUR PORTE FACET4_GCH)
(COLLE_SUR FENETRE_1 FACET4_DT)
(COLLE_SUR FENETRE_2 FACET4_DEV)
(ASSEMBLER FACET4_DEV FACET4_DT FACET4_DER FACET4_GCH)

(EGAL zmax_ FACET4_DEV zmax_FACET4_DER zmax_FACET4_DT zmax_FACET4_GCH)
(EGAL h_FENETRE_2 h_FENETRE_1)

/10/ **ARBRE1** → **TRONC FEUILLES**

h_FEUILLES ≥ 1.4 * h_TRONC

(VERTICAL TRONC)
(POSE_SUR FEUILLES TRONC)

/11/ **ARBRE2** → **TRONC FEUILLES**

h_FEUILLES ≥ 1.7 * h_TRONC

(VERTICAL TRONC)
(POSE_SUR FEUILLES TRONC)

/12/ **BOUQUET** → **SAPIN_1 SAPIN_2 SAPIN_3**

h_SAPIN_2 ≥ h_SAPIN_1 + 1
h_SAPIN_2 ≥ 1.3 * h_SAPIN_3

/13/ **BOUQUET** → **SAPIN**

/14/ SAPIN → TRONC CONE

h_CONE ≥ 2 * h_TRONC
alpha_CONE ≥ 12

(VERTICAL TRONC)
(POSE_SUR CONE TRONC)
(VERTICAL CONE)

/15/ ARBRE2 → CONE

h_ARBRE2 ≥ 11
alpha_CONE ≥ 10

(VERTICAL CONE)

/16/ TRONC → CYLINDRE

h_TRONC ≥ 3
h_TRONC ≤ 9

/17/ FEUILLES → SPHERE

r_SPHERE ≥ 15
r_SPHERE ≤ 40

/18/ GARAGE → GMURS FACET4_TOIT

(POSE_SUR FACET4_TOIT GMURS)
(POINTS_EGAUX P1_FACET4_TOIT P2_GMURS)
(POINTS_EGAUX P2_FACET4_TOIT P1_GMURS)
(POINTS_EGAUX P3_FACET4_TOIT P3_GMURS)
(POINTS_EGAUX P4_FACET4_TOIT P4_GMURS)

/19/ GMURS → FACET4_DEV FACET4_DER FACET4_DT FACET4_GCH PORTE

zmax_FACET4_DT ≤ zmax_FACET4_GCH + 1
h_PORTE ≥ h_GMURS - 13
(PARALLELE XZ FACET4_GCH FACET4_DT)
(PARALLELE YZ FACET4_DEV FACET4_DER)
(COLLE_SUR PORTE FACET4_DT)
(ASSEMBLER FACET4_DEV FACET4_DT FACET4_DER FACET4_GCH)
(POINTS_EGAUX P1_GMURS P2_FACET4_DER)
(POINTS_EGAUX P2_GMURS P1_FACET4_DER)
(POINTS_EGAUX P3_GMURS P2_FACET4_DEV)
(POINTS_EGAUX P4_GMURS P1_FACET4_DEV)
(EGAL zmax_FACET4_DEV zmax_FACET4_DER zmax_FACET4_GCH)

/20/ GMURS → FACET4_DER FACET4_DT FACET4_GCH FACET4_VAS

(PARALLELE XZ FACET4_GCH)
(PARALLELE YZ FACET4_DEV FACET4_DER)
(COLLE_SUR FACET4_VAS FACET4_DEV)
(ASSEMBLER FACET4_DER FACET4_GCH FACET4_DEV)
(POINTS_EGAUX P1_GMURS P2_FACET4_DER)
(POINTS_EGAUX P2_GMURS P1_FACET4_DER)
(POINTS_EGAUX P3_GMURS P2_FACET4_DEV)
(POINTS_EGAUX P4_GMURS P1_FACET4_DEV)
(EGAL zmax_FACET4_DEV zmax_FACET4_DER zmax_FACET4_GCH)

/21/ PORTE → FACET4

/22/ FENETRE → VOLET_1 VOLET_2 FACET4_VIT

(ASSEMBLER VOLET_1 FACET4_VIT VOLET_2)
(MEME_PLAN VOLET_1 VOLET_2 FACET4_VIT)
(POINTS_EGAUX P1_FENETRE P1_VOLET_1)
(POINTS_EGAUX P2_FENETRE P2_VOLET_2)
(POINTS_EGAUX P3_FENETRE P3_VOLET_2)
(POINTS_EGAUX P4_FENETRE P4_VOLET_1)

/23/BALLON → SPHERE

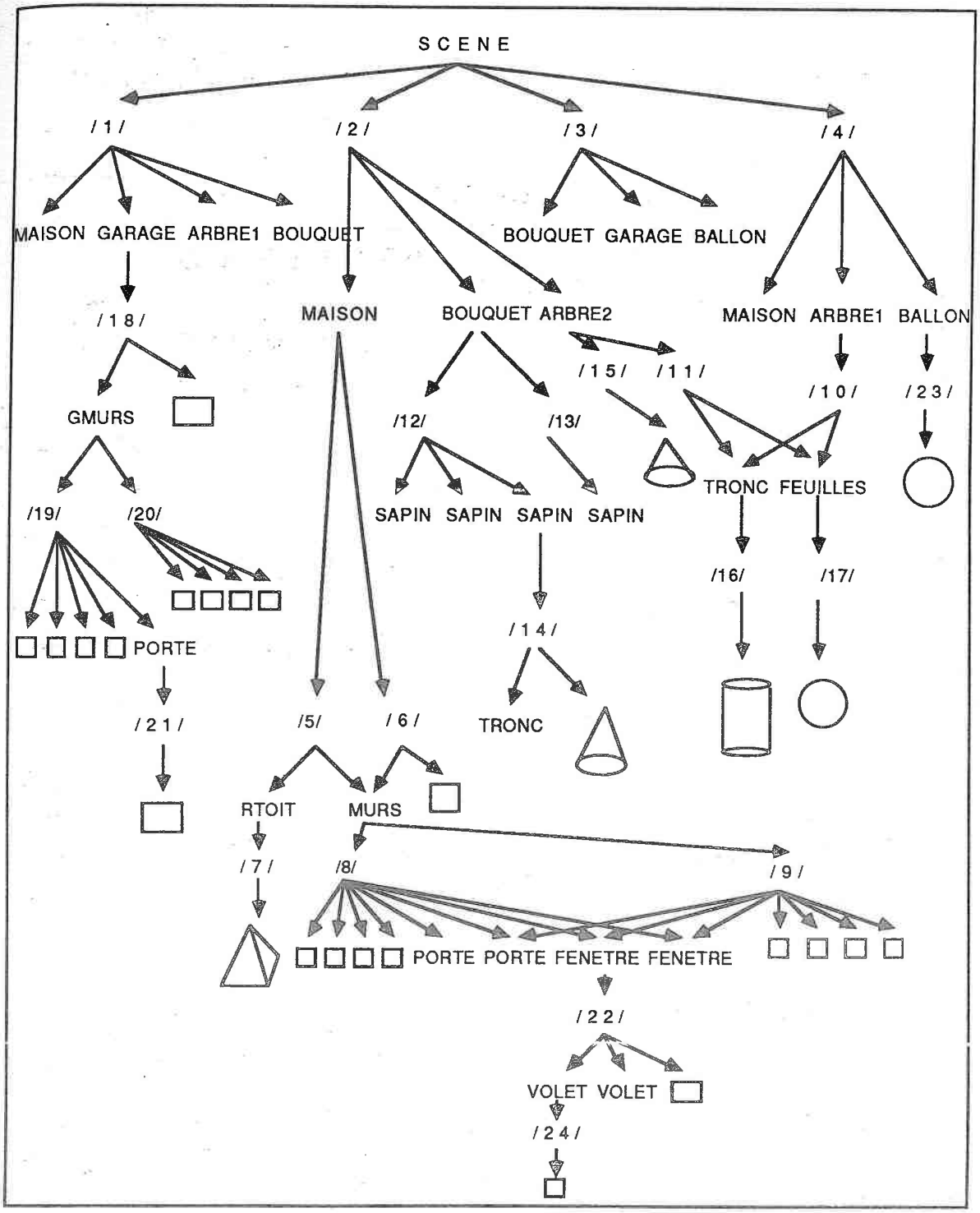
r_SPHERE ≤ 5

/24/VOLET → FACET4

GRAPHE ET/OU ASSOCIE AU MODELE

On notera que l'on peut assimiler un tel modèle à un **graphe ET / OU** avec contraintes. Chaque noeud OU est étiqueté par un nom de forme : ses successeurs éventuels sont des noeuds ET, qui représentent les différentes alternatives possibles sur la structure de cette forme. Chacun de ces successeurs est lui-même un noeud ET, étiqueté par un numéro de règle : les successeurs d'un noeud ET correspondent aux formes apparaissant dans la partie droite de la règle.

Le schéma de la page suivante donne le graphe ET/OU associé au modèle donné en exemple. Pour clarifier le dessin, les noms de primitives ont été remplacés par la figure correspondante. De plus, chaque nom de forme a été dupliqué partout où il apparait dans un membre droit de règle : cela évite d'avoir des arcs dans tous les sens.


SCENE
/1/
/2/
/3/
/4/
MAISON GARAGE ARBRE1 BOUQUET
BOUQUET GARAGE BALLON
MAISON ARBRE1 BALLON
/18/
MAISON
BOUQUET ARBRE2
/15/ /11/
/10/
/23/
GMURS
/12/
/13/
TRONC FEUILLES
/19/
/20/
SAPIN SAPIN SAPIN SAPIN
/16/
/17/
/14/
PORTE
/21/
/5/
/6/
TRONC
RTOIT
MURS
/7/
/8/
/9/
PORTE PORTE FENETRE FENETRE
/22/
VOLET VOLET
/24/

3.2.10 - CONCLUSION

Ce paragraphe était consacré à la définition précise du modèle utilisé par TRIDENT. Une telle représentation est basée sur le schéma de **représentation CSG** vue en détail dans le chapitre 1. En effet, dans notre modèle, les objets sont définis par composition de primitives bornées, le seul opérateur utilisé étant l'union.

Dans ces limites, nous avons vu qu'un tel schéma possède de nombreuses propriétés comme la facilité de validation et de création, la concision et la complétude. Afin d'étendre le domaine des objets représentables, nous avons généralisé la représentation CSG en introduisant des contraintes définissant les modalités d'unions des primitives. Malheureusement, il s'avère que, si elle augmente considérablement le domaine, la représentation obtenue perd toutes les propriétés citées précédemment. C'est la raison pour laquelle nous avons été amenés à développer un outil de conception de modèles nécessitant de la part de l'utilisateur le moins de travail possible, ainsi qu'un **outil de validation automatique** qui fait l'objet de cette thèse, et qui sera décrit en détail au chapitre 3.

3.3 - COMPILATION DU MODELE

Le modèle, tel qu'il a été décrit au paragraphe précédent, est d'une grande **complexité**, lorsque l'on s'attaque à la définition de scènes quasi-naturelles. Pour accélérer la phase d'interprétation, il s'avère indispensable d'exploiter les informations contenues implicitement dans le modèle.

Ces différentes informations sont rendues explicites par un **compilateur de description** : celui-ci est chargé d'extraire du modèle des ensembles d'informations qui sont vraies globalement sur le modèle, et qui sont utilisées par l'interpréteur pour faire des déductions et des recoupements. On retrouve ici les idées développées dans MIRABELLE [MASI83], pour l'analyse de dessins.

Nous allons expliciter l'ensemble des informations extraites du modèle par le compilateur, ainsi que l'usage qui en est fait. Précisons toutefois que la validité des formules qui vont suivre est montrée dans [CAMO85]. Par la suite, nous noterons **REG (f)** l'ensemble des règles décrivant la forme f. Rappelons également que SF_r désigne l'ensemble des sous-formes apparaissant en partie droite de la règle r.

La première information essentielle à connaitre est l'ensemble de toutes les **sous-formes** possibles d'une forme donnée. L'interpréteur, lors de l'analyse d'une forme, peut ainsi limiter son traitement à l'ensemble des composants possibles de cette forme. Cet ensemble s'obtient par fermeture transitive du graphe associé au modèle; c'est aussi la plus petite solution du système à point fixe suivant :

$$\textbf{SS-FORMES (f)} = \textbf{si } f \in P \textbf{ alors } \{f\} \textbf{ sinon } \{f\} \cup \left(\bigcup_{r \in REG(f)} \left(\bigcup_{y \in SF_r} \textbf{SS-FORMES (y)} \right) \right)$$

Prenons l'ensemble de règles de la figure 1. Les ensembles des sous-formes des objets SCENE et A sont donnés respectivement par :

SS-FORMES (SCENE) = $\mathcal{F}$, l'ensemble de toutes les formes possibles
SS-FORMES (A) = { A, a, b, c, d, e }

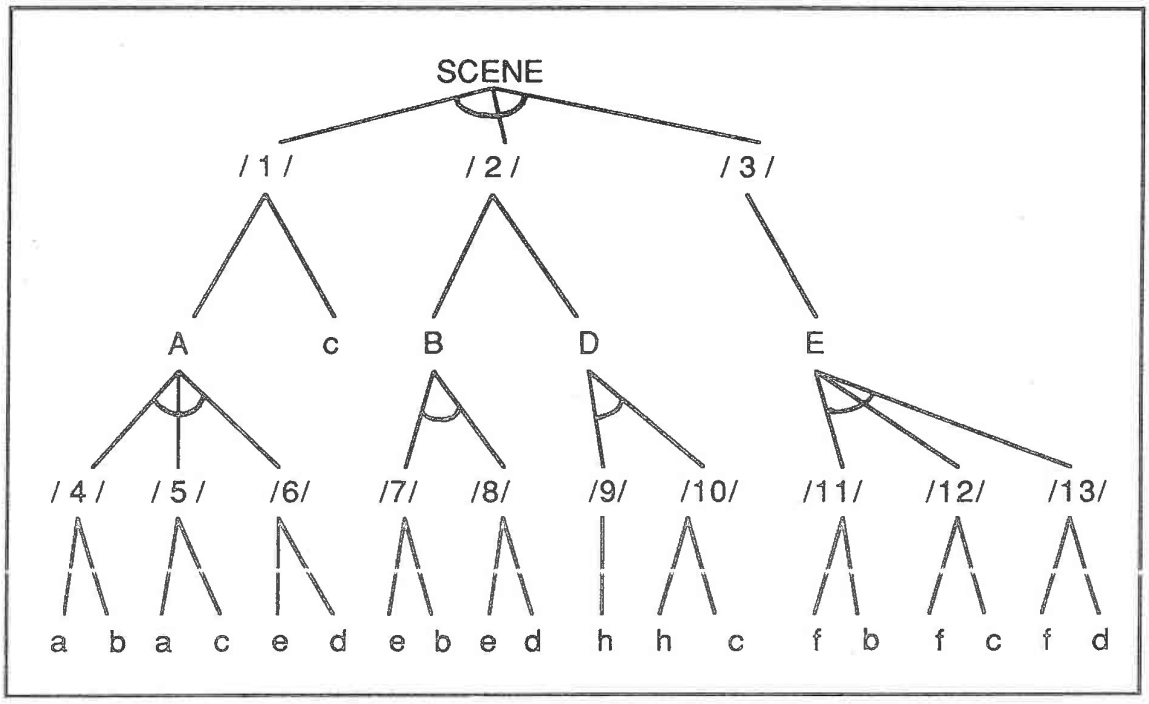
FIGURE 1

$\mathcal{P}$ = { a, b, c, d, e, f, h }

$\mathcal{N}$ = { SCENE, A, B, D, E }

/1/	SCENE	→	A c	**/8/**	B	→	e d
/2/	SCENE	→	B D	**/9/**	D	→	h
/3/	SCENE	→	E	**/10/**	D	→	h c
/4/	A	→	a b	**/11/**	E	→	f b
/5/	A	→	a c	**/12/**	E	→	f c
/6/	A	→	e d	**/13/**	E	→	f d
/7/	B	→	e b				

GRAPHE ASSOCIE AU MODELE



La relation **caractéristique** est également source d'informations : nous dirons que *l'objet O1 est caractéristique de l'objet O2* si tout chemin du graphe reliant l'axiome à O1 passe par O2. Autrement dit, cela signifie que la présence de O1 entraine la présence de O2. Le calcul de l'ensemble des formes caractéristiques d'une forme donnée f, noté **CARAC (f)**, nécessite la fermeture transitive d'un sous-graphe du modèle, obtenu en supprimant tous les noeuds étiquetés f. CARAC (f) est alors l'ensemble des formes qui ne sont plus sous-formes de l'axiome.

Dans l'exemple, a est caractéristique de A donc $a \in CARAC(A)$.

La relation réciproque, appelée **OBLIGATOIRE** donne l'ensemble des formes appartenant à une forme donnée, quelque soit la règle qui la décrit. Cela nous permet de déterminer toutes les sous-formes obligatoirement présentes lorsqu'une forme est présente. Cet ensemble, noté **OBLIG (f)**, est donné par la plus grande solution du système à point fixe suivant :

$$\mathbf{OBLIG}(f) = \begin{cases} f & \text{si } f \in P \\ \{f\} \cup \left(\bigcap_{r \in REG(f)} \left(\bigcup_{x \in SFr} OBLIG(x) \right) \right) & \text{sinon} \end{cases}$$

Dans l'exemple : e est obligatoire de C donc $e \in OBLIG(C)$.

TRIDENT n'utilise pas seulement ces deux propriétés de façon indépendante, car leur conjonction fournit un renseignement important : en effet, si l'objet O1 est à la fois caractéristique et obligatoire de l'objet O2, cela signifie que *O1 est sous-forme de O2 seulement*, et que, de plus, O1 est *obligatoirement présent* si O2 l'est. Une telle information est précieuse à l'interpréteur, car celui-çi sait alors que O1 doit être présent sur la scène (en faisant abstraction des parties cachées).

Dans l'exemple : f est caractéristique et obligatoire de E

$f \in CARAC(E)$ et $f \in OBLIG(E)$

La dernière information extraite par le compilateur de description concerne les objets dont la présence est exclue par la présence d'un objet donné. Un tel ensemble, noté **EXCLUS (f)** et appelé **ensemble d'exclusion de f**, permet en particulier de restreindre les hypothèses d'identification d'un objet donné, au fur et à mesure de l'interprétation. Si l'on appelle **POSS (f)** l'ensemble des formes appartenant à tous les chemins du graphe passant par f, l'ensemble EXCLUS (f) représente en fait le complémentaire de POSS (f) sur $\mathcal{F}$.

Le calcul de l'ensemble des possibles de f utilise des informations déjà obtenues par la relation SS-FORME :

$$\textbf{POSS (f)} = \textbf{SOUS-FORMES (f)} \cup \textbf{POSS1 (f)}$$

$$\textbf{POSS1 (f)} = \textbf{si f est l'axiome alors } \varnothing \textbf{ sinon } \bigcup_{r \in \text{REG2 (f)}} \; \bigcup_{f \in SF_r} (\cup \textbf{ SS-FORMES (f)}) \cup \textbf{POSS1 } (SF_r) \cup \{F_r\})$$

où REG2 (f) est l'ensemble des règles où f apparait en second membre.

Ainsi, on obtient :

$$\textbf{EXCLUES (f)} = \mathcal{F} \textbf{ POSS (f)}$$

Dans l'exemple : EXCLUS (h) = { A, E, a, f }

Le calcul des ensembles SS-FORMES, CARACT, OBLIG et EXCLUS a été détaillé sur les formes uniquement. Cependant, des calculs similaires peuvent être effectués sur les **règles** puisque l'interpréteur travaille alternativement sur les formes et les règles. L'extension de ces notions aux attributs est actuellement à l'étude : elle permettra de tenir compte par exemple des ensembles de formes exclues par la présence d'un objet rouge.

Pour finir, il faut préciser que tous ces calculs sont effectués par le compilateur, une fois pour toutes : ils pourront donc être utilisés directement à chaque essai d'interprétation. Bien évidemment, dans certains cas, l'interpréteur aura à sa disposition des résultats calculés inutilement. Face à cet inconvénient mineur, il est important de noter que les calculs sont très rapides - car ils se ramènent à des fermetures transitives de graphe - et que de cette façon, la structure de l'interpréteur est simplifiée.

4 - Le système de déduction ou interpréteur

L'interpréteur de TRIDENT a pour tâche d'effectuer la correspondance entre les indices visuels extraits d'images de la scène par le système de perception, et les indices objets contenus dans le modèle de l'univers. Il est donc chargé d'**étiqueter les régions** d'une image avec les noms des objets qu'elles représentent, en se servant des règles de description du modèle.

Le contenu d'une image représente ce qui est perçu d'une scène en trois dimensions, selon un point de vue particulier. Il y a donc obligatoirement des parties cachées. Ainsi, l'interpréteur doit fonctionnner par **émissions d'hypothèses** sur un ensemble de faits constatés par le bas-niveau et déduits de la structure de l'univers, à partir des informations fournies par le compilateur de description.

La stratégie de l'interpréteur de TRIDENT obéit à un cycle **prédiction-vérification-propagation**. La prédiction correspond à une phase **ascendante** : un objet OBJ déjà reconnu est supposé être une composante d'un objet plus élaboré. La vérification - ou phase descendante - consiste à localiser et identifier sur l'image les composantes encore inconnues de OBJ. Reste à résoudre l'indéterminisme soulevé par les deux problèmes suivants ; comment choisir l'objet servant à amorcer le processus ? En phase ascendante ou descendante, quelle règle choisir parmi celles possibles ? Ces deux problèmes sont partiellement résolus, en utilisant les informations extraites du modèle par le compilateur de description. En effet, on peut remarquer que les primitives caractéristiques et obligatoires de l'axiome fournissent des points de départ idéaux. Dans la même idée, les ensembles d'exclusion attachés aux objets déjà reconnus et aux règles déjà utilisées permettent de restreindre l'éventail des règles possibles en phases ascendante et descendante. Ces informations n'étant pas toujours suffisantes pour lever l'indéterminisme, l'interpréteur augmente sa puissance de raisonnement en menant en parallèle plusieurs analyses, fonctionnant de la manière décrite précedemment, en s'appuyant sur des ilôts de confiance.

Un tel système manipule une grande quantité d'informations qui évoluent constamment : ensembles d'exclusions, contraintes, domaine de variation des attributs, hypothèses d'identification,.... Par conséquent, une troisième phase, dite de **propagation**, est indispensable. A une étape donnée, elle consiste à intégrer dans la base de connaissances, les informations produites par les phases de prédiction et de vérification. Dans le cas où la propagation conduit à une inconsistance de l'une ou l'autre des connaissances, il y *échec* et donc remise en cause d'un choix. La propagation est essentielle : elle permet non seulement de renforcer ou de réfuter le plus rapidement possible les hypothèses émises par les diverses analyses menées en parallèle, mais aussi de restreindre au maximum les éventails de choix. La puissance de l'interpréteur en est augmentée.

Le lecteur interessé pourra consulter [MASI85] pour de plus amples détails.

CHAPITRE III

MEPHISTO : UN OUTIL DE VALIDATION DE SCENES TRI-DIMENSIONNELLES

1 - INTRODUCTION

Comme nous l'avons vu au chapitre précédent, **le modèle de l'univers** utilisé par le système d'interprétation de scènes TRIDENT est complexe. Rappelons que ce modèle est constitué d'un ensemble de règles, chacune d'elles donnant une *décomposition* possible d'une forme en sous-formes plus simples, ces sous-formes étant soit des primitives, soit définies par d'autres règles. A chaque règle du modèle est associé un ensemble de *contraintes* portant sur des attributs propres à chaque forme : elles permettent de préciser les caractéristiques des formes (dimensions, position, couleur, ...). Un tel ensemble de règles forme un graphe **ET/OU** avec contraintes.

Le modèle de description d'une scène réelle peut atteindre un haut degré de complexité lorsque les règles sont nombreuses. Un tel modèle n'est utilisable par TRIDENT que s'il est **cohérent**. Le problème de la validité de modèles géométriques est important, mais, dans les systèmes développés jusqu'à présent il était résolu par le concepteur lui-même. Cette solution fastidieuse et bien évidemment source d'erreurs n'est absolumment pas envisageable pour des modèles de la taille et de la complexité de ceux manipulés par TRIDENT. Il devient alors indispensable d'automatiser cette vérification de façon à alléger la tâche du concepteur.

Vérifier la consistance d'un tel modèle est un problème probablement **indécidable**, du fait de l'existence de contraintes quelconques portant sur des variables continues - dans le cas linéaire, le problème est exponentiel. Cependant, dans la plupart des cas *ordinaires*, l'expérience semble montrer que l'on peut se contenter de l'existence prouvée de quelques **instances** du modèle. En d'autres termes, montrer la consistance d'un modèle revient à *construire les chemins du graphe* ET/OU associé au modèle, pour lesquels les contraintes liées aux règles utilisées forment un ensemble *consistant*. Chacune de ces instances est donc une *arborescence* décrivant hiérarchiquement une scène particulière parmi la famille de scènes proposée par le modèle.

Ainsi, la tâche du système **MEPHISTO** consiste à extraire du modèle proposé par le concepteur, une ou plusieurs instances, s'il en existe. Au paragraphe 2, nous donnerons un exemple d'instance extraite d'un modèle par MEPHISTO.

Comme nous le verrons au paragraphe 4 où sera décrit précisément le fonctionnement de MEPHISTO, la validation de modèles pose deux types de problèmes essentiels rencontrés dans la construction d'un chemin du graphe ET/OU. Le premier problème concerne la mise en place de *stratégies* générales visant à minimiser le coût de la validation : des précisions seront données au paragraphe 3.1. A chaque étape de la construction d'un chemin, se pose également le problème de la *cohérence* d'un ensemble de contraintes : ce second point sera résolu de diverses manières dans le paragraphe 3.2.

2 - EXEMPLE

Prenons l'exemple incomplet suivant :

/1/ **SCENE** → **TABLE**
$0 < h(TABLE) \leq 100$

/2/ **SCENE** → **MAISON ARBRE**

/3/ **MAISON** → **MURS TOIT**
h (MAISON) = h (MURS) + h (TOIT)
$h(MURS) \geq h(TOIT)$
POSE_SUR (TOIT , MURS)

/4/ **MURS** → **PARALLELEPIPEDE**
h (MURS) = h (PARALLELEPIPEDE)
$h(MURS) \leq 5$

/5/ **TOIT** → **PRISME**
$h(TOIT) \geq 6$

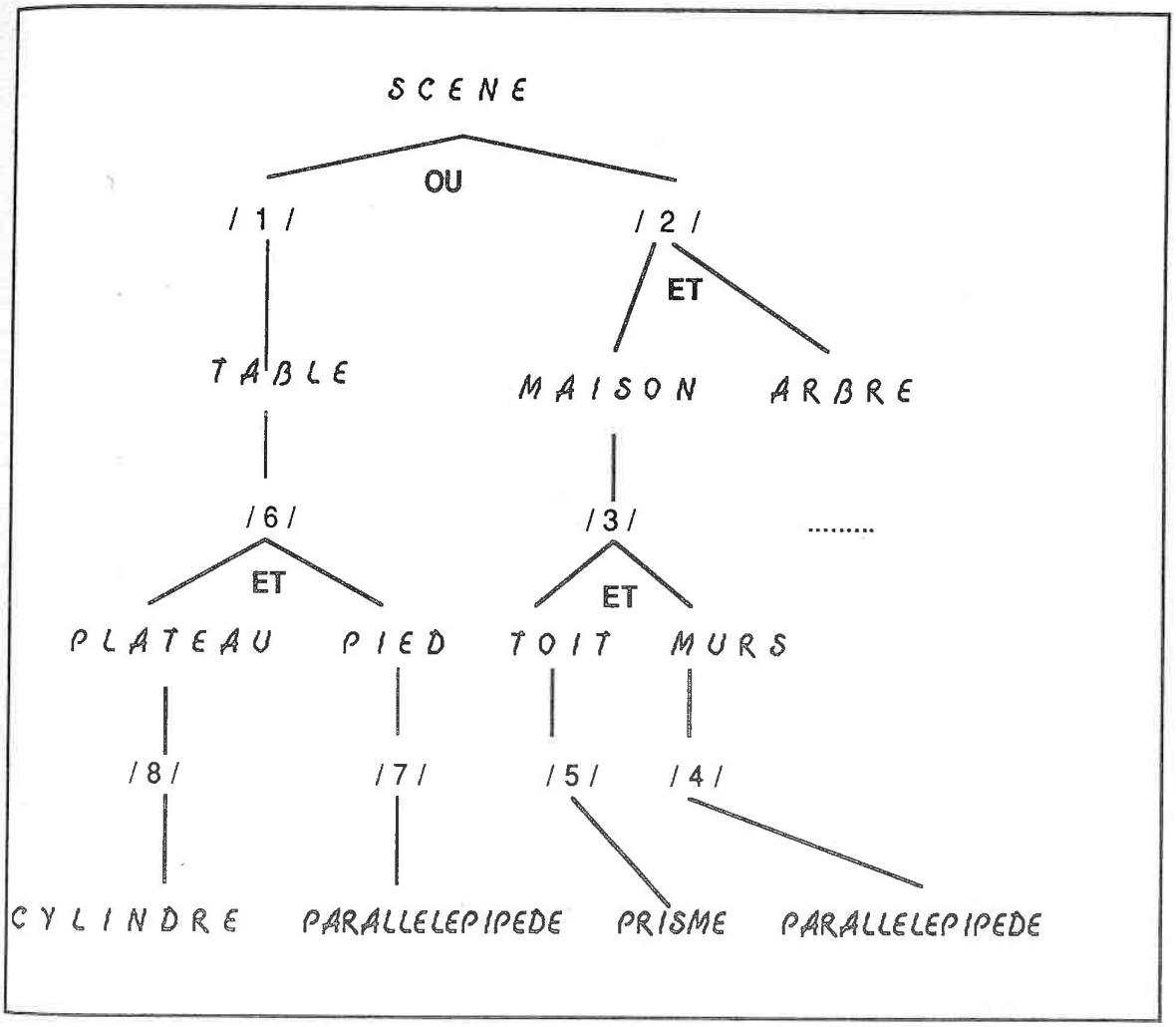
/6/ **TABLE** → **PLATEAU PIED**
h (TABLE) = h (PLATEAU) + h (PIED)

/7/ **PIED** → **PARALLELEPIPEDE**
h (PIED) = h (PARALLELEPIPEDE)
h (PARALLELEPIPEDE) ≥ 90

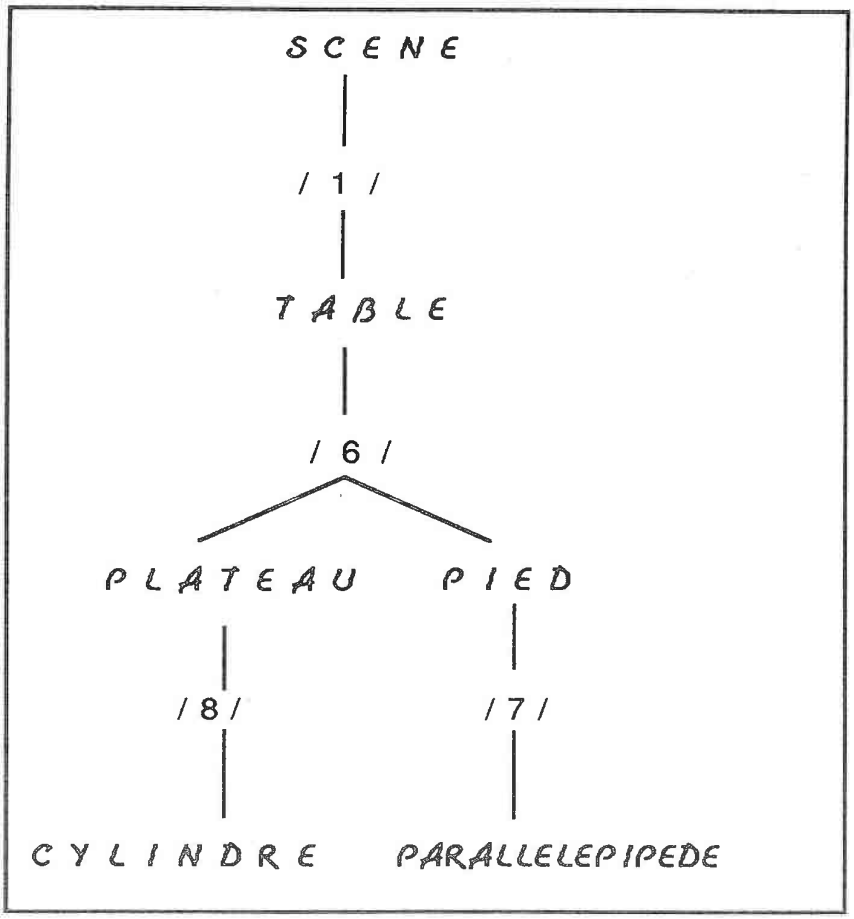
/8/ **PLATEAU** → **CYLINDRE**
h (PLATEAU) = h (CYLINDRE)

/9/ **PLATEAU** → **PARALLELEPIPEDE**
h (PLATEAU) = h (PARALLELEPIPEDE)

Ce modèle est représenté par le graphe ET/OU suivant, SCENE étant le point d'entrée de ce graphe.



Une instance de ce modèle satisfaisant les contraintes peut être :



AVEC h (TABLE) = 100

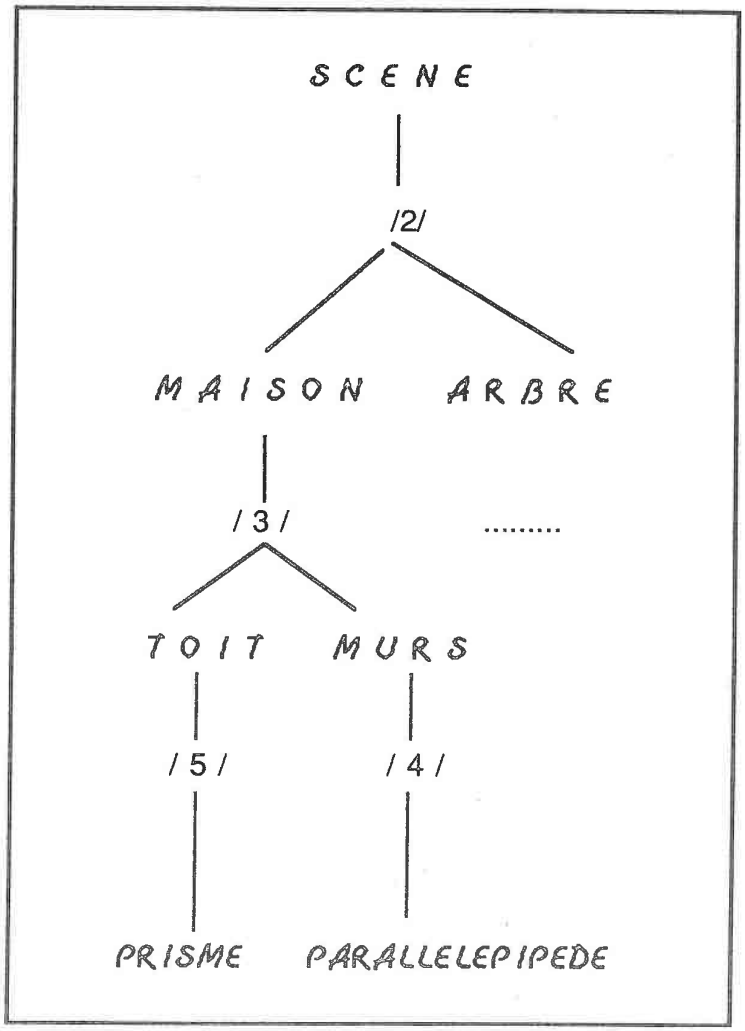
h (PLATEAU) = 10

h (PIED) = 90

h (CYLINDRE) = 10

h (PARALLELEPIPEDE) = 90

Lors de la construction de cette instance particulière, le système MEPHISTO constate une **inconsistance partielle** et la signale à l'utilisateur. En effet, d'après les données fournies par le modèle, il est impossible de construire une scène partiellement définie par l'arbre suivant :



Ceci est du à **l'incohérence** d'un sous-ensemble des contraintes associées à ce chemin, c'est-à-dire des contraintes des règles /2/, /3/, /4/, et /5/. L'*ensemble de contraintes inconsistant* est le suivant :

$$h\ (MURS) \geq h\ (TOIT)$$
$$h\ (MAISON) = h\ (MURS) + h\ (TOIT)$$
$$h\ (MURS) \leq 5$$
$$h\ (TOIT) \geq 6$$

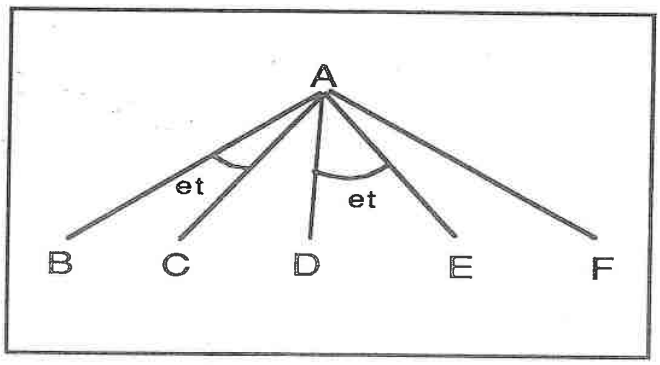
Le système MEPHISTO est alors en mesure de préciser au concepteur une des sources possibles de l'erreur détectée : ici, il signale que l'inconsistance a pu être provoquée par l'attribut h (MURS). Ceci permet d'orienter l'utilisateur dans la correction du modèle, puisqu'il sait alors que l'erreur provient des contraintes portant sur l'attribut h (MURS), ainsi que sur tout autre attribut en relation avec celui-çi.

3 - STRATEGIES DE RESOLUTION D'UN GRAPHE ET/OU

3.1 - GRAPHE ET/OU

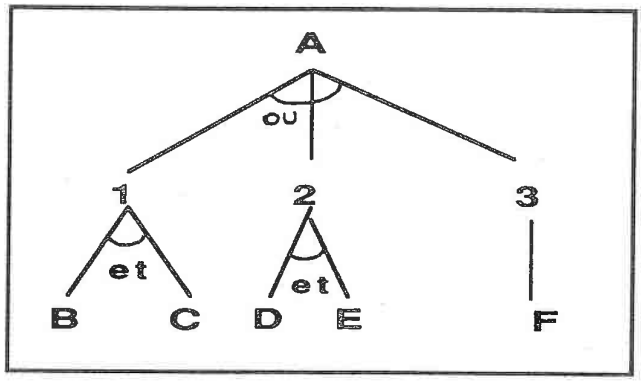
La représentation par **réduction** est une méthode générale de résolution de problèmes en intelligence artificielle [WINS79] [NILS71]. Cette technique très répandue consiste à décomposer le problème initial en sous-problèmes, de même pour chacun des sous-problèmes, jusqu'à ce que le problème se réduise à un ensemble de problèmes triviaux primitifs. Les exemples d'application sont abondants dans la littérature : citons tout d'abord les jeux comme les échecs, le go, le jeu de dame mais également, dans le domaine des mathématiques, le problème de l'intégration symbolique ou encore celui de la démonstration de théorèmes en géométrie plane [MART73] [MART78].

Habituellement, on représente la décomposition d'un problème en sous-problèmes par une **structure de graphe**. Supposons que le problème A puisse être résolu soit par les sous-problèmes B et C, soit par D et F, soit par F. Une telle décomposition est montrée par la figure suivante :



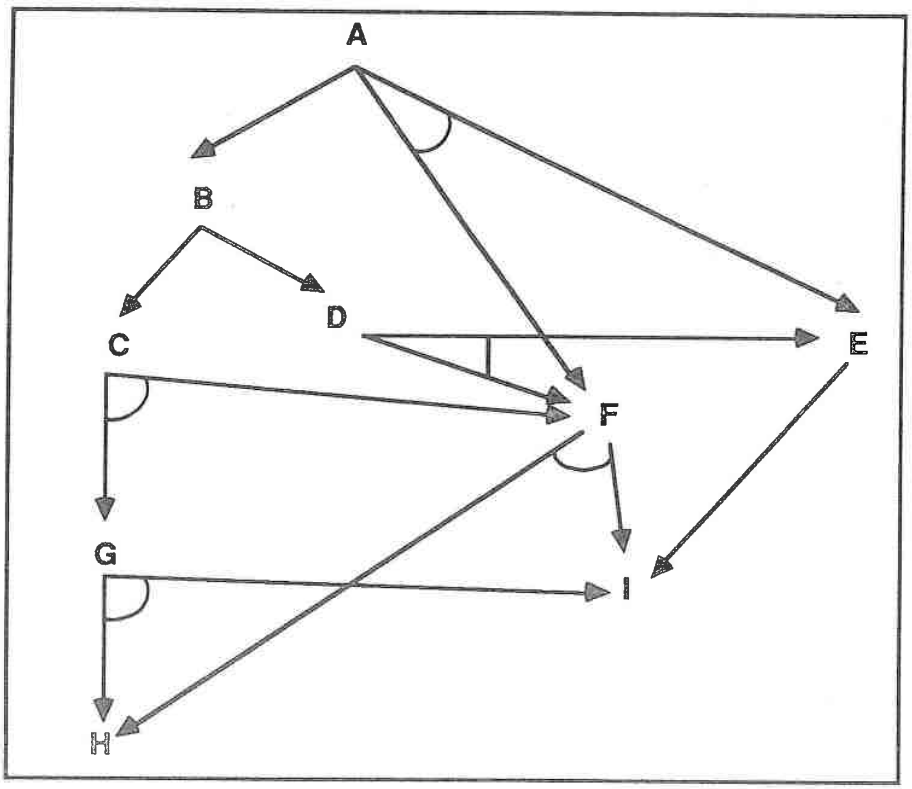
Dans ce graphe, chaque noeud est étiqueté par le problème qu'il représente. Le noeud A a trois ensembles de successeurs : {B C}, {D E} et {F}.

Dans un souci de simplicité de traitement, on préfère habituellement introduire un nouveau type de noeud, correspondant à un ensemble de sous-problèmes. Ici, on introduit donc trois noeuds notés 1, 2 et 3 et l'on obtient le graphe suivant :



Dans ce graphe, on fait la distinction entre deux types de noeuds. Un noeud dit de **type OU** correspond à un problème à résoudre, et ses successeurs représentent les diverses façons de le résoudre : A peut être résolu par 1, 2 ou 3. De façon symétrique, pour résoudre le noeud 1, il est nécessaire d'avoir résolu les noeuds B et C : le noeud 1 est un noeud dit de **type ET**. L'opérateur ET est appelé généralement **opérateur de réduction.** Une telle structure est appelée **graphe ET/OU**. L'un des noeuds OU, appelé **point d'entrée** correspond au problème initial. C'est le seul noeud du graphe qui ne représente pas un sous-problème d'un autre problème. Inversement, les noeuds OU qui ne sont pas décomposés représentent des problèmes primitifs : ils sont appelés les **feuilles** du graphes, ou les noeuds **terminaux**.

Voici un exemple de graphe ET/OU plus complet. Le point d'entrée de ce graphe est le noeud A. L'ensemble des feuilles est { H, I }.



Un graphe ET/OU est défini par $\gamma = (\mathcal{O}, \mathcal{F}, \mathcal{E}, \mathcal{X}, \mathcal{A})$ où

- $\mathcal{O}$ est l'ensemble des noeuds de type OU
- $\mathcal{F}$ est l'ensemble des noeuds feuilles
- $\mathcal{E}$ est l'ensemble des noeuds de type ET
- $\mathcal{X}$ est le point d'entrée du graphe , supposé unique $\mathcal{X} \in \mathcal{O}$
- $\mathcal{A}$ est l'ensemble des arcs orientés reliant les noeuds

$$\mathcal{A} \subset (\mathcal{O} \cup \mathcal{F}) \times \mathcal{E} \cup \mathcal{E} \times (\mathcal{O} \cup \mathcal{F})$$

Hall [HALL73] a montré l'équivalence d'une telle représentation et des grammaires à contexte libre. Une telle grammaire serait définie par :

$$G = (\mathcal{O}, \mathcal{F}, \mathcal{R}, \mathcal{X})$$

où $\mathcal{R}$ est l'ensemble des règles de la forme $r : \alpha \rightarrow \beta$ telle que

- à r correspond un élément unique de $\mathcal{E}$
- il existe un arc de $\mathcal{A}$ orienté de α à r
- $\forall b \in \mathcal{O} \cup \mathcal{F}$ dans β, il existe un arc de $\mathcal{A}$ orienté de r à b

Pour l'exemple précédent, la grammaire équivalente est définie par :

$$G = (\{ A, B, C, D, E, F, G \} , \{ H, I \} , \mathcal{R} , A)$$

avec

$$\begin{aligned}
A &\rightarrow B \mid F\ E \\
B &\rightarrow C \mid D \\
C &\rightarrow G\ F \\
D &\rightarrow E\ F \\
E &\rightarrow F \mid I \\
F &\rightarrow H\ I \\
G &\rightarrow H\ I
\end{aligned}$$

Résoudre un graphe ET/OU signifie rechercher un chemin dans ce graphe, à partir du point d'entrée : cela consiste à fixer une *décomposition possible* de chaque problème en sous-problèmes. Le chemin obtenu est un **sous-graphe** donnant les diverses décompositions choisies. Les algorithmes de recherche de chemin dans des graphes ET/OU sont abondants dans la littérature : ce sont des variations des algorithmes fondamentaux basés sur les recherches en profondeur et en largeur d'abord.

De même que dans les graphes ordinaires, on associe souvent un coût aux feuilles et aux arcs. Si **C(N,M)** est le coût de l'arc reliant le noeud N de type OU à l'un de ses successeurs M de type ET, et **COUT (M)** est le coût associé au noeud feuille M, la fonction de calcul du coût d'un noeud quelconque se définit récursivement par :

$$COUT\ (N) = C\ (N, M) + COUT\ (M)$$

si N est un noeud de type OU et M l'un de ses successeurs

$$COUT\ (N) = max\ (\ C\ (N, M_i\ ) + COUT\ (\ M_i\ )\ )$$

si N est un noeud de type ET et les M_i ses successeurs

Les coûts associés aux arcs et aux feuilles sont rarement définies avec précision. Dans la pratique, on est amené à utiliser une approximation de ces coûts. Les algorithmes de recherche de chemin optimaux, donc de coûts minimaux sont nombreux [WINS79] [NILS71] [NILS80].

3.2 - UNE STRATEGIE DE RECHERCHE DE CHEMIN

Dans le cadre du système MEPHISTO, nous nous sommes intéressés plus généralement à des **graphes ET/OU avec contraintes** : dans de tels graphes, à chaque arc reliant un noeud ET à un noeud OU est associé un ensemble de *contraintes quelconques*. Dans ce cas, le problème de la résolution est plus complexe, puisqu'il s'agit de construire un chemin respectant les contraintes imposées par les arcs empruntés. Nous développons ici une stratégie de construction de chemin, visant une minimisation du coût de la résolution.

On choisit de construire un tel chemin pas à pas, une étape consistant à compléter une des extrémités du chemin. Le choix de l'extrémité est un choix dit de **type ET**, puisqu'il faudra en effet compléter toutes les extrémités. Laurière [LAUR76] propose une stratégie qui consiste à compléter l'extrémité ayant le moins de successeurs : autrement dit, à chaque étape, on résoud le problème dont le nombre de décompositions possibles est le plus faible.

De façon symétrique, une fois l'extrémité choisie, le choix de la décomposition à appliquer est un choix dit de **type OU**, puisque seule une de celles possibles sera choisie. La stratégie du **meilleur d'abord** est la plus populaire : elle consiste à appliquer la décomposition dont l'espérance de succès est la plus grande.

La stratégie proposée ici combine les deux stratégies précédemment décrites. Elle suppose l'existence de deux fonctions définies pour chacun des noeuds OU et ET :

S : noeud $\rightarrow [0..1]$	est l'*espérance de succès* de résolution du noeud donné
C : noeud $\rightarrow \mathbb{N}$	est l'*estimation du coût* de la résolution du noeud donné.

3.2.1 - Cas d'un choix de type OU

Pour simplifier l'exposé, supposons tout d'abord que la résolution d'un problème conduit seulement à la résolution de l'un des deux **sous-problèmes N_1 et N_2**. Supposons également que les fonctions S et C sont des estimations exactes des espérances de succès et de coût. Nous noterons :

$C_i = C(N_i)$, $S_i = S(N_i)$ $(i = 1, 2)$

Le coût de la stratégie **choisir d'abord N_1, puis N_2 en cas d'échec de N_1** est donné par :

$$OU(N_1, N_2) = C_1 + (1 - S_1) * C_2$$

C_1 : coût de N_1 ; $(1 - S_1) * C_2$: coût de N_2 en cas d'échec de N_1

De façon symétrique, le coût de la stratégie **choisir d'abord N_2, puis N_1 en cas d'échec de N_2** est donné par :

$$OU(N_2, N_1) = C_2 + (1 - S_2) * C_1$$

Si $OU(N_1, N_2) < OU(N_2, N_1)$, on décide de mettre en oeuvre la première stratégie, sinon ce sera la seconde.

Cependant, en général, un choix de type OU n'est pas limité à deux noeuds mais porte sur un nombre de noeuds quelconque : dans ce cas, on effectue un tri sur les noeuds, en tenant compte du résultat suivant :

proposition

La relation induite sur les noeuds par la fonction OU et définie par

$$N_1 < N_2 \quad \Leftrightarrow \quad OU\,(N_1, N_2) < OU\,(N_2, N_1)$$

est un préordre total.

Démonstration

* la relation est *réflexive*, puisque

$$N_1 < N_1 \Leftrightarrow OU\,(N_1, N_1) < OU\,(N_1, N_1)$$

* la *transitivité* demande une démonstration; montrons que si

	$OU\,(N_1, N_2) < OU\,(N_2, N_1)$	[1]
et	$OU\,(N_2, N_3) < OU\,(N_3, N_2)$	[2]
alors	$OU\,(N_1, N_3) < OU\,(N_3, N_1)$	[3]

[1] et [2] se réécrivent de la façon suivante :

$$C_1 + (1 - S_1) * C_2 < C_2 + (1 - S_2) * C_1 \quad [1']$$
$$C_2 + (1 - S_2) * C_3 < C_3 + (1 - S_3) * C_2 \quad [2']$$

ou encore :

$$C_2 / S_2 > C_1 / S_1 \quad [1'']$$
$$C_3 / S_3 > C_2 / S_2 \quad [2'']$$

[1''] et [2''] impliquent

$$C_3 / S_3 > C_1 / S_1 \qquad [3'']$$

[3''] se réécrit ainsi :

$$C_1 + (1 - S_1) * C_3 < C_3 + (1 - S_3) * C_1 \qquad [3']$$

et [3'] est équivalent à [3].

CQFD

* le préordre est **total** car il revient à comparer les rapports réels de la forme C/S comme nous l'avons fait dans la démonstration de la transitivité.

Ainsi, lorsque se pose un problème de choix OU entre n noeuds, le préordre indiqué permet de trier les noeuds et fournit ainsi l'ordre dans lequel ils doivent être examinés.

3.2.2 - Cas d'un choix de type ET

Supposons que la résolution d'un problème conduise seulement à la résolution de **deux sous-problèmes, notés N1 et N2.** L'ordre d'examen a encore une importance puisque l'échec de la résolution de l'un des deux permettra de conclure à l'insolubilité du problème posé, sans examen de l'autre sous-problème. Ici encore, on est donc amené à établir le coût de la stratégie qui consiste à **résoudre d'abord N1, puis N2 seulement en cas de succès**; il est donné par :

$$ET(N_1, N_2) = C_1 + S_1 * C_2$$

C_1 : coût de N_1 ; C_2 : coût de N_2, en cas de succès de N_1

De façon similaire à ce que nous avons vu pour la fonction OU, la fonction ET introduit un **préordre total** qui permet de trier les noeuds dans l'ordre d'examen pour lequel le coût total est le plus faible.

3.3 - Comparaisons avec les méthodes existantes

Dans les algorithmes habituellement utilisés, n'interviennent que rarement les deux fonctions coût et espérance que nous avons introduites. Il est tout de même possible de se ramener à des stratégies existantes en supposant tout d'abord que la fonction coût est constante. La méthode proposée revient alors à :

* la stratégie du *meilleur d'abord*, pour un choix de type OU
* la stratégie du plus *mauvais d'abord*, pour un choix de type ET.

On remarque ainsi que la stratégie populaire du meilleur d'abord n'a de sens que pour un choix de type OU, et à condition que les coûts des diverses possibilités soient égaux, sinon comparables.

De façon symétrique, si l'on travaille à espérances de succès constantes, la méthode proposée revient à traiter les noeuds par ordre de *coûts croissants*, que ce soit pour les noeuds ET ou bien les noeuds OU. Dans les problèmes combinatoires, on considère raisonnablement que le coût de résolution d'un problème est proportionnel au nombre de solutions possibles pour le résoudre. A espérances constantes, notre méthode revient donc à explorer en premier les noeuds ayant le moins de possibilités : on retrouve alors la stratégie de Laurière.

Si notre méthode est comparable aux méthodes existantes à coût ou à espérances constantes, elle offre cependant l'avantage de combiner l'utilisation des deux fonctions, ce qui dans certains cas, diminue considérablement le temps de recherche d'un chemin.

Illustrons les différentes méthodes par un exemple numérique simple. On se place dans le cas d'un choix de type OU, portant sur les noeuds N_1, N_2 et N_3. Pour chacun d'eux, on définit les fonctions coût et espérance de succès de la façon suivante : .

$C_1 = C(N_1) = 60$ $\quad$ $S_1 = S(N_1) = 0.4$
$C_2 = C(N_2) = 30$ $\quad$ $S_2 = S(N_2) = 0.5$
$C_3 = C(N_3) = 10$ $\quad$ $S_3 = S(N_3) = 0.2$

La stratégie dite du *meilleur d'abord* conduit à traiter les noeuds dans l'ordre des espérances décroissantes, c'est-à-dire N_2, N_1, N_3. Le coût de cette stratégie est donnée par

$$\text{Coût}(N_2, N_1, N_3) = C_2 + C_1 * (1 - S_2) + C_3 * (1 - S_2) * (1 - S_1) = 62$$

alors que la stratégie proposée conduit à traiter les noeuds dans l'ordre croissant des rapports coût / espérance, c'est-à-dire N_3, N_2, N_1, puisque

$$C_3 / S_3 = 50 \quad C_2 / S_2 = 60 \quad C_1 / S_1 = 150$$

Le coût de cette stratégie est donnée par

$$\text{Coût}(N_3, N_2, N_1) = C_3 + C_2 * (1 - S_3) + C_1 * (1 - S_3) * (1 - S_2) = 58$$

La pire des combinaisons possibles, donnée par la comparaison des coûts de toutes les solutions, est N_1, N_2 et N_3 : le coût de cette stratégie est de 81.

Les gains obtenus avec notre méthode peuvent être très variables selon les cas. Cependant, il faut noter que cette stratégie est utilisée à chaque étape : c'est l'accumulation des différents gains qui va permettre de minimiser le coût global de la construction d'un chemin, processus qui est fondamentalement exponentiel.

3.4 - LIMITES DE CETTE METHODE

La stratégie proposée suppose l'existence de deux fonctions exactes, *coût* et *espérance* de *succès*, associées à chaque noeud du graphe ET/OU. La définition précise de telles fonctions est parfois difficile à obtenir, voire même impossible. Ainsi, l'approximation des coûts et des espérances peut remettre en cause l'intérêt de la stratégie. En effet, si dans l'absolu les gains sont évidents, ils peuvent devenir nuls si les estimations sont trop éloignées des fonctions exactes.

A cette difficulté de définition des fonctions C et S, s'ajoute parfois un autre problème complexe, du à la *dépendance* existant entre les différents problèmes à résoudre. Par exemple, dans le problème bien connu du placement de reines sur un échiquier, le fait de fixer la position de l'une d'elles influe sur l'espace des solutions des reines restant à placer. De même, dans le domaine de l'interprétation de scènes, la découverte de la présence d'une forme sur une image peut interdire l'existence d'autres formes sur celle-ci.

Donc, dans certains cas, les fonctions *coût et espérance de succès* introduites précédemment ne sont pas des constantes attachées à chaque noeud, mais dépendent du contexte, c'est-à-dire des problèmes déjà résolus.

Dans la pratique, cela signifie que la méthode proposée ne peut être qu'un guide auquel il est nécessaire d'intégrer des heuristiques propres au problème à résoudre. Nous verrons une application de cette méthode dans le chapitre suivant.

4 - RESOLUTION DE SYSTEMES DE CONTRAINTES LINEAIRES CONTINUES

< Règle : quand un carré et autre terme de l'inconnue est engagé dans le reste, alors, après avoir multiplié les deux cotés de l'équation par une quantité supposée, on doit leur ajouter quelque chose de sorte que le côté puisse donner une racine carrée. Egaler encore la racine carrée du nombre absolu à la racine carrée de l'inconnue; on trouve la valeur de l'inconnue à partir de cette équation. Si l'on ne peut parvenir ainsi à la solution, cette valeur doit être obtenue grâce à la propre ingénuité du calculateur > BHASKARA La résolution d'équations, XIII siècle;

Une **contrainte linéaire continue** est une équation ou une inéquation portant sur des variables prenant leurs valeurs dans $\mathbb{R}$. Elle a la forme suivante :

$$expression\_linéaire \quad \begin{matrix} = \\ \leq \\ \geq \end{matrix} \quad expression\_linéaire$$

où chaque expression linéaire est une somme de monômes.

Résoudre un ensemble de contraintes consiste à déterminer le domaine de variation de chacune des variables concernées. On notera $\mathcal{V}$ l'ensemble des variables et $\mathcal{S}$ le système à résoudre. Dans un premier temps, nous allons nous limiter aux inéquations et présenter deux méthodes, l'une développée par BLEDSOE [BLED75], et l'autre par MOHR [MOHR82]. Comme nous le verrons par la suite, le premier algorithme traite des ensembles d'inégalités reliées par des ET, alors que le second fournit une méthode de résolution de systèmes ET/OU. Cependant, aucune de ces deux méthodes ne tenant compte des égalités, une contrainte portant sur le connecteur = doit être transformée en deux inégalités équivalentes. Par exemple, l'égalité

$$X + Y + 2*Z = D - 3$$

devient

$$X + Y + 2*Z \leq D - 3$$
$$X + Y + 2*Z \geq D - 3$$

Ce n'est que sous cette forme que l'égalité pourra être traitée par l'un des deux algorithmes.

Cependant, la substitution d'une égalité par deux inégalités augmente la complexité du système à résoudre et entraine une perte d'information. En effet, l'égalité

$$X = 2 * Y$$

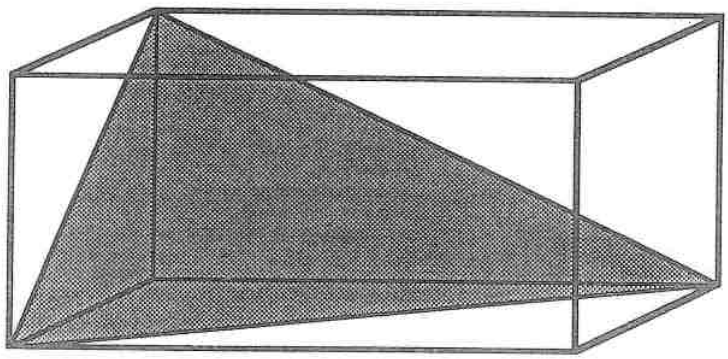
permet de supprimer une variable, puisque partout, on peut remplacer X par 2*Y. Le nombre de variables à traiter est ainsi plus faible. Au paragraphe 3.2.3, nous proposons une méthode permettant de limiter la complexité de la résolution d'un système de contraintes quelconques, en simplifiant les inégalités par les égalités : en moyenne, plus le nombre d'égalités sera important, plus la complexité du système à résoudre sera faible.

4.1 - METHODE SUP-INF

La méthode SUP-INF développée par BLEDSOE [BLED75] détermine la validité ou l'invalidité d'un ensemble d'inéquations linéaires, en calculant les **bornes inférieure et supérieure** des variables. Lorsque le système est consistant, SHOSTACK a démontré que cette méthode fournit le **domaine exact** de variation de chaque variable [SHOS77].

L'ensemble des solutions des inéquations est un convexe de $\mathbb{R}^n$ (n est le nombre de variables). Cette méthode permet de calculer le domaine de variation de chacune des variables de manière optimale. Géométriquement, elle construit le parallélépipède rectangle dont les faces de dimension n-1 sont parallèles aux hyperplans du repère et s'appuient sur le convexe des solutions.

Remarquons que l'existence de ce parallélépipède garantit l'existence d'une solution, mais que malheureusement ne permet pas d'en exhiber une. En particulier, l'intuition géométrique peut laisser croire que le centre du parallélépipède est toujours dans le convexe des solutions. Ceci est faux dès la dimension 3, comme le prouve le dessin ci-dessous.



Cet algorithme se décompose en deux étapes :

1 - **A partir de S, pour chaque variable v de $\mathcal{V}$, on détermine la fonction I(v) des expressions linéaires minorant v et la fonction S(v) des expressions linéaires majorant v. On a donc $I(v) \leq v \leq S(v)$.**

2 - **Les bornes inférieure et supérieure de chaque variable de $\mathcal{V}$ sont calculées à l'aide de deux fonctions récursives INF et SUP. Si pour l'une des variables l'intervalle obtenu est vide, le système d'inéquations est inconsistant.**

La fonction **SUP** a trois arguments :

- J : expression normalisée dont on veut calculer la borne supérieure; elle est sous l'une des formes suivantes :

  -- une constante numérique $\in \mathbb{R}$, par exemple **3** ou **65.7**

  -- l'infini, noté $-\infty$ et $+\infty$

  -- une variable, par exemple **X**

  -- un monôme, par exemple **2.5 * X**

  -- une expression linéaire, par exemple **1 + 2.5 * X - Y**

  -- une expression quasi-linéaire, par exemple **Min (23, 2*X, Y - 1.2)**

  au premier appel, J est une variable

- $H \subset \mathcal{V}$, ensemble des variables en fonction desquelles on exprime la borne supérieure de J; au premier appel, $H = \varnothing$

- $\mathcal{S}$: système d'inégalités à résoudre.

Dans les deux algorithmes qui vont suivre, les opérateurs +, −, /, *Min* et *Max* désignent des opérations formelles sur des expressions normalisées. Par exemple :

(2*X + Y - 3*Z + 1) + (X - Y - 1) = 3*X - 3*Z .

L'algorithme de *SUP (J, H, $\mathcal{S}$)* évalue la borne supérieure de la variable J, en fonction des variables de *H*, pour le système $\mathcal{S}$; il s'écrit de la façon suivante :

1. si J est une **constante numérique** ou d'**infini** : SUP (J, H, $\mathcal{S}$) $\leftarrow$ J

2. si J est une **variable** :

   2.1. si J $\in$ H : SUP (J, H, $\mathcal{S}$) $\leftarrow$ J

   2.2. si J $\notin$ H :

   - on borne supérieurement les majorants de J en fonction des variables de H $\cup$ {J} par Z = SUP ($\mathcal{S}$(J), H $\cup$ {J}, $\mathcal{S}$)
   - on résoud J $\leq$ Z par SUP (J, H, $\mathcal{S}$) $\leftarrow$ SUPP (J, Z)

3. si J est un **monôme** de la forme J = a * J', où a est une constante numérique et J' une variable

   3.1. si a > 0 alors SUP (J, H, $\mathcal{S}$) $\leftarrow$ a * SUP (J', H, $\mathcal{S}$)

   3.2. si a < 0 alors SUP (J, H, $\mathcal{S}$) $\leftarrow$ a * INF (J', H, $\mathcal{S}$)

4. si J est une **expression linéaire** de la forme J = a * J' + B, où a est une constante numérique, J' une variable et B soit une constante, soit une variable, soit une expression linéaire : on borne B supérieurement en fonction des variables de H $\cup$ {J'} par Z = SUP (B, H $\cup$ {J'}, $\mathcal{S}$)

   4.1. si Z s'exprime en fonction de J' alors
   SUP (J, H, $\mathcal{S}$) $\leftarrow$ SUP (a * J' + Z, H, $\mathcal{S}$)

   4.2. si Z ne s'exprime pas en fonction de J' alors
   SUP (J, H, $\mathcal{S}$) $\leftarrow$ SUP (a * J', H, $\mathcal{S}$) + Z

5. si J est une **expression quasi-linéaire** de la forme Min (J1, J2) : on calcule la borne supérieure de J1 en fonction de H par $Z = SUP(J1, H, \mathcal{S})$

   5.1. si $Z = -\infty$ alors $SUP(J, H, \mathcal{S}) \leftarrow -\infty$

   5.2. sinon $SUP(J, H, \mathcal{S}) \leftarrow$ **Min** $(Z, SUP(J2, H, \mathcal{S}))$

La fonction **SUPP (J, Z)** calcule la borne supérieure de J telle que $J \leq Z$. J est une variable et Z une expression normalisée.

L'algorithme de SUPP (J, Z) s'écrit :

1. si Z ne s'exprime pas en fonction de J : $SUPP(J, Z) \leftarrow Z$

2. si $Z = J$: $SUPP(J, Z) \leftarrow +\infty$

3. si Z est de la forme Min $(Z_1, ..., Z_n)$:
   $SUPP(J, Z) \leftarrow$ **Min** $(SUPP(J, Z_1), ..., SUPP(J, Z_n))$

4. sinon Z est obligatoirement de la forme $a * J + Z'$

   4.1. si $a > 1$ alors on ne peut pas borner J : $SUPP(J, Z) \leftarrow +\infty$

   4.2. si $a < 1$: $SUPP(J, Z) \leftarrow Z' / (1 - a)$

   4.3. si $a = 1$ alors

      4.3.1. si Z' n'est pas un nombre, on ne peut pas borner J :
      $SUPP(J, Z) \leftarrow +\infty$

      4.3.2. si Z' est un nombre et $Z' \geq 0$ alors la solution de $J \leq Z$ est indéterminée :
      $SUPP(J, Z) \leftarrow +\infty$

      4.3.3. si Z' est un nombre et $Z' < 0$ alors il n'y a pas de solution :
      $SUPP(J, Z) \leftarrow -\infty$

Les algorithmes des fonctions INF et INFF s'écrivent de façon identique.

La complexité des algorithmes SUP et INF est difficile à déterminer avec précision. Elle dépend bien sûr des nombres de variables et d'inéquations à traiter, mais principalement de la forme des inéquations. En effet, la résolution du système

$$
\begin{aligned}
a &\leq 10 \\
b &\leq 5 \\
c &\leq 3 \\
e &\leq 1.4 \\
d &\leq 5 \\
k &\leq 100
\end{aligned}
$$

est bien évidemment plus simple que celle du système

$$
\begin{aligned}
a &\leq b + c - d + 2*k \\
b &\leq c - 2*d - e \\
c &\leq a - 2*e + k \\
d &\leq b - 2 \\
k &\leq 1 - a - c
\end{aligned}
$$

même si les nombres des variables et d'inéquations sont comparables.

Il est donc important de noter que plus les variables seront liées entre elles, plus la résolution du système sera complexe.

Dans les pages suivantes, le lecteur trouvera quelques exemples de résolution de systèmes d'inéquations utilisant la méthode de Shostak. Dans chaque cas, on fournit les inéquations traitées, la solution trouvée ainsi qu'un temps CPU (sur VAX 785) approximatif nécessaire à cette résolution.

```
nom du fichier d essai(fin si arret)
? tsho9

Les contraintes
  10 - y    >=  0
  2 - x - y    >=  0
  + x    >=  0
  10 - x    >=  0

*** SHOSHO mouline

  -infini <= y <= 2
  0. <= x <= 10

tout ceci vous est calcule dans le temps record de    .1099997 secondes


          ====================================


nom du fichier d essai(fin si arret)
? tsho8

Les contraintes
  2 + x    >=  0
  2 - x    >=  0
  - y    >=  0

*** SHOSHO mouline

  -2. <= x <= 2
  -infini <= y <= 0

tout ceci vous est calcule dans le temps record de    .03999996 secondes


          ====================================


nom du fichier d essai(fin si arret)
? tsho7

Les contraintes
  + x    >=  0
  -2 + y    >=  0
  + z    >=  0
  5 - x    >=  0
  -8 + y    >=  0
  25 - z    >=  0

*** SHOSHO mouline

  0. <= x <= 5
  8. <= y <= +infini
  0. <= z <= 25

tout ceci vous est calcule dans le temps record de    .03999996 secondes
```

```
nom du fichier d essai(fin si arret)
? tsho6

Les contraintes
  + x   >=  0
  -1 + y   >=  0
  + z   >=  0



*** SHOSHO mouline
  0. <= x <= +infini
  1. <= y <= +infini
  0. <= z <= +infini

tout ceci vous est calcule dans le temps record de   .02999973 secondes


          ======================================


nom du fichier d essai(fin si arret)
? tsho5

Les contraintes
  4 + x + y + z   >=  0
  2 - x -2*y -5*z   >=  0

*** SHOSHO mouline

  -infini <= x <= +infini
  -infini <= y <= +infini
  -infini <= z <= +infini

tout ceci vous est calcule dans le temps record de   .8599997 secondes


          ======================================


nom du fichier d essai(fin si arret)
? tsho4

Les contraintes
  4 + x + y + z   >=  0

*** SHOSHO mouline

  -infini <= x <= +infini
  -infini <= y <= +infini
  -infini <= z <= +infini

tout ceci vous est calcule dans le temps record de   .1299992 secondes
```

```
nom du fichier d essai(fin si arret)
? tsho3

Les contraintes
  4 + x + y + z   >=  0
  2 - x -2*y -5*z   >=  0
  4 + y -2*x   >=  0

*** SHOSHO mouline

  -infini <= z <= 3.
  -5.999999 <= y <= +infini
  -infini <= x <= +infini

tout ceci vous est calcule dans le temps record de   2.860001 secondes



          ====================================
nom du fichier d essai(fin si arret)
? tsho2

Les contraintes
  4 + x + y + z   >=  0
  2 - x -2*y -5*z   >=  0
  4 + y -2*x   >=  0
  3 - y - x   >=  0

*** SHOSHO mouline

  -7. <= z <= 3.
  -5.999999 <= y <= 34.
  -30.99999 <= x <= 2.333333

tout ceci vous est calcule dans le temps record de   4.860001 secondes



          ====================================
nom du fichier d essai(fin si arret)
? tsho1

Les contraintes
  4 - x - y - z   >=  0
  4 + x + y + z   >=  0
  4 - x - y + z   >=  0
  4 + x + y - z   >=  0
  4 - x + y + z   >=  0
  4 - x + y - z   >=  0
  4 + x - y - z   >=  0
  4 + x - y + z   >=  0

*** SHOSHO mouline

  -4. <= x <= 4.
  -4. <= y <= 4.
  -4. <= z <= 4.

tout ceci vous est calcule dans le temps record de   109.83 secondes
```

4.2 - SYSTEMES ET/OU D'INEQUATIONS

De façon plus générale, nous avons été amenés à résoudre des **systèmes ET/OU d'inéquations linéaires.** Voici un exemple simple d'un tel système :

$$\mathcal{S} \quad \begin{array}{ll} & 0 \le X \le 10 \\ \text{ET} & Y \ge 0 \\ \text{ET} & \\ & X \le Y \quad \text{et} \quad Y \le 2 - X \\ \text{OU} & X \le Y + 7 \quad \text{et} \quad X \ge Y - 1 \quad \text{et} \quad Y \ge 2 - X \end{array}$$

Une solution consiste à se ramener à un OU sur un ensemble de systèmes ET, en prenant toutes les combinaisons possibles. Pour l'exemple, on obtient :

$$\begin{array}{l} 0 \le X \le 10 \\ Y \ge 0 \\ X \le Y \\ Y \le 2 - X \end{array}$$

OU

$$\begin{array}{l} 0 \le X \le 10 \\ Y \ge 0 \\ X \le Y + 7 \\ X \ge Y - 1 \\ Y \ge 2 - x \end{array}$$

L'étape suivante consiste à résoudre chacun des systèmes ET obtenus. La solution du système initial est obtenue par la *réunion des solutions des différents systèmes*. Cette méthode présente un gros avantage : elle est très simple et nécessite uniquement l'existence d'un système de résolution de systèmes ET d'inéquations linéaires, comme celui présenté au paragraphe précédent. Cependant, si n est le nombre maximum d'imbrications de OU, il y a alors 2^n systèmes ET à résoudre, ce qui n'est pas réaliste. De plus, la solution fournie ne peut être qu'un encadrement approché, qui ne rend pas compte des composantes de l'union.

Il serait donc intéressant de développer une autre méthode de résolution de tels systèmes, en espérant qu'elle soit plus rapide sinon plus efficace. C'est ce que propose MOHR dans [MOHR82], sa méthode étant appliquée plus particulièrement aux règles de description de formes.

Précisons tout d'abord le vocabulaire et les notations utilisés. Nous noterons $\mathcal{V}$ l'ensemble des formes et $\mathcal{R}$ l'ensemble des règles : chacune d'elles est constituée d'un *membre gauche* qui est la forme définie par la règle et d'un *membre droit* qui représente l'ensemble des composants de la forme. A chaque règle est associé un ensemble de *contraintes linéaires* portant sur des variables ou *attributs* propres à chaque forme de la règle. L'ensemble des attributs est noté $\mathcal{A}$. Nous supposerons de plus que chaque contrainte est ramenée à la forme *canonique* suivante, pour chaque variable x de la contrainte :

$$max\ (x) \leq \Phi\ (\ max\ (y_1), \ldots, max\ (y_n), min\ (y_1), \ldots, min\ (y_n)\ )$$

$$min\ (x) \geq \Psi\ (\ max\ (y_1), \ldots, max\ (y_n), min\ (y_1), \ldots, min\ (y_n)\ )$$

où les $y_1 \ldots y_n$ sont des expressions

Φ est une fonction croissante des max. et décroissante des min.

Ψ est une fonction décroissante des max. et croissante des min.

Par exemple, la contrainte $x \leq y - z$ devient, pour la variable x :

$$\max(x) \leq \max(y) - \min(z)$$
$$\min(x) \geq \min(y) - \min(z)$$
$$\min(x) \leq \max(y) - \max(z)$$

L'occurrence de l'attribut a dans les contraintes de la règle r est noté a_r.

La détermination des intervalles de variations des attributs indépendamment de toute règle conduit à la résolution d'un système ET/OU de contraintes linéaires. Dans ce cas, MOHR propose une méthode d'encadrement des attributs, de mise en oeuvre simple et donnant de bons résultats dans des cas simples.

Pour $a \in \mathcal{A}$ et $r \in \mathcal{R}$, on définit la fonction Ψa_r par

$$\Psi\ a_r = \sup\ (\Psi_i, \min\ (a)\ )$$

où Ψ_i est la fonction exprimant la contrainte sur le minimum de a_r en fonction des autres attributs apparaissant dans les contraintes de la règle

min (a) est la valeur minimum que peut prendre l'attribut a, indépendamment de tout contexte.

De façon symétrique, nous définissons la fonction $\phi\, a_r$ par

$$\phi\ a_r = \inf(\phi_i, \max(a))$$

avec les mêmes conventions de notation que ci-dessus.

Les fonctions Ψ et Φ nous permettent alors d'exprimer les minima et maxima de a, à l'aide des deux définitions suivantes :

$$\Psi a = \sup(\inf(\Psi a_r), \inf(\Psi a_r))$$

$$r \in g(a) \qquad r \in d(a)$$

$$\Phi a = \inf(\sup(\Phi a_r), \sup(\Phi a_r))$$

$$r \in g(a) \qquad r \in d(a)$$

où *g (a)* est l'ensemble des règles où la forme relative à l'attribut a apparait dans le premier membre de la règle
d (a) est l'ensemble des règles où la forme relative à l'attribut a apparait dans le second membre de la règle.

Ainsi définies, ces fonctions nous permettent de borner l'attribut a de la façon suivante :

$$\Psi a \leq a \leq \Phi a$$

ce qui signifie simplement que le minimum de la valeur de l'attribut, indépendamment de tout contexte, est la plus grande des deux valeurs définies respectivement par le minima des valeurs de l'attribut lorsqu'il apparait en premier et en second membre. Les résultats sont symétriques pour la fonction Φa.

Dans la suite, nous noterons $\mathcal{I}$ l'ensemble des intervalles de $\mathbb{R}$ de la forme [a, b] ou $]-\infty, b[$ ou $[a, +\infty[$. On définit la fonction γ par

$$\gamma : \mathcal{I}(\mathcal{A} \times \mathcal{R}) \cup \mathcal{A} \rightarrow \mathcal{I}(\mathcal{A} \times \mathcal{R}) \cup \mathcal{A}$$

$$\gamma([a_i, b_i]) = ([c_i, d_i]) \quad \text{pour } i \in (\mathcal{A} \times \mathcal{R}) \cup \mathcal{A}$$

$$\text{avec } c_k = \Psi_k([a_i, b_i]) \text{ et } d_k = \Phi_k([a_i, b_i])$$

$$\text{pour } k \in (\mathcal{A} \times \mathcal{R}) \cup \mathcal{A}$$

Théorème

Toute solution $S = (a_i)$ *pour* $i \in (\mathcal{A} \times \mathcal{R}) \cup \mathcal{A}$
du système d'inéquations vérifie :

$$S \in \text{limite}_{n \rightarrow \infty} \; g(]-\infty, +\infty[)$$

La démonstration complète de ce théroème est donnée dans [MOHR82], elle ne sera donc pas détaillée ici. En d'autres termes, ce théroème signifie que, par applications successives de la fonction γ et par là-même des fonctions Ψ et Φ, jusqu'à stabilité des intervalles, on obtient un encadrement de la valeur des attributs, indépendamment de toute règle.

Regardons le principe de la résolution d'un système ET/OU sur un exemple très simple, limité à une seule règle, pour la clarté de l'exposé.

/1/ $A \rightarrow B\ C$

$0 \leq c_1 \leq 10$

$| b_1 - b_2 | \leq 5$

$a_1 \geq b_1 \geq 0$

$b_2 \leq a_1 + 6$

$b_2 \geq a_1 + 6$

Les attributs a_1, b_1 et b_2, c_1 sont relatifs respectivement aux formes A, B et C.

Les fonctions Φ et Ψ s'écrivent de la façon suivante :

$\Phi\, a_{11} = \min(a_{11}) = \sup(\min(a_{11}), \min(b_{11}), \min(b_{21}) - 6)$

$\Psi\, a_{11} = \max(a_{11}) = \inf(\max(a_{11}), \max(b_{21}) - 6)$

$\Phi\, b_{11} = \min(b_{11}) = \sup(\min(b_{11}), 0, \min(b_{21}) - 5)$

$\Psi\, b_{11} = \max(b_{11}) = \inf(\max(b_{11}), \max(a_{11}), \max(b_{21}) + 5)$

$\Phi\, b_{21} = \min(b_{21}) = \sup(\min(b_{21}), \min(a_{11}) + 6, \min(b_{11}) - 5)$

$\Psi\, b_{21} = \max(b_{21}) = \inf(\max(b_{21}), \max(a_{11}) + 6, \max(b_{11}) + 5)$

$\Phi\, c_{11} = \min(c_{11}) = \sup(\min(c_{11}), 0)$

$\Psi\, c_{11} = \max(c_{11}) = \inf(\max(c_{11}), 10)$

Au départ, l'intervalle de variation de chaque variable est fixé à [-10, +10]. A chaque itération, on applique la fonction γ. Ce qui donne :

$$\begin{array}{l} -10 \le a_1 \le 10 \\ -10 \le b_1 \le 10 \\ -10 \le b_2 \le 10 \\ -10 \le c_1 \le 10 \end{array} \xrightarrow{\gamma} \begin{array}{l} -10 \le a_1 \le 4 \\ 0 \le b_1 \le 4 \\ -4 \le b_2 \le 9 \\ 0 \le c_1 \le 10 \end{array} \xrightarrow{\gamma}$$

$$\begin{array}{l} 0 \le a_1 \le 3 \\ 0 \le b_1 \le 2 \\ 6 \le b_2 \le 8 \\ 0 \le c_1 \le 10 \end{array} \xrightarrow{\gamma} \begin{array}{l} 0 \le a_1 \le 2 \\ 1 \le b_1 \le 2 \\ 6 \le b_2 \le 7 \\ 0 \le c_1 \le 10 \end{array} \xrightarrow{\gamma}$$

$$\begin{array}{l} 1 \le a_1 \le 1 \\ 1 \le b_1 \le 1 \\ 7 \le b_2 \le 6 \\ 0 \le c_1 \le 10 \end{array}$$

A la quatrième itération, le domaine de variation de la variable b_2 devient vide. Dans ce cas, la limite est vide et il n'existe pas de solution : le système est inconsistant. La variable b_1, ainsi que toutes les autres variables qui lui sont liées par une inéquation, sont causes de l'erreur : ici, ce sont donc les variables a_1, b_1 et b_2.

De tout cela, il ressort que la résolution d'un système ET/OU d'inéquations peut se faire par des applications successives de la fonction γ, jusqu'à stabilité des intervalles de variation. Afin d'éviter une itération infinie, on pourra s'arrêter dès que la diminution est inférieure à un seuil. Une fois la limite obtenue, on sait que toute solution du système est dans cette limite, et que si la limite est vide, le système n'a pas de solution. Il est toutefois impossible de connaitre la différence entre la limite et l'espace des solutions.

4.3 - LIMITATION DE LA COMPLEXITE D'UN ENSEMBLE DE CONTRAINTES LINEAIRES

La résolution d'un système ET de contraintes par la méthode proposée par BLEDSOE nécessite **la substitution des équations par des inéquations équivalentes** (il en est de même dans le cas de systèmes OU). Le nombre de variables et d'inéquations à traiter étant un facteur de la complexité de la résolution, il s'avère donc intéressant d'utiliser les égalités pour simplifier les inégalités. En effet, chaque égalité peut être transformée de façon à donner une définition d'une variable. Par exemple, l'égalité

$$X + 2*Y - 3 = Z + 1$$

est équivalente à

$$X = Z - 2*Y + 4$$

qui représente une définition de la variable X, en fonction des variables Y et Z.

Une telle information est très importante, puisqu'elle permet de substituer X par sa définition dans toutes les inégalités, qui peuvent s'en trouver simplifiées. Plus le nombre d'égalités est important, plus le nombre de variables supprimées sera important.

Prenons un exemple : la résolution du système

$$\begin{aligned}
&X = Y - 1\\
&Z = U + K\\
&X + 3*Z - 1 = U + 2*K\\
&X \geq 2 + Z\\
&K \leq X - 2*Y + 3
\end{aligned}$$

avec la méthode proposée par BLEDSOE, sans aucune simplification, nécessite le remplacement des égalités par deux inégalités. Le système d'inéquations à résoudre est composé des 8 inégalités suivantes :

$$\begin{aligned}
&X \leq Y - 1\\
&X \geq Y - 1\\
&Z \leq U + K\\
&Z \geq U + K\\
&X + 3*Z - 1 \leq U + 2*K\\
&X + 3*Z - 1 \geq U + 2*K\\
&X \geq 2 + Z\\
&K \leq X - 2*Y + 3
\end{aligned}$$

il porte sur les cinq variables {X, Y, Z, U, K} .

Par contre, en substituant d'abord X par $Y-1$, puis Z par $U + K$ et Y par $-2*U - K$, on obtient les deux inégalités suivantes :

$$3*U + 2*K + 1 \leq 0$$
$$U \leq 2$$

Ce système à résoudre ne porte plus que sur les deux variables {U, K}. Le domaine de définition des variables éliminées {X, Y, Z} s'obtient par la suite par calcul en utilisant les définitions suivantes :

$$X = -2*U - K - 1$$
$$Z = U + K$$
$$Y = -2*U - K$$

Outre la **limitation du nombre de variables à traiter** et par là même du temps de résolution, cette simplification présente un autre avantage : une **incohérence éventuelle** dans le système de contraintes peut déjà être détectée dans la phase de simplification, avant même de résoudre les inégalités.

Par exemple, dès la phase de simplification, une *incohérence* sera détectée simplement dans les égalités du système suivant :

$$C = D$$
$$A \leq B + C$$
$$B \geq 0$$
$$B = C$$
$$B = D + 1$$

une autre erreur sera détectée globalement dans l'ensemble de contraintes suivant :

$$A \geq B + C$$
$$B = A$$
$$C = 1$$

puisque, par substitution dans l'inégalité de B et C par leurs valeurs respectives, on obtient

$$A \geq A + 1$$

qui est incohérent.

De telles incohérences sont trouvées facilement et rapidement dans la phase de simplification, alors que leur détection risque d'être longue si elle doit être faite par le système de résolution.

Comment effectuer cette simplification ?

On détermine tout d'abord, à partir des égalités du système à résoudre, le plus grand ensemble de variables pouvant être définies en fonction des autres. Cet ensemble est noté $\mathcal{V}e$. Les éléments de $\mathcal{V}e$ seront éliminés de l'ensemble des inéquations. Les variables restantes sont dites **variables principales**, leur ensemble est noté $\mathcal{V}p$.

On a $\mathcal{V} = \mathcal{V}e \cup \mathcal{V}p$.

A chacun des éléments de $\mathcal{V}e$, est donc associée une définition portant uniquement sur des variables de $\mathcal{V}p$. L'algorithme qui détermine les ensembles $\mathcal{V}e$ et $\mathcal{V}p$ est similaire à un algorithme de calcul de classes d'équivalence d'une relation : ici, la relation traitée est ***est défini par.*** Par construction, il fournit l'ensemble $\mathcal{V}e$ de cardinal maximum.

Dès qu'une variable de $\mathcal{V}$ a une nouvelle définition, on effectue la substitution correspondante dans les inégalités. Lorsque toutes les égalités ont été traitées, le système d'inéquations obtenu ne porte plus que sur des variables principales.

Algorithme de simplification

1. Chaque variable est supposée être principale
   $\forall\ v \in \mathcal{V}p \quad DEF(v) = v$

2. *Pour chaque égalité eg du système à résoudre faire*

   2.1. remplacer chaque variable v de *eg* par sa définition DEF (v) : on obtient une nouvelle égalité **eg'** que l'on met sous la forme
   expression = 0

   2.2. *si* l'expression est réduite à une constante ...

   ... nulle : l'égalité est **redondante** : elle est ignorée

   ... non nulle : le système d'égalités est **incohérent** : **ERREUR**

   2.3. *sinon*, on **isole une variable** v dans *eg'* et on obtient une définition de la forme
   v = expression

   DEF (v) ← expression

   on remplace v par DEF (v) dans les définitions des autres variables ainsi que dans toutes les inégalités en vérifiant qu'aucune incohérence n'apparaisse.

En pratique, le choix de la variable à isoler dans une égalité (phase 2.3.) tient compte de *l'heuristique* suivante : puisque l'on cherche à simplifier au maximum le système d'inéquations, autant choisir d'isoler la variable qui entrainera le plus de simplifications possibles. Voici un exemple simple : pour l'égalité

$$A + B - C = 0$$

et le système d'inéquations suivant :

$$A \leq C - 2*D$$
$$D + B/2 \geq 0$$

nous avons la possibilité d'isoler de l'égalité l'une ou l'autre des variables { A, B, C } : les divers cas sont étudiés dans le tableau ci-dessous: chaque ligne fournit la définition d'une variable, le nouveau système d'inéquations obtenues par substitution, la variation du nombre d'attributs entre inégalités initiales et inégalités simplifiées, et la variation globale.

VARIABLE CHOISIE	DEFINITION	NOUVELLES INEQUATIONS	VARIATION DU NOMBRE D'ATTRIBUTS	AU TOTAL
A	C - B	B >= 0 D + B/2 >= 0	-2 0	-2
B	C - A	A <= C + 2*D D/2 + C - A >= 0	0 1	1
C	A + B	B >= 0 D + B/2 >= 0	-2 0	-2

Ici, il est donc plus intéressant de définir A ou C en fonction des autres variables puisque dans ce cas, la simplification globale est la plus grande (la variation globale du nombre d'attributs est la plus faible). Bien sûr, dans de petits exemples comme celui-ci, le gain n'est pas très significatif, mais pour le traitement de grands systèmes d'équations, la simplification globale est considérable.

Appliquons l' algorithme de simplification sur l'ensemble de contraintes suivant :

$$A + B - 3 = 0$$
$$A \geq B + C - 1$$
$$A + + C + D + H + 1 = 0$$
$$B \geq K$$
$$D - H = 0$$
$$C + K \leq H + 2$$
$$B + Z + D + A - 1 = 0$$
$$D \geq Z + C$$

On détermine les **variables principales**. Initialement, on a

DEF (A) = A
DEF (B) = B
DEF (C) = C
DEF (D) = D
DEF (Z) = Z
DEF (H) = H

Chaque égalité est traitée l'une après l'autre.

* ***A + B - 3 = 0***

la substitution des variables par leurs définitions ne change rien

l'application de l'heuristique précédente nous indique d'isoler au choix les variables A ou B, pour lesquels la variation globale du nombre d'attributs est de -1 : **on choisit A** :

$$A = 3 - B$$

seule DEF (A) va changer : **DEF (A) ← 3 - B**

* ***A + C + D + H + 1 = 0***

on remplace A par DEF (A) : l'égalité devient -B + C + D + H + 4 = 0

l'heuristique nous permet de **choisir D** : **D = B - C - H - 4**

seule DEF (D) va changer : **DEF (D) ← B - C - H - 4**

* ***D - H = 0***

on remplace D par DEF (D) : l'égalité devient B - C - 2*H - 4 = 0

l'heuristique nous indique de choisir C ou H : on **choisit C : C = B - 2*H - 4**

les définitions de C et D sont modifiées

DEF (C) = B - 2*H - 4
DEF (D) = H

* ***B + Z + D + A - 1 = 0***

on remplace A et D par leurs définitions : l'égalité devient H + Z + 2 = 0

l'heuristique désigne **H : H = -Z - 2**

les définitions de C, D et H sont mises à jour :

DEF (C) = B + 2*Z
DEF (D) = -Z - 2
DEF (H) = -Z - 2

Les ensembles $\mathcal{V}e$ et $\mathcal{V}p$ sont définis par :

$\mathcal{V}e$ = {A, C, D, H}
$\mathcal{V}p$ = {Z, B}

L'ensemble d'inéquations simplifiées est finalement

$4 \geq 3*B + 2*Z$
$B \geq K$
$B + 3*Z + K \geq 0$
$B \geq 2$

avec de plus
A = 3 - B
C = B + 2*Z
D = -Z - 2
H = -Z - 2

Non seulement le nombre de variables a diminué, mais la complexité des inéquations est également plus faible.

5. RESOLUTION DE SYSTEMES DE CONTRAINTES DISCRETES

présentation du problème

Une **contrainte discrète** porte sur des variables prenant leurs valeurs dans un domaine discret quelconque, noté $\mathcal{D}$. Elle a l'une des formes suivantes :

variable = variable
variable ≠ variable
variable ∈ ensemble_de_valeurs $\subset \mathcal{D}$
variable ∉ ensemble_de_valeurs $\subset \mathcal{D}$

Résoudre un système de contraintes discrètes consiste à déterminer l'ensemble des valeurs possibles de chaque variable. Ce problème s'apparente à la *preuve de consistance d'un graphe*. L'ensemble $\mathcal{N}$ des variables constitue les noeuds du graphe. A chaque noeud i ∈ $\mathcal{N}$, est associé un ensemble d'étiquettes noté **DOM(i)** représentant les valeurs possibles de la variable. L'ensemble des arcs est noté $\mathcal{A}$. Un *arc (i, j)* représente une relation binaire = ou ≠ entre les deux variables i et j. Le problème se ramène à un étiquetage des noeuds du graphe, en respectant les contraintes imposées par les arcs.

Le traitement des **égalités** ne pose aucun problème. Comme on le verra plus tard dans ce paragraphe, il se réduit à la construction des classes d'équivalence de la relation **est égal à**, sur l'ensemble des variables $\mathcal{N}$.

Le traitement des **différences** a conduit à diverses solutions d'efficacités variables [MACK77]. L'algorithme le plus performant est celui proposé par MOHR et HENDERSON [MOHR85]. Son efficacité est évaluée en $O(al^2)$ où **a** est le cardinal de $\mathcal{A}$ et **l** le cardinal de l'ensemble de toutes les étiquettes possibles $\mathcal{D}$.

L'idée de base repose sur l'algorithme de MACKWORTH et FREUDER. Dans cet algorithme, lorsqu'une étiquette est retirée de l'ensemble possible des étiquettes d'un noeud i, seuls sont remis en cause les étiquetages des noeuds j pour lesquels il existe un arc (i,j). L'amélioration apportée à cette technique s'appuie sur des aspects techniques du système ALICE [LAUR76]. Elle repose sur la notion de **correspondant**.

Considérons par exemple l'*étiquette x au noeud i*. Tant que x est compatible avec au moins une étiquette de tous les noeuds j tels que $i \neq j$, l'étiquette x reste possible pour le noeud i. Dès que cette condition n'est plus vérifiée, nous dirons que l'étiquette x pour le noeud i a perdu tous ses correspondants : elle doit donc être éliminée de L(i).

Cette mise à jour doit être **propagée** sur les autres variables : il convient donc de signaler à tous les noeuds ayant une étiquette compatible avec x que ce correspondant leur fait défaut. **L'inconsistance** du graphe est montrée dès qu'un ensemble L(i) devient vide.

Quoique très efficace, cette technique est limitée car elle ne permet qu'une vérification **locale** de la consistance et non globale. Citons l'exemple suivant bien connu :

$\mathcal{N} = \{ n_1, n_2, n_3 \}$

$L(n_1) = L(n_2) = L(n_3) = \{ a\ b \}$

$n_1 \# n_2 \quad n_2 \# n_3 \quad n_3 \# n_1$

Ici, les noeuds sont compatibles 2 à 2, mais globalement, le système est *inconsistant* puisqu'il n'y a que deux étiquettes pour 3 noeuds qui doivent être étiquetés tous différemment.

De telles situations ne peuvent être détectées qu'en appliquant un **algorithme de consistance globale** qui examine toutes les combinaisons d'étiquetages possibles et élimine celles qui sont inconsistantes. La construction de l'ensemble des labels possibles se fait par une technique de retour-arrière dont le coût est bien évidemment exponentiel selon le nombre de noeuds du graphe. Après avoir choisi un noeud i et une étiquette possible dans DOM(i), on propage le fait que toutes les autres étiquettes sont impossibles pour le noeud i.

Cette opération se répète tant qu'il existe un noeud j pour lequel DOM(j) ne soit pas réduit à un élément. Dès qu'un ensemble DOM(k) devient vide ou dès qu'une combinaison consistante a été trouvée, le dernier choix est remis en cause, afin d'étudier une nouvelle combinaison.

Le caractère exponentiel de cet algorithme mène évidemment à l'étude de stratégies visant à minimiser le nombre de retour-arrières effectués en cas d'échec. Un panorama des différentes stratégies possibles a été fait par LAURIERE dans ALICE [LAUR76]. L'une des plus simples à mettre en oeuvre consiste à effectuer chaque choix en priorité dans l'ensemble de cardinal le plus petit.

Algorithme de consistance locale

Rappelons qu'il s'agit d'associer à chaque variable un domaine de valeurs possibles, respectant les contraintes imposées par un système $\mathcal{S}$ de contraintes discrètes. Dans un souci d'efficacité, l'algorithme utilisé effectue en parallèle la construction du graphe et la vérification de sa consistance.

Afin de simplifier au maximum le traitement des égalités de variables, un noeud du graphe ne représente pas seulement une variable mais plutôt une **classe d'équivalence** de la relation *est égal à* sur les variables de $\mathcal{V}$. De cette façon, les différences entre variables se traduisent par des différences de classes d'équivalence. De plus, dans le graphe de consistance construit par l'algorithme, les différences, donc les arcs ne sont plus conservées lorsqu'elles deviennent toujours vérifiées : cela permet de limiter la complexité du graphe à traiter ainsi que la place mémoire nécessaire.

Dans la suite, nous noterons :

$\mathcal{N}$	l'ensemble des noeuds du graphe, c'est-à-dire l'ensemble des classes d'équivalence de la relation
DOM($\mathcal{C}_i$)	l'ensemble des labels possibles pour les variables de la classe $\mathcal{C}_i$
VAR($\mathcal{C}_i$)	l'ensemble des variables de la classe $\mathcal{C}_i$
DIFF($\mathcal{C}_i$)	l'ensemble des classes des variables différentes des variables de la classe $\mathcal{C}_i$.

L'algorithme de vérification de consistance locale est décrit ci-dessous. Il utilise la fonction CLASSE-DE ($\mathcal{N}$, v) qui retourne la classe d'équivalence de la variable v dans l'ensemble des classes $\mathcal{N}$.

début

$\mathcal{N} \leftarrow \varnothing$

Pour chaque c **dans** $\mathcal{S}$ **faire**

cas c *de la forme*

var1 = var2 : EGALITE ($\mathcal{N}$, CLASSE-DE ($\mathcal{N}$, var1), CLASSE-DE ($\mathcal{N}$, var2))

var1 # var2 : DIFFERENCE ($\mathcal{N}$, CLASSE-DE ($\mathcal{N}$, var1),
CLASSE-DE ($\mathcal{N}$,var2))

var $\in \mathcal{E}$: MAJ-DOMAINE ($\mathcal{N}$, CLASSE-DE ($\mathcal{N}$, var), $\mathcal{E}$)

var $\notin \mathcal{E}$: MAJ-DOMAINE ($\mathcal{N}$, CLASSE-DE ($\mathcal{N}$, var), $\mathcal{D}$ - $\mathcal{E}$)

fincas

finpour

Procédure EGALITE ($\mathcal{N}$, $\mathcal{C}1$, $\mathcal{C}2$)

mise à jour de l'ensemble des classes $\mathcal{N}$ en tenant compte de l'égalité des variables des classes $\mathcal{C}1$ et $\mathcal{C}2$

début

si $\mathcal{C}1 \in$ DIFF($\mathcal{C}2$) *ou* $\mathcal{C}2 \in$ DIFF($\mathcal{C}1$)

alors **INCONSISTANCE LOCALE**

sinon MAJ-DOMAINE ($\mathcal{N}$, $\mathcal{C}1$, DOM($\mathcal{C}1$) $\cap$ DOM ($\mathcal{C}2$))

VAR ($\mathcal{C}1$) $\leftarrow$ VAR ($\mathcal{C}1$) + VAR ($\mathcal{C}2$)

$\mathcal{N} \leftarrow \mathcal{N}$ - { $\mathcal{C}2$ }

DIFF ($\mathcal{C}1$) $\leftarrow$ DIFF ($\mathcal{C}1$) $\cup$ DIFF ($\mathcal{C}2$)

finsi

fin

Procédure DIFFERENCE ($\mathcal{N}$, $\mathcal{C}1$, $\mathcal{C}2$)

mise a jour de l'ensemble des classes $\mathcal{N}$ en tenant compte de la différence des variables des classes $\mathcal{C}1$ et $\mathcal{C}2$

début

si $\mathcal{C}1 = \mathcal{C}2$

alors **INCONSISTANCE LOCALE**

sinon INTER $\leftarrow$ DOM ($\mathcal{C}1$) $\cap$ DOM ($\mathcal{C}2$)

si INTER = $\varnothing$

alors *la contrainte de différence est toujours vérifiée*
elle est ignorée

sinon *étude des cardinaux des domaines des classes*

cas

card (DOM($\mathcal{C}1$)) = card (DOM($\mathcal{C}2$)) = 1 : **INCONSISTANCE LOCALE**

card (DOM($\mathcal{C}1$)) = 1 : MAJ-DOMAINE ($\mathcal{N}$, $\mathcal{C}1$, INTER)
DIFF ($\mathcal{C}1$) $\leftarrow$ DIFF ($\mathcal{C}1$) - { $\mathcal{C}2$ }
DIFF ($\mathcal{C}2$) $\leftarrow$ DIFF ($\mathcal{C}2$) - { $\mathcal{C}1$ }

card (DOM($\mathcal{C}2$)) = 1 : MAJ-DOMAINE ($\mathcal{N}$, $\mathcal{C}2$, INTER)
DIFF ($\mathcal{C}1$) $\leftarrow$ DIFF ($\mathcal{C}1$) - { $\mathcal{C}2$ }
DIFF ($\mathcal{C}2$) $\leftarrow$ DIFF ($\mathcal{C}2$) - { $\mathcal{C}1$ }

sinon DIFF ($\mathcal{C}1$) $\leftarrow$ DIFF ($\mathcal{C}1$) $\cup$ { $\mathcal{C}2$ }
DIFF ($\mathcal{C}2$) $\leftarrow$ DIFF ($\mathcal{C}2$) $\cup$ { $\mathcal{C}1$ }

fincas

finsi

finsi
fin

Procédure MAJ-DOMAINE ($\mathcal{N}$, $\mathcal{C}$, V)

mise à jour du domaine de la classe $\mathcal{C}$ avec l'ensemble de valeurs V ; propagation de cette mise à jour aux classes différentes de $\mathcal{C}$

début

si $V = \varnothing$

alors **INCONSISTANCE LOCALE**

sinon DOM ($\mathcal{C}$) $\leftarrow$ V

Pour chaque cl *dans* DIFF ($\mathcal{C}$) *faire*

DIFFERENCE ($\mathcal{N}$, $\mathcal{C}$, cl)

finpour

finsi

fin

Algorithme de consistance globale

Procédure CONSIST-GLOBALE ($\mathcal{N}$)

début

ENS-CHOIX ← { $\mathcal{C}$i ∈ $\mathcal{N}$ tel que card (DOM ($\mathcal{C}$i)) > 1 }

si ENS-CHOIX ≠ ∅

alors $\mathcal{C}$j ← CHOIX (ENS-CHOIX)

Pour chaque x *dans* DOM ($\mathcal{C}$j) *faire*

MAJ-DOMAINE ($\mathcal{N}$, $\mathcal{C}$j, { x })

CONSIST-GLOBALE ($\mathcal{N}$)

finpour

finsi

fin

La fonction **CHOIX (ENS-CHOIX)** choisit et retourne une classe parmi celles de ENS-CHOIX. Afin de limiter le caractère exponentiel de cet algorithme, il est indispensable que ce choix ne soit pas effectué au hasard. L'heuristique la plus simple à mettre en oeuvre est de sélectionner la classe dont le domaine contient le plus petit nombre d'éléments : le traitement de la classe choisie sera donc plus rapide.

EXEMPLE

Les algorithmes de consistance locale et globale décrits ci-dessus ont été intégré à un outil conversationnel permettant à un utilisateur de vérifier la cohérence de contraintes discrètes. Comme le montre l'exemple ci-après, l'utilisateur peut, à tout moment, soit donner un nouvelle contrainte à intégrer à celles fournies précédemment, soit demander l'impression du graphe construit par le système. A chaque nouvelle contrainte introduite, la consistance locale de l'ensemble obtenu est vérifiée. La consistance globale doit être demandée explicitement par l'utilisateur.

```
************************************************************
***   RESOLUTION DE SYSTEMES DE CONTRAINTES DISCRETES      ***
************************************************************

 Une contrainte discrete a l'une des 4 formes suivantes

       (var1 = var2)
       (var1 # var2)
       (var app val1 ... valk)
       (var napp val1 ... valk)

 Quel est l'ensemble de toutes les valeurs possibles ? (1 2 3 4 5 6 7 8 9 10)

 Pour imprimer le graphe, taper  (IMPRIME)
 Pour lancer la consistance globale, taper  (GLOBALE)
 Pour arreter, taper  (FIN)


 A vous ? (a app 1)

 A vous ? (b app 9 10)

 A vous ? (c app 3 4)

 A vous ? (a = f)

 A vous ? (imprime)

    ***  GRAPHE OBTENU  ***

     Classe =class1
         Variables   (a f)
         Domaine     (1)
         Differences ()

     Classe =class3
         Variables   (c)
         Domaine     (3 4)
         Differences ()

     Classe =class2
         Variables   (b)
         Domaine     (9 10)
         Differences ()

 A vous ? (d app 1 10 3 5 7)

 A vous ? (a = d)

 A vous ? (a # b)

 A vous ? (imprime)

    ***  GRAPHE OBTENU  ***

     Classe =class1
         Variables   (a f d)
         Domaine     (1)
         Differences ()
```

```
    Classe =class3
        Variables   (c)
        Domaine     (3 4)
        Differences ()

    Classe =class2
        Variables   (b)
        Domaine     (9 10)
        Differences ()

A vous ? (h # a)

A vous ? (g # d)

A vous ? (imprime)

   ***  GRAPHE OBTENU  ***

    Classe =class7
        Variables   (g)
        Domaine     (2 3 4 5 6 7 8 9 10)
        Differences ()

    Classe =class6
        Variables   (h)
        Domaine     (2 3 4 5 6 7 8 9 10)
        Differences ()

    Classe =class1
        Variables   (a f d)
        Domaine     (1)
        Differences ()

    Classe =class3
        Variables   (c)
        Domaine     (3 4)
        Differences ()

    Classe =class2
        Variables   (b)
        Domaine     (9 10)
        Differences ()

A vous ? (i # c)

A vous ? (i = h)

A vous ? (imprime)

   ***  GRAPHE OBTENU  ***

    Classe =class8
        Variables   (i h)
        Domaine     (2 3 4 5 6 7 8 9 10)
        Differences (=class3)

    Classe =class7
        Variables   (g)
        Domaine     (2 3 4 5 6 7 8 9 10)
        Differences ()

    Classe =class1
        Variables   (a f d)
        Domaine     (1)
        Differences ()
```

```
        Classe =class3
            Variables   (c)
            Domaine     (3 4)
            Differences (=class8)

        Classe =class2
            Variables   (b)
            Domaine     (9 10)
            Differences ()

    A vous ? (g = c)

    A vous ? (imprime)

       ***  GRAPHE OBTENU  ***

        Classe =class7
            Variables   (g c)
            Domaine     (3 4)
            Differences (=class8)

        Classe =class8
            Variables   (i h)
            Domaine     (2 3 4 5 6 7 8 9 10)
            Differences (=class3)

        Classe =classl
            Variables   (a f d)
            Domaine     (1)
            Differences ()

        Classe =class2
            Variables   (b)
            Domaine     (9 10)
            Differences ()

    A vous ? (h # b)

    A vous ? (imprime)

       ***  GRAPHE OBTENU  ***

        Classe =class7
            Variables   (g c)
            Domaine     (3 4)
            Differences (=class8)

        Classe =class8
            Variables   (i h)
            Domaine     (2 3 4 5 6 7 8 9 10)
            Differences (=class3 =class2)

        Classe =classl
            Variables   (a f d)
            Domaine     (1)
            Differences ()

        Classe =class2
            Variables   (b)
            Domaine     (9 10)
            Differences (=class8)

    A vous ? (fin)

= ()
```

```
************************************************************
***   RESOLUTION DE SYSTEMES DE CONTRAINTES DISCRETES      ***
************************************************************


  Une contrainte discrete a l'une des 4 formes suivantes

        (var1 = var2)
        (var1 # var2)
        (var app val1 ... valk)
        (var napp val1 ... valk)

  Quel est l'ensemble de toutes les valeurs possibles ? (1 2 3)

  Pour imprimer le graphe, taper  (IMPRIME)
  Pour lancer la consistance globale, taper  (GLOBALE)
  Pour arreter, taper  (FIN)


  A vous ? (a app 1 2)

  A vous ? (b app 1 2)

  A vous ? (c app 1 2)

  A vous ? (a # b)

  A vous ? (b # c)

  A vous ? (c # a)

  A vous ? (imprime)

     ***  GRAPHE OBTENU  ***

      Classe =class3
          Variables   (c)
          Domaine     (1 2)
          Differences (=class2 =class1)

      Classe =class2
          Variables   (b)
          Domaine     (1 2)
          Differences (=class1 =class3)

      Classe =class1
          Variables   (a)
          Domaine     (1 2)
          Differences (=class2 =class3)

  A vous ? (globale)

?#$@%<>*&?%^#  Inconsistance
  A vous ? (fin)

= ()
```

6 - LE SYSTEME MEPHISTO

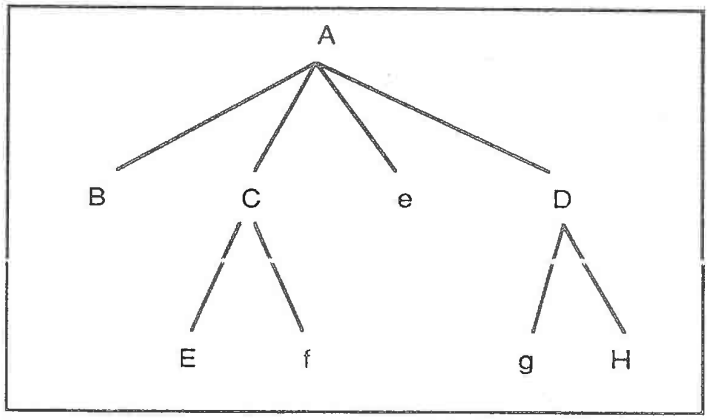
6.1 - PRINCIPE GENERAL

Pour montrer la validité d'un modèle, le système **MEPHISTO** en extrait les différentes **instances** possibles, les unes après les autres. L'utilisateur a la possibilité de fixer le nombre d'instances différentes que MEPHISTO doit extraire.

Rappelons qu'une **instance** est un des chemins du graphe ET/OU associé au modèle, chemins pour lesquels les contraintes attachées aux arcs sont globalement compatibles. Une telle instance est représentée sous forme d'une *arborescence* donnant la décomposition choisie de chaque forme en sous-formes. Si on fait le parallèle entre un graphe ET/OU et une grammaire, il s'agit donc de construire un des mots du langage engendré par cette grammaire.

La technique de construction d'une arborescence est simple : à chaque étape, l'arborescence partiellement construite est complétée soit de façon **ascendante**, en étendant la racine, soit de façon **descendante**, en étendant l'une de ses feuilles. **Etendre un noeud** - la racine ou une feuille - signifie greffer sur ce noeud l'arborescence d'une règle faisant intervenir le noeud. En phase *ascendante*, il s'agit d'une règle où la forme étiquetant le noeud apparait en partie droite. La forme décrite par cette règle devient alors la nouvelle racine de l'arbre. De façon symétrique, en phase *descendante*, il s'agit d'une règle qui décrit la forme étiquetant la feuille considérée. Les sous-formes de la règle deviennent alors autant de nouvelles feuilles à traiter ultérieurement.

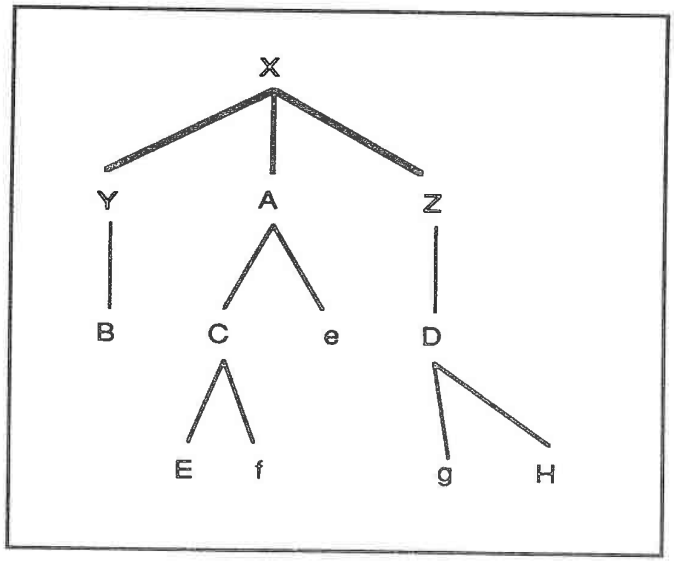
Prenons l'exemple suivant : l'instance en cours de construction est représentée par l'arbre suivant :



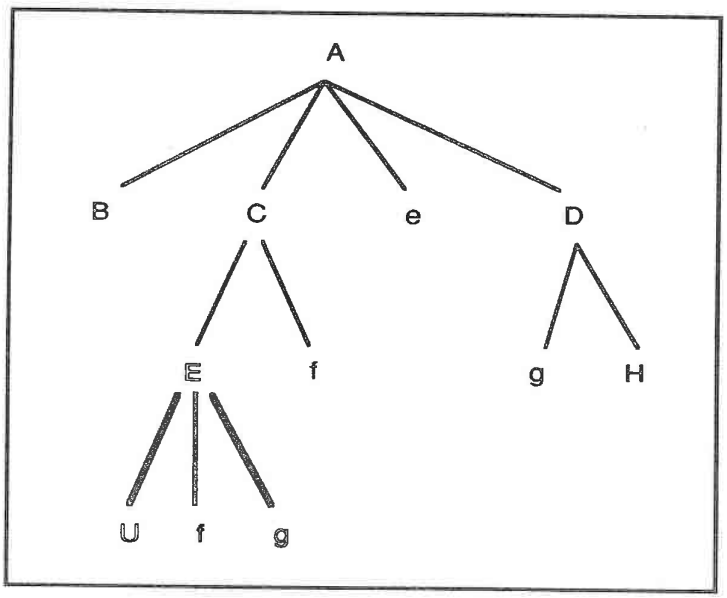
Si le modèle contient en particulier les règles suivantes :

/10/ $X \rightarrow Y\ A\ z$
/11/ $T \rightarrow A\ m\ K$
/12/ $E \rightarrow U\ f\ g$
/13/ $E \rightarrow K\ e$

La phase ascendante consiste à étendre la *racine*, en utilisant l'une des règles /10/ et /11/, où la forme A apparait en partie droite. Par exemple, en prenant la règle /10/, on obtient :



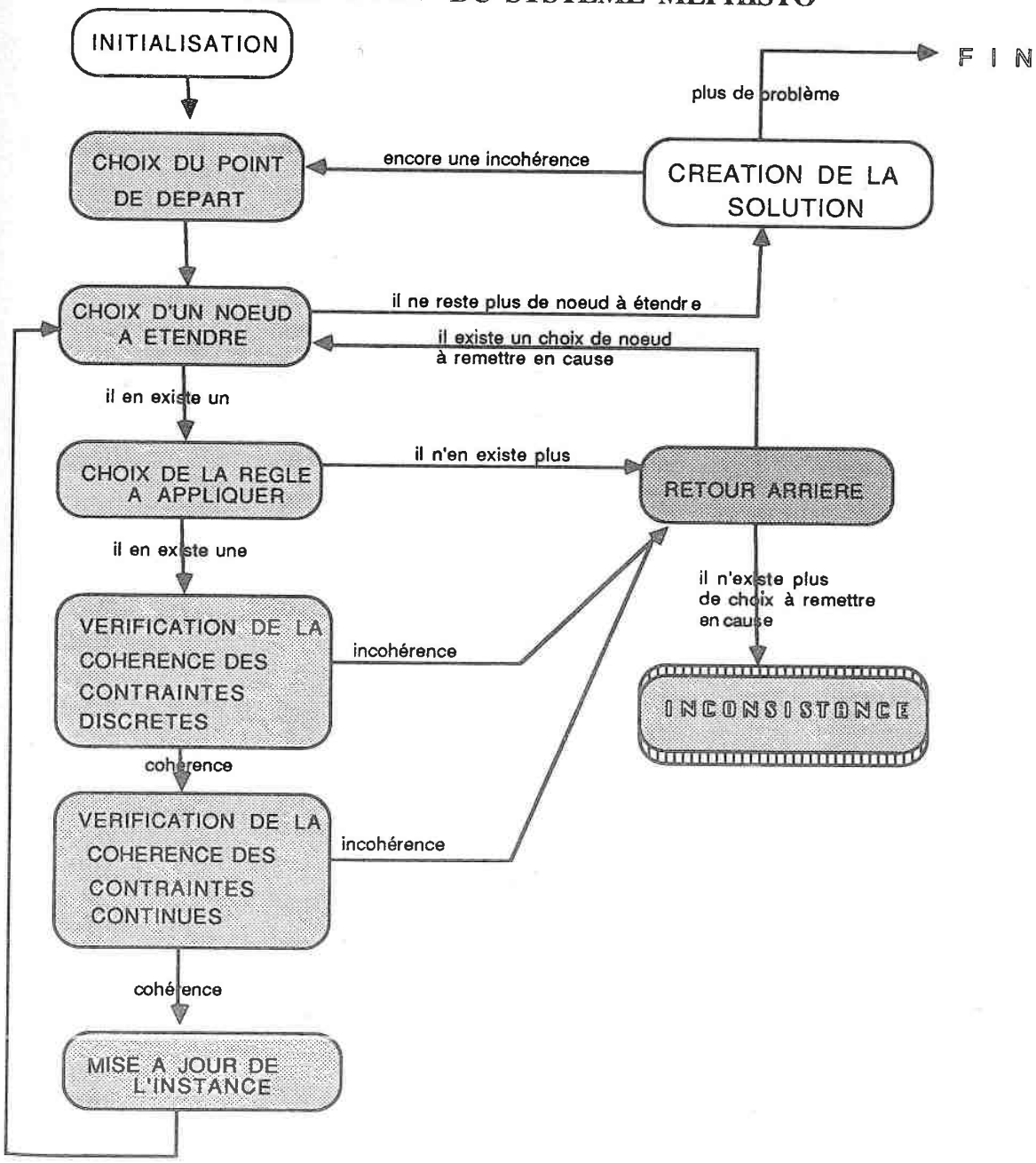
Une étape descendante consiste à étendre l'*une des feuilles possibles*, c'est-à-dire { B, E, H } : par exemple, l'extension de E par la règle /12/ donne :



La construction d'une instance est terminée lorsque tous les noeuds possibles sont étendus, c'est-à-dire dès que la racine est étiquetée par *l'axiome*, et chaque feuille par un *nom de primitive*.

Ce principe de construction permet d'entamer la construction d'une instance par une forme quelconque : celle-ci peut être imposée par l'utilisateur, ou bien choisie automatiquement par le système parmi les **formes obligatoires** du modèle, puisqu'une telle forme appartient à tous les chemins possibles du graphe ET/OU. Le calcul de l'ensemble des formes obligatoires d'un modèle a été donné au chapitre 2 § 3.3. Cet ensemble n'est jamais vide, car par construction, il contient toujours l'axiome. Lorsque celui-ci est choisi comme point de départ, la construction se fera uniquement par des phases descendantes successives.

STRUCTURE DU SYSTEME MEPHISTO



Comme le montre la figure précédente, chaque étape, **ascendante** ou **descendante**, est constituée de quatre parties : chacune d'elles sera détaillée dans un paragraphe ultérieur. Une étape débute par le **choix du noeud** à étendre parmi ceux restant à traiter. S'il n'en existe plus, c'est-à-dire si l'instance construite est complète, le système effectue une dernière vérification de la cohérence des contraintes, avant de proposer la solution trouvée au concepteur. Une fois un noeud choisi parmi ceux possibles, le système MEPHISTO **choisit une règle** parmi toutes les règles permettant d'étendre ce noeud. Une **inconsistance partielle** est détectée par MEPHISTO lorsque la règle choisie provoque une incohérence des contraintes continues ou discrètes. Dans ce cas, le système effectue un **retour-arrière**, en remettant en cause le dernier choix effectué.

Si, lors d'une étape, aucune inconsistance partielle n'est détectée, MEPHISTO intègre les contraintes de la règle choisie à celles de l'instance partiellement construite. Dans une phase de propagation, il teste ensuite la validité de l'ensemble des contraintes obtenues : on distinguera **propagation continue**, qui traite les inégalites continues et **propagation discrète** qui traite les contraintes discrètes. MEPHISTO met à jour également sa base de connaissances dynamiques et l'étape est terminée, avec succès.

L'inconsistance globale est montrée lorsque toutes les arborescences possibles ont été construites sans succès.

6.2 - CHOIX DE NOEUD ET DE REGLE

A une étape donnée, le système doit **choisir l'un des noeuds** de l'arbre, afin de le compléter soit de manière ascendante, soit de manière descendante. Pour chacun de ces noeuds, plusieurs règles sont applicables.

Etant donné la complexité et la taille des modèles traités, le choix de noeud et de règle ne peut pas être laissé au hasard. Nous allons tenter de **limiter le caractère exponentiel** du processus de validation, en introduisant des **heuristiques spécialisées**. Celles-ci doivent permettre de trouver le plus rapidement possible, soit une instance satisfaisante, soit une inconsistance dans le modèle.

Avec la terminologie introduite au paragraphe 3, le choix de noeud est de type **ET** : en effet, il faudra bien étendre **tous les noeuds** pendants de l'arbre qui ne sont pas des primitives. De façon symétrique, le choix de règle est de type **OU**, puisque, pour étendre un noeud, **une seule des règles** possibles sera utilisée. Nous allons donc pouvoir appliquer ici les résultats obtenus précédemment.

Rappelons que cette stratégie propose, dans le cas de choix OU, de comparer les règles deux à deux, en utilisant le préordre défini par :

$$R_1 < R_2 \quad \Leftrightarrow \quad OU\,(R_1, R_2)$$

$$\text{avec} \quad OU\,(R_1, R_2) \;=\; C_1 + (1 - S_1) * C_2$$

De même, dans le choix ET, la comparaison de deux 2 noeuds à étendre utilise la fonction suivante :

$$ET\;(N_1, N_2) \;=\; C_1 + S_1 * C_2$$

L'intérêt de cette stratégie ayant déjà été montré, il reste à trouver la définition des fonctions *coût* et *espérance de succès*. Dans notre cas, la détermination de ces fonctions ne conduira qu'à des résultats approchés.

6.2.1 - La fonction COUT

Sur quels *critères* peut-on **estimer le coût** d'un noeud et celui d'une règle ? De façon générale, étendre un noeud donné par une règle donnée fait l'objet d'une étape de la validation. La détermination du coût du choix d'un noeud ou d'une règle doit donc être guidée par le coût de l'étape. En examinant les différentes phases, on remarque que ce coût est directement lié à celui de la phase de vérification de consistance des contraintes.

Si on confond coût et temps d'exécution, nous définissons donc le coût d'une étape comme étant proportionnel au temps d'exécution de la phase de propagation continue. Comme nous le verrons au paragraphe 6.3, ce temps est très élevé, en raison de la complexité des algorithmes utilisés. Il dépend bien évidemment de la **complexité du système de contraintes** à traiter.

Le premier critère d'estimation peut être le nombre de contraintes du système à traiter. Dans un premier temps, on peut considérer que plus ce nombre est grand, plus le temps de propagation, donc le coût de l'étape, sera élevé. Toutefois, cette estimation simple ne donne qu'un résultat très grossier. En effet, selon ce critère, le coût du système .

$$\begin{aligned} X &\leq 12 \\ Y &\leq 10 \\ Z &\geq 22 \\ K &\leq 21 \end{aligned}$$

est supérieur au coût du système

$$\begin{aligned} X + Y &\leq Z + K - 2 \\ X - 2*K &\geq Y + 1 \end{aligned}$$

alors que les inégalités du premier système sont plus simples à étudier que celles du second.

Le nombre de contraintes du système à traiter n'est donc pas assez significatif pour être retenu. Il en est de même pour un critère tenant compte uniquement du nombre de variables total intervenant dans les contraintes. Ceci est encore bien montré par l'exemple précédent où les deux systèmes à comparer portent sur les mêmes variables { X, Y, Z, K }.

En fait, la complexité dépend plutôt de la forme de chaque contrainte. En effet, la propagation de la contrainte $X \leq 12$ ne coûte rien, puisque, dans ce cas, le travail du système de vérification se limite à modifier la borne supérieure de X. Par contre, la contrainte

$$X + Y \leq Z + K - 2$$

nécessite l'examen des variables Y, K et Z, et des contraintes qui portent sur elles.

En tenant compte de cette remarque, on estime donc que la **complexité** d'une inégalité est égale au **nombre d'attributs** - de variables - qui y apparaissent. De cette façon, le coût de la phase de validation, et donc d'une étape, est défini comme la somme des coûts des inéquations à traiter.

Comme une étape correspond à l'extension d'un noeud par une règle donnée, on assimile le coût d'une règle à celui de l'étape. On arrive donc à la définition suivante :

$$C(r) = \sum_{contr \in CT_r} \text{NB-ATT}(contr) \qquad r \in \mathcal{R}$$

où la fonction NB-ATT (contr) retourne le nombre d'attributs apparaissant dans contr

Comment définir le coût d'un noeud?

Il parait raisonnable de considérer que celui-ci dépend du nombre de possibilités que le noeud introduit, c'est-à-dire du nombre de règles possibles pour son extension. Nous estimons donc le coût d'un noeud par la somme des coûts des règles possibles : en phase ascendante, il s'agit des règles qui utilise la forme étiquetant le noeud :

$$C(n) = \sum_{r \in \mathcal{R} \text{ tq } \text{FORME-DE}(n) \in SF_r} C(r)$$

où FORME-DE (n) fournit le nom de la forme étiquetant le noeud n

En phase descendante, il s'agit des règles qui décrivent la forme étiquetant le noeud :

$$C(n) = \sum_{r \in \mathcal{R} \text{ tq FORME-DE}(n) = F_r} C(r)$$

où FORME-DE (n) fournit le nom de la forme étiquetant le noeud n

Pour finir, il nous reste à préciser à quel moment doit se faire le calcul du coût d'une règle. La première solution consiste à effectuer ce calcul **statiquement**, à l'acquisition du modèle. Déterminer le coût revient à compter le nombre d'attributs apparaissant dans les inégalités de la règle. Par exemple, pour la règle définie par

/1/ X → A B C

$$\begin{aligned} a &\leq b + c + 1 \\ b - d &\leq x + 1 \\ d - a &\leq 0 \\ x + k &\leq a - c - 2 \\ a &= 2*c - d \\ d &= a + 2*x - 2. \end{aligned}$$

le coût est estimé à 12.

Ce calcul ne tient pas compte des égalités de la règle. Si on simplifie les inégalités par les égalités (cf § 4.3 de ce chapitre) l'ensemble d'inéquations obtenu est réduit à

$$\begin{aligned} -x &\leq c \\ b - c &\leq 2*x \\ x - 1 &\leq 0 \\ 2*x + k &\leq -1 \end{aligned}$$

avec $a = c - x + 1$
$d = c - x - 1$

Après simplification, le coût de la règle est donc estimé à 8. Ainsi, on se rend compte que la prise en compte des égalités a une influence importante sur le calcul du coût d'une règle, même dans le cas d'une évaluation statique.

Une évaluation statique du coût des règles a l'avantage de ne pas ralentir la construction d'une arborescence. Cependant, elle ne tient pas compte du tout de l'environnement dans lequel la règle sera utilisée. Effectuer l'estimation du coût **dynamiquement** dès qu'un problème de choix se pose, permet de prendre en considération toutes les égalités associées à l'arborescence en cours de création : par simplification des inégalités de la règle par les égalités de la règle et de l'arbre, l'estimation du coût sera plus précise. Il est important de noter que cette simplification ne garantit en aucun cas une diminution du coût, mais plutôt une *meilleure approximation* de celui-ci.

Illustrons ceci sur l'exemple précédent. Si les égalités associées à l'arbre sont .

$$c = x$$
$$a = z - k$$
$$u = d$$

le nouvel ensemble d'inégalités de la règle /1/ simplifiées par l'ensemble de toutes les égalités devient

$$0 \leq b + x$$
$$b - 3*x \leq 0$$
$$d \leq 1$$
$$z + x \leq 0$$

avec $a = 1$
$d = 2*c - 1$
$k = z - 1$
$u = 2*c - 1$

Le coût est ici estimé à 7.

6.2.2 - La fonction ESPERANCE

Contrairement au coût, l'**espérance de succès** d'une règle ou d'un noeud ne peut pas être évaluée indépendamment de tout contexte : son évaluation est **dynamique** car elle doit tenir compte des hypothèses émises sur la construction de l'arborescence.

Comment utiliser au mieux les informations dynamiques pour donner une bonne estimation de l'espérance d'une règle ? Nous avons vu au paragraphe 6.1 décrivant la structure générale de MEPHISTO, qu'un échec ne peut être provoqué que par une inconsistance des contraintes continues ou discrètes. Ainsi, la fonction espérance doit donner un résultat inversement proportionnel au **risque d'inconsistance**, en cas d'utilisation de la règle considérée.

A partir de cette remarque, il reste à évaluer le risque d'inconsistance d'un ensemble de contraintes. En pratique, on constate que l'ensemble des contraintes discrètes est peu important : il nous parait donc raisonnable de limiter l'évaluation de ce risque aux contraintes continues.

De façon intuitive, on peut estimer que le risque d'inconsistance d'un ensemble de contraintes continues est d'autant plus grand que l'influence des contraintes entre elles est importante. Par exemple, le risque d'inconsistance du système

(1)
$$\begin{aligned} X &\leq A + 1 \\ Y &\leq Z + 2 \\ U &\leq T + 1 \end{aligned}$$

parait plus faible que celui du système

(2)
$$\begin{aligned} Z &\leq K + A \\ X &\leq A + K - 1 \\ X &\geq 2B - 1 \\ Y + A &\leq Z + 2K - 3 \end{aligned}$$

En effet, les domaines de variation des attributs { A, B, K, X, Y, Z } du système (2) peuvent être influencés par les quatre contraintes : l'un d'entre eux a donc plus de chances de devenir vide. Alors que dans le système (1), les attributs sont simplement liés deux par deux : le risque parait donc moins grand.

L'extension de ce principe à un ensemble quelconque d'attributs $\mathcal{A}$ et de contraintes $\mathcal{S}$ fait intervenir la relation **est lié à**, notée ∇ , portant sur des attributs apparaissant dans les contraintes. Cette relation d'ordre total est définie par

$a_k \in \mathcal{A} \qquad a_l \in \mathcal{A}$

$a_k \nabla a_l$ si et seulement si a_k et a_l apparaissent dans la même contrainte

On peut maintenant estimer que le risque d'inconsistance d'un système de contraintes $\mathcal{S}$ est **inversement proportionnel au nombre de classes** d'équivalence de la relation ∇, puisque, intuitivement, plus le nombre de classes est grand, plus les attributs sont indépendants, et donc plus leur domaine de variation a des chances de peu varier. En cas d'égalité du nombre de classes, on évalue le risque en fonction du cardinal de chaque classe.

Dans l'exemple précédent, les classes d'équivalence de la relation pour les systèmes (1) et (2) sont respectivement

{ X, A }
{ Y, Z } et { A, B, X, Y, Z, K }
{ U, T }

Le nombre de classes derivées du système (1) étant le plus grand, le risque d'inconsistance est estimé comme étant le plus faible : 0.33 au lieu de 1.

Toutefois, cette estimation ne fournira pas les résultats attendus dans tous les cas. Ceci se voit clairement sur l'exemple suivant :

(1) $a \leq 10$
$a \geq 12$

(2) $Z \leq K + a$
$x \leq a + K + t$

Les classes dérivées respectivement des systèmes (1) et (2) sont

{ a }	**pour le système (1)**
{ a x t Z K }	**pour le système (2)**

L'heuristique décrite précédemment nous conduit à attribuer au système (1) un risque d'inconsistance plus faible qu'au système (2), alors que le risque du système (1) devrait être maximum. Ceci vient du fait que cette stratégie ne tient pas compte de la forme réelle des contraintes, mais uniquement des attributs concernés. Cependant, la prise en compte de ces informations nécessiterait une propagation complète des contraintes, ce qui est abherrant.

L'estimation choisie ne sera donc pas entièrement fiable, mais rappelons qu'une mesure exacte de l'espérance de succès est impossible à déterminer.

En résumé, on estime que l'espérance de succès d'une règle est directement proportionnelle au nombre de classes d'équivalence de la relation **est lié à**.

$$S(r) = \mathrm{card}(\mathcal{C}\mathbb{L}) / 1000$$

où $\mathcal{C}\mathbb{L}$ = { classes d'équivalence de la relation ∇ sur les attributs de $\mathcal{C}$ }

avec $\mathcal{C}$ = { contraintes de la règle r } $\cup$ { contraintes de l'arbre }

Il nous reste à définir l'*espérance de succès d'un noeud*. Tout comme le coût d'un noeud est fonction des coûts des règles possibles pour son extension, l'espérance d'un noeud dépend des espérances des règles possibles; elle est estimée par la moyenne des différentes espérances. En phase ascendante, l'espérance d'un noeud est défini par :

$$S(n) = \text{Moyenne}_{r \in \mathcal{R} \text{ tq FORME-DE}(n) \in SF_r} \left(S(r) \right)$$

Et en phase descendante,

$$S(n) = \text{Moyenne}_{r \in \mathcal{R} \text{ tq FORME-DE}(n) = F_r} \left(S(r) \right)$$

6.2.3 - Un exemple

Le modèle suivant décrit les formes SCENE, B, C, D et E. Les primitives sont P1, P2 et P3.

/1/ **SCENE** → **B C**

$x(SCENE) \geq x(B) + x(C)$
$y(SCENE) = y(B)$
$h(SCENE) \geq h(B) - 10$
$h(SCENE) \leq h(C) + 23$
$25 \leq z(SCENE) \leq 200$
$y(SCENE) = y(C)$

/2/ **SCENE** → **B D E**

$x(SCENE) \geq 44 - y(E) + y(D)$
$y(B) \geq 20$

/3/ **B** → **P1 P2**

$h(P1) \leq 10 + h(B)$
$y(B) \geq y(P1) + y(P2) - z(P2)$
$x(B) + x(P1) \leq x(P2)$

/4/ **B** → **D E P2**

$y(P2) + y(E) \leq 2 * y(D) - z(B) + 11$
$x(B) \leq 34$
$y(E) \leq 123$

/5/ **C** → **P1 P3**

$2 * x(P1) + x(P3) \geq x(C)$
$y(C) - y(P1) \geq 25$
$z(P3) = 12$

/6/ C → **P1 P2**

$x(C) - x(P1) \geq 28$
$y(C) \geq y(P1) * 2$

/7/ D → **C P3**

$x(D) + x(C) \geq x(P3)$
$z(P3) - z(C) \geq y(D) + 1$
$y(P3) \geq 10$
$z(C) \leq 2 * z(D) - z(P3) + 21$

/8/ E → **D P1**

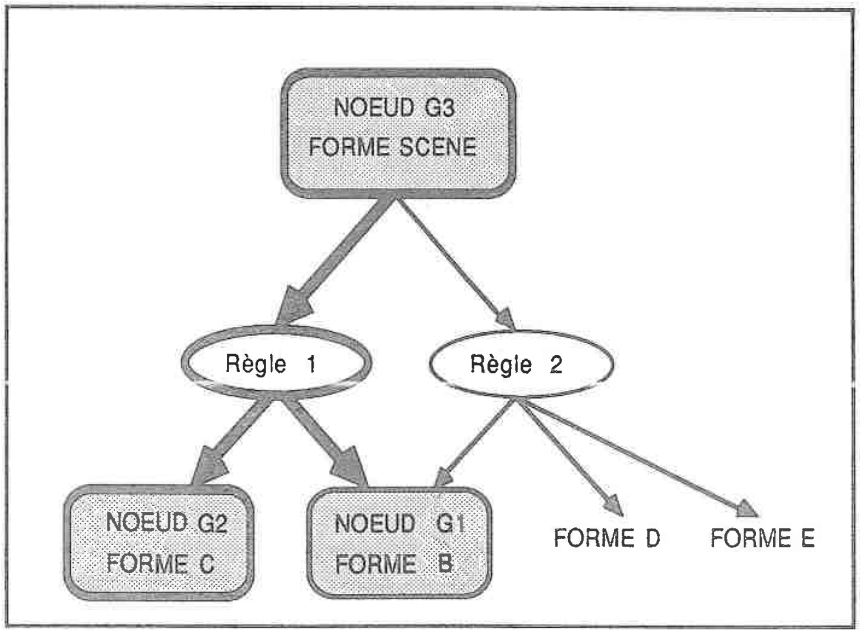
$y(E) \leq y(D) - y(P1) + 36$

La création d'une instance débute par la forme B. Les pages suivantes donnent le déroulement de la création. A chaque étape, on indique quels sont les **choix possibles d'extension de noeud**. Pour effectuer ce choix, on calcule le coût et l'espérance de succès de chaque règle applicable à chacun de ces noeuds. L'application des formules établies dans les paragraphes précédents permet, dans un premier temps de choisir un noeud de l'instance à compléter. Par la suite, on **choisit également la règle** à appliquer à ce noeud : il s'agit de la règle de rapport coût / espérance de succès minimal.

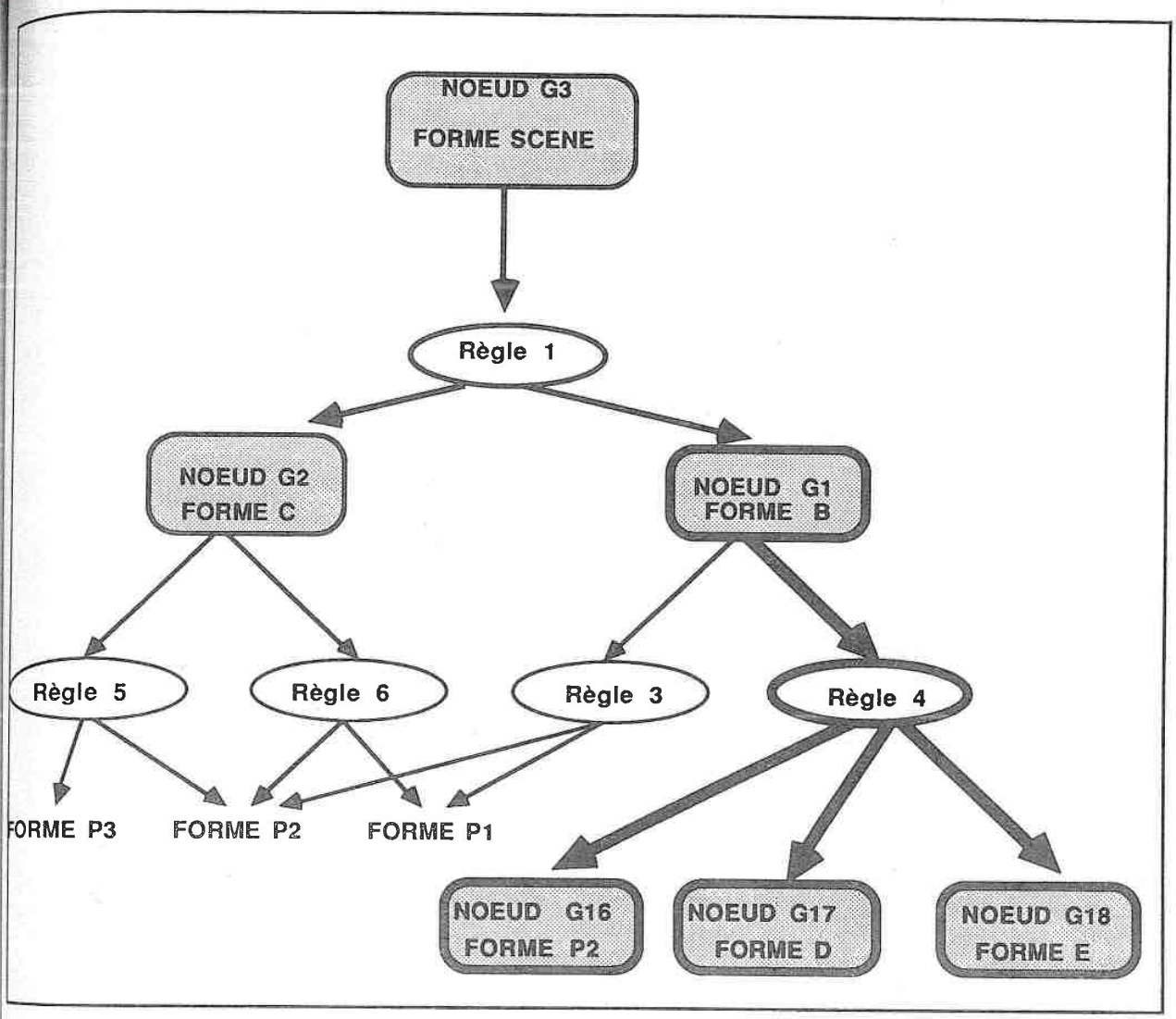
A la fin de chaque étape, on trouvera également un schéma de l'instance en cours de construction. Pour chaque noeud à étendre, il indique les différentes possiblilités d'extension. Le chemin choisi est dessiné en gras.

Les étapes n'offrant aucun intérêt pour l'illustration des choix de noeud et de règle ne sont pas détaillées.

```
**************************************

E T A P E  0

*** CHOIX D'UN NOEUD PARMI (g1)

    Noeud g1  Forme associee b

*** Collecte des contraintes continues de la regle r_2
    Cout de la regle r_2 9
    Nombre de composantes connexes de r_2 3

*** Collecte des contraintes continues de la regle r_1
    Cout de la regle r_1 9
    Nombre de composantes connexes de r_1 4

    Noeud choisi g1

*** ANALYSE ASCENDANTE ***

*** CHOIX D'UNE REGLE PARMI (r_2 r_1)
 >> r_2 d'esperance .003 et de cout 9
 >> r_1 d'esperance .004 et de cout 9
    Choix dans (r_2 r_1) de la regle de rapport cout/esperance minimal

    Regle choisie r_1

*** L'ETAPE 0 S'EST BIEN PASSEE ***

*** INSTANCE
(scene (c) (b) )
```



```
**************************************

E T A P E  1

*** CHOIX D'UN NOEUD PARMI (g2 g1)

    Noeud g2  Forme associee c

*** Collecte des contraintes continues de la regle r_6
    Cout de la regle r_6 4
    Nombre de composantes connexes de r_6 4

*** Collecte des contraintes continues de la regle r_5
    Cout de la regle r_5 5
    Nombre de composantes connexes de r_5 5

    Noeud g1  Forme associee b

*** Collecte des contraintes continues de la regle r_4
    Cout de la regle r_4 6
    Nombre de composantes connexes de r_4 5

*** Collecte des contraintes continues de la regle r_3
    Cout de la regle r_3 9
    Nombre de composantes connexes de r_3 4

    Noeud choisi g1

*** ANALYSE DESCENDANTE ***

*** CHOIX D'UNE REGLE PARMI (r_4 r_3)
 >> r_4 d'esperance .005 et de cout 6
 >> r_3 d'esperance .004 et de cout 9
    Choix dans (r_4 r_3) de la regle de rapport cout/esperance minimal

    Regle choisie r_4

*** L'ETAPE 1 S'EST BIEN PASSEE ***

*** INSTANCE
(scene (c) (b (p2) (e) (d)))
```

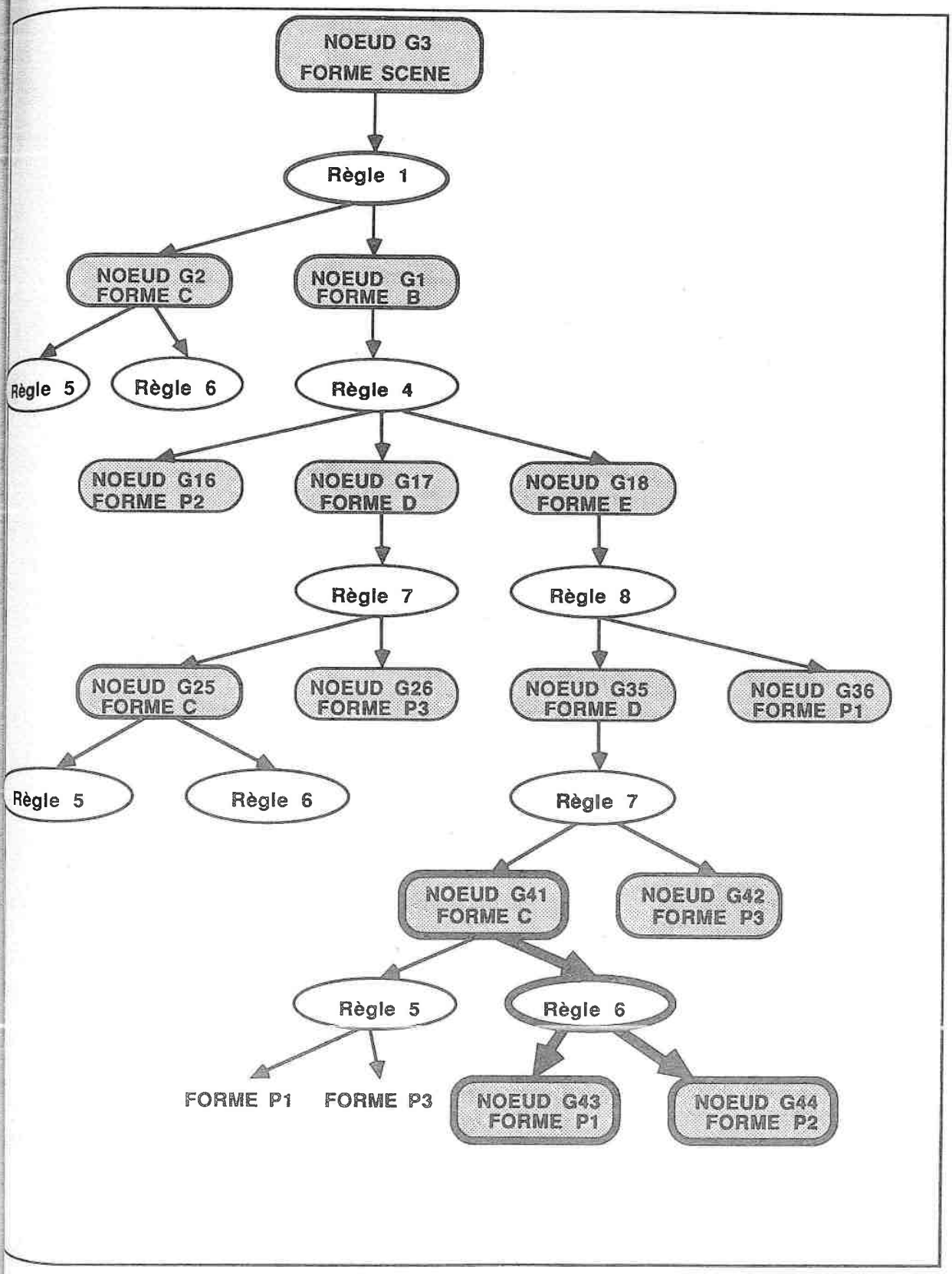


CHOIX DE NOEUD ET DE REGLE
A L'ETAPE 1

```
**************************************

E T A P E  5

*** CHOIX D'UN NOEUD PARMI (g41 g25 g2)

    Noeud g41  Forme associee c

*** Collecte des contraintes continues de la regle r_6
    Cout de la regle r_6 4
    Nombre de composantes connexes de r_6 10

*** Collecte des contraintes continues de la regle r_5
    Cout de la regle r_5 5
    Nombre de composantes connexes de r_5 11

    Noeud g25  Forme associee c

*** Collecte des contraintes continues de la regle r_6
    Cout de la regle r_6 4
    Nombre de composantes connexes de r_6 10

*** Collecte des contraintes continues de la regle r_5
    Cout de la regle r_5 5
    Nombre de composantes connexes de r_5 11

    Noeud g2  Forme associee c

*** Collecte des contraintes continues de la regle r_6
    Cout de la regle r_6 4
    Nombre de composantes connexes de r_6 9

*** Collecte des contraintes continues de la regle r_5
    Cout de la regle r_5 5
    Nombre de composantes connexes de r_5 10

    Noeud choisi g41

*** ANALYSE DESCENDANTE ***

*** CHOIX D'UNE REGLE PARMI (r_6 r_5)
 >> r_6 d'esperance .01 et de cout 4
 >> r_5 d'esperance .011 et de cout 5
    Choix dans (r_6 r_5) de la regle de rapport cout/esperance minimal

    Regle choisie r_6


*** L'ETAPE 5 S'EST BIEN PASSEE ***

*** INSTANCE
(scene (c) (b (p2) (e (p1) (d (p3) (c (p2) (p1)))) (d (p3) (c))))
```

CHOIX DE NOEUD ET DE REGLE

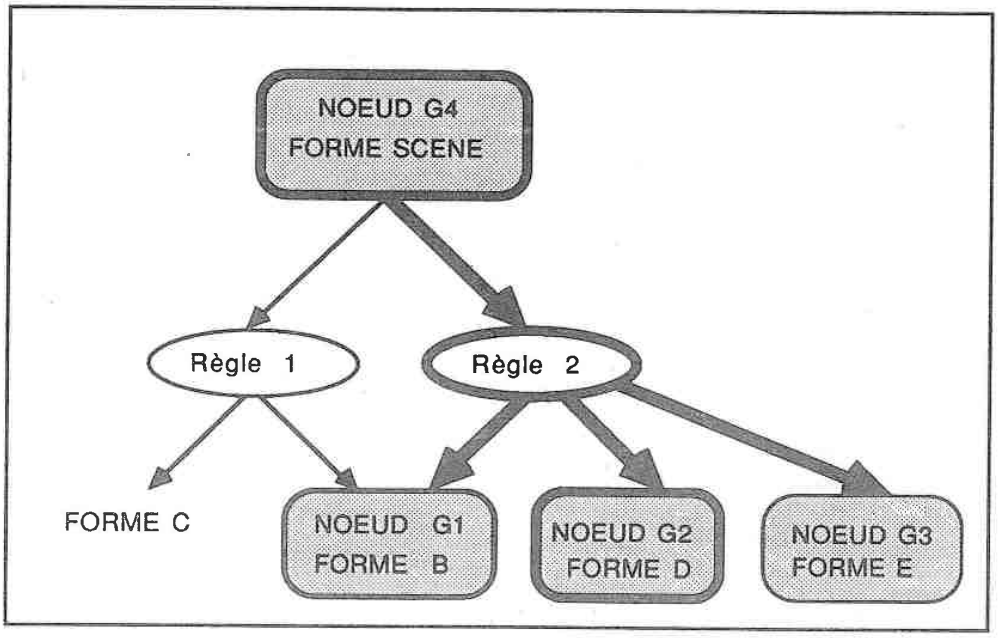
A L'ETAPE 5



La modification de l'ensemble des contraintes associées à l'une des règles modifie totalement les choix effectués. Par exemple, si on ajoute à la règle /2/ les contraintes

$$h(B) + h(D) - h(E) \leq 400$$
$$x(SCENE) \leq x(B) + 10$$
$$x(D) = x(B) - x(E)$$

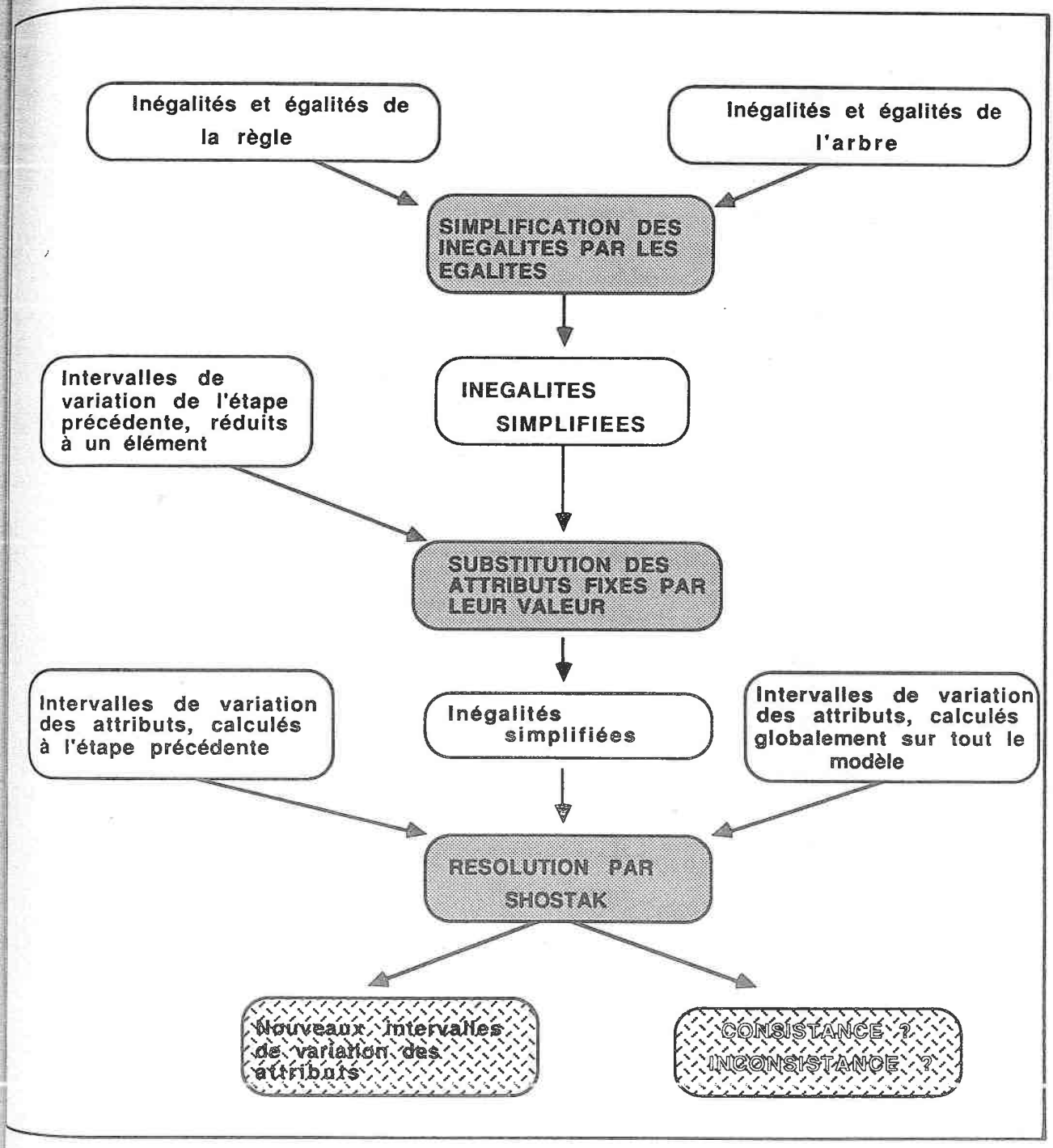
dès l'étape 0, le choix de règle à appliquer est différent : à coûts comparables, la règle /2/ a une espérance de succès plus faible que celle de la règle /1/ : c'est donc la règle /2/ qui est choisie. L'instance que l'on construit dans ce cas sera totalement différente de celle construite précédemment.



6.3 - PROPAGATION CONTINUE

L'ensemble des contraintes continues à traiter est composé de l'ensemble consistant des **contraintes continues associées à l'arbre** depuis le début de la construction et de l'**ensemble des contraintes continues associées à la règle** considérée. On retrouve donc un problème développé et résolu au paragraphe 4.1 de ce chapitre.

Le schéma suivant donne la structure générale du module de propagation continue.



PROPAGATION CONTINUE

La première étape consiste à **simplifier** l'ensemble complet des inégalités par celui des égalités. L'ensemble obtenu peut alors être traité par la méthode de résolution proposée par BLEDSOE. Rappelons toutefois que cet algorithme procède par diminutions successives des intervalles de variation des variables en fonction des inégalités traitées. La convergence vers l'intervalle final ou l'intervalle vide sera obtenue d'autant plus rapidement que l'intervalle de départ sera petit : ainsi, il s'avère intéressant de mémoriser, d'une étape à l'autre, les domaines de variation des attributs : ceci évite à l'algorithme de résolution d'effectuer à chaque étape à nouveau les mêmes calculs.

En d'autres termes, cela signifie qu' à l'étape i, on utilise les i*ntervalles de variation* des attributs calculés à l'étape i-1, - pour les attributs qui existaient à cette étape bien sûr. Ceci revient en fait à tenir compte des résultats des propagations antérieures afin de simplifier celle de l'étape courante.

Ce mécanisme de réutilisation des domaines des attributs existant à l'étape i-1 *accélère* la phase de validation. De plus, il permet une simplification supplémentaire des inégalités : en effet, si à l'étape i-1, l'attribut x a comme intervalle de variation [3 .. 3], il est substitué par la valeur 3 partout où il apparait.

Il reste toutefois à considérer le cas des attributs introduits seulement par les contraintes de la règle à l'étape i : ils n'existaient pas à l'étape i-1, et leur domaine est donc indéfini.

Une solution consiste à laisser une initialisation commune à tous les nouveaux attributs, par exemple] $-\infty$, $+\infty$ [, comme cela était prévu au départ. Cependant, étant donné l'importance d'une diminution initiale des intervalles dans le module de résolution, il serait intéressant d'utiliser, pour chaque attribut, un encadrement approché qui soit indépendant de tout contexte. Cet intervalle initial doit donc être calculé sur l'ensemble des contraintes du modèle. C'est ce que nous permet de faire la méthode de résolution de systèmes ET/OU d'inéquations décrite au paragraphe 4.2. Ainsi, avant toute création d'instance, MEPHISTO applique cette méthode pour déterminer le domaine de variation de chaque attribut, indépendamment de toute règle. Cela fournit un encadrement qui permettra de réduire l'indéterminisme lors de la première utilisation de chacun de ces attributs.

6.4 - PROPAGATION DISCRETE

L'ensemble des contraintes discrètes à traiter est composé des **contraintes discrètes associées à l'arbre** auxquelles on adjoint celles associées à la règle considérée. Le problème de la **vérification de cohérence** d'un tel système a été résolu au paragraphe 5 de ce chapitre.

A partir d'un ensemble de contraintes discrètes, le module de propagation discrète construit *le graphe des relations* entre les attributs : relation *d'égalité* ou de *différence*. Rappelons qu'un noeud du graphe représente une **classe d'équivalence** de la relation **est égal à** sur les attributs. Un arc du graphe traduit une différence entre les attributs de deux classes d'équivalence. L'algorithme utilisé essaie de montrer la consistance du graphe en affectant à chaque classe (donc à chaque noeud du graphe) un ensemble d'étiquettes le plus petit possible.

Le graphe est complété au fur et à mesure de la construction d'une arborescence. Au départ, il est vide; à chaque étape, le module de propagation des contraintes discrètes y inclut les contraintes de la règle appliquée.

Comme dans le cas de la propagation continue, l'algorithme procède par affinements successifs de l'ensemble des valeurs possibles des classes : plus le cardinal de l'ensemble de départ sera faible, plus court sera le temps de propagation. Il est donc essentiel de mémoriser à chaque étape les nouveaux domaines des classes, afin que ceux-ci soient réutilisés à l'étape suivante. Pour les attributs introduits à l'étape i, l'algorithme initialise l'ensemble des valeurs possibles au domaine tout entier.

Pour finir, remarquons que l'algorithme utilisé ne vérifie que la consistance locale du graphe construit (cf § 5.2). Ce qui signifie qu'une fois une arborescence construite, l'ensemble des contraintes discrètes associées aux règles de l'arbre n'est pas forcément globalement consistant.

6.5 - RETOUR-ARRIERE

Lors d'une étape k, un **retour-arrière** est provoqué lorsque la règle choisie pour étendre un noeud de l'arbre provoque une inconsistance des contraintes continues ou discrètes.

Lorsque l'incohérence provient des **contraintes continues**, c'est en fait l'adjonction des contraintes de la règle à celles de l'arborescence partiellement construite qui a provoqué l'échec, puisque, par construction, l'ensemble des contraintes de l'étape k-1 était consistant. Sans pouvoir en être absolumment sûr, on peut toutefois supposer qu'il faut *remettre en cause* le dernier choix de règle effectué, soit à l'étape k s'il reste encore des règles à appliquer au noeud choisi, soit à l'étape k-1 sinon. Il est donc inutile de mettre en place des techniques de retour-arrière sophistiquées.

Par contre, si l'inconsistance vient des **contraintes discrètes**, la cause principale de l'échec ne se trouve pas forcément à l'étape k, puisque, comme nous l'avons vu au paragraphe précédent, seule la consistance locale est vérifiée à chaque étape. L'échec peut donc être du à l'inconsistance *locale* des contraintes de la règle appliquée à l'étape k ou bien à l'inconsistance *globale* des contraintes associées à l'arborescence de l'étape k. Dans ce cas, peut-être a-t-on intérêt à retrouver une cause plus précise de l'échec. Examinons tout d'abord ce problème à travers quelques simples.

Exemple 1 : Contraintes de l'arborescence

a = b
b # c

Contraintes de la règle appliquée à l'étape k

d = c
d # c

Dans cet exemple, l'incohérence est provoquée uniquement par les contraintes discrètes de la règle.

Exemple 2 : Contraintes de l'arborescence

$a = b$
$b \# c$

Contraintes de la règle appliquée à l'étape k

$d = c$
$a = d$

Dans cet exemple, l'incohérence est provoquée par l'ensemble des contraintes.

Exemple 3 : Contraintes de l'arborescence

$a \# b$
$b \# c$
$a \# c$
$a \in \{ 1, 2 \}$
$b \in \{ 1, 2 \}$
$c \in \{ 1, 2 \}$

Contraintes de la règle appliquée à l'étape k

$a \in \{ 1 \}$

Dans cet exemple, l'incohérence existait déjà à l'étape k, mais n'avait pas été détectée

Les exemples précédents illustrent simplement les divers cas possibles. Dans les cas 1 et 2, il suffit de remettre en cause le choix de règle effectué à l'étape k, puisque ses contraintes discrètes ont de fortes chances d'avoir provoqué l'échec. Par contre, dans le dernier cas, l'utilisation d'une autre règle, donc la poursuite de l'étape k, n'est pas envisageable, puisque l'arborescence partiellement construite à l'étape k-1 contient un ensemble de contraintes discrètes inconsistant. Il est donc essentiel de remettre en cause le choix de règle effectué à l'étape k-1.

Reste à déterminer lequel de ces deux ensembles de contraintes discrètes, celui associé à la règle appliquée à l'étape k et celui associé à l'arborescence en cours, est cause de l'inconsistance. La technique employée consiste, lors d'une incohérence détectée à l'étape k, à montrer la consistance globale (cf § 5.3) de l'ensemble des contraintes de l'arborescence. Si MEPHISTO y parvient, cela signifie que les contraintes discrètes de la règle sont cause de l'échec; sinon, cela prouve que le problème vient des contraintes des règles utilisées avant l'étape k.

Dans le cas d'une *inconsistance locale* des contraintes discrètes de la règle en cours, MEPHISTO poursuit la construction d'une arborescence en appliquant une autre règle à l'étape k, s'il en existe une. S'il n'en existe plus, c'est-à-dire si elles ont toutes été essayées en vain, MEPHISTO provoque la remise en cause du dernier choix de règle effectué à l'étape k-1. Si l'étape k-1 n'existe pas dans l'historique de la construction de l'arbre, une inconsistance générale du modèle est détectée et signalée à l'utilisateur.

Dans le cas d'une *inconsistance globale* de l'ensemble des contraintes discrètes des règles appliquées avant l'étape k, MEPHISTO provoque sans tarder un retour-arrière à l'étape k-1, supposée en partie cause de l'erreur. De même que précédemment, pour savoir si c'est effectivement la règle appliquée à l'étape k-1, MEPHISTO lance l'algorithme de **propagation globale** sur l'ensemble des contraintes associées à l'arborescence à l'étape k-2. Ce même processus est itéré en remettant en question la construction de l'arbre étape par étape, jusqu'à trouver la règle effectivement cause de l'inconsistance.

Pour finir, précisons que chaque fois que le système MEPHISTO remet en cause un choix de règle, il signale ce fait à l'utilisateur, puisque cela signifie qu'une contrainte au moins de la règle appliquée est incohérente avec les autres. Afin de permettre à l'utilisateur de localiser précisément la cause de l'échec, MEPHISTO lui communique le sous-ensemble des attributs concernés par cet échec. En effet, même si l'inconsistance est détectée sur un attribut discret ou continu particulier, rien ne permet de dire si cet attribut est vraiment cause de tout : l'incohérence provient d'un ou de plusieurs attributs liés à celui-ci par une contrainte.

6.6 CREATION D'UNE SOLUTION

Lorsque le système MEPHISTO a trouvé une **instance satisfaisante**, le problème de la validité n'est pas encore tout à fait résolu. En effet, rappelons que l'instance constituée ne tient compte, pour les attributs dits continus, que des contraintes linéaires fournies par le concepteur du modèle. Les contraintes prédéfinies non linéaires (cf § 3.2.7 Chapitre 2) ont été volontairement ignorées, par manque de méthode précise de résolution. Il convient donc de vérifier que l'instance créée indépendamment des contraintes non linéaires ne contient pas d'inconsistance, lorsque ces contraintes sont prises en compte.

C'est à ce stade que se sont posées les plus grandes difficultés. A l'heure actuelle, il n'existe pas de méthode mathématique permettant de connaitre *exactement* l'ensemble des solutions d'un système quelconque d'inéquations et d'équations non linéaires portant sur des variables réelles. On ne trouve que quelques méthodes qui, dans certains cas favorables, fournissent une *solution approchée*. Dans les autres cas, ces méthodes ne permettent pas de décider si le système a ou non une solution.

Ce problème peut être résolu en utilisant une **extension de la méthode de Bledsoe** (cf § 4.1 Chapitre 3) au traitement des contraintes non linéaires. L'extension de l'algorithme a été proposée par BROOKS dans le système ACRONYM [BROO81]. Cette solution, envisagée dans un premier temps a été abandonnée. Elle a l'avantage de s'intégrer assez simplement au reste du système. Mais, le coût de l'algorithme est prohibitif par rapport à la qualité des résultats.

La seconde technique etudiée relève des **méthodes d'analyse numérique**, et plus particulièrement *d'optimisation de fonctions avec contraintes* (minimisation ou maximisation). De façon générale, ces méthodes tentent de résoudre des problèmes de la forme :

Soient $\mathcal{F}$ une fonction quelconque définie sur $\mathbb{R}$

$\mathcal{S}$ un ensemble de contraintes, de cardinal p

$\mathcal{V}$ l'ensemble des variables, de cardinal n

Chercher l'optimum de $\mathcal{F}$, sous les contraintes données par l'ensemble $\mathcal{S}$, c'est-à-dire, chercher $m^* \in \mathcal{V}$ vérifiant $\mathcal{S}$ et tel que $\mathcal{F}(m^*) = \text{optimum } \mathcal{F}(x) \quad x \in \mathcal{V}$

Dans le cas qui nous préoccupe, la fonction à minimiser peut être tout à fait *quelconque*, puisque la condition la plus importante est que la valeur m* trouvée vérifie bien toutes les contraintes de l'ensemble $\mathcal{S}$.

La plupart des méthodes résolvant ce type de problème imposent que les contraintes fournies soient *continument dérivables*. Parmi les techniques existantes, nous avons étudié celles proposées par ROSENBROCK et UZAWA.

A - Méthode de ROSENBROCK

ROSENBROCK procède par **itérations successives** à partir d'une idée de la solution ou à défaut d'un point quelconque de $\mathbb{R}^n$. La méthode tente d'approcher pas à pas l'une des solutions. A chaque itération, à partir du point courant de $\mathbb{R}^n$, on recherche la *meilleure direction* à prendre. Il existe 2n directions possibles. Un des gros problèmes de cette méthode réside dans le choix de la direction à prendre.

Cette technique est de mise en oeuvre relativement simple, dès que l'on a trouvé un bon critère de choix de direction. Elle a donc été implantée. En pratique, il s'est avéré que les résultats sont à peu près satisfaisants, sauf lorsque les contraintes présentent un caractère *oscillatoire*. Dans ce cas, on a toutes les chances d'osciller autout des solutions, sans jamais parvenir à les atteindre. La méthode de Rosenbrock a donc été abandonnée, car les contraintes non linéaires à traiter ici utilisent toutes des fonctions trigonométriques.

β - Méthode d'UZAWA

La méthode d'**UZAWA** est plus complexe. C'est une méthode itérative qui recherche le minimum à chaque itération. Les contraintes doivent être sous la forme :

$$\varphi_i(m) \leq 0 \quad \text{où} \quad m \in \mathbb{R}^n$$

A l'itération k, on calcule, dans un premier temps, m_k tel que

$$\mathcal{L}(m_k, \lambda^k) = \inf_{m \in \mathbb{R}^n} \mathcal{L}(m, \lambda^k)$$

où
- $L(m, \lambda)$ est le lagrangien de $\mathcal{F}$ c'est-à-dire

$$L(m, \lambda) = \mathcal{F}(m) + \sum_{i=1}^{p} \lambda_i \varphi_i(m)$$

- m_k est le point minimum trouvé à l'itération k
- λ_k est le poids donné aux contraintes à l'itération k.

On se ramène donc à un problème de minimisation de fonction à plusieurs variables sans contrainte, la fonction à minimiser étant :

$$L(m, \lambda) = \mathcal{F}(m) + \sum_{i=1}^{p} \lambda_i \varphi_i(m)$$

Ce problème peut être résolu en utilisant la méthode du gradient conjugué.

β - Méthode d'UZAWA

La méthode d'**UZAWA** est plus complexe. C'est une méthode itérative qui recherche le minimum à chaque itération. Les contraintes doivent être sous la forme :

$$\varphi_i(m) \leq 0 \quad \text{où} \quad m \in \mathbb{R}^n$$

A l'itération k, on calcule, dans un premier temps, m_k tel que

$$L(m_k, \lambda^k) = \inf_{m \in \mathbb{R}^n} L(m, \lambda^k)$$

où
- $L(m, \lambda)$ est le lagrangien de $\mathcal{F}$ c'est-à-dire

$$L(m, \lambda) = \mathcal{F}(m) + \sum_{i=1}^{p} \lambda_i \varphi_i(m)$$

- m_k est le point minimum trouvé à l'itération k
- λ_k est le poids donné aux contraintes à l'itération k.

On se ramène donc à un problème de minimisation de fonction à plusieurs variables sans contrainte, la fonction à minimiser étant :

$$L(m, \lambda) = \mathcal{F}(m) + \sum_{i=1}^{p} \lambda_i \varphi_i(m)$$

Ce problème peut être résolu en utilisant la méthode du gradient conjugué.

Dans un second temps, on calcule le nouveau vecteur λ^{k+1}; chaque coordonnée i de ce vecteur est définie par :

$$\lambda_i^{k+1} = max\ (\lambda_i^k + \rho\ \varphi_i(m_k)\ ,0)$$

La valeur de ρ est à déterminer par l'utilisateur, en fonction du problème étudié. Les itérations s'arrêtent lorsque :

$$| m_k - m_{k+1} | \leq \varepsilon$$

et $$| F(m_k) - F(m_{k+1}) | \leq \varepsilon'$$

Cette méthode n'a pas été implantée, étant donnée sa complexité de mise en oeuvre.

C - Solution choisie

La solution retenue pour résoudre ce problème nous a été dictée par l'étude des contraintes non linéaires à traiter. La *complexité* de celles-çi provient essentiellement des fonctions trigonométriques (sinus et cosinus). Si ces fonctions pouvaient disparaitre, les contraintes redeviendraient linéaires, donc plus simples à résoudre. Pour cela, nous avons choisi de **discrétiser** le domaine de variation des angles impliqués dans ces fonctions. Cette restriction parait raisonnable, d'autant plus que le pas de discrétisation n'est pas fixé a priori.

De cette façon, la résolution du système de contraintes non linéaires se ramène à celle d'un système ET/OU de contraintes linéaires. Par exemple, la résolution du système $\mathcal{S}$

$$y = a \sin(b) + x \cos(c*d)$$

$$z \leq a \cos(c*b) + y \cos(b)$$

avec $x, y, z, a, b, c, d \in \mathbb{R}$

se ramène à la résolution du système $\mathcal{S}$, avec

$$x, y, z \in \mathbb{R}$$

et $$a, b, c, d \in [\, 0,\ \pi/4,\ \pi/2,\ 3\pi/4,\ 2\pi\,]$$

On recherche le n-uplet (x, y, z, a, b, c, d) vérifiant l'un des 4^5 systèmes de contraintes obtenus.

Il reste à trouver une technique de résolution d'un ensemble de contraintes linéaires. Contrairement aux problèmes déjà rencontrés, on recherche ici une **solution possible** à ces contraintes, et non pas un encadrement de toutes les solutions. Ce problème a été analysé de deux façons différentes.

Applications successives de la méthode de Shostak

Une *première solution* consiste à associer successivement une valeur à chacune des variables, dans son domaine de variation. Cette affectation ne peut pas se faire en parallèle sur chaque variable, puisque, outre les domaines de variation qui restreignent les valeurs possibles, les variables sont encore liées par les contraintes. Il est nécessaire de fixer les valeurs une à une, en propageant la modification aux autres variables. On obtient l'algorithme suivant :

Pour chaque variable v *de* $\mathcal{V}$ *faire*

- **associer** à v une valeur dans son domaine de variation
- **propager** la modification dans les contraintes, et en déduire les nouveaux domaines des autres variables

finpour

La valeur à associer à chaque variable v peut être quelconque, du moment qu'elle fait partie de son domaine de variation. La propagation de la modification se fait, tout d'abord, en *substituant* globalement dans les contraintes la variable v par sa valeur, puis par une application de la méthode de *Shostak* (§ 4.1.) : on trouve ainsi le nouvel intervalle de variation des variables restantes.

Cette technique nécessite un *nombre de propagations* comparable au nombre de variables à traiter. On peut l'améliorer en fixant à chaque étape, non pas une seule variable, mais toutes les variables indépendantes. Ici encore, on est amené à utiliser la relation **est lié à** sur l'ensemble des variables, ainsi que les classes d'équivalence associées. Si k est le nombre de classes d'équivalence, on peut fixer à chaque étape k variables, prises chacune dans une classe différente. Dans la suite, on note $\mathcal{EC}$ l'ensemble des classes d'équivalence.

$\mathcal{NF} \leftarrow \mathcal{V}$; l'ensemble des variables non encore fixées

Tantque $\mathcal{NF}$ non vide *faire*

Pour chaque cl *dans* $\mathcal{EC}$ *faire*

- choisir une variable v dans la classe d'équivalence cl
- associer à v une valeur prise dans son domaine de variation
- $\mathcal{NF} \leftarrow \mathcal{NF}$ - { v }

finpour

Propager la modification dans les contraintes, en déduire les nouveaux intervalles de variation des variables, ainsi que les nouvelles classes d'équivalence

fintantque

De cette façon, le nombre de propagations est comparable au cardinal de la plus grande classe d'équivalence de la relation.

Cette solution a l'avantage d'être simple à implanter, puisque l'on dispose déjà du module de propagation, utilisé dans la première partie de MEPHISTO. L'algorithme a été implanté dans un premier temps, puis abandonné, car même en tenant compte des liens entre les variables pour améliorer l'algorithme, le nombre de propagations est encore prohibitif.

LE SIMPLEXE

Pour finir, nous avons choisi d'appliquer une des méthodes les plus courantes d'optimisation, restreinte aux cas de linéarité des contraintes. Le **simplexe** est une technique d'optimisation d'une fonction linéaire, sous des contraintes linéaires quelconques [FAU71]. Soit $x_1, x_2, ..., x_n$, les n variables naturelles à traiter. La fonction à optimiser, appelée *fonction économique* est de la forme :

$$MAX \sum_{i=1}^{n} c_i x_i \qquad \text{ou} \qquad MIN \sum_{i=1}^{n} c_i x_i$$

sous des contraintes de la forme

$$\sum_{i=1}^{n} a_{j_1 i} x_i \leq d_{j1} \qquad \forall j_1 \in J_1$$

$$\sum_{i=1}^{n} a_{j_2 i} x_i = d_{j2} \qquad \forall j_2 \in J_2$$

$$\sum_{i=1}^{n} a_{j_3 i} x_i \geq d_{j3} \qquad \forall j_3 \in J_3$$

avec $x_{i1} \geq 0 \qquad \forall i_1 \in I_1$

$x_{i2} \leq 0 \qquad \forall i_2 \in I_2$

On a $I = I_1 \cup I_2 \qquad J = J_1 \cup J_2 \cup J_3$

Le problème n'est pas résolu dans cette forme canonique mixte, mais plutôt dans une *forme standard* définie par :

$$J_1 = J_3 = I_2 = \varnothing$$

La **forme standard** n'autorise que des contraintes d'égalités, portant sur des variables positives ou nulles. Ces égalités s'obtiennent simplement à partir des inégalités en rajoutant des **variables d'écarts**. A partir du système standard, on constitue une **base**, c'est-à-dire un sous-ensemble de l'ensemble des variables naturelles et d'écarts, tel que toutes les autres variables, appelées *variables hors-base*, puissent s'exprimer en fonction des variables de base.

On note **B** la base, et **H** l'ensemble des variables hors-base. On a $J = B \cup H$. On obtient une **solution de base**, en fixant toutes les variables hors base à 0. On appelle alors *programme de base*, une solution de base *réalisable*, c'est-à-dire une solution pour laquelle les contraintes sont vérifiées et les variables de la base sont positives ou nulles.

Le principe du **simplexe** consiste à passer d'un programme de base à un autre, jusqu'à en trouver un qui soit optimal. Dans la suite, nous noterons x_B les variables de base, et x_H les variables hors-base.

A l'étape k-1, on a un programme de base, défini par :

$$X_H(k-1) = 0$$

$$X_B(k-1) = d(k-1) = (A_B(k-1))^{-1} * d(0) \geq 0$$

où $X_H(k-1)$ désigne les variables hors base à l'étape k-1

$X_B(k-1)$ désigne les variables de base à l'étape k-1

Si ce programme de base n'est pas optimum, on tente de l'améliorer en modifiant au mieux les ensembles H et B de l'étape k-1. Une variable de B, notée x_{Bi} doit sortir de la base, au profit d'une variable hors-base de H, appelée x_{Hi}.

Dans la suite, $A_B(k-1)[v,w]$ désigne le coefficient de la variable hors-base x_w, dans la v-ième contrainte, obtenue à partir de la base de l'étape k-1.

Les formules relatives à ce changement de base sont :

$$d_l^{(k)} = d_l^{(k-1)} - A_B(k-1)[l,j] * \frac{d_i^{(k-1)}}{A_B(k-1)[i,j]}$$

$$A_B(k)[l,m] = A_B(k-1)[l,m] - A_B(k-1)[l,j] * \frac{A_B(k-1)[i,m]}{A_B(k-1)[i,j]} \qquad l \neq i$$

$$A_B(k)[i,m] = \frac{A_B(k-1)[i,m]}{A_B(k-1)[i,j]}$$

La nouvelle fonction à optimiser s'écrit :

$$F^{(k)} = F^{(k-1)} + \frac{d_i^{(k-1)}}{A_B(k-1)[i,j]} * c_j = F^{(k-1)} + R * c_j$$

Comme on désire que $F^{(k)} \geq F^{(k-1)}$, il faut choisir j tel que c_j soit positif. Le rapport R est positif car $d_i^{(k-1)}$ est strictement positif et $A_B(k-1)[i,j]$ sera pris positif. En principe, on gagne à prendre c_j le plus grand possible (*premier critère de Dantzig*).

De plus, il faut que $\forall$ l, $d_l^{(k)}$ soit positif ou nul, c'est-à-dire :

$$d_l^{(k)} = d_l^{(k-1)} - A_B(k-1)[l,j] * R \geq 0$$

ce qui s'écrit aussi

$$d_l^{(k-1)} \geq A_B(k-1)[l,j] * R$$

Si $A_B(k-1)[l,j]$ est **négatif**, l'inégalité est vérifiée puisque R et $d_l^{(k-1)}$ sont positifs.

Lorsque $A_B(k-1)[l,j]$ est **positif**, l'inégalité s'écrit :

$$\frac{d_i^{(k-1)}}{A_B(k-1)[i,j]} \leq \frac{d_l^{(k-1)}}{A_B(k-1)[l,j]}$$

Cette condition est vérifiée si l'on a soin de prendre parmi les quotients $d_i^{(k-1)} / A_B(k-1)[i,j]$ le plus petit positif d'entre eux (*deuxième critère de Dantzig*).

L'optimum est atteint dès que le premier critère n'est plus applicable, c'est-à-dire lorsque tous les $d_l^{(k-1)}$ relatifs aux variables hors base sont négatifs, ceux relatifs aux variables de base étant nuls.

7 - EXEMPLES

Cette dernière partie illustre tous les propos tenus dans cette thèse. Le système MEPHISTO a été entièrement réalisé en Le_Lisp, sur VAX 785, sauf l'algorithme du simplexe, qui est implanté en Fortran.

Les exemples détaillés ici sont simples, car il s'est avéré que les modèles importants nécessitent un temps d'exécution prohibitif, s'ils ne sont pas totalement inutilisables. En effet, rappelons que le système MEPHISTO ajoute aux contraintes fournies par le concepteur, un ensemble de contraintes prédéfinies précisant les liens existant entre les attributs prédéfinis. Pour un modèle de 5 règles, on atteint vite un ensemble de 150 contraintes à traiter.

EXEMPLE 1

Dans cet exemple, l'ensemble des primitives est donné par

$$\mathcal{P} = \{ \text{PSPHERE, PCONE, PCYLINDRE} \}$$

Les cinq règles suivantes décrivent une scène sans alternative sur la composition des formes.

/1/ SCENE → TABLE ORANGE

h (TABLE) ≥ h (ORANGE)
ymax (TABLE) - ymax (ORANGE) ≥ 10 + ymin (ORANGE) - ymin (TABLE)
xmax (TABLE) - xmax (ORANGE) ≥ 20 + xmin (ORANGE) - xmin (TABLE)
r (ORANGE) ≤ 10

/2/ TABLE → PLATEAU PIED

h (TABLE) = h (PLATEAU) + h (PIED)
zmax (TABLE) = zmax (PLATEAU)
zmin (TABLE) = zmin (PIED)
(VERTICAL PIED)
(VERTICAL PLATEAU)
(POSE_SUR PLATEAU PIED)
couleur (PIED) ≠ couleur (PLATEAU)
couleur (PLATEAU) ∈ { noir, brun, rouge, blanc }

/3/ **PLATEAU→ PCYLINDRE**

$10 \leq h\,(PLATEAU) \leq 20$
$100 \leq r\,(PCYLINDRE)$
$h\,(PLATEAU) = h\,(PCYLINDRE)$
$phi\,(PLATEAU) = phi\,(PCYLINDRE)$
$teta\,(PLATEAU) = teta\,(PCYLINDRE)$
$couleur\,(PLATEAU) = couleur\,(PCYLINDRE)$

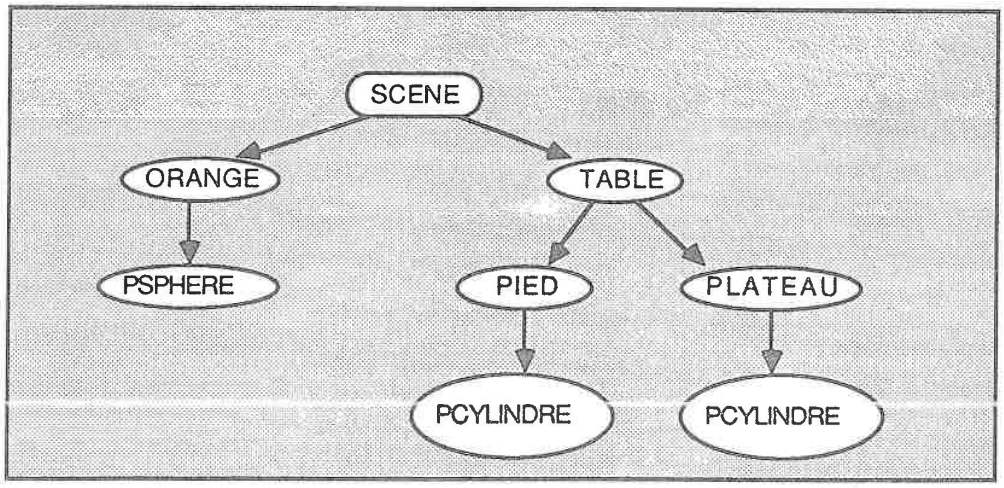
/4/ **PIED → PCYLINDRE**

$10 \leq r\,(PCYLINDRE) \leq 20$
$60 \leq h\,(PIED) \leq 100$
$h\,(PIED) = h\,(PCYLINDRE)$
$phi\,(PIED) = phi\,(PCYLINDRE)$
$teta\,(PIED) = teta\,(PCYLINDRE)$
$couleur\,(PIED) = couleur\,(PCYLINDRE)$

/5/ **ORANGE → PSPHERE**

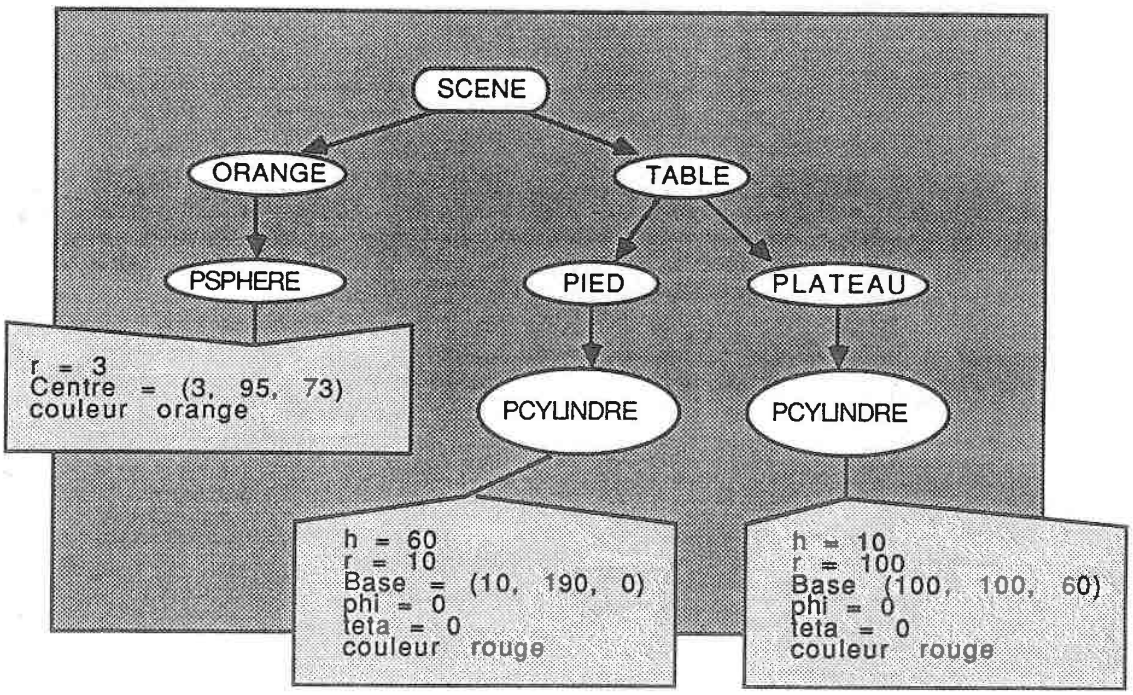
$3 \leq r\,(PSPHERE) \leq 7$
$couleur\,(ORANGE) = couleur\,(PSPHERE)$
$couleur\,(PSPHERE) \in \{ orange \}$

Dans un premier temps, MEPHISTO construit l'instance suivante :

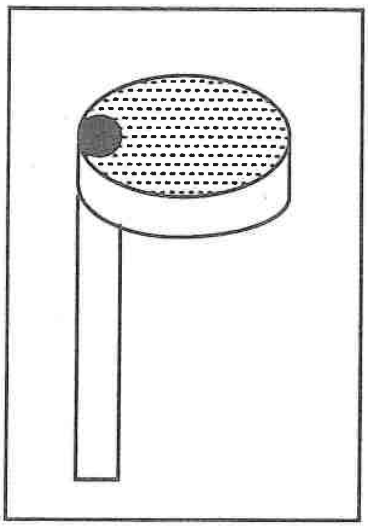


A chaque attribut des formes composant la scène, est associé un intervalle de variation définissant les valeurs possibles de chacun d'eux. A l'instance construite est associé l'ensemble des contraintes liant les attributs entre eux, c'est-à-dire l'ensemble des contraintes des règles /1/, /2/, /3/, /4/ et /5/.

La seconde étape consiste à résoudre cet ensemble d'inéquations et d'équations, afin d'associer une valeur à chaque attribut de primitive. Les résultats sont donnés par le schéma suivant :



La scène obtenue est la suivante :



Inutile d'être très attentif pour se rendre compte d'un problème de construction de la table. Ceci vient du fait que l'ensemble des contraintes fourni par le concepteur n'est pas suffisamment précis. Dans la règle /2/, il faudrait ajouter des contraintes précisant que le pied est à peu près au milieu du plateau.

Le détail du déroulement de la construction est donné dans les pages suivantes.

```
? (debug (MODELE) )
NOM DU FICHIER CONTENANT LE MODELE  ? don-initl


LA SOLUTION DOIT-ELLE CONTENIR UNE FORME PARTICULIERE  ? o
NOM DE CETTE FORME ? TABLE
Regles exclues  ()


INITIALISATION DES INTERVALLES DE VARIATION
Impression du resultat ? n


**************************************

E T A P E  0

*** CHOIX D'UN NOEUD PARMI (g1)

    Noeud choisi g1

*** ANALYSE ASCENDANTE ***

*** CHOIX D'UNE REGLE PARMI (r_1)

    Regle choisie r_1

*** Collecte des contraintes discretes de la regle r_1

*** Collecte des contraintes continues de la regle r_1
    ........................
*** Simplification d'inegalites par des egalites

*** SHOSHO mouline

*** L'ETAPE 0 S'EST BIEN PASSEE ***

*** HISTORIQUE ((0 g1 a-ascend))

*** INSTANCE de racine g3 ***

NOEUD g3 Forme scene
Attributs et leurs etats
  g15 lg (0 300.)
  g16 l (0 300.)
  g17 h (1. 300.)
  g18 xmin (0 279.)
  g19 xmax (21. 300)
  g20 ymin (0 289.)
  g21 ymax (11. 300)
  g23 zmin (0 0)
  g25 zmax (1. 300.)
NOEUD g2 Forme orange
Attributs et leurs etats
  g4 h (0 300)
  g6 ymin (0 289.)
  g9 ymax (1. 300)
  g10 xmin (0 279.)
  g13 xmax (1. 300)
  g14 r (0 10)
  g26 zmin (0 300)
  g27 zmax (0 300)
NOEUD g1 Forme table
Attributs et leurs etats
  g5 h (0 300)
```

```
    g7 ymin (0 299)
    g8 ymax (11. 300)
    g11 xmin (0 299)
    g12 xmax (21. 300)
    g22 zmin (0 0)
    g24 zmax (0 300)
  (scene (orange) (table))


**************************************

E T A P E  1

*** CHOIX D'UN NOEUD PARMI (g2 g1)

    Noeud g2  Forme associee orange

*** Collecte des contraintes continues de la regle r_5
    ........................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_5 8
    Nombre de composantes connexes de r_5 2

    Noeud g1  Forme associee table

*** Collecte des contraintes continues de la regle r_2
    ........................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_2 32
    Nombre de composantes connexes de r_2 6

    Noeud choisi g1


*** ANALYSE DESCENDANTE ***

*** CHOIX D'UNE REGLE PARMI (r_2)

    Regle choisie r_2

*** Collecte des contraintes discretes de la regle r_2
    ..
*** Collecte des contraintes continues de la regle r_2
    ........................
*** Simplification d'inegalites par des egalites


*** SHOSHO mouline

*** L'ETAPE 1 S'EST BIEN PASSEE ***

*** HISTORIQUE ((1 g1 a-descend) (0 g1 a-ascend))

*** INSTANCE de racine g3 ***

NOEUD g3 Forme scene
Attributs et leurs etats
  g15 lg (0 300.)
  g16 l (0 300.)
  g17 h (1. 300.)
  g18 xmin (0 279.)
  g19 xmax (21. 300)
  g20 ymin (0 289.)
  g21 ymax (11. 300)
  g23 zmin (0 0)
  g25 zmax (1. 300.)
```

```
 NOEUD g2 Forme orange
 Attributs et leurs etats
   g4 h (0 300)
   g6 ymin (0 289.)
   g9 ymax (1. 300)
   g10 xmin (0 279.)
   g13 xmax (1. 300)
   g14 r (0 10)
   g26 zmin (1. 300)
   g27 zmax (0 300)
 NOEUD g1 Forme table
 Attributs et leurs etats
   g40 lg (0 300.)
   g41 l (0 300.)
   g5 h (1. 300)
   g7 ymin (0 299)
   g8 ymax (11. 300)
   g11 xmin (0 299)
   g12 xmax (21. 300)
   g22 zmin (0 0)
   g24 zmax (1. 300)
 NOEUD g29 Forme pied
 Attributs et leurs etats
   g32 xmin (0 299)
   g34 xmax (0 300)
   g36 ymin (0 299)
   g38 ymax (0 300)
   g44 zmin (0 0)
   g45 zmax (0 300)
   g47 h (0 300)
   g48 teta (0 0)
   g31 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
NOEUD g28 Forme plateau
Attributs et leurs etats
   g33 xmax (1. 300)
   g35 xmin (0 300)
   g37 ymax (1. 300)
   g39 ymin (0 300)
   g42 zmin (0. 300)
   g43 zmax (1. 300)
   g46 h (0 300)
   g49 teta (0 0)
   g30 couleur (blanc rouge brun noir)
(scene (orange) (table (pied) (plateau)))


**************************************

E T A P E  2

*** CHOIX D'UN NOEUD PARMI (g29 g28 g2)

    Noeud g29  Forme associee pied

*** Collecte des contraintes continues de la regle r_4
    ..............................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_4 23
    Nombre de composantes connexes de r_4 8

    Noeud g28  Forme associee plateau

*** Collecte des contraintes continues de la regle r_3
    ..............................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_3 24
```

```
     Nombre de composantes connexes de r_3 8

     Noeud g2  Forme associee orange

 *** Collecte des contraintes continues de la regle r_5
     ........................
 *** Simplification d'inegalites par des egalites
     Cout de la regle r_5 8
     Nombre de composantes connexes de r_5 4

     Noeud choisi g28


 *** ANALYSE DESCENDANTE ***

 *** CHOIX D'UNE REGLE PARMI (r_3)

     Regle choisie r_3

 *** Collecte des contraintes discretes de la regle r_3
     .
 *** Collecte des contraintes continues de la regle r_3
     ..........................
 *** Simplification d'inegalites par des egalites

 *** SHOSHO mouline

 *** SHOSHO mouline

 *** L'ETAPE 2 S'EST BIEN PASSEE ***


 *** HISTORIQUE ((2 g28 a-descend) (1 g1 a-descend) (0 g1 a-ascend))

 *** INSTANCE de racine g3 ***

NOEUD g3 Forme scene
Attributs et leurs etats
  g15 lg (0 300.)
  g16 l (0 300.)
  g17 h (10. 300.)
  g18 xmin (0 279.)
  g19 xmax (21. 300)
  g20 ymin (0 289.)
  g21 ymax (11. 300)
  g23 zmin (0 0)
  g25 zmax (10. 300.)
NOEUD g2 Forme orange
Attributs et leurs etats
  g4 h (0 300)
  g6 ymin (0 289.)
  g9 ymax (1. 300)
  g10 xmin (0 279.)
  g13 xmax (1. 300)
  g14 r (0 10)
  g26 zmin (10. 300)
  g27 zmax (0 300)
NOEUD g1 Forme table
Attributs et leurs etats
  g40 lg (0 300.)
  g41 l (0 300.)
  g5 h (10. 300)
  g7 ymin (0 299)
  g8 ymax (11. 300)
  g11 xmin (0 299)
  g12 xmax (21. 300)
```

```
    g22 zmin (0 0)
    g24 zmax (10. 300)
  NOEUD g29 Forme pied
  Attributs et leurs etats
    g32 xmin (0 299)
    g34 xmax (0 300)
    g36 ymin (0 299)
    g38 ymax (0 300)
    g44 zmin (0 0)
    g45 zmax (0 290.)
    g47 h (0 290.)
    g48 teta (0 0)
    g31 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
  NOEUD g28 Forme plateau
  Attributs et leurs etats
    g53 lg (0 300.)
    g54 l (0 300.)
    g68 phi (0 300)
    g33 xmax (1. 300)
    g35 xmin (0 299.)
    g37 ymax (1. 300)
    g39 ymin (0 299.)
    g42 zmin (0. 290.)
    g43 zmax (10. 300)
    g46 h (10. 20)
    g49 teta (0 0)
    g30 couleur (rouge noir blanc brun)
  NOEUD g50 Forme pcylindre
  Attributs et leurs etats
    g52 r (100. 300)
    g55 lg (0 300.)
    g56 l (0 300.)
    g57 h (10. 20)
    g58 xpb (0 300)
    g59 xmin (0 299.)
    g60 xmax (1. 300)
    g61 ypb (0 300)
    g62 ymin (0 299.)
    g63 ymax (1. 300)
    g64 zpb (0 300)
    g65 zmin (0 290.)
    g66 zmax (10. 300)
    g67 phi (0 300)
    g69 teta (0 0)
    g51 couleur (rouge noir blanc brun)
(scene (orange) (table (pied) (plateau (pcylindre))))


****************************************

E T A P E  3


*** CHOIX D'UN NOEUD PARMI (g29 g2)

    Noeud g29  Forme associee pied

*** Collecte des contraintes continues de la regle r_4
    ..............................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_4 23
    Nombre de composantes connexes de r_4 10

    Noeud g2  Forme associee orange

*** Collecte des contraintes continues de la regle r_5
```

```
    ..........................
*** Simplification d'inegalites par des egalites
    Cout de la regle r_5 8
    Nombre de composantes connexes de r_5 6

    Noeud choisi g29


*** ANALYSE DESCENDANTE ***


*** CHOIX D'UNE REGLE PARMI (r_4)

    Regle choisie r_4

*** Collecte des contraintes discretes de la regle r_4
    .
*** Collecte des contraintes continues de la regle r_4
    ..............................
*** Simplification d'inegalites par des egalites



*** SHOSHO mouline

*** L'ETAPE 3 S'EST BIEN PASSEE ***

*** HISTORIQUE ((3 g29 a-descend) (2 g28 a-descend) (1 g1 a-descend) (0 g1
a-ascend))

*** INSTANCE de racine g3 ***

NOEUD g3 Forme scene
Attributs et leurs etats
  g15 lg (0 300.)
  g16 l (0 300.)
  g17 h (70. 300.)
  g18 xmin (0 279.)
  g19 xmax (21. 300)
  g20 ymin (0 289.)
  g21 ymax (11. 300)
  g23 zmin (0 0)
  g25 zmax (70. 300.)
NOEUD g2 Forme orange
Attributs et leurs etats
  g4 h (0 120.)
  g6 ymin (0 289.)
  g9 ymax (1. 300)
  g10 xmin (0 279.)
  g13 xmax (1. 300)
  g14 r (0 10)
  g26 zmin (70. 120.)
  g27 zmax (0 300)
NOEUD g1 Forme table
Attributs et leurs etats
  g40 lg (0 300.)
  g41 l (0 300.)
  g5 h (70. 120.)
  g7 ymin (0 299)
  g8 ymax (11. 300)
  g11 xmin (0 299)
  g12 xmax (21. 300)
  g22 zmin (0 0)
  g24 zmax (70. 120.)
NOEUD g29 Forme pied
Attributs et leurs etats
```

```
    g73 lg (0 300.)
    g74 l (0 300.)
    g88 phi (0 300)
    g32 xmin (0 299)
    g34 xmax (1. 300)
    g36 ymin (0 299)
    g38 ymax (1. 300)
    g44 zmin (0 0)
    g45 zmax (60. 100.)
    g47 h (60. 100.)
    g48 teta (0 0)
    g31 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
 NOEUD g70 Forme pcylindre
 Attributs et leurs etats
    g72 r (10. 20)
    g75 lg (0 300.)
    g76 l (0 300.)
    g77 h (60. 100.)
    g78 xpb (0 300)
    g79 xmin (0 299)
    g80 xmax (1. 300)
    g81 ypb (0 300)
    g82 ymin (0 299)
    g83 ymax (1. 300)
    g84 zpb (0. 100.)
    g85 zmin (0 0)
    g86 zmax (60. 100.)
    g87 phi (0 300)
    g89 teta (0 0)
    g71 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
 NOEUD g28 Forme plateau
 Attributs et leurs etats
    g53 lg (0 300.)
    g54 l (0 300.)
    g68 phi (0 300)
    g33 xmax (1. 300)
    g35 xmin (0 299.)
    g37 ymax (1. 300)
    g39 ymin (0 299.)
    g42 zmin (60. 100.)
    g43 zmax (70. 120.)
    g46 h (10. 20)
    g49 teta (0 0)
    g30 couleur (rouge noir blanc brun)
 NOEUD g50 Forme pcylindre
 Attributs et leurs etats
    g52 r (100. 300)
    g55 lg (0 300.)
    g56 l (0 300.)
    g57 h (10. 20)
    g58 xpb (0 300)
    g59 xmin (0 299.)
    g60 xmax (1. 300)
    g61 ypb (0 300)
    g62 ymin (0 299.)
    g63 ymax (1. 300)
    g64 zpb (60. 120.)
    g65 zmin (60. 100.)
    g66 zmax (70. 120.)
    g67 phi (0 300)
    g69 teta (0 0)
    g51 couleur (rouge noir blanc brun)
 (scene (orange) (table (pied (pcylindre)) (plateau (pcylindre))))
```



```
**************************************
```

```
E T A P E  4

*** CHOIX D'UN NOEUD PARMI (g2)

    Noeud choisi g2

*** ANALYSE DESCENDANTE ***


*** CHOIX D'UNE REGLE PARMI (r_5)

    Regle choisie r_5

*** Collecte des contraintes discretes de la regle r_5
    ..
*** Collecte des contraintes continues de la regle r_5
    ........................
*** Simplification d'inegalites par des egalites

*** SHOSHO mouline

*** SHOSHO mouline

*** L'ETAPE 4 S'EST BIEN PASSEE ***


*** HISTORIQUE ((4 g2 a-descend) (3 g29 a-descend) (2 g28 a-descend) (1 g1
a-descend) (0 g1 a-ascend))

*** INSTANCE de racine g3 ***

NOEUD g3 Forme scene
Attributs et leurs etats
  g15 lg (0 300.)
  g16 l (0 300.)
  g17 h (76. 300.)
  g18 xmin (0 279.)
  g19 xmax (21. 300)
  g20 ymin (0 289.)
  g21 ymax (11. 300)
  g23 zmin (0 0)
  g25 zmax (76. 300.)
NOEUD g2 Forme orange
Attributs et leurs etats
  g94 lg (6. 14.)
  g95 l (6. 14.)
  g4 h (6. 14.)
  g6 ymin (0 289.)
  g9 ymax (6. 300)
  g10 xmin (0 279.)
  g13 xmax (6. 293.)
  g14 r (0 10)
  g26 zmin (70. 120.)
  g27 zmax (76. 134.)
  g92 couleur (orange)
NOEUD g90 Forme psphere
Attributs et leurs etats
  g93 r (3. 7.)
  g96 lg (6. 14.)
  g97 l (6. 14.)
  g98 h (6. 14.)
  g99 xmax (6. 293.)
  g100 xmin (0 279.)
  g101 ymax (6. 300)
  g102 ymin (0 289.)
```

```
    g103 zmax (70. 194.)
    g104 zmin (62. 128.)
    g105 x (3. 286.)
    g106 y (3. 296.)
    g107 z (69. 131.)
    g91 couleur (orange)
  NOEUD g1 Forme table
  Attributs et leurs etats
    g40 lg (0 300.)
    g41 l (0 300.)
    g5 h (70. 120.)
    g7 ymin (0 299)
    g8 ymax (11. 300)
    g11 xmin (0 292.)
    g12 xmax (21. 300)
    g22 zmin (0 0)
    g24 zmax (70. 120.)
  NOEUD g29 Forme pied
  Attributs et leurs etats
    g73 lg (0 300.)
    g74 l (0 300.)
    g88 phi (0 300)
    g32 xmin (0 299)
    g34 xmax (1. 300)
    g36 ymin (0 299)
    g38 ymax (1. 300)
    g44 zmin (0 0)
    g45 zmax (60. 100.)
    g47 h (60. 100.)
    g48 teta (0 0)
    g31 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
  NOEUD g70 Forme pcylindre
  Attributs et leurs etats
    g72 r (10. 20)
    g75 lg (0 300.)
    g76 l (0 300.)
    g77 h (60. 100.)
    g78 xpb (0 300)
    g79 xmin (0 299)
    g80 xmax (1. 300)
    g81 ypb (0 300)
    g82 ymin (0 299)
    g83 ymax (1. 300)
    g84 zpb (0. 100.)
    g85 zmin (0 0)
    g86 zmax (60. 100.)
    g87 phi (0 300)
    g89 teta (0 0)
    g71 couleur (rouge jaune vert bleu noir blanc orange brun gris rose)
  NOEUD g28 Forme plateau
  Attributs et leurs etats
    g53 lg (0 300.)
    g54 l (0 300.)
    g68 phi (0 300)
    g33 xmax (1. 300)
    g35 xmin (0 299.)
    g37 ymax (1. 300)
    g39 ymin (0 299.)
    g42 zmin (60. 100.)
    g43 zmax (70. 120.)
    g46 h (10. 20)
    g49 teta (0 0)
    g30 couleur (rouge noir blanc brun)
  NOEUD g50 Forme pcylindre
  Attributs et leurs etats
    g52 r (100. 300)
```

```
    g56 l (0 300.)
    g57 h (10. 20)
    g58 xpb (0 300)
    g59 xmin (0 299.)
    g60 xmax (1. 300)
    g61 ypb (0 300)
    g62 ymin (0 299.)
    g63 ymax (1. 300)
    g64 zpb (60. 120.)
    g65 zmin (60. 100.)
    g66 zmax (70. 120.)
    g67 phi (0 300)
    g69 teta (0 0)
    g51 couleur (rouge noir blanc brun)
 (scene (orange (psphere)) (table (pied (pcylindre)) (plateau (pcylindre))))


 *** CONSTRUCTION D'UNE SOLUTION A PARTIR DE L'ARBRE ***

 FEUILLES DE L'ARBRE (g90 g70 g50)
 AUTRES NOEUDS (g3 g2 g1 g29 g28)


 *** TRAITEMENT DES CONTRAINTES DISCRETES ***

     Attributs discrets non fixes
         g91 de valeur (couleur g90)
         g71 de valeur (couleur g70)
         g51 de valeur (couleur g50)


    Je fixe g91 a orange
    Je fixe g71 a rouge
    Je fixe g51 a rouge


 *** TRAITEMENT DES CONTRAINTES CONTINUES ***
 Attributs fixes ((g85 0) (g89 0) (g69 0) (g23 0) (g22 0) (g44 0) (g48 0) (g49
 0))

 *** Simplification d'inegalites par des egalites


 SOLUTION TROUVEE


 (psphere 4)
     couleur = orange
     r = 3.
     x = 3.
     y = 95.
     z = 73.
 (pcylindre 5)
     couleur = rouge
     xpb = 10.
     ypb = 190.
     zpb = 0
     h = 60.
     r = 10.
     phi = 0
     teta = 0
 (pcylindre 3)
     couleur = rouge
     xpb = 100.
     ypb = 100.
```

```
     zpb = 60.
     h = 10.
     r = 100.
     phi = 0
     teta = 0
= ()
?
? (end)
Que Le_Lisp soit avec vous.
```

EXEMPLE 2

Dans cet exemple, l'ensemble des primitives est

$$\mathcal{P} = \{ P_1, P_2, P_3 \}$$

Nous n'utilisons pas les primitives solides de base (sphère, cône, cylindre ...) afin de créer un modèle de règles avec contraintes, sans y faire intervenir les contraintes prédéfinies nécessaires dans les autres cas.

/1/ SCENE → B C

$x(SCENE) \geq x(B) + x(C)$
$y(SCENE) = y(B)$
$h(SCENE) \geq h(B) - 10$
$h(SCENE) \leq h(C) + 23$
$z(SCENE) \leq 200$
$z(SCENE) \geq 25$
$y(SCENE) = y(C)$
$couleur(SCENE) \neq couleur(B)$
$couleur(B) \neq couleur(C)$

/2/ SCENE → B D E

$x(SCENE) \geq 44 - y(E) + y(D)$
$y(B) \geq 20$
$h(B) + h(D) - h(E) \leq 400$
$x(SCENE) \leq x(B) + 10$
$x(D) = x(B) - x(E)$
$couleur(SCENE) = couleur(B)$

/3/ B → P_1 P_2

$h(P_1) \leq 10 + h(B)$
$y(B) \geq y(P_1) + y(P_2) - z(P_2)$
$x(B) + x(P_1) = x(P_2)$
$couleur(B) = couleur(P_1)$
$couleur(P_1) = couleur(P_2)$
$couleur(P_1) \in \{$ bleu, rouge, vert, noir, blanc $\}$

/4/ B $\rightarrow$ D E P_2

$y(P_2) + y(E) \geq 2\,y(D) - z(B) + 11$
$x(B) \geq 34$
$y(E) \leq 123$
couleur (B) $\neq$ couleur (D)

/5/ C $\rightarrow$ P_1 P_3

$2\,x(P_1) + x(P_3) \geq x(C)$
$y(C) - y(P_1) \geq 25$
$z(P_3) = 12$
couleur (C) $\in$ { rouge, vert, noir, blanc }
couleur (P_1) = couleur (C)

/6/ C $\rightarrow$ P_1 P_2

$x(C) - x(P_1) \geq 28$
$y(C) \geq 2\,y(P_1)$
couleur (P_2) $\in$ { noir }
couleur (C) = couleur (P_2)

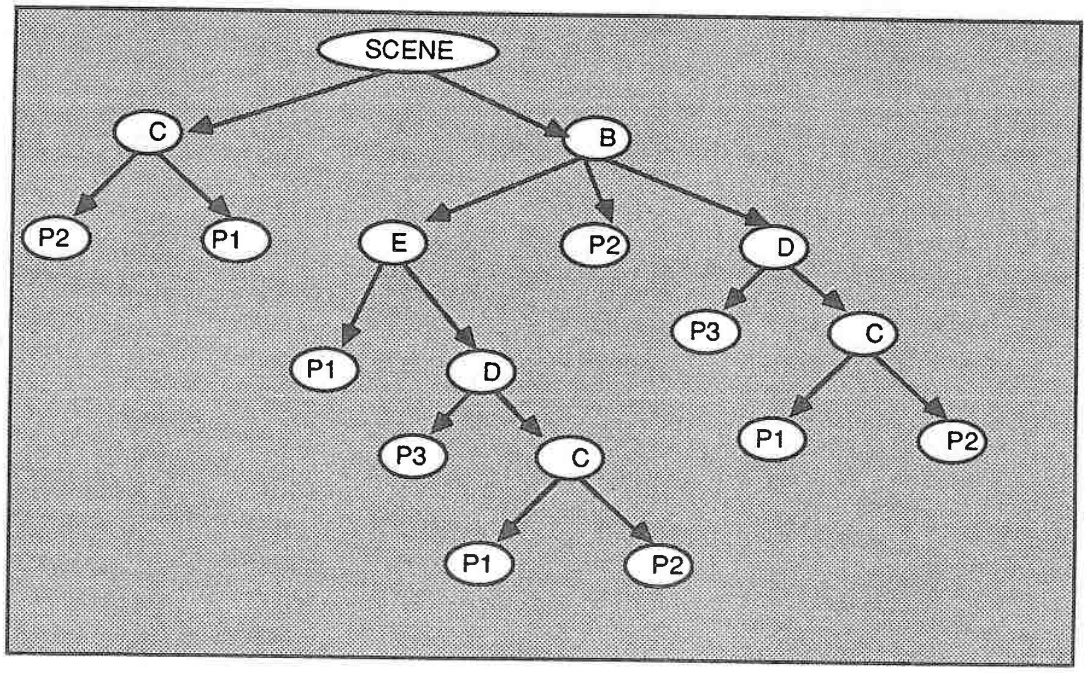
/7/ D $\rightarrow$ C P_3

$x(D) + x(C) \geq x(P_3)$
$z(P_3) - z(C) \geq y(D) + 1$
$y(P_3) \geq 10$
$z(C) \leq 2\,z(D) - z(P_3) + 21$
couleur (D) = couleur (C)

/8/ E $\rightarrow$ D P_1

$y(E) \leq y(D) - y(P_1) + 36$
couleur (D) $\neq$ couleur (E)

MEPHISTO construit l'instance suivante :



EXEMPLE 3

On reprend l'exemple précédent, en modifiant légèrement les règles /2/ et /4/.

/2/ SCENE $\rightarrow$ **B D E**

$x(SCENE) \geq 44 - y(E) + y(D)$
$y(B) \geq 20$
couleur (SCENE) = couleur (B)

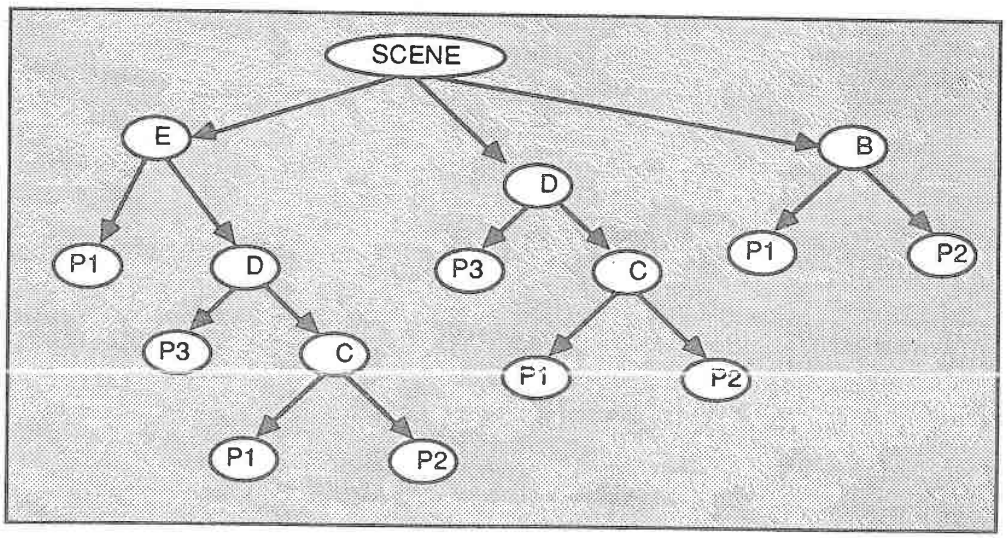
/4/ B $\rightarrow$ **D E** $\mathbf{P_2}$

$y(P_2) + y(E) \geq 2\,y(D) - z(B) + 11$
$x(B) \geq 34$
$y(E) \leq 123$
$x(D) - x(E) = x(P_2) + y(E)$

$y(B) \leq y(D) + y(E) - 1$
$z(B) \leq z(P_2)$

couleur (B) $\neq$ couleur (D)

En modifiant l'ensemble des contraintes associées à ces deux règles, on provoque la création d'une autre instance, car MEPHISTO effectue ces choix différemment.



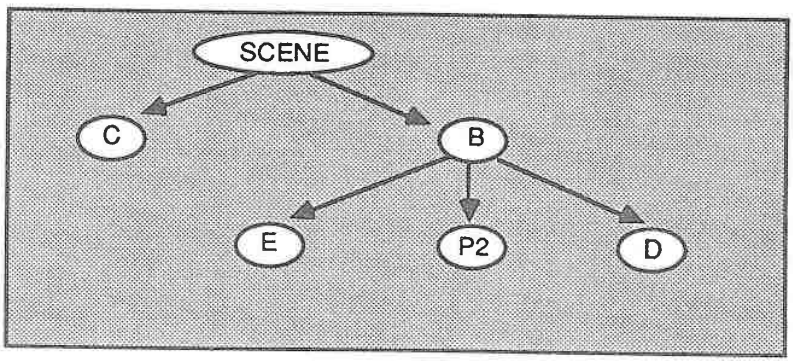
EXEMPLE 4

On reprend l'exemple numéro 2, en modifiant légèrement la règle /7/.

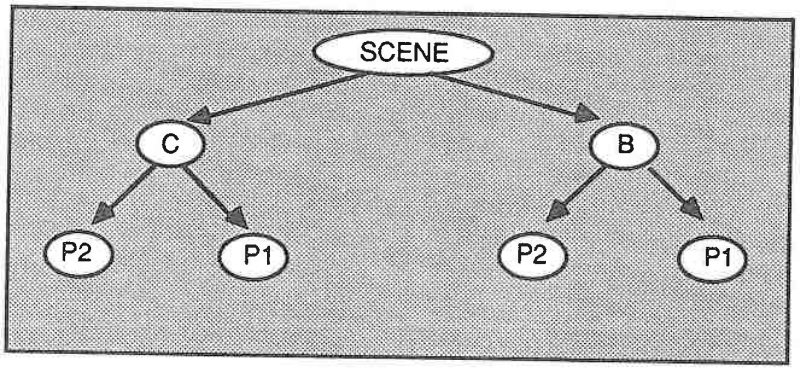
/7/ **D** $\rightarrow$ **C** P_3

$x(D) + x(C) \geq x(P_3)$
$z(P_3) - z(C) \geq y(D) + 1$
$y(P_3) \geq 10 + y(D)$
$y(P_3) \leq 9 + z(C) - z(P_3)$
$z(C) \leq 2\ z(D) - z(P_3) + 21$
couleur (D) = couleur (C)

Dans un premier temps, le système choisit la règle /1/ pour décrire SCENE, puis la règle /4/ pour décrire B.



Par la suite, il décide de continuer l'extension de D, par la règle /7/. Une **inconsistance** de l'ensemble des contraintes continues des règles /1/ , /4/ et /7/ est alors détectée. Le système provoque un retour-arrière : il commence par remettre en cause le choix de la règle /7/ pour décrire D. Comme aucune autre règle ne permet de décrire cette forme, il remet en cause le choix précédent, c'est-à-dire celui de la règle /4/ pour décrire B. Il décide alors d'utiliser la règle /3/ et construit l'instance suivante :



CONCLUSION

Le problème de la **validité de modèles géométriques** a été résolu par le système MEPHISTO, dans le cadre du modèle utilisé par le système TRIDENT. Etant donné la généralité des modèles utilisés, ce problème est indécidable. Le système MEPHISTO ne peut qu'extraire une instance du modèle. Rien ne permet donc d'assurer au concepteur que toutes les instances possibles du modèle sont valides, puisqu'il en existe une infinité.

L'étude de la validité d'un modèle nous a conduit à développer une **stratégie de construction** d'une instance, c'est-à-dire d'un chemin dans un graphe ET/OU. Elle applique une heuristique minimisant en moyenne les coûts de construction, à partir d'une évaluation des espérances de succès de cette construction.

Les capacités du système MEPHISTO sont limitées : le temps nécessaire à la validation de modèles importants est prohibitif. Ce temps ne reste raisonnable que pour des petits ensembles de règles associées à un ensemble réduit de contraintes. Il apparait donc inexploitable sur un modèle géométrique complexe. Toutefois, il peut être utilisé sur un modèle simple ou bien localement, sur un sous-ensemble de règles d'un modèle complet. En pratique, cette restriction reste satisfaisante.

Les problèmes rencontrés viennent essentiellement du fait que ce modèle a été conçu dans un but d'**interprétation de scènes et non de création**. A l'origine, le modèle effectivement utilisé par Trident n'a pas besoin d'être complet : la description de toutes les caractéristiques physiques des objets n'est pas utile à l'interpréteur, puisqu'il reconstitue des objets en s'appuyant à la fois sur le modèle et sur une image de la scène. Il dispose donc de deux types d'informations, qui se complètent, ce qui autorise des imprécisions dans le modèle. Mais, pour vérifier automatiquement la validité, le système MEPHISTO est amené à compléter la description des objets, car le modèle est la seule base sur laquelle il peut s'appuyer pour construire un ou plusieurs objets. Par exemple, le système doit connaitre les relations géométriques reliant les différents attributs d'un cylindre : ces contraintes sont intégrées à celles fournies par le concepteur et alourdissent énormément la modélisation.

Notre démarche globale, qui consiste à *distinguer conception et validation* est probablement à remettre en cause. L'étude des différentes modélisations développées dans les systèmes les plus récents montre que, la plupart du temps, conception et validation sont étroitement liés. Ces systèmes offrent toujours au concepteur un outil **interactif** de création de modèle. Les objets construits sont visualisés pas à pas. Les contrôles de validité peuvent être faits au fur et à mesure et assurent une localisation simplifiée des erreurs de conception. De plus, la visualisation facilite la représentation de contraintes comme "le téléphone est posé sur la table". Il est en effet plus aisé de dessiner un téléphone posé sur une table que d'écrire l'ensemble des inéquations spécifiant cette contrainte.

Cette technique pourrait être adaptée à la saisie du modèle utilisé par TRIDENT. Afin de respecter la généralité du modèle, il faudrait étendre le principe afin d'autoriser la création de modèles génériques. Une solution consiste à saisir "graphiquement" un objet de la famille générique, par exemple une table composée de 4 pieds cylindriques et d'un plateau parallélépipèdique. La contrainte "le plateau est posé sur les 4 pieds de la table" se ramène à un problème de saisie graphique. Tous les objets de cette famille seront ainsi constitués; on autorise ensuite la paramétrisation de chacune des longueurs et des positions des primitives constituant l'objet décrit, en faisant des contrôles de validité locaux.

Le système de saisie de modèles, appelé **modeleur**, pourrait construire de cette façon un ensemble d'objets génériques, en utilisant une représentation interne sous forme de CSG. Chaque objet est défini dans un repère universel par composition d'objets plus simples; à chacun d'eux est associé un ensemble de paramètres fixes, ou variables dans un certain domaine. Cette modélisation ne permettrait pas autant de généralité que celle utilisée dans le système TRIDENT. De plus, comme le modèle obtenu doit être utilisable directement par l'interpréteur de TRIDENT, cela nécessiterait une conversion entre les deux représentations. Pour finir, notons que cette démarche est très couramment utilisée dans les systèmes de modélisation; on peut citer par exemple le système PADL-2 [BROW82] qui, pour la saisie interactive, utilise une représentation interne sous forme de CSG puis, pour les calculs, une représentation par facettes.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[AGIN76] **AGIN G.J., BINFORD T.O.**

Computer descriptions of curved objects

IEEE Transactions Computer, vol 25, 1976

[AGIN81] **AGIN G.J.**

Hierarchical representations of three dimensional objects using verbals models

IEEE Transcations on PAMI, vol 3, no 2, 1981

[AYAC82] **AYACHE M., BOISSONNAT J.D., FAUGERAS O., GERMAIN F., HEBERT M., PANCHON E., PONCE J.**

Vers un système de vision flexible

Premières Journées annuelles du programme ARA, pp 153-169, 1982

[ATHE78] **ATHERTON P., WEILER K. GREENBERG D.**

Polygon shadow generation

SIGGRAPH, pp275-281, 1978

[BADL78] **BADLER N., BAJCSY R.**

3D representations for computer graphics and computer vision

ACM Computer Graphics, vol 12, no 3, pp 153-159, 1978

[BALL82] **BALLARD D.H., BROWN C.M.**

Computer vision

Prentice Hall, 1982

[BARN84] **BARNETT J.A.**

How much is control knowledge worth ?

Artificial Intelligence, vol 22, pp 77-89, 1984

[BARR84] **BARR A.H.**

Global and Local Deformations of solid primitives

Computer Graphics ACM, 1984

[BELA86] **BELAID Y., CAMONIN M., MASINI G., MOHR R., THIRION E.**

Le système de vision TRIDENT

Rapport final du contrat ADI no 83061, Université de NANCY 1, CRIN 86-R-012

[BLED75] **BLEDSOE W.W.**

A new method for proving certain Presburger formulas

4th IJCAI, Tbilissi, 1975

[BLIN77] **BLINN J.F.**

Models of light reflection for computer synthesized pictures

SIGGRAPH, pp 192-198, 1977

[BROO78] **BROOKS R.A., BINFORD T.O.**

The Acronym model based vision system

Proceedings ARPA Image Understanding Workshop, Pittsburgh, pp 145-151, 1978

[BROO81] **BROOKS R.A.**

Symbolic reasonning among 3D models and 2D images

Ph. D. Thesis, Stanford University, 1981

[BROO] **BROOKS R.A.**

Model based three dimensional interpretation of 2D images

Artificial Intelligence Laboratory, Stanford University, Stanford

[BROW81] **BROWN C.M.**

Some mathematical and representational aspects of solid modeling

IEEE Transactions on PAMI, vol 3, no 4, 1981

[BROW82] **BROWN C.M.**

PADL-2 : A Technical Summary

IEEE Computer Graphics and Applications, 1982, pp 69-84

[CAMO83] **CAMONIN M.**

Validation de modèles de description de scènes 3D

Rapport de DEA, Université de Nancy 1, CRIN 83-R-047

[CAMO85] **CAMONIN M., MASINI G.**

Conception et compilation du modèle dans le système de vision Trident

Actes de COGNITIVA 85, Paris 1985

[CHAN71] **CHANG C.L., SLAGLE J.R.**

An admissible and optimal algorithm for searching AND/OR graphs

Artificial Intelligence, vol 2, pp 117-128, 1971

[DANE81] **DANE C., BAJCSY R.**

Three dimensional segmentation using the Gaussian image and spatial information

Proceedings of Conference on Pattern Recognition and Image Processing, Dallas pp 54-56, 1981

[DAVI81] **DAVIS S., HENDERSON C.**

Hierarchical constraint processes for shape analysis

IEEE Transactions on PAMI, vol 3, no 3, pp 265-277, 1981

[DODD79] **DODD G.D., ROSSOL L.**

Computer Vision and Sensor-based robots

Plenum Press, New-York, 1979

[DOUG81] **DOUGLAS R.J.**

Interpreting Three dimensional scenes : a model building approach

Computer Graphics and Image Processing, vol 17, pp 91-113, 1981

[FAUR71] **FAURE R.**

Eléments de recherche opérationnelle

Editions GAUTHIER-VILLARS, 1971

[GARD85] **GARDAN Y.**

Mathématiques et CAO

Hermès, 1985

[GOSL83] **GOSLING J.**

Algebraic constraints

Department of computer science, Carnegie Mellon University, mai 1983

[GOUR71] **GOURAUD H.**

Computer display of curved surfaces

IEEE Transactions on PAMI, vol 6, 1971

[GRIM80] **GRIMSON W.E.L.**

Aspects of a computational theory of human stereo vision

Proceedings of the Image Understanding Workshop, pp 128-149, 1980

[HALI85] **HALIN G.**

Regroupement de régions sur des images 3D

Rapport de DEA, Université de Nancy 1, CRIN 85-R-086

[HALL73] **HALL P.A.V.**

Equivalence between and/or graphs and context free grammars

CACM, vol 16, pp 444-445

[HEBE82] **HEBERT M., PONCE J.**

A new method for segmenting three dimensional scenes into primitives

Proceeding of the 6th International conference on Pattern recognition, Munich, vol 2, pp 836-838, 1982

[HUFF71] **HUFFMAN D.**

Impossible objects as nonsense sentences

Machine Intelligence 6, 1971

[JACK80] **JACKINS C.L., TANIMOTO S.L.**

Oct-trees and thier use in representing three dimensional objects

Computer Graphics and Image Processing 14, p249-270, 1980

[KARP83] **KARP R.M., PEARL J.**

Searching for an optimal path in a tree with random costs

Artificial Intelligence, vol 21, pp 99-116, 1983

[KUMA83] **KUMAR V., KANAL L.N.**

A general Branch and Bound synthezing AND/OR tree search procedures

Artificial Intelligence, vol 21, pp 179-198, 1983

[KUNU83] **KUNU T.L.**

Computer Graphics - Theory and Applications

Springer Verlag, 1983

[LATO77] **LATOMBE J.C.**

Une application de l'intelligence artificielle à la conception assistée par ordinateur

Thèse d'état, Université de Grenoble, 1977

[LAUR76] **LAURIERE J.L.**

Un langage et un programme pour énoncer et résoudre des problèmes combinatoires

Thèse d'état, Université de Paris VI, 1976

[LENA82] **LENAT D.B.**

The nature of heuristics

Artificial Intelligence, vol 19, pp 189-249, 1982

[LEVI76] **LEVI G., SIROVICH F.**

Generalized AND/OR graphs

Artificial Intelligence, vol 7, pp 243-259, 1976

[LIEB74] **LIEBERMAN L.I.**

Computer recognition and description of natural scenes

Ph. Thesis, University of Pennsylvania, 1974

[LUX84] **LUX A., SOUVIGNIER V.**

PVV - Un système de vision appliquant une stratégie de prédiction - vérification

Actes du 4ième congrès AFCET RFIA, Paris 1984, pp 223-234, tome 1

[MACK77] **MACKWORTH A.K.**

Consistency in networks of relations

Artificial Intelligence, no 8, pp 99-118, 1977

[MART73] **MARTINELLI A., MONTANARI V.**

Additive and/or graphs

IJCAI 3, 1973, pp 1-11

[MART78] **MARTINELLI A., MONTANARI V.**

Optimizing decision trees through heuristically guided search

CACM 21, no 12, pp 1025-1039, 1978

[MASI83] **MASINI G., MOHR R.**

MIRABELLE : A system for structural analysis of drawings

Pattern Recognition, vol 16, no 4, pp 363-372, 1983

[MASI84] **MASINI G., MOHR R., MULLER V., ZAROLI F.**

Analyse de scènes 3D

Premier rapport intermédiaire du contrat ADI no 83.601, CRIN-R-064, 1984

[MASI85] **MASINI G., THIRION E., MOHR R.**

Stratégie de perception pour un modèle hiérarchique

Actes du 5ième congrès AFCET RFIA, Grenoble 1985, pp 631-640, tome 2

[MAZA84] **MASINI G., ZAROLI F.**

Présentation de Trident : un système d'interprétation d'images tri-dimensionnelles

Actes du 4-ième congrès AFCET RFIA, Paris, 1984

[MEAG82] **MEAGHER D.**

Geometric Modeling using Octree encoding

Computer Graphics and Image Processing, vol 19, 1982

[MOHR79] **MOHR R.**

Descriptions structurées et analyse de formes complexes

Thèse d'état, Université de Nancy 1, 1979

[MOHR82] **MOHR R., MASINI G.**

Utilisation du contexte en analyse d'images

Rapport final du contrat ADI no 19/79, Université de Nancy 1, CRIN 82-R-087

[MOHR84] **MOHR R., WROBEL B.**

La correspondance en stéréovision vue comme une recherche d'un chemin optimal

Actes du 4ième congrès AFCET RFIA, Paris, pp 71-79, 1984

[MOHR85] **MOHR R., HENDERSON T.C.**

Arc and path consistency revisited

Université de Nancy 1, Rapport interne CRIN 85-R-065

[MORT84] **MORTENSON M.E.**

Geometric modeling

John Wiley & Sons

[MULL84] **MULLER Y.**

Détection de surfaces par la transformée de Hough en vision tridimensionnelle

Thèse de 3ième cycle, Université de Nancy 1, 1984

[NEVA77] **NEVATIA R., BINFORD T.O.**

Description and recognition of curved objects

Artificial Intelligence, vol 8, pp 77-98, 1977

[NEVB77] **NEVATIA R., BINFORD T.O.**

Structured descriptions of complex objetcs

Project n° SD-183 Stanford Artificial Intelligence Laboratory, Computer science department

[NEVA78] **NEVATIA R.**

Characterization and requirements of computer vision systems

Computer vision systems, Hanson and Riseman, Academic Press, pp 81-87, 1978

[NEWM79] **NEWMAN W.E., SPROULL R.F.**

Principles of Interactive Computer Graphics

Mac Graw-Hill, 1979

[NILS71] **NILSSON N.J.**

Problem solving methods in Artificial Intelligence

Mac Graw-Hill, 1971

[NILS80] **NILSSON N.J.**

Principles of artificial intelligence

Tioga Publishing Company, Palo Alto, California, 1980

[OROU79] **O'ROURKE J., BADLER N.**

Decomposition of 3D objects into spheres

IEEE Transactions on PAMI, vol 1, pp 295-305, 1979

[OSHI79] **OSHIMA M., SHIRA Y.**

A scene description method using 3D informations

Pattern recognition, vol 11, no 1, pp 9-17, 1979

[POHL] **POHL I.**

First results on the effect of error in heuristic search

IBM Research Division

[REQU80] **REQUICHA A.A.G.**

Representations for rigid solids : theory, methods ans systems

Computing Surveys, vol 122, no 4, 1980

[REVO77] **REQUICHA A.A.G., VOELCKER H.B.**

Constructive Solid Geometry

Production automation project, University of Rochester, Rochester, 1977

[REVO82] **REQUICHA A.A.G., VOELCKER H.B.**

Solid modeling : a historical summary and contemporary assessment

IEEE Transactions on PAMI, 1982

[SEQU86] **SEQUEIRA J.**

Modélisation de solides pour création de scènes

Etude du centre scientifique d'IBM France, Paris 1986

[SHOS77] **SHOSTAK R.E.**

On the SUP-INF method for proving Presburger formulas

JACM, vol 24, no 4, 1977

[WESL80] **WESLEY M.A.**

Construction and use of geometric models

Lecture notes in computer science, no 89

[WINS79] **WINSTON P.M.**

Artificial Intelligence

Addison Wesley Publishing Company

[WINS86] **WINSTON P.M.**

LISP

Addison Wesley Publishing Company, 1986

[WROB87] **WROBEL B.**

Stéréovision : coopération entre l'extraction et la mise en correspondance symbolique des régions

CESTA - MARI, Paris, mai 1987

[WROM87] **WROBEL B., MONGA O.**

Segmentation d'images naturelles : coopération entre détecteur-contour et détecteur-région

11 ième colloque du GRETSI, Nice, juin 1987

NOM DE L'ETUDIANT : CAMONIN Martine

NATURE DE LA THESE : Doctorat de 3e cycle en INFORMATIQUE

VU, APPROUVE ET PERMIS D'IMPRIMER 1132

NANCY, le 7 JUIL. 1987

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

R. MAINARD

UNIVERSITÉ DE NANCY I
Le Président

Le **système MEPHISTO** est un outil de validation automatique de modèles géométriques tridimensionnels, développé dans le cadre d'un système d'interprétation de scènes tridimensionnelles (TRIDENT). Le modèle choisi permet de décrire des familles d'objets *génériques* construits par unions de primitives. La description de scènes est donnée par un ensemble de *règles* donnant une hiérarchie de composition des formes. Des ensembles de *contraintes* associées aux règles permettent de préciser l'assemblage des formes entre elles ainsi que leurs positions et dimensions respectives. La tâche du système MEPHISTO est de décider de la cohérence d'un modèle fourni par l'utilisateur, avant qu'il ne soit utilisé par TRIDENT. Dans le contexte de la représentation choisie , un modèle peut être vu comme un graphe ET/OU avec contraintes. Le problème de la vérification de validité se ramène à celui de la **construction d'une instance d'un graphe ET/OU avec contraintes**. Ceci nous a conduit à l'étude des techniques de résolution de graphes ET/OU et de résolution de systèmes de contraintes continues et discrètes. En particulier, nous avons développé une stratégie de recherche de chemin dans un graphe ET/OU, minimisant en moyenne les coûts de construction, à partir d'une évaluation des espérances de succès de cette construction.

MOTS-CLES

modélisation géométrique, validation de modèles, graphe ET/OU, résolution d'inéquations.