N° d'ordre : 990

THÈSE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE 3ème CYCLE

par

Corinne PARENT

ETUDE ET SPECIFICATIONS D'UN EDITEUR D'IMAGES. APPLICATIONS À LA BUREAUTIQUE

Soutenue le 5 octobre 1982 devant la Commission d'Examen

Membres du Jury	V. CORDONNIER	Président
	C. CARREZ	Rapporteur
	M. DAUCHET	Examinateur
	M. MERIAUX	Examinateur
	N. NAFFAH	Invité
	M. LOUF	Invité

PROFESSEURS 1ère CLASSE

M.	BACCHUS Pierre	Mathématiques
M.	BEAUFILS Jean-Pierre (dét.)	Chimie
M.	BIAYS Pierre	G.A.S.
M.	BILLARD Jean	Physique
M.	BONNOT Ernest	Biologie
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie
M.	CELET Paul	Sciences de la Terre
M.	COEURE Gérard	Mathématiques
M.	CONSTANT Eugène	I.E.E.A.
M.	CORDONNIER Vincent	I.E.E.A.
M.	DEBOURSE Jean-Pierre	S.E.S.
M.	DELATTRE Charles	Sciences de la Terre
M.	DURCHON Maurice	Biologie
M.	ESCAIG Bertrand	Physique
M.	FAURE Robert	Mathématiques
M.	FOURET René	Physique
M.	GABILLARD Robert	I.E.E.A.
M.	GRANELLE Jean-Jacques	S.E.S.
M.	GRUSON Laurent	Mathématiques
M.	GUILLAUME Jean	Biologie
M.	HECTOR Joseph	Mathématiques
M.	HEUBEL Joseph	Chimie
M.	LABLACHE COMBIER Alain	Chimie
M.	LACOSTE Louis	Biologie
M.	LANSRAUX Guy	Physique
M.	LAVEINE Jean-Pierre	Sciences de la Terre
M.	LEBRUN André	C.U.E.E.P.
M.	LEHMANN Daniel	Mathématiques
Mme	LENOBLE Jacqueline	Physique
M.	LHOMME Jean	Chimie
M.	LOMBARD Jacques	S.E.S.
M.	LOUCHEUX Claude	Chimie
M.	LUCQUIN Michel	Chimie

M.	MAILLET Pierre	S.E.S.
M.	MONTREUIL Jean	Biologie
M.	PAQUET Jacques	Sciences de la Terre
M.	PARREAU Michel	Mathématiques
M.	PROUVOST Jean	Sciences de la Terre
M.	SALMER Georges	I.E.E.A.
Mme	SCHWARTZ Marie-Hélène	Mathématiques
M.	SEGUIER Guy	I.E.E.A.
M.	STANKIEWICZ François	Sciences Economiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie
M.	VIDAL Pierre	I.E.E.A.
M.	VIVIER Emile	Biologie
M.	WERTHEIMER Raymond	Physique
M.	ZEYTOUNIAN Radyadour	Mathématiques

PROFESSEURS 2ème CLASSE

M.	AL FAKIR Sabah	Mathématiques
M.	ANTOINE Philippe	Mathématiques
M.	BART André	Biologie
Mme	BATTIAU Yvonne	Géographie
M.	BEGUIN Paul	Mathématiques
M.	BELLET Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOBE Bernard	S.E.S.
M.	BODART Marcel	Biologie
M.	BOILLY Bénoni	Biologie
M.	BONNELLE Jean-Pierre	Chimie
M.	BOSQ Denis	Mathématiques
M.	BREZINSKI Claude	I.E.E.A.
M.	BRUYELLE Pierre (Chargé d'enseignement)	Géographie
M.	CAPURON Alfred	Biologie
M.	CARREZ Christian	I.E.E.A.
M.	CHAMLEY Hervé	E.U.D.I.L.

M. CHAPOTON Alain	C.U.E.E.P.
M. COQUERY Jean-Marie	Biologie
Mme CORSIN Paule	Sciences de la Terre
M. CORTOIS Jean	Physique
M. COUTURIER Daniel	Chimie
Mle DACHARRY Monique	Géographie
M. DEBRABANT Pierre	E.U.D.I.L.
M. DEGAUQUE Pierre	I.E.E.A.
M. DELORME Pierre	Biologie
M. DEMUNTER Paul	C.U.E.E.P.
M. DE PARIS Jean-Claude	Mathématiques
M. DEVRAINNE Pierre	Chimie
M. DHAINAUT André	Biologie
M. DORMARD Serge	S.E.S.
M. DOUKHAN Jean-Claude	E.U.D.I.L.
M. DUBOIS Henri	Physique
M. DUBRULLE Alain	Physique
M. DUEE Gérard	Sciences de la Terre
M. DYMENT Arthur	Mathématiques
M. FLAMME Jean-Marie	E.U.D.I.L.
M. FONTAINE Hubert	Physique
M. GERVAIS Michel	S.E.S.
M. GOBLOT Rémi	Mathématiques
M. GOSSELIN Gabriel	S.E.S.
M. GOUDMAND Pierre	Chimie
M. GREVET Patrice	S.E.S.
M. GUILBAULT Pierre	Biologie
M. HANGAN Théodore	Mathématiques
M. HERMAN Maurice	Physique
M. JACOB Gérard	I.E.E.A.
M. JACOB Pierre	Mathématiques
M. JOURNEL Gérard	E.U.D.I.L.
M. KREMBEL Jean	Biologie
M. LAURENT François	I.E.E.A.
Mle LEGRAND Denise	Mathématiques
Mle LEGRAND Solange	Mathématiques (Calais)
Mme LEHMANN Josiane	Mathématiques

M. LEMAIRE Jean	Physique
M. LENTACKER Firmin	G.A.S.
M. LEVASSEUR Michel	I.P.A.
M. LHENAFF René	G.A.S.
M. LOCQUENEUX Robert	Physique
M. LOSFELD Joseph	I.E.E.A.
M. LOUAGE Francis	E.U.D.I.L.
M. MACKE Bruno	Physique
M. MAIZIERES Christian	I.E.E.A.
Mle MARQUET Simone	Mathématiques
M. MESSELYN Jean	Physique
M. MIGEON Michel	E.U.D.I.L.
M. MIGNOT Fulbert	Mathématiques
M. MONTEL Marc	Physique
Mme NGUYEN VAN CHI Régine	G.A.S.
M. PARSY Fernand	Mathématiques
Mle PAUPARDIN Colette	Biologie
M. PERROT Pierre	Chimie
M. PERTUZON Emile	Biologie
M. PONSOLLE Louis	Chimie
M. PORCHET Maurice	Biologie
M. POVY Lucien	E.U.D.I.L.
M. RACZY Ladislas	I.E.E.A.
M. RICHARD Alain	Biologie
M. RIETSCH François	E.U.D.I.L.
M. ROGALSKI Marc	M.P.A.
M. ROUSSEAU Jean-Paul	Biologie
M. ROY Jean-Claude	Biologie
M. SALAMA Pierre	S.E.S.
Mme SCHWARZBACH Yvette (CCP)	M.P.A.
M. SCHAMPS Joël	Physique
M. SIMON Michel	S.E.S.
M. SLIWA Henri	Chimie
M. SOMME Jean	G.A.S.
Mle SPIK Geneviève	Biologie
M. STERBOUL François	E.U.D.I.L.
M. TAILLIEZ Roger	Institut Agricole

M.	TOULOTTE Jean-Marc	I.E.E.A.
M.	VANDORPE Bernard	E.U.D.I.L.
M.	WALLART Francis	Chimie
M.	WATERLOT Michel	Sciences de la Terre
Mme	ZINN JUSTIN Nicole	M.P.A.

CHARGES DE COURS

M.	TOP Gérard	S.E.S.
M.	ADAM Michel	S.E.S.

CHARGES DE CONFERENCES

M.	DUVEAU Jacques	S.E.S.
M.	HOFLACK Jacques	I.P..A
M.	LATOUCHE Serge	S.E.S.
M.	MALAUSSENA DE PERNO Jean-Louis	S.E.S.
M.	OPIGEZ Philippe	S.E.S.

REMERCIEMENTS

Je tiens à remercier

Monsieur le Professeur V. CORDONNIER, qui m'a fait l'honneur d'accepter de présider ce Jury et qui m'a constamment aidé de ses conseils au cours de mes études ;

Monsieur le Professeur C. CARREZ qui a dirigé cette thèse en ne me ménageant ni son temps, ni sa patience. Il a su, avec une gentillesse inlassable, m'apporter l'aide de sa compétence au long des nombreuses discussions qui ont accompagné la construction de cette Thèse ;

Monsieur le Professeur N. NAFFAH, Directeur de Recherche à l'INRIA et Monsieur M. LOUF, Ingénieur chez Matra qui ont bien voulu participer à ce Jury et honorer ce travail du parrainage de leur autorité ;

Monsieur le Professeur M. DAUCHET qui a accepté d'examiner cette étude. Son assistance bienveillante a joué un rôle déterminant dans la formalisation de ce travail ;

Monsieur M. MERIAUX, Attaché de Recherche au CNRS qui a aussi accepté d'examiner ce travail. Ses conseils fructueux, ses encouragements m'ont accompagné de façon permanente depuis mes débuts dans l'informatique graphique.

Je remercie également Mademoiselle B. FIEVET qui a courageusement réalisé la frappe de ce document, ainsi que Madame DEBOCK qui en a diligemment assuré l'impression ;

Je remercie enfin ma famille et tous les membres du Laboratoire qui, non contents de me supporter, on su me réconforter en temps utile.

A mes grands-parents,
à mes parents,
à Philippe.

INTRODUCTION

La synthèse d'images par ordinateur prend une place de plus en plus importante dans de nombreux domaines : Conception Assistée par Ordinateur, Bureautique, Arts Graphiques, Enseignement, etc... Il est nécessaire que l'utilisateur puisse agir très facilement sur ses images. On constate souvent que les systèmes proposés, i.e. les éditeurs graphiques, présentent des lacunes : difficulté d'emploi, inadéquation au type d'images (seuls les dessins au trait sont manipulés interactivement), implémentation souvent empirique ou non reproductible ; le chapitre I présente plus en détail ces différentes motivations.

L'étude poursuivie a donc pour objectif de spécifier un éditeur graphique sur le système STYX (de conception d'images par taches de couleur) conciliant la démarche de l'utilisateur avec les possibilités offertes par les machines actuelles ou à venir ; nous étudions en première partie les besoins de l'utilisateur et les notions qui en résultent. La seconde définit à partir de ces résultats les spécifications d'un éditeur de base et propose quelques extensions.

La troisième partie est une formalisation partielle de cet éditeur, destinée à fournir une référence rigoureuse pour l'implémenteur et pour l'usager.

CHAPITRE I

MOTIVATION DE L'ETUDE D'UN EDITEUR GRAPHIQUE

CHAPITRE I

MOTIVATION DE L'ETUDE

L'infographie interactive est l'une des réalisations informatiques les plus populaires actuellement ; son essor, dont l'origine venait de projets complexes de conception assistée par ordinateur, de contrôle de processus et même d'art graphique, se répercute actuellement - en égard au progrès technologique (LSI, VLSI) et à la baisse des coûts de production - sur des applications plus proches du grand public ; c'est ainsi que de nombreux jeux dits "électroniques" apparaissent sur le marché et des consoles graphiques de plus en plus perfectionnées sont commercialisées. Dans ce cadre d'utilisation, nous décrivons des voies de recherche en cours.

I.1. L'INFOGRAPHIE INTERACTIVE

Suite aux efforts de standardisation des systèmes graphiques, différents modèles ont été retenus ; le modèle du "GSPC" Core System [Herzog] peut se résumer selon la figure I.1.

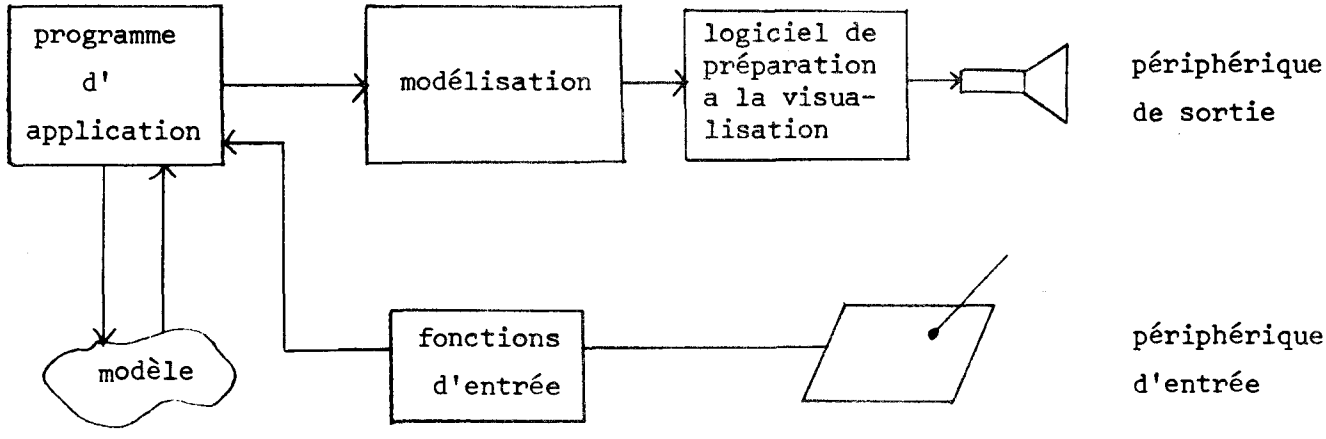


Figure I.1. : Modèle utilisé par le "GSPC Core Systèm".

Ce modèle, controversé par les lacunes qu'il présente au niveau des fonctions d'entrées, concurrence un modèle plus symétrique [Lucas, Kilgour] schématisé dans la figure II.2.

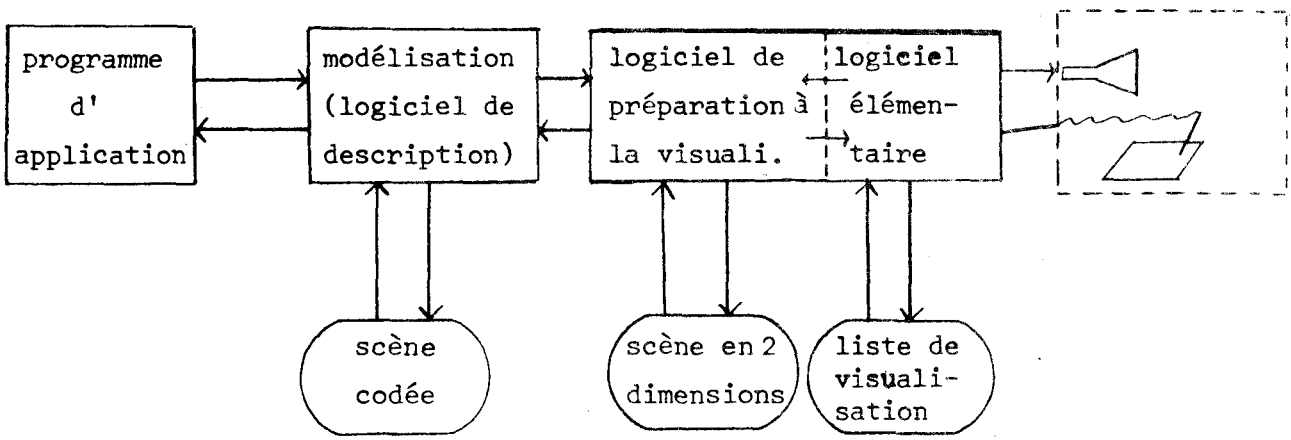


Figure II.2. : Schéma général d'un logiciel graphique interactif.

Ce dernier modèle met en évidence les interactions existant aux différents niveaux de logiciel. Ces interactions, essentielles pour la communication homme-machine, sont l'objet de recherches pour une utilisation simple des systèmes graphiques ; alors que les dispositifs matériels et logiciels de sortie sont adaptés au travail en mode graphique, les primitives d'entrée agissent rarement sur la scène codée traitée par le logiciel de description et le programme d'application, exception faite des primitives de transformations géométriques, qui peuvent servir aussi bien pour la mise en page finale sur l'écran que pour la description de la scène [Core, GKS]. De façon générale, il est assez difficile d'accéder, par des moyens exclusivement visuels à des éléments de la scène codée ou à des données de l'application. Des travaux ont été entrepris depuis plusieurs années, et fournissent quelques réalisations [Staudhammer] mais le débat porte encore sur la méthodologie de l'interaction, particulièrement dans le domaine graphique [Guedj, Baecker, Van Den Bos] .

Dans la plupart des cas, nous constatons que l'interaction est, soit limitée dans les hiérarchies des objets-graphiques, soit relativement peu commode à utiliser, les mécanismes ne permettant pas d'accéder aisément à des noeuds d'arborescences ; de plus, elle s'applique généralement à la synthèse d'images "dessinées au trait" (wire frame images) et non pas directement à des images considérées comme des taches de couleur, analogues à celles visualisées sur des écrans à balayage de type "télévision" ("raster screen"). Ce type d'écran est largement répandu, et des études sont entreprises depuis quelques années dans la conception de systèmes graphiques pour ce type d'écran.

Dans le domaine de la création et manipulation interactives d'images, quelques "éditeurs" ont été définis et développés récemment ; nous en présentons quelques uns en I.3, dont l'étude est chronologiquement légèrement antérieure ou parallèle à notre travail, et qui sont adaptés au dessins au trait ou aux images en couleur.

I.2. INFORMATISATION DU TRAVAIL DE BUREAU : LA BUREAUTIQUE

De façon relativement peu liée à l'informatique graphique, qui nécessite l'existence de processeurs spécialisés, l'informatisation s'étend, grâce à la miniaturisation et aux coûts compétitifs, dans les PME et PMI, les plus petites d'entre-elles ne s'étant informatisées qu'à l'apparition des mini et micro-ordinateurs. Ce phénomène d'informatisation s'applique au domaine du travail de bureau : commercialisation d'éditeurs de textes, plus ou moins perfectionnés et autonomes, ou qui s'intègrent dans les systèmes de gestion de bases de données documentaires, par exemple [Trefle].

Les fonctions de dialogue utilisées dans les systèmes de traitement de textes sont intéressantes à étudier ; on remarque, de façon générale, qu'un système sera d'autant mieux accepté que l'apprentissage des fonctions paraît naturel et que les commandes se présentent indifféremment sous forme condensée ou explicite selon l'aisance acquise par l'utilisateur pour les différentes manipulations. L'ensemble des documents textuels peut,

par de tels systèmes, être constitué, modifié facilement, visualisé, archivé... mais les problèmes surgissent dès qu'il y a introduction de schémas dans les documents, car l'infographie est rarement utilisée conjointement à de tels systèmes : la conception des schémas se fait alors "à la main" et l'archivage donne lieu à des traitements d'exceptions (quand il est prévu !).

Des efforts ont donc été poursuivis afin de remédier à cette lacune ; par la conception d'éditeurs de textes et d'images compatibles, permettant la création de documents mélangeant texte et images [Shepherd, Kayak].

Plusieurs tendances se développent : d'une part la conception d'éditeurs de textes et d'éditeurs graphiques spécialisés, indépendants, mais dont les fichiers peuvent être (parfois) tous archivés de façon compatible ; c'est la démarche la plus fréquemment suivie lorsque l'une des deux données (texte ou image) est relativement très peu importante. D'autre part, la conception d'éditeurs à fonctions multiples, intégrant l'édition de textes et l'éditeur d'images (éditeur Plume de Kayak). Cette démarche assure une meilleure unité de conception de documents pour l'utilisateur, toutes les données étant structurées de façon assez voisines, compte tenu de la nature des informations [Naffah, Borron].

Nous allons maintenant étudier quelques éditeurs d'images, adaptés à la bureautique ou à des applications plus vastes, par exemple, les arts graphiques, la décoration...

I.3. PRESENTATION DE QUELQUES EDITEURS GRAPHIQUES

Les premiers éditeurs graphiques ont été réalisés pour les besoins de la C.A.O, et sont, de ce fait, très adaptés au dessin industriel [CADAM, LGIVD,... ; ils sont destinés à un public restreint, très qualifié dans ce domaine et présentent donc un degré de complexité et une puissance importante.

Parallèlement, des éditeurs graphiques sur mini et micro-ordinateurs se développent ; ils sont destinés à un public moins qualifié que le précédent dans le domaine informatique ; nous étudions succintement quelques-uns de ces éditeurs d'images, significatifs de l'état de l'art actuel : l'éditeur "PLUME" de KAYAK, l'éditeur graphique GRED, PAM, et le logiciel de STYX.

PLUME [Naffah 81]

Spécifié à partir des besoins et des contraintes (matérielles, logicielles, ergonomiques et économiques) mis en évidence par le phénomène bureautique, cet éditeur est conçu de façon à être aisément utilisable, et à correspondre aux activités d'édition de textes et d'images, qui représentent une part importante du traitement de l'information dans un bureau (21 % de saisie de documents [Borron]). Dans le cadre général du travail de bureau, moins de 10 % des originaux comportent des graphiques ou des dessins. Le buroviseur est donc principalement conçu pour l'édition de textes et de graphiques. Ces derniers sont classés en deux types : le "dessin géométral" (droite, cercle, ligne polygonale, courbe) et le dessin à main levée, quelconque ; ils correspondent à la majorité des schémas que l'on peut trouver dans des documents, et leur complexité est relativement faible. Nous pouvons cependant noter que les "images" ne sont pas structurées et n'ont pas de composante couleur, que possèdent les photographies, d'ailleurs rares dans les documents classiques.

Notre but est d'étendre cet aspect bureautique d'édition de dessins à la synthèse d'images en couleur par tache, dans la perspective d'une utilisation dans des domaines d'activités plus orientés vers l'esthétique : décoration intérieure, architecture, décoration paysagiste...

GRED [Voirol 81]

GRED est un éditeur graphique de dessins "au trait". Les objets graphiques sont hiérarchiques (modèle le plus couramment utilisé en informatique graphique [Mallgreen et Shaw]) ; les objets sont composés d'éléments primitifs simples et sont utilisables de la même manière que ces derniers ; on peut composer des objets et interpréter le graphisme (des traits) à chaque niveau de composition, le nombre de niveaux n'étant pas limité. L'interaction est obtenue soit par un programme de dialogue agissant sur les objets de la scène, soit par le programme d'application lui-même. Il n'y a pas actuellement d'interaction possible par identification visuelle des objets de niveau quelconque : les entrées sont essentiellement alpha-numériques et les mécanismes d'entrée graphiques ne sont pas (encore) définis. Nous pouvons également constater une certaine confusion entre des objets de nature différente : les objets répétitifs et les objets récursifs : alors que la définition d'un objet répétitif est utilisée à une transformation géométrique et un graphisme près, un certain nombre de fois et de la même façon (construction d'un escalier par répétition de l'objet MARCHE), les objets récursifs s'utilisent eux-mêmes dans leur propre description, ce qui complique la description, la visualisation et l'identification de tels objets. En conclusion, cet éditeur est un bon début d'insertion de l'informatique graphique à faible coût dans le cadre d'applications diverses, mais sa puissance est limitée au dessin au trait et son emploi manque un peu de naturel !

PAM [Lakin 80]

Ce système graphique, appelé PAM pour "Pattern Manipulating" possède les caractéristiques suivantes : il associe l'édition de textes et l'édition graphique de dessins "au trait". Les objets graphiques manipulés sont structurés de façon hiérarchique et les fonctions d'entrée agissent sur la structure de ces objets ou sur leur placement dans l'espace ;

la hiérarchie est représentée par des lignes horizontales (les niveaux) et verticales (les éléments reliés à un niveau) reliées aux éléments du dessin ; les fonctions agissant sur la structure sont indépendantes de celles agissant sur les informations spatiales (transformations géométriques et fonctions de visualisation), ce qui facilité la tâche de l'utilisateur. Nous pouvons néanmoins reprocher quelques inconvénients : le manque de clarté de l'écran dû aux traits représentant les hiérarchies, et l'incapacité de fournir des images en couleur ; de plus, les fonctions de visualisation), bien qu'adaptées à la structure des données, ne s'avèrent pas toujours correspondre parfaitement aux besoins de l'utilisateur.

STYX [Rauch, Grimonprez 79]

Le logiciel STYX est conçu pour la synthèse interactive en couleur, sans obliger l'utilisateur à concevoir ses images par l'intermédiaire de dessins "au trait" (cette démarche consistant à dessiner les contours puis à calculer les images est souvent utilisée [Staudhammer]). Les objets graphiques sont hiérarchiques, de profondeur quelconque, et les objets élémentaires sont visualisés par des taches monochromes dont la forme est commandée par les informations liées aux feuilles.

Des fonctions structurelles permettent de créer, de visualiser ces images en couleur, mais nous constatons des lacunes au niveau des transformations géométriques et de l'aspect réaliste des images, car les taches sont monochromes (pas de texture, d'ombrage, de variations d'éclairement). Nous constatons également que tous les attributs (forme, taille, position, couleur...) sont déterminés au niveau des feuilles ; de plus l'absence de clôture de visualisation et de fenêtre sur le "monde réel" entraîne que l'écran est un repère absolu pour la visualisation. Ces particularités devraient être modifiées lors de la conception d'un éditeur d'images plus élaboré.

En guise de conclusion, et d'après ses caractéristiques, ce système de base (relativement peu riche au niveau des fonctions d'édition d'images) constitue un cadre privilégié pour l'étude d'un éditeur d'images hiérarchiques en couleur utilisable de façon interactive par des non-spécialistes en informatique.

I.4. VERS UNE FORMALISATION DES LANGAGES

La démarche de formalisation, maintenant habituelle pour les langages de programmation de type classique (sémantique d'ADA), est originale à notre connaissance pour un logiciel graphique interactif, bien qu'évoquée succintement dans BUMPS.

Les motivations d'un tel travail sont les mêmes que pour un langage classique : il s'agit de savoir très précisément "ce que signifie" chaque instruction sur les objets manipulés, i.e., sur l'environnement. Ceci permet d'avoir grâce à la possibilité de prouver chaque instruction une **une référence indiscutable et admise** de tous pour le langage considéré. L'enjeu en est la qualité du langage : la fiabilité, l'extensibilité, la transportabilité...

Il est remarquable que l'approche formelle nous ait amené à modifier certains points de notre langage et conduit à quelques retouches qui se sont avérées comme empiriquement pertinentes.

La donnée de cette sémantique peut être la référence pour l'implémenteur ; celui-ci n'aura qu'à simuler l'outil formel utilisé, i.e., les règles de transducteur, qui sont des références souples offrant de larges latitudes de choix d'implémentation (il suffira de prouver l'équivalence avec la référence) et d'extension (il suffira de rajouter des règles).

L'outil formel que nous utilisons est inspiré de la notion de transducteur d'arbres. Les transducteurs d'arbres sont nés de préoccupations de compilation par attributs (syntax directed translators), l'idée étant là de conpiler de façon guidée par la syntaxe, donc guidée par l'arbre syntaxique du programme, et en dernier ressort, de transformer des arbres.

Les objets que nous manipulons (cf. II, III) dans notre langage ayant une structure logique d'arbres, le choix de cet outil était tout indiqué, et la sémantique de notre langage sera donc donnée par la sémantique d'un transducteur d'arbres.

Les sémanticiens reconnaîtrons en cette description une sémantique algébrique en ce sens que l'outil-transducteur est un objet syntaxique, et qu'il ramène la sémantique à des manipulations formelles. On peut poser des équations, les résoudre, trouver des propriétés des programmes et en prouver l'invariance (nous le faisons dans le chapitre VI à titre d'exemple). Notre sémantique a donc le mérite d'être aussi dénotationnelle. Elle est enfin opérationnelle en ce sens que chaque règle décrit l'effet sur l'environnement de chaque instruction.

I.5. PRESENTATION DE L'ETUDE

Nous avons cherché à définir un logiciel interactif de synthèse d'images en couleurs par taches qui corresponde aux applications bureautiques (au sens le plus large) de l'informatique graphique. Contrairement à de nombreux logiciels de synthèse de dessins "au trait" complétés par des fonctions de remplissage des dessins, le logiciel que nous proposons est basé sur la notion d'image composée de taches de couleurs [STYX].

Nous étudions au chapitre II la démarche de l'utilisateur face à un tel système, quelles sont ses méthodes de conception d'images, quelles sont les fonctions nécessaires, pour obtenir un logiciel attrayant. Nous mettons en évidence quelques problèmes à résoudre.

Dans le chapitre III, nous présentons les principales fonctions de l'éditeur graphique appliquées sur les objets visualisés : accès, modifications structurelles et spatiales, et nous abordons des problèmes de représentation interne des transformations géométriques.

Le chapitre IV décrit des objets particuliers du système : les objets catalogués qui permettent l'accès aux objets graphiques non visualisés, les objets motifs qui donnent la possibilité d'engendrer des répétitions d'objets, et les objets récursifs qui se décrivent en utilisant leur propre description.

Dans le chapitre V, nous proposons une extension possible du système, réalisée par des objets graphiques incomplètement spécifiés qui servent de modèles : ce sont les objets-patrons.

Le chapitre VI comporte la formalisation des fonctions d'accès décrites au chapitre III. Il s'agit là d'un exemple de définition formelle des fonctions de l'infographie interactive c'est le premier pas vers une implémentation rigoureuse et sûre des fonctions de l'éditeur d'images. Pour illustrer l'intérêt pratique de notre formalisme, nous développons un exemple, celui d'une fonction appelée "1er noeud commun" (VI.6.).

CHAPITRE II

LA DEMARCHE DE L'UTILISATEUR INTERACTIF

CHAPITRE II

LA DEMARCHE DE L'UTILISATEUR INTERACTIF

Par analogie avec les éditeurs de textes, qui sont des aides incontestables dans la conception de documents divers dans des environnements bureautiques ou de programmes dans les centres informatiques, nous définissons un éditeur d'images pour des utilisateurs de culture informatique variable, a priori faible. Après avoir décrit ce que voit l'utilisateur et défini les notions de base qu'il est souhaitable d'avoir, nous présentons la méthodologie de l'utilisateur sur le plan du dialogue homme-machine, puis surtout sur le plan de la synthèse d'images proprement dite.

Nous énonçons enfin quelques problèmes inhérents aux manipulations d'objets à partir de leur représentation sur un écran de visualisation.

II.1. LES OBJETS DE L'UTILISATEUR

Un des outils essentiels de l'infographie interactive est l'écran de visualisation ou plus simplement, l'écran. Il est perçu comme une surface rectangulaire (ou carrée) constitué de points dans laquelle des images sont visualisées. Chacune de ces images est contenue dans une portion de l'écran définie au gré de l'utilisateur : c'est la *clôture* de l'image. L'image la plus élémentaire est une tache monochrome, i.e., une zone "d'un seul bloc" délimitée par un contour fermé qui partage l'espace en deux parties distinctes : l'intérieur et l'extérieur de la tache. Le contour est supposé appartenir à l'intérieur de la tache. D'une façon générale, une image est composée d'un ensemble de taches. Elle résulte de la visualisation d'un certain objet informatique appelé objet-image qui définit la place, la nature, la taille de ces taches.

Un objet-image est un descripteur de structure qui contient :

- des informations liées à la structure de l'objet-image (i.e., les éléments qui le composent),

- des informations liées à la géométrie et à la visualisation (i.e., la forme des éléments, la couleur, la texture, la transparence...).

Ce sont les attributs graphiques. La visualisation d'un objet-image est une image ; cette image est la réunion des images de tous les éléments qui composent l'objet-image. Les objets-images ainsi décrits sont dits "structurés". La décomposition à un ou deux niveaux est expérimentalement suffisante dans de nombreuses utilisations, mais un peu restrictive [GRIGRI]. On peut considérer cette décomposition d'objets, dont les plus élémentaires sont visualisés par des taches, comme potentiellement infinie.

Nous appelons "*feuille*" un objet-image qui n'est pas décomposable dans la structure. Un noeud est un élément quelconque dans la structure d'un objet.

Tout objet-image est défini par un noeud appelé "*racine*" de l'objet-image ; l'objet-image est visualisé par l'ensemble des images des éléments qui le composent, c'est-à-dire par l'ensemble des images des feuilles utilisées dans la description de la racine. Cette description des objets est classique [Nw-Sp],[GRED] car elle correspond au mode de pensée de l'utilisateur, nomenclatures, organigrammes d'entreprise... C'est une façon très naturelle de concevoir la description d'objets à visualiser. Par exemple une maison est composée de murs, d'un toit, d'un garage ; chaque mur est lui-même composé de maçonnerie, de fenêtres et de portes éventuellement, le toit peut aussi se décrire plus finement etc... Une façon de structurer une telle maison serait la suivante :

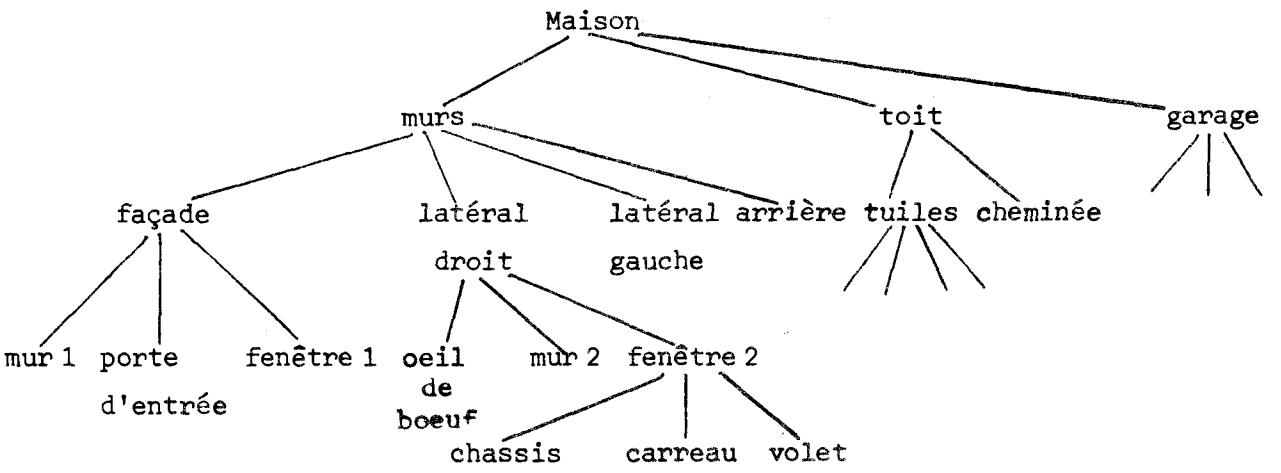


Figure II.1. : Exemple de structure d'un objet-image.

On dira qu'un objet-image O est une arborescence logique de racine O, dont les noeuds intermédiaires sont des sous-objets-images. Les éléments qui composent un noeud sont les descendants de ce noeud ; les feuilles sont les noeuds qui n'ont pas de descendant. Sur l'exemple, porte d'entrée, fenêtre 1, cheminée sont des feuilles tandis que murs, toit, façade sont des noeuds intermédiaires de l'objet-image Maison. Tous les noeuds sont considérés comme des objets-images, qu'ils aient des descendants (on les dit structurés) ou pas. Ce concept général d'objet-image respecte les conditions de simplicité, de compréhension et de conception qui sont essentielles pour l'utilisateur non-spécialiste de l'informatique.

II.2. NOTIONS DE CONTEXTE ET DE NOEUD COURANT

Pour permettre de visualiser une partie d'objet-image, correspondant à un sous-objet-image, dans une clôture, nous introduisons la notion de contexte de visualisation, ou encore de *contexte*. Un contexte repère un noeud logique n de l'objet-image. Il définit l'image affichée dans une clôture comme la visualisation du sous-objet-image de racine n.

Dans une clôture déterminée de l'écran, l'utilisateur peut visualiser successivement différents contextes d'un objet-image, chacun de ces contextes occupe l'espace maximum contenu dans la clôture de façon à ne pas modifier les proportions de l'image du noeud. Par exemple, si le noeud est un carré, et la clôture un rectangle, l'image du noeud sera un carré dont la longueur des côtés est égale à la plus petite dimension de la clôture. La notion de contexte est très utile pour l'utilisateur : c'est l'environnement dans lequel il désire visualiser les objets qu'il manipule. Cette démarche se retrouve dans des domaines diversifiés :

- En bureautique, la visualisation d'une lettre entière pour travailler sur un ou plusieurs paragraphes mis en évidence au sein de la lettre entière.

- En architecture, l'architecte conçoit un chalet, mais cherche à s'assurer que cette habitation s'intègre bien dans le cadre qui sera le sien.

Le contexte doit être étendu ou réduit aisément de façon à "voir" les images dans des environnements de plus en plus vastes, ou de plus en plus limités jusqu'à la restriction de l'environnement d'un objet à lui-même. Dans ce cas, l'objet est visualisé de façon "intrinsèque", c'est-à-dire, tel qu'il est, indépendamment de tout environnement. Par exemple, c'est un paragraphe de lettre qui est élaboré et modifié indépendamment du texte qui le contient ; c'est aussi la constitution d'un chalet, sur lequel l'utilisateur concentre son attention : le chalet doit apparaître dans l'espace maximum de la clôture, plutôt que réduit, perdu dans la montagne visualisée dans cette clôture.

A un instant donné, l'utilisateur se positionne sur une sous-image du contexte. Cette sous-image est la visualisation d'un noeud appelé *noeud courant* sur lequel l'utilisateur travaille. Tout noeud descendant d'un noeud contexte peut, à des moments distincts, être le noeud courant. En particulier, le noeud courant peut être le noeud contexte : l'utilisateur travaille sur le noeud visualisé.

Prenons la maison comme elle est structurée dans la figure II.1.

- la figure II.2. représente le noeud courant "mur latéral droit" visualisé dans le contexte de la maison.

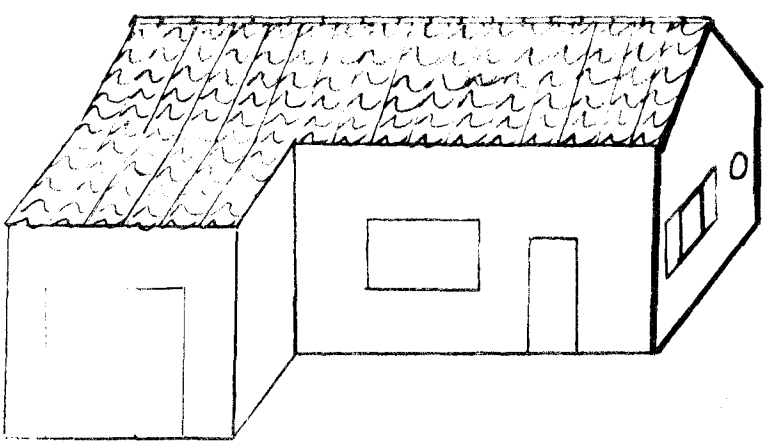


Figure II.2. : Visualisation du noeud courant "mur latéral droit" dans le contexte "maison". Le contour du noeud courant (i.e., ce mur) est graissé de façon à le distinguer du reste de la maison.

- la figure II.3. représente le noeud contexte "mur latéral droit", le noeud courant est le noeud contexte.

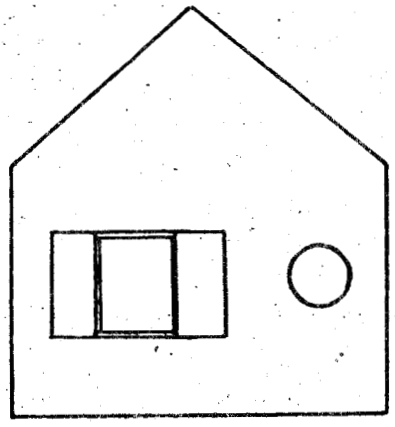


Figure II.3. : Visualisation du noeud courant "mur latéral droit" dans le contexte "mur latéral droit".

II.3. NOTION DE CONTEXTE MULTIPLE ET DE CONTEXTE UNIQUE

L'utilisateur peut désirer voir un même objet de façon simultanée dans différents contextes sur le même écran. Par exemple, l'architecte visualise un chalet dans une clôture de l'écran et la porte de son chalet dans une autre clôture. S'il modifie cette porte, il verra immédiatement l'effet de sa modification dans le contexte du chalet. Toujours sur cet écran, l'utilisateur peut avoir envie de voir des objets distincts : chaque clôture dans l'écran constitue en quelque sorte un sous-écran dans lequel est visualisé un contexte : on obtient l'équivalent d'un poste de travail multi-écran correspondant au nombre de contextes visualisés simultanément, d'où son appellation de *contexte multiple*. Ce contexte doit permettre de désigner et de travailler successivement sur des images appartenant à des clôtures différentes.

En contexte multiple, dans chaque clôture est constamment visualisé un noeud contexte, qui contient le noeud courant pour ce contexte : c'est le dernier noeud sur lequel l'utilisateur a été positionné. De plus, à un instant donné, l'utilisateur est positionné sur une et une seule image, (un noeud courant) dans un seul contexte contenu dans une seule clôture : c'est la clôture de "travail", unique. Changer de clôture de travail revient à se positionner dans la clôture désirée, sur le noeud courant du contexte visualisé dans cette clôture.

De façon symétrique, ce même usager est susceptible de ne vouloir visualiser qu'un et un seul contexte sur l'écran ; dans ce cas il est souhaitable que ce noeud contexte occupe le maximum de place sur l'écran, étant donné qu'il est seul visualisé. Ce mode de visualisation permet d'isoler un contexte parmi les contextes visualisés en contexte multiple : c'est le contexte visualisé dans la clôture de travail, et l'on dit que l'utilisateur travaille en *contexte unique* sur l'objet-image visualisé.

Ces modes "*contexte unique*" et "*contexte multiple*" ne doivent évidemment n'affecter que la visualisation des objets-images ; ce sont en quelque sorte des modes de fonctionnement ; le passage d'un mode à l'autre ne doit pas modifier les noeuds contextes et les noeuds courants.

II.4. METHODOLOGIE DE L'UTILISATEUR

II.4.1. Communication homme-machine

L'utilisateur interactif communique avec le système graphique selon le principe schématisé dans la figure II.4. :

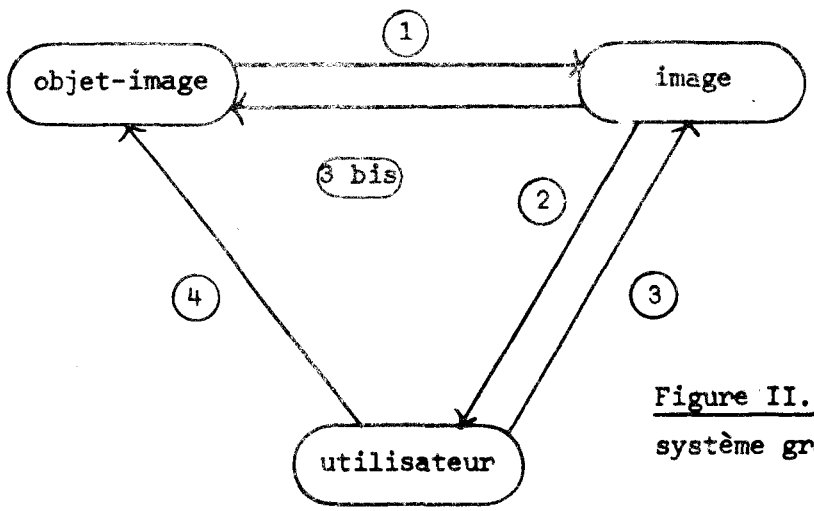


Figure II.4. : Principe du système graphique interactif.

(1) : l'objet-image est interprété pour donner une représentation : son image.

(2) : l'utilisateur voit l'image sur l'écran. S'il veut la modifier il utilise soit (3) et (3 bis), soit (4).

(3) : l'utilisateur se base sur l'image de l'objet qu'il élabore pour décrire une transformation.

(3 bis) : la transformation est effectivement appliquée sur l'objet-image après interprétation de la commande par (1), il y a visualisation de l'objet-image transformé.

(4) : l'utilisateur agit directement sur l'objet-image, sans se baser sur la représentation de cet objet-image.

Le cheminement (3) - (3 bis) paraît le plus naturel pour l'utilisateur.

La communication se fait à plusieurs niveaux :

- au niveau des fonctions de dialogue, l'utilisateur doit disposer de moyens lui permettant de manipuler aisément ses images.

- au niveau de la méthodologie de construction et de manipulation d'images. Nous retenons deux types principaux de fonctions, celles concernant la structure des objets-images : accès, modification de structure et celles spécifiquement graphiques : saisie ou calculs de contours et de formes, affichage, modifications des attributs graphiques (couleur, forme, position, texture) qui ne changent généralement pas la structure des objets-images.

II.4.2. Les fonctions de dialogue

L'interaction est une caractéristique essentielle de l'éditeur graphique ; les moyens de dialogue mis à la disposition de l'utilisateur sont essentiellement les dispositifs de l'infographie interactive.

Nous retiendrons les quatre fonctions suivantes en tant qu'éléments de base du dialogue [Luc 77].

* Le menu, qui permet à l'opérateur de choisir une action à effectuer dans une liste de commandes.

* L'introduction de valeurs alphanumériques permettant d'affecter une valeur à un ou plusieurs paramètres.

* L'identification d'un (ou plusieurs) composant d'une image, permettant ainsi d'indiquer l'élément auquel est appliqué un certain traitement.

* L'acquisition d'un élément d'image, permettant ainsi de construire une représentation destinée au calculateur.

Ces quatre éléments peuvent être combinés entre-eux et nous nous efforcerons de définir les actions de l'utilisateur en fonction de ces éléments de base, sans supposer l'existence de dispositif de dialogue particulier - en égard à la grande diversité des procédés disponibles sur le marché. Nous nous contentons de rappeler à l'aide du schéma II.3 l'adéquation des dispositifs de communication (les plus usités) aux fonctions de dialogue.

	Identification	Menu	Valeurs	Coordonnées
Photostyle	Désignation directe sur l'écran du composant à identifier	Désignation directe sur l'écran de la commande	Entrée carac. par caractère à partir d'une zone alphabet	a) Point par point b) en continu par poursuite
Clavier alphanumérique	- Donnée du nom de l'élément - guidage d'un symbole	Frappe du nom de la commande	Frappe de la chaîne de caractères	Frappe des suites de coordonnées
Clavier de fonctions	Guidage d'un symbole par touches directionnelles	Appui sur une touche		
tablettes	Guidage d'un symbole	a) Guidage d'un symbole b) Découpage en en zones c) Reconnaisseur de symboles	- Découpage en zones - Reconnaisseur de caractères	a) point par point b) en continu
Réticule	Désignation directe	Désignation directe		Point par point
Manche à balai	Guidage d'un symbole	Guidage d'un symbole		a) point par point b) en continu

Les cases barrées indiquent des fonctions qu'il n'est pas raisonnable de simuler avec le dispositif considéré.

Figure II.5. : Adéquation des dispositifs de communication aux fonctions de dialogue.

II.4.3. Comment construit-on une image ? Conséquences

Regardons la démarche d'un utilisateur potentiel parfaitement ignorant de l'informatique : le comportement le plus naturel qui soit est celui d'un enfant qui dessine et fait des coloriages. Il s'installe avec son papier, ses crayons de couleur et commence à griffonner, qui un soleil, qui un arbre (d'abord le tronc puis le feuillage), qui une maison (en plusieurs morceaux).

A partir d'images simples (qui correspondent souvent à des tâches de couleur), il compose des images plus complexes, par exemple, la maison, jusqu'à obtenir l'image finale, la plus élaborée : c'est la construction ascendante qui procède par regroupements successifs d'éléments d'images et de groupes d'éléments.

Un autre comportement se retrouve assez fréquemment : l'utilisateur a une idée de ce qu'il veut construire, et il décompose en "morceaux" de manière à réduire la complexité de la scène qu'il élabore ; c'est le cas de l'architecte qui crée un immeuble : il est composé d'un ensemble d'étages, et chaque étage se décompose lui-même en appartements...

Ces deux méthologies se complètent : alors que la première effectue une sur-définition d'un objet, i.e., une définition condensée globale pour un ensemble d'éléments, la seconde permet une sous-définition d'un objet dont on affine la description. Dans tous les cas, il s'agit d'une structuration logique d'objets hiérarchiques. Il y a parfaite concordance entre la structure des objets et la méthodologie de conception et d'utilisation ; le tout répond au besoin de simplicité de conception qui a été évoqué en II.1.

II.5. LES PROBLEMES D'IDENTIFICATION ET D'ACCES

II.5.1. Présentation des types d'accès

Il existe deux types d'accès :

* l'accès à un objet-image non visualisé ; c'est par exemple l'accès à un objet-image dans la base de données.

* l'accès à un objet-image visualisé (éventuellement en partie).

Le premier type d'accès ne peut se faire que de façon explicite, c'est-à-dire par nommage de l'objet : le nom de l'objet est la seule information qui permet d'y accéder. Les problèmes posés par les accès aux objets dans une base de données sont très généraux, nous en étudions quelques aspects au chapitre IV.

Le second type d'accès peut, lui, se réaliser de deux façons :

- de façon explicite, on accède à un objet par nommage ; c'est un objet-catalogue.

- de façon implicite, on accède à un objet-image par son identification "visuelle", à partir de son image sur l'écran.

On dit que l'on désigne un objet-image quand on l'identifie à partir de sa visualisation (que l'objet-image soit visualisé complètement ou partiellement). La désignation d'une image identifie l'objet-image correspondant ; c'est la façon la plus naturelle d'accéder à un objet-image, car l'utilisateur voit (au sens propre du terme) sur quel noeud il agit, et quelle est la portée des fonctions qu'il désire appliquer sur ses objets-images.

Le tableau suivant résume les façons d'accéder à un objet selon qu'il est ou non visualisé.

Types d'accès / Façon	Objet non visualisé	Objet visualisé
Explicite : nommage	oui	oui
Implicite : désignation	non	oui

Les éléments soulignés du tableau correspondent aux identifications privilégiées selon les types d'accès.

Nous étudions principalement les accès par désignation des objets, car ils sont à la base de la conception d'un éditeur graphique simple d'emploi. Nous supposons qu'il existe un dispositif permettant les accès implicites, et ne précisons pas de dispositif de dialogue particulier (II.2.2.). Ce dispositif est capable de retrouver une feuille de l'arborescence d'un objet-image à partir de coordonnées (correspondant à un point de son image sur l'écran de visualisation).

II.5.2. Problèmes d'identification des noeuds

Considérons les différents problèmes d'identification des noeuds sur un exemple simple. Soit l'objet-image "Maison" du II.1 visualisé comme dans les figures II.2 et II.3.

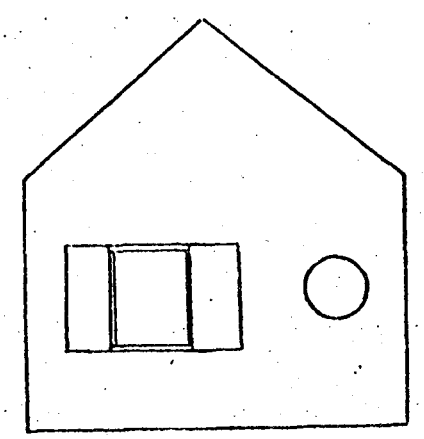


Figure II.6. : Rappel de la figure II.3 : Visualisation du mur latéral droit.

Nous allons considérer quelques cas particuliers explicites, qui correspondent à la position des volets de la fenêtre.

- 1er cas : les volets sont fermés, le chassis et le carreau ne sont pas du tout visibles.
- 2ème cas : un seul des volets est fermé : la moitié de l'encadrement est visible (idem pour le carreau).
- 3ème cas : les volets sont ouverts : c'est le mur qui est maintenant partiellement visible.

De façon plus générale, on peut également rencontrer, selon la façon dont on a conçu les objets-images, des parties d'images communes à deux éléments de l'image (jonction des volets fermés en assemblage à mi-bois) et des parties incluses dans d'autres (par exemple, découpage du carreau en petits carreaux).

Les principaux problèmes à résoudre sont les suivants :

* Comment peut-on identifier un noeud qui n'est pas visible ?

* Comment peut-on identifier un noeud partiellement visible?

* Comment résoudre l'ambiguïté de désignation par un point qui appartient à deux éléments de l'image ?

* Qu'est-ce qui est identifié lorsqu'un élément d'image est inclus dans un autre ? Comment obtenir sans ambiguïté l'un ou l'autre de ces éléments ? (i.e., l'englobé ou l'englobant).

* Comment reconnaître le noeud courant ?

* Comment accéder simplement aux noeuds de la structure ? (par exemple, à un mur particulier de la maison, ou au toit, ou à une porte...).

* Comment transformer géométriquement un noeud quelconque de façon globale ? (par exemple, réduire globalement une fenêtre et son volet).

* Comment créer, ajouter, supprimer des éléments d'image dans une structure ?

Des problèmes mineurs d'allègement de la tâche de l'usager doivent également être étudiés pour que l'éditeur ne soit pas ennuyeux à utiliser :

* Que faut-il faire pour cataloguer la maison et l'utiliser sans avoir à la redécrire ?

* Comment pourrait-on utiliser cette structure grossière comme une ébauche pour la conception de diverses maisons toutes construites sur ce même modèle ?

* Comment pourrait-on structurer une feuille de façon à avoir des détails répétitifs ? par exemple, pour obtenir un mur de briques sans avoir à décrire et positionner chacune des briques.

Des réponses à ces différents problèmes vont être apportées par les mécanismes que nous proposons dans les chapitres suivants.

II.5.3. Les accès. Notion de chemin

Les dispositifs de dialogue ne permettent d'identifier que des feuilles d'arborescences. Les fonctions d'accès doivent donc permettre de "remonter" dans la structure pour modifier un noeud, puis de "redescendre" sur des noeuds plus élémentaires.

Les opérations de montée sont simplifiées par le fait qu'un noeud a au plus un ascendant direct ; Par contre, les primitives de descente doivent permettre de revenir "de là d'où l'on vient", et d'accéder sans surprise pour l'utilisateur à l'un des descendants d'un noeud. Les parcours effectués à partir de l'identification d'un noeud doivent toujours être réalisés dans le même mode de cheminement, ce qui assure une bonne cohérence des accès. A priori, il faudrait que chaque primitive puisse en annuler une

autre (qui lui est symétrique) par exemple, la descente annule la montée dans une arborescence ; mais la descente peut être demandée sur un noeud dont on ne vient pas par un de ses descendants. Il faut donc décider quel noeud descendant choisir ! Ce mode de parcours s'appelle le *"cheminement courant"*.

Une autre façon de concevoir les parcours est de définir de façon générale toutes les fonctions de parcours, sans se soucier des "retours" à des noeuds auxquels on a accédé antérieurement, ces retours étant réalisés par une fonction explicite spécialement conçue dans ce but. Ce mode de parcours s'appelle le *"cheminement trace"*, car on conserve la trace des accès pour le retour.

Dans tous les cas, on accède à des noeuds qui sont donc visualisés, et contenus dans un certain contexte de visualisation.

Lors de la désignation d'un objet-image descendant du noeud contexte, l'objet sélectionné devra être mis en évidence par un artifice graphique (clignotement, sur-éclairement, changement de brillance...). Ces moyens de repérage d'une image sont comparables aux inversions-vidéo ou clignotement utilisés pour mettre en évidence des mots dans des textes.

II.5.4. Conclusion

Nous avons présenté quelques besoins de l'utilisateur et des caractéristiques que doit vérifier un éditeur graphique pour qu'il soit attrayant, simple d'emploi.

En particulier, les fonctions d'accès doivent toujours fournir le noeud le plus "logique" vis-à-vis de la compréhension de la structure par l'utilisateur.

Nous allons maintenant proposer des mécanismes d'accès aux images qui répondent à ces contraintes, puis nous étudierons les fonctions de création, d'ajout et de suppression d'objets graphiques. Le chapitre III

s'achèvera sur l'étude des transformations géométriques et graphiques et de leur influence sur la structure des données en mémoire centrale.

CHAPITRE III

LES FONCTIONS DE L'EDITEUR GRAPHIQUE.

MECANISMES PROPOSES.

CONSEQUENCES

CHAPITRE III

LES FONCTIONS DE L'EDITEUR GRAPHIQUE

MECANISMES PROPOSES. CONSEQUENCES

III.1. DEFINITIONS

Les objets du système correspondent aux objets de l'utilisateur. Définissons les :

Un *objet-image* est un descripteur de structure qui contient des informations relatives à la visualisation. Considérons un objet-image O composé des sous-objets $O_{i_1}, \ldots, O_{i_p}$. L'objet O forme une arborescence de profondeur quelconque (*). La racine, les noeuds, les feuilles sont des objets-images.

L'écran E est un ensemble discret de points :

$$E = \{(x, y)/\alpha \leq x \leq \beta \text{ et } \gamma \leq y \leq \delta\}$$

L'image est la représentation sur E de tous les éléments qui composent l'objet-image. L'image la plus élémentaire est une tache.

Une tache est la restriction dans l'espace discret de l'écran d'un ensemble connexe intérieur à un contour fermé Γ. On rappelle que

- soit (E, d) un espace métrique, et $A \subset E$, on dit que A est connexe si :

$$\forall P \text{ et } Q \text{ ouverts}, \left[\begin{array}{c} A \subset P \cup Q \\ \text{et } A \cap P \cap Q = \emptyset \end{array}\right] \Longrightarrow \left[\begin{array}{c} A \subset P \\ \text{ou} \\ A \subset Q \end{array}\right]$$

(*) Rappel

Nous appelons
- branche d'arborescence, l'ensemble des noeuds reliant une feuille à la racine ;
- profondeur, la longueur de la branche comportant le plus grand nombre de noeuds (i.e., la plus longue lignée de descendants) ;
- niveau, l'ensemble des descendants de même génération par rapport à la racine.

- Un contour est le graphe d'une application continue ; il est fermé si l'on peut délimiter une zone intérieure et une zone extérieure au contour.

- Le contour sera supposé appartenir à l'intérieur de la tache.

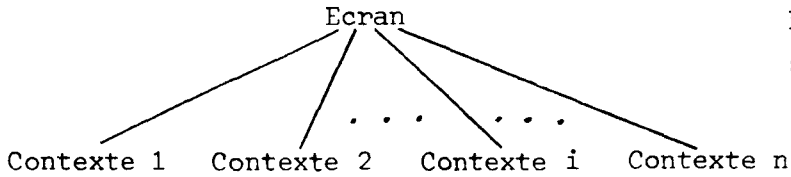
La notion de *contexte* est utilisée pour permettre de visualiser une ou plusieurs parties d'objets-images simultanément sur l'écran. Un contexte associe, à un noeud d'un objet-image (noeud contexte), une clôture qui délimite un emplacement de l'écran et qui est déterminée par la position et la taille de cet emplacement : il définit le sous-objet-image visualisé dans cette clôture, et c'est dans cette clôture que seront affichés les différents contextes successifs de l'objet-image.

Tous les noeuds d'un objet-image sont visualisables, il y a donc autant de contextes potentiels que de noeuds dans une arborescence.

Chaque contexte contient un noeud appelé *noeud courant* qui est le dernier noeud sur lequel l'utilisateur s'est positionné dans le contexte correspondant.

Le contexte multiple est composé de la liste des contextes visualisés dont l'écran est le repère absolu commun. Il permet donc de visualiser différents noeuds contextes appartenant à des arborescences quelconques.

La structure de visualisation de l'écran est la suivante :



pour n contextes visualisés simultanément.

Dans ce chapitre, nous étudions les "fonctions" de l'éditeur d'images, qui s'appliquent sur les objets visualisés. Nous allons donc définir les fonctions de visualisation puis étudier des mécanismes d'accès basés sur une identification implicite des noeuds. Enfin, nous aborderons les fonctions de création, d'ajouts et de suppressions d'objets-images et achèverons ce chapitre avec les transformations géométriques et une proposition de structure de données graphiques.

III.2. LES FONCTIONS DE VISUALISATION DE CONTEXTES

III.2.1. Modes de fonctionnement. Définition d'une clôture

La visualisation d'un objet-image ne peut se faire qu'à l'intérieur d'une clôture. Si l'objet-image 0 est créé en mode de visualisation *contexte unique*, la clôture est prédéfinie : c'est l'écran et l'objet-image 0 créé est visualisé. Si l'on veut ensuite visualiser 0 en mode *contexte multiple*, ou créer et visualiser d'autres objets-images également en "contexte multiple", il faut qu'une clôture soit définie pour chaque contexte ; un objet-image créé en contexte unique est alors visualisé en contexte multiple dès qu'une clôture lui a été attribuée. Cette clôture se définit par la fonction "DEFINITION CLOTURE" : La clôture implicitement définie est homothétique de l'écran. Elle est déterminée par un point de référence de la clôture (le centre, ou un des coins) et la longueur de la clôture (ou un rapport d'homothétie) car la largeur est calculée selon le même rapport d'homothétie que la longueur. La clôture est alors une simple réduction homothétique de l'écran, et toute image dans l'écran est réduite de la même façon ; les proportions de l'image sont toujours conservées. La clôture peut également être forcée à un rectangle non homothétique de l'écran : il y a changement d'échelle selon les axes de l'écran. Dans ce cas, l'utilisateur doit fournir la longueur et la largeur de la clôture. Un objet-image visualisé en contexte unique puis en contexte multiple dans une clôture non homothétique de l'écran subit le même rapport d'échelle que la clôture par rapport à l'écran, mais ce changement d'échelle n'affecte que la visualisation de l'objet-image, et non pas l'objet-image.

III.2.2. Les fonctions de visualisation

On dispose de cinq fonctions principales agissant sur la visualisation des noeuds ; elles ne modifient pas leur structure, mais elles permettent de modifier aisément la visualisation des objets-images.

III.2.2.1. Validation. Invalidation

* La validation est le mécanisme qui permet d'activer une fonction f lorsque f est paramétrée : la validation fixe le (les) paramètre(s) de f qui peut alors s'exécuter. La détermination des paramètres peut alors se réaliser par tatonnements de la part de l'utilisateur ; quand il est positionné sur le noeud qu'il désire, il faut qu'il active effectivement la fonction en attente du noeud paramètre : La fonction "VALIDATION" fournit le noeud sur lequel l'utilisateur est positionné en paramètre d'où le déclenchement de l'exécution de la fonction.

* La notion d'invalidation est également très utile dans le cadre interactif de l'éditeur d'images. En effet, toute fonction demandée modifie l'état du système. Il faut donc préserver l'utilisateur des erreurs de manipulations et lui permettre de revenir à l'état dans lequel il était avant l'appel de la fonction qu'il désire annuler. En particulier, les fonctions paramétrées par un (ou des) noeud(s) à identifier modifient l'état du système avant même que la fonction ne soit activée (car l'identification d'un noeud change l'état du système). Pour ces fonctions qui ne peuvent s'exécuter que si elles sont validées, la primitive "INVALIDATION" permet de retourner à l'état précédant l'appel de la fonction sans que les identifications n'aient détruit les informations antérieures.

Dans la suite, quand nous présentons une fonction paramétrée qui doit être validée, nous signalons le fait qu'elle nécessite une validation pour être effective, et nous étudions implicitement ses actions dans ce cas. L'INVALIDATION annulant toute fonction, nous ne considérons plus ce cas.

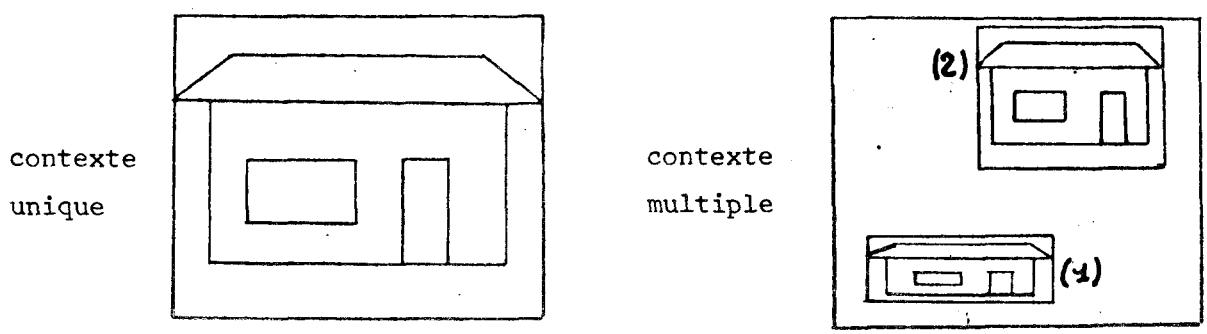
III.2.2.2. Les fonctions de visualisation proprement dites

1) "DEFINITION CONTEXTE" : quand il a créé une clôture, l'utilisateur lui associe un contexte N par la primitive "DEFINITION CONTEXTE" (N) ou N est le noeud choisi comme contexte, qui pouvait être visualisé dans une autre clôture (éventuellement). La fonction est paramétrée par l'identification d'un noeud ; elle doit donc être validée. Nous n'étudions pas ici le moyen d'identifier N, ceci se fait à l'aide des primitives d'accès décrites en III.3 ou par nommage si N n'est pas visualisé. Cette fonction se fait indifféremment en contexte unique ou en contexte multiple.

2) "CONTEXTE UNIQUE" : Cette primitive passe du mode de fonctionnement contexte multiple au mode contexte unique. Le contexte de travail, i.e., celui dans lequel l'utilisateur est positionné à l'appel de la fonction est visualisé seul dans la clôture de l'écran ; la position et la taille de l'image relatives à la clôture sont constantes.

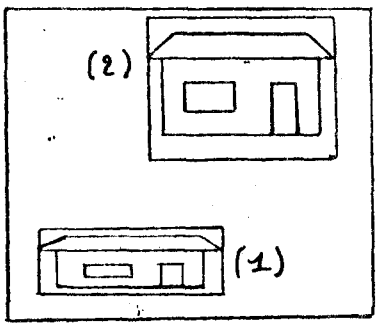
3) "CONTEXTE MULTIPLE" : Cette primitive entraîne la visualisation de l'ensemble des noeuds contextes dans les clôtures définies par l'utilisateur. Le noeud contexte sur lequel l'utilisateur travaillait est le contexte de travail en contexte multiple. C'est la primitive symétrique de "CONTEXTE UNIQUE".

La figure III.1 est un exemple de changement de mode pour des objets créés dans des contextes différents

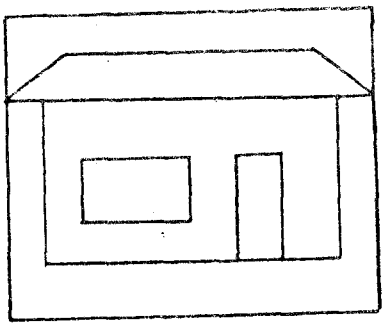


a) Définition d'une maison en contexte unique, puis visualisation en contexte multiple dans une clôture non homothétique de l'écran (1) et dans une clôture homothétique de l'écran (2).

contexte multiple



contexte unique



b) Définition de la maison en contexte multiple puis visualisation en contexte unique (1).

Figure III.1. : Exemple de visualisation dans les contextes uniques et multiples.

4) "CHANGEMENT CONTEXTE" : Cette primitive n'est utilisé qu'en contexte multiple : l'utilisateur change de clôture de travail, les noeud courants des contextes correspondants sont inchangés.

5) "REDUCTION CONTEXTE" : Cette primitive permet de réduire le noeud contexte au noeud courant. Cette fonction ne modifie que le contexte sur lequel on travaille, que l'on soit en contexte multiple ou en contexte unique.

III.3. LES PRIMITIVES D'ACCES

III.3.1. La désignation

Nous appelons désignation l'identification "visuelle" d'une feuille F, i.e., l'identification de F à partir de son image sur l'écran. Que ce soit en contexte unique ou en contexte multiple, une feuille d'arborescence peut

être en partie et parfois même complètement cachée par d'autres images.

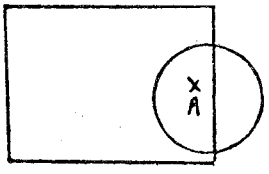
Si la feuille (ou un noeud quelconque) est cachée (caché) par une autre image, une méthode pour la (le) rendre visible consiste à affecter l'attribut de transparence aux objets-images qui la (le) cachent, car on ne peut identifier visuellement un objet non visible !

Une feuille F peut être partiellement cachée par des images provenant de noeuds visualisés dans des clôtures différentes de celle de F si l'on travaille en contexte multiple (cas de clôtures non disjointes). En contexte unique, une feuille peut aussi être partiellement cachée par d'autres images composant le noeud contexte.

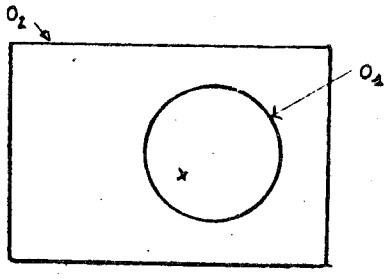
Dans tous les cas, la désignation sur l'écran doit être sûre, quel que soit le chevauchement des images :

- En cas d'ambiguïté, un message de demande de désignation non ambiguë est envoyée et il y a attente d'une nouvelle désignation ; la première est alors annulée.

Exemple de situation ambiguë : deux taches de même couleur dans un même plan, et la désignation est faite à partir des coordonnées d'un point commun aux deux taches (A)



- En cas d'ambiguïté totale (images incluses), l'objet désigné est celui qui est inclus dans l'autre ce qui correspond à l'exemple suivant.



O_1 et O_2 sont de même couleur et appartiennent au même plan.

La désignation ponctuelle (visualisée sur le dessin par +) fournit l'objet O_1, car il est inclus dans O_2.

Ces mécanismes permettent bien de résoudre les problèmes de désignation évoqués au chapitre II. Nous pouvons donc étudier maintenant les primitives d'accès aux objets-images.

III.3.2. Les primitives d'accès aux voisins immédiats. Notion de chemin

III.3.2.1. Introduction

Que la méthodologie de manipulations des objets-images soit "descendante" ou " ascendante", les accès nécessaires à l'utilisateur se résument en quelques primitives de base complétées éventuellement de quelques fonctions dites "d'édition".

Les primitives de base sont :

- Accès à l'ascendant du noeud courant (i.e., sur lequel on est positionné) ; cette primitive s'appelle *"haut"* ou "↑".

- Accès à un descendant du noeud courant ; cette primitive s'appelle *"bas"* ou "↓".

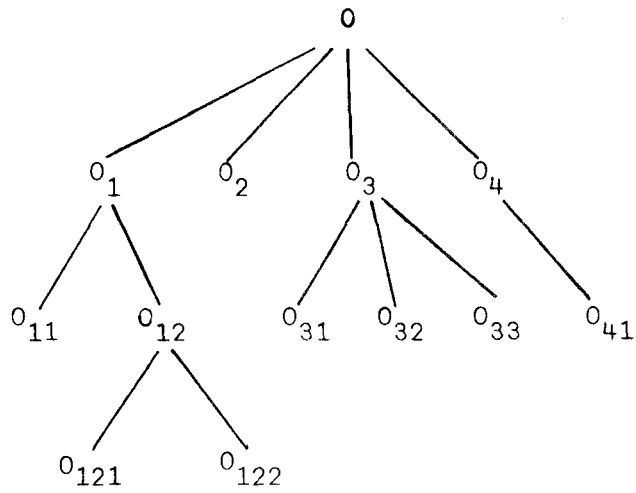
- Accès à un frère du noeud courant :

  * au frère droit ; cette primitive s'appelle *"droit"* ou "→" ;
  * au frère gauche ; cette primitive s'appelle *"gauche"* ou "←".

- Identification d'une feuille par désignation d'un point ; cette primitive s'appelle *"désignation"* ou "⊕".

III.3.2.2. Définition précise de quelques primitives de base

Regardons l'action des primitives de base sur un objet-image O quelconque. Prenons l'exemple de structure de la figure III.2 pour O.



* "⊕" : on peut désigner n'importe quelle feuille (0_{11}, 0_{121}, 0_{122}, 0_2, 0_{31}, 0_{32}, 0_{33}, 0_{41}).

* "↑" sur 0_{32} fournit son unique ascendant : 0_3 : On peut ainsi accéder à la racine d'une arborescence à partir de la désignation d'une feuille.

* Si 0_3 est visualisé, "→" sur 0_{32} fournit le premier frère droit de 0_{32} : 0_{33}.

* Si 0_3 est visualisé, "←" sur 0_{32} fournit le premier frère gauche de 0_{32} : 0_{31}.

* "↓" sur 0_3 est ambiguë ! En effet, il faut choisir un descendant parmi les trois potentiels (0_{31}, 0_{32} ou 0_{33}).

Nous définissons ultérieurement (en III.3.2.3.) la notion de chemin qui supprime toute ambiguïté à la fonction "↓". Nous étudions maintenant le comportement des primitives de base dans des cas particuliers.

Le noeud que doit fournir une primitive de base n'existe pas toujours ; dans chacun des cas de "non-existence", une option par défaut détermine un noeud particulier ;

("↑") appliqué sur une racine fournit celle-ci en résultat
("↓") appliqué sur une feuille fournit celle-ci en résultat
} idempotence

Sur l'exemple "↑" sur 0 fournit 0
"↓" sur 0_{32} fournit 0_{32} (même principe pour chaque feuille).

("↳") (respectivement ("↰")) appliqué sur le dernier descendant droit (respectivement gauche) fournit le premier descendant gauche (respectivement droit) dépendant du même père : les descendants d'un noeud sont configurés en liste circulaire, ce qui permet de revenir aisément au premier descendant (i.e., en tête de liste). Notons que "↳" et "↰" sur le noeud contexte lui-même n'ont pas d'effet, car les frères ne sont pas visualisés.

Sur l'exemple, "↳" sur 0_{33} fournit 0_{31}
"↰" sur 0_{31} fournit 0_{33}
} sur des feuilles

"↳" sur 0_{12} fournit 0_{11}
"↰" sur 0_{11} fournit 0_{12}
} sur des noeuds quelconques

"↳" sur 0_{41} fournit 0_{41}
"↰" sur 0_{41} fournit 0_{41}
} cas très particulier où il n'y a pas de frère ; la fonction est l'identité.

Dans chacun de ces cas, un artifice graphique ou un message signalera que la fonction est appliquée dans l'une de ces situations.

Ces définitions précises de "↳", "↰" ont pour conséquences que ces fonctions s'annulent mutuellement, ce que nous pouvons écrire sous forme de règles :

$$(\rightarrow, \leftarrow) \vdash \Lambda$$

$$(\leftarrow, \rightarrow) \vdash \Lambda$$

Il est évident que de plus, quelque soit le choix de N_i effectué pour la fonction "↓" sur le noeud N, l'action ultérieure de "↑" refournira N. Nous avons donc également : $(\downarrow, \uparrow) \vdash \Lambda$.

Ces règles fournissent ce que nous pourrions appeler un certain "droit à l'erreur" de l'utilisateur, qui nous paraît important. La notion de chemin et de mode de parcours va nous permettre de définir "↓" comme annulation de "↑" (parcours "courant") ou de définir une fonction "RETOUR" pour cette annulation en laissant le libre choix de la définition de "↓".

III.3.2.3. Le mode de parcours "courant"

Dans ce mode de parcours, nous désirons obtenir la règle :

$$(\uparrow, \downarrow) \vdash \Lambda$$

Pour cela, il faut mémoriser les noeuds par lesquels on est passé en appliquant la primitive "↑". Ceci détermine une branche d'arborescence qui part de la racine, mais n'est pas obligatoirement définie jusqu'à une feuille : c'est ce que nous appellerons le chemin courant. Notons que le noeud courant (celui sur lequel on est positionné) peut ne pas appartenir au chemin courant (accès par les fonctions "↳" et "↰").

La fonction "↓" est alors définie comme suit :

"↓" sur une feuille fournit cette feuille ; (cf. III.3.2.2.).

"↑" sur un noeud N qui n'est pas une feuille fournit le descendant de N appartenant au chemin courant s'il existe, ou sinon le premier descendant de N.

De plus, les actions des primitives de base sur le chemin courant C sont définies comme suit :

"↳" et "↰" ne modifient pas le chemin courant ;

"↑" sur N ne modifie pas C si N ∈ C, sinon redéfinit C comme le chemin qui relie N à la racine ;

"↓" sur N et fournissant D, ne modifie pas C si D ∈ C, sinon redéfinit C comme le chemin qui relie D à la racine.

Notons que "↑" et "↓" peuvent faire perdre le chemin courant précédent. Il n'y a donc pas de "droit à l'erreur" total.

III.3.2.4. Le mode de parcours "trace"

Un des inconvénients du mode de parcours courant est le fait que la fonction "↓" ne fournit pas toujours le même noeud selon que le chemin courant "passe ou ne passe pas" par l'un des descendants du noeud courant ; de plus, l'utilisateur ne conserve qu'une trace très réduite des accès qu'il a effectué. Il peut désirer conserver la trace des noeuds auxquels il a accédé à partir de la désignation d'une feuille pour pouvoir y retourner en "pas à pas".

La trace complète, qui inclut tous les cycles réalisés par les accès est très redondante, et de ce fait, nous paraît peu commode pour l'utilisateur. Nous définissons la trace réduite ou *trace* comme l'ensemble des noeuds obtenus par les fonctions d'accès à partir d'une désignation, dont on a enlevé les cycles par annulation des fonctions dont l'action résultante est vide (cf. III.3.2.2.).

Le mode trace est essentiellement justifié par la possibilité d'un Retour, i.e., des accès aux noeuds en "marche arrière" pas à pas.

La fonction RETOUR est définie par son action sur le noeud courant dans différents cas suivants :

$(\rightarrow, \text{RETOUR}) \vdash\!\!- \Lambda$
$(\leftarrow, \text{RETOUR}) \vdash\!\!- \Lambda$
$(\uparrow, \text{RETOUR}) \vdash\!\!- \Lambda$
$(\downarrow, \text{RETOUR}) \vdash\!\!- \Lambda$

La fonction RETOUR permet d'annuler n'importe quelle primitive de base. En particulier, elle annule la fonction "$\uparrow$". De ce fait, le choix d'une définition de la fonction "$\downarrow$" est libre, car dégagé de la contrainte d'annulation de "$\uparrow$". Afin d'obtenir une définition simple, non ambiguë et commune à l'ensemble des noeuds d'arborescence, la fonction "$\downarrow$" est déterminée de la façon suivante :

"$\downarrow$" sur une feuille fournit cette feuille (cf. III.3.2.2.).

"$\downarrow$" sur un noeud N qui n'est pas une feuille fournit le premier descendant de N.

Remarques :

(1) tout comme dans le mode de parcours courant, "$\downarrow$" sur une feuille est l'identité.

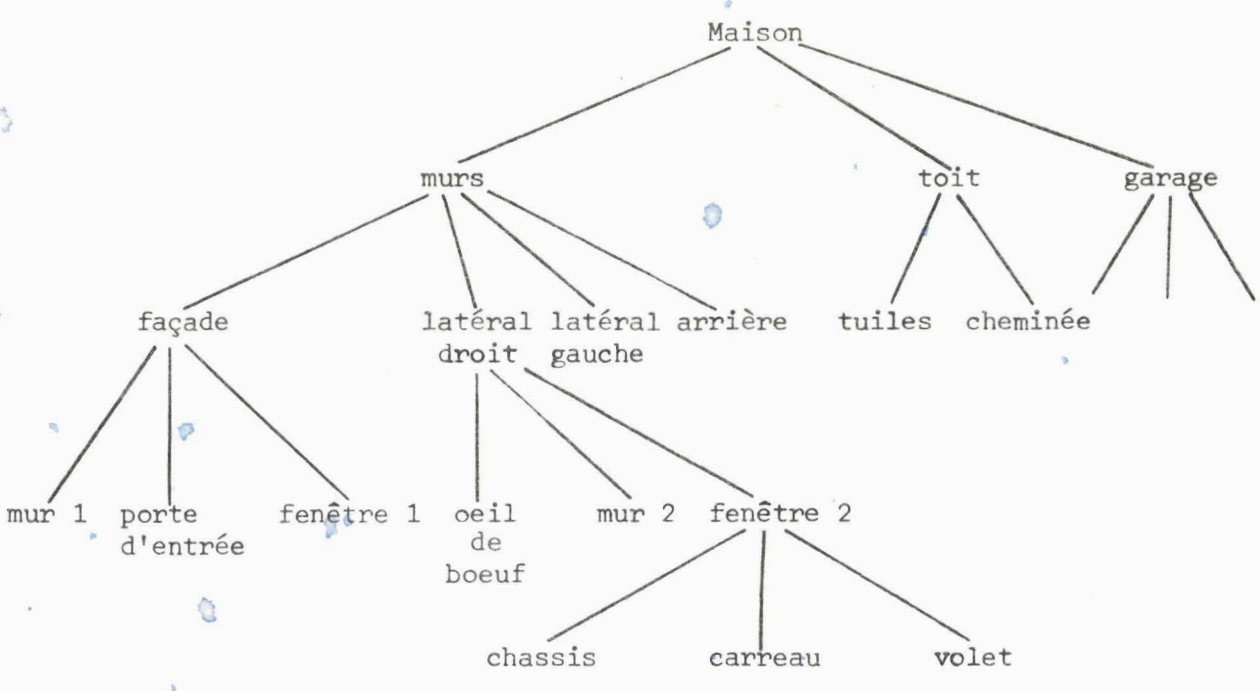
(2) La fonction "RETOUR" n'est définie que dans le mode trace ; tout comme la fonction "$\downarrow$" en mode courant, la fonction "RETOUR" ne restreint jamais le contexte.

III.3.2.5. Choix

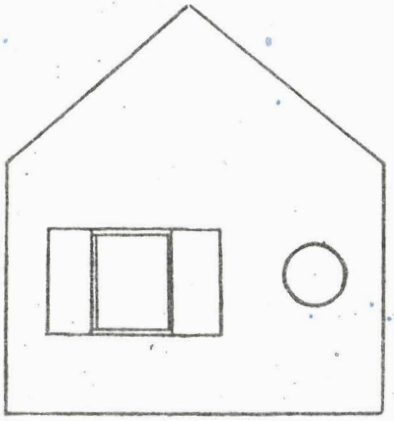
L'option de mode de parcours par défaut est le mode courant. On passe du mode courant au mode trace par la fonction "MODE TRACE" ; On retourne au mode courant par la fonction "MODE COURANT". Chacune de ces fonctions provoque la perte des accès aux noeuds obtenus antérieurement aux noeuds courants des différents contextes (ceux de la trace ou ceux du chemin courant mémorisés). Ces noeuds courants ainsi que les noeuds contextes correspondants ne sont pas modifiés.

III.3.2.6. Exemple

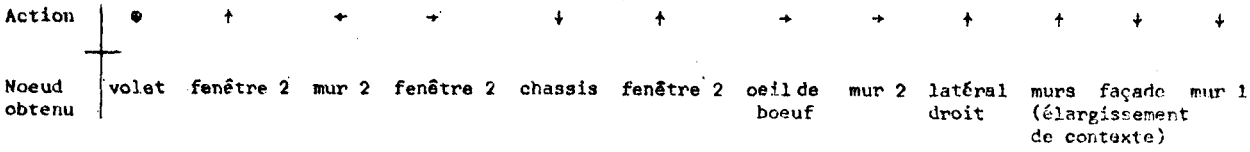
Illustrons les modes de parcours sur l'exemple de structure de maison de la figure II.1.



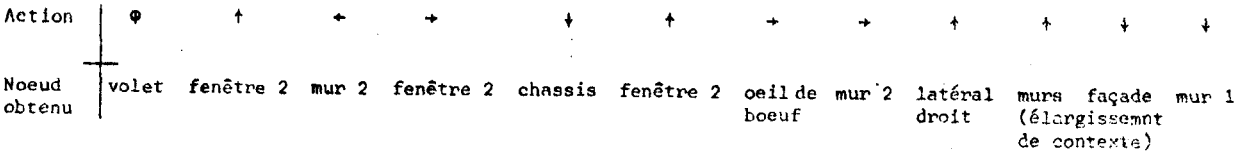
Le mur latéral droit est le noeud contexte, le volet est désigné.

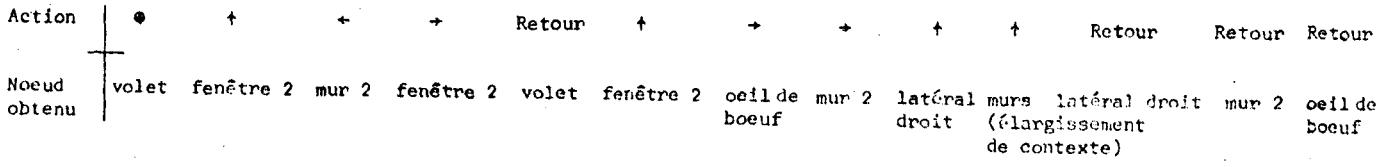


Etudions la suite des accès obtenus en mode courant



Etudions les suites d'accès en mode trace





III.3.3. Primitives d'édition. Extensions du système

Quand l'utilisateur a oublié la structure exacte de ses objets-images, il faut qu'il puisse identifier simplement les noeuds d'arborescence ; s'il désire retrouver un noeud ascendant d'une feuille, il faut qu'il ait un moyen commode de le retrouver (sans avoir à faire des actions symétriques de celles qu'il a effectuées depuis la détermination de la feuille. (cf. Mode courant). Nous proposons des demandes d'accès plus élaborées que les primitives de base. Nous allons en définir une qui est "associative" selon le critère de la structure des objets-images :

Déterminer le premier noeud qui compte parmi ses descendants :

- le noeud courant,

- une feuille que l'on désigne.

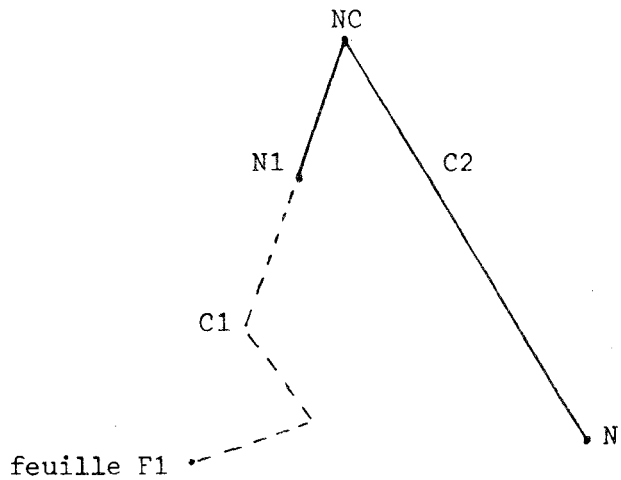
Cette fonction s'appelle : "1er NOEUD COMMUN" (N) où N est un noeud identifié par l'utilisateur. En terme de chemin, elle se définit de la façon suivante :

Soit le noeud courant appartenant au chemin C1.

Soit un chemin C2 (déterminé à partir de la désignation d'une feuille).

Chercher le premier noeud commun à C1 et à C2 ; i.e., quelle est la jonction des deux chemins ?

Remarque : Le noeud résultat peut être le noeud courant lui-même.



Si les chemins appartiennent à des arborescences différentes, il y a visualisation des deux objets séparément (par exemple, clignotement inversé entre les objets...).

Tout comme la fonction "DEFINITION CONTEXTE" en III.2, la fonction "1er NOEUD COMMUN" nécessite qu'un noeud paramètre soit identifié. Cette fonction doit donc être validée pour être effective. Si elle est invalidée, la fonction "1er NOEUD COMMUN" est annulée.

Remarque :

Cette fonction d'édition compense la perte de la "trace" des actions effectuées depuis la désignation d'un noeud ; En effet, par désignations et appels successifs de cette fonction, l'utilisateur peut retrouver au moins en partie, les noeuds auxquels il a accédé.

Extension du système

D'autres fonctions d'accès associatifs peuvent être définies ; Les critères des accès sont essentiellement liés à la visualisation : Déterminer tous les objets dont les éléments de base sont des octogones, toutes les maisons bleues... La mise en oeuvre des fonctions d'édition doit être extensible.

III.3.4. Action des primitives d'accès sur les contextes

Il n'y a de modification automatique de contexte que dans le sens d'un élargissement de contexte si cela est nécessaire.

* La primitive ⊕ ne modifie jamais le contexte, car c'est la désignation d'une feuille visualisée, qui est donc incluse dans un contexte.

* Les fonctions "↳" et "↰" ne peuvent s'appliquer que sur des noeuds dont le père est visualisé ; elles ne modifient pas le contexte.

* La fonction "↓" ne réduit pas le contexte.

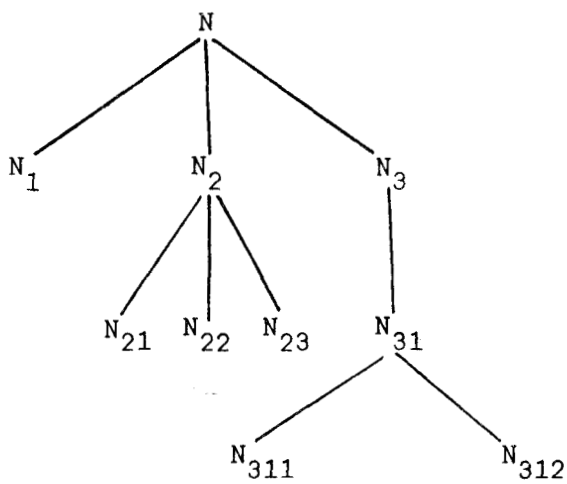
* La fonction "↑" élargit le contexte au noeud auquel on accède s'il n'était pas visualisé. Sinon, il n'y a pas de modification de contexte.

* La fonction "1er NOEUD COMMUN" ne modifie pas le contexte car le noeud courant et le noeud désigné sont visualisés dans un même contexte : leur premier ascendant commun est donc également visualisé dans ce contexte.

* Les fonctions de changement de mode de parcours, de validation et d'invalidation ne modifient pas les contextes.

Illustrons ces propriétés à l'aide d'un exemple.

Soit l'objet N ayant la structure ci-dessous :



* ⊕ N_{22}. On ne peut pas désigner visuellement un objet qui n'est pas représenté ; le contexte est . soit N_{22},
. soit un ascendant quelconque de N_{22}.

Il n'y a pas de modification de contexte.

* Pour rester dans un cas général, soit N_2 le noeud contexte, et le chemin composé de N_{22}, N_2, N.

* "↓" sur N_2 fournit N_{22}, visualisé dans le noeud N_2 : la fonction "↓" ne réduit jamais le contexte.

* "↑" sur N_2 fournit N, et il y a visualisation de tout le noeud N : la fonction "↑" élargit le contexte car elle s'applique sur un noeud contexte.

* "↑" sur N_{22} fournit N_2 ; il n'y a pas d'élargissement de contexte car le noeud obtenu est déjà inclus dans le contexte.

* "→" ("←") sur N_2 est impossible car N_3 (resp. N_1) n'est pas visualisé.

* "→" ("←") sur N_{22} fournit N_{23} (N_{21}) visualisé dans le noeud N_{21} : le contexte est ascendant du noeud fourni, il reste donc inchangé.

* 1^{er} NOEUD COMMUN (N_2, descendant de N_2) fournit N_2 : il n'y a pas de modification de contexte.

III.4. LES CONSEQUENCES ET LES CONTRAINTES D'UTILISATION. LA FENETRE D'UN OBJET-IMAGE

III.4.1. La normalisation des données

Il n'y a pas de distinction fondamentale entre un objet-image et ses sous-objets-images : l'utilisateur doit pouvoir accéder à un objet (et ses attributs) de la même façon qu'à un sous-objet (et les attributs propres à ce sous-objet) ; les fonctions d'accès définies en III.3 s'appliquent sur tous les objets-images, quel que soit leur niveau dans les

arborescences ; les seuls objets un peu particuliers sont les feuilles d'arborescence que l'on peut désigner (f et ●) et visualiser directement : une feuille fournit l'image tache. Tous les noeuds qui ne sont pas des feuilles sont visualisables par la visualisation de l'ensemble de leurs descendants ; la notion de contexte implique que tout noeud soit visualisable par rapport à l'un quelconque de ses ascendants.

Les caractères généraux des primitives d'accès et de leurs effets sur les images conduisent à une normalisation des objets-images : tout objet-image est inclus dans une "fenêtre".

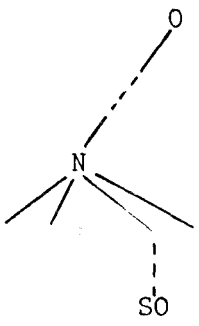
III.4.2. Première définition de la fenêtre d'un objet-image

La fenêtre d'un objet-image est un référentiel dans lequel est repéré un objet-image ; de prime abord, c'est un système d'axes orthonormés selon les axes de l'écran ; il est implicite pour l'objet. Tout objet est défini de façon unique dans une fenêtre qui lui est propre, et il y a autant de fenêtres que de noeuds dans une arborescence.

On obtient ainsi un système de repérage des objets-images qui consiste à

- repérer un objet-image dans sa fenêtre,
- repérer la fenêtre d'un noeud par rapport à la fenêtre de son père.

Soit par exemple le schéma suivant :



SO est repéré dans sa fenêtre F_{SO},

N est repéré dans sa fenêtre F_N,

O est repéré dans sa fenêtre F_O.

Repérer SO par rapport à N, c'est repérer F_{SO} par rapport à F_N (indépendamment de F_O).

F_N étant repéré par rapport à F_O, on peut ainsi repérer SO par rapport à O.

Les objets-images sont repérés de façon intrinsèque dans leur fenêtre, donc par des coordonnées relatives à leur fenêtre ; cette dernière est aussi repérée de façon relative par rapport à la fenêtre du noeud ascendant.

III.4.3. Propriétés

- Les primitives d'accès ne modifient pas les fenêtres des objets-images.

- Tout noeud est inclus dans une fenêtre, donc tous les descendants de ce noeud sont visualisables dans cette fenêtre. En particulier, lorsqu'un noeud est un noeud contexte, sa fenêtre est "cadrée" dans la clôture du contexte et ses noeuds descendants sont visualisés dans cette clôture en tenant compte de la position de leurs fenêtre respective par rapport à la fenêtre du noeud contexte.

- Modifier la position d'un noeud par rapport à un de ses ascendants consiste à modifier la position de la fenêtre de ce noeud par rapport à la fenêtre du noeud ascendant.

- Les changements de contexte ne provoquent que des modifications de la visualisation dues au changement de repère d'une fenêtre à une autre ; les fenêtres ne sont pas modifiées les unes par rapport aux autres.

- On obtient un système de repérages relatifs :

  - des objets O_i dans leur fenêtre F_i,
  - des fenêtres entre-elles : O_i descendant de O (de fenêtre F).

Repérer O_i dans F nécessite la conversion :

coordonnées (O_i) dans F_i → coord (O_i) dans F.

- Tout objet-image est inclus dans une fenêtre ; les fenêtres sont donc emboîtées : la fenêtre d'un noeud contient les fenêtres des objets qui le composent : La notion de fenêtre admet une composante Taille en plus de celle de référentiel. Nous en parlerons en détail lors des fonctions de création, d'ajout et de suppression, et des transformations géométriques (III.5).

- Le problème de la visualisation d'un objet-image dans une clôture est résolu aisément : il suffit de prendre comme dimension de la fenêtre, la plus grande taille telle qu'elle est incluse dans la clôture (en conservant ses proportions).

Conclusion sur le mécanisme de fenêtre

Le mécanisme de fenêtre permet d'accéder facilement à un niveau quelconque d'une arborescence, et d'isoler les niveaux hiérarchiques. La fenêtre assure un découpage logique des objets-images de l'utilisateur, et des images correspondantes.

III.4.4. Les contraintes d'utilisation

* Le système graphique composé des objets-images définis précédemment, présente une caractéristique qui alourdit considérablement son emploi. Dans une arborescence donnée, un objet ne peut être utilisé plusieurs fois en utilisant le même nom. Les conséquences sont multiples :

(1) Il n'y a pas de répétition simple possible d'un même noeud dans une arborescence du point de vue utilisateur, il faut définir autant de dalles que nécessaires pour obtenir un carrelage.

(2) Obligation de définir complètement un objet (éventuellement complexe) même si d'autres objets déjà définis sont très "ressemblants". Du point de vue utilisateur, il n'y a pas moyen de calquer des objets-images sur des modèles.

(3) Impossibilité de définir des objets récursifs.

* Les fonctions d'assistance à la synthèse d'images sont limitées. Nous définissons au chapitre IV des objets un peu particuliers, tels que leur utilisation simplifie la tâche de l'usager. Ils permettront de résoudre (au moins en partie) les problèmes évoqués ci-dessus.

Le chapitre V contient la généralisation des objets utilisés dans le système. Celui-ci en sera à la fois plus puissant et plus souple d'emploi, ce qui lui donne un caractère plus attrayant pour l'utilisateur.

III.5. MISE A JOUR DES OBJETS-IMAGES

Introduction

Grâce aux fonctions d'accès (cf. III.3.), l'utilisateur peut se positionner sur n'importe quel noeud N d'un objet-image, visualisé dans un contexte qui le contient. Ce noeud N est le noeud courant, sur lequel l'utilisateur peut effectuer différentes modifications. Elles comportent plusieurs fonctions de base pour l'éditeur graphique :

* la création d'objets, sans laquelle on ne peut définir ni visualiser d'objet-image ;

* les modifications de structure des objets-images, qui comprennent les ajouts et les suppressions de noeuds, mais également des modifications locales d'organisation de structure ;

* les modifications du noeud, qui permettent de changer l'image de ce noeud ; ce sont des modifications géométriques ou graphiques.

Toutes ces fonctions doivent être interactives et simples ; certaines d'entre-elles (création, modification de structure...) sont l'adaptation graphique des fonctions d'édition de textes, mais les fonctions sont plus variées, car les objets sont plus complexes que les textes.

Nous étudierons successivement ces différentes fonctions et ne nous intéresserons uniquement qu'à des noeuds visualisés en contexte unique ou multiple. Nous préciserons si nécessaire le mode de fonctionnement demandé pour la réalisation des fonctions. Le sous-chapitre se décompose donc en :

III.5.1. Création d'une feuille
III.5.2. Modification de structure
III.5.3. Modification de noeud

Nous ne considérons pas la création des noeuds qui ne sont pas des feuilles, car nous ne nous préoccupons ici que des objets visualisables (et visualisés). Un noeud n'étant visualisable que par les feuilles qui le composent, un noeud créé autre qu'une feuille n'est pas visualisable ; on ne peut donc y accéder par les primitives décrites en III.3, tant qu'ils ne sont pas insérés dans la structure d'un objet-image. Nous supposerons par la suite que l'insertion d'un noeud dans la structure d'un objet-image comporte automatiquement la création de ce noeud ; la fonction d'insertion permet alors de compléter sa définition (cf. III.5.2.). Le seul mécanisme de création qui nous intéresse est donc celui de création d'une feuille d'arborescence.

III.5.1. Création d'une feuille

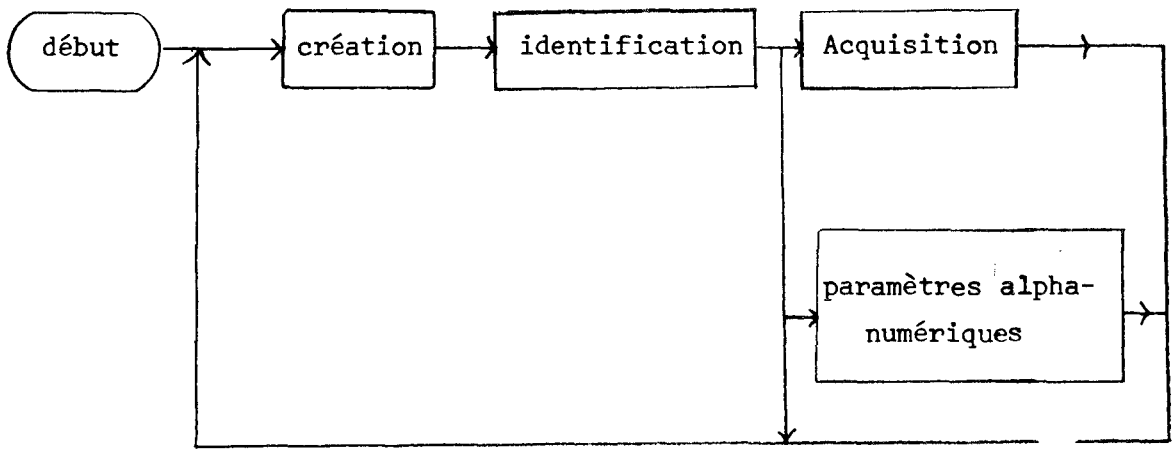
C'est la fonction qui consiste à créer un descripteur de structure et à déterminer les informations nécessaires à la visualisation de cette feuille. Du point de vue de l'utilisateur, la création se fait très sim-

plement : dans une clôture de l'écran, il active la création d'une tache de couleur par l'appel d'une fonction indiquant sa forme ; par exemple en 2D : DISQUE, RECTANGLE, ou en 3D : SPHERE, PARALLELEPIPEDE... Les formes les plus courantes en 2D sont les taches déterminées par un cercle, un carré, une ligne polygonale fermée ou une ligne courbe, dont les points caractéristiques sont repérés dans la clôture ; ensuite, l'utilisateur doit attribuer une couleur "brute", codée en fonction de la décomposition Rouge, Vert, Bleu par exemple. Après avoir déterminé la forme, l'emplacement et la couleur de la tache, la feuille créée est visualisée à l'emplacement souhaité dans la clôture. L'utilisateur peut alors, s'il le désire, lui donner un nom, compléter sa description en lui attribuant des caractéristiques graphiques (transparence, brillance, texture...) qui la rendront plus réaliste, mais il ne s'agit plus à proprement parler de création, mais de modification de la feuille ; nous étudierons ces différentes modifications en III.5.3.

La création se fait indépendamment du mode de fonctionnement contexte unique ou contexte multiple ; en effet, le passage d'un mode de contexte à un autre ne modifie pas les proportions d'une image par rapport à sa clôture dans laquelle elle est visualisée, d'où l'intérêt de choisir une clôture homothétique de l'écran en contexte multiple, la clôture en contexte unique étant toujours l'écran (cf. III.1.). Par contre, la visualisation de l'objet créé dépend de ce mode et demande, en contexte multiple, que la définition de la clôture précède la création de l'objet-image, car celle-ci provoque implicitement la visualisation de l'objet-image créé dans cette clôture.

A la création de l'objet-image, la fenêtre implicite est la clôture (éventuellement l'espace de l'écran) : elle est automatiquement définie de cette façon, tant que l'objet n'est pas géométriquement complètement défini. Dès qu'il l'est, la fenêtre est "réajustée". Grâce à ce mécanisme, l'objet est défini de façon absolue dans l'écran ; ainsi, l'écran est le repère absolu qui permet, en contexte multiple, de créer, d'ajouter et de supprimer des objets-images dont on peut connaître les tailles relatives.

Les actions attendues à la création d'une feuille peuvent se schématiser de la façon suivante :

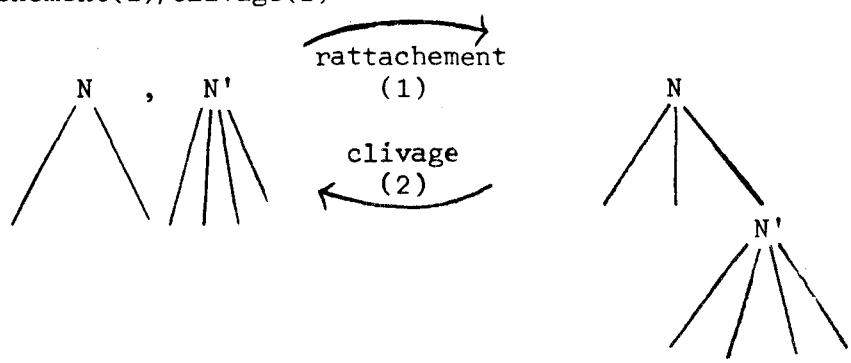


Nous allons maintenant étudier les fonctions de modifications de structure des objets-images ; elles sont à la base de la création des objets structurés et des modifications de ces arborescences.

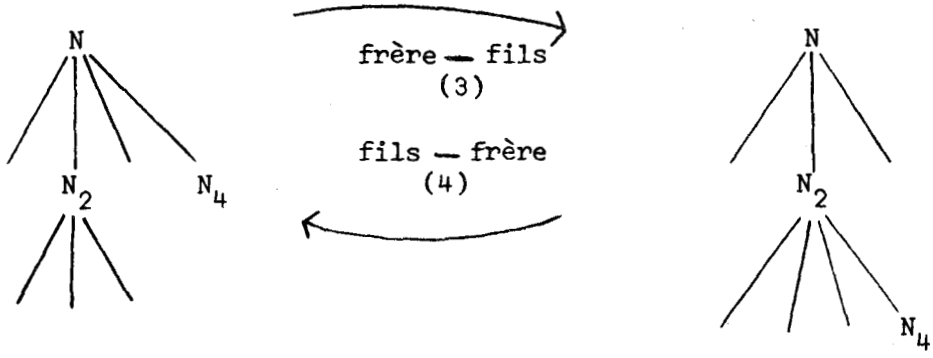
III.5.2. Modifications structurelles

On peut considérer trois types principaux de modifications : celles qui visent à ajouter des éléments dans une arborescence, celles qui visent à ôter des éléments, et celles qui ne modifient pas le nombre de noeuds mais effectuent des déplacements locaux dans les structures. Les primitives de base sont telles que toute fonction a sa fonction inverse qui l'annule ; par exemple, l'insertion d'un noeud est annulée par la suppression de ce noeud... Les primitives de base peuvent s'écrire selon les règles suivantes :

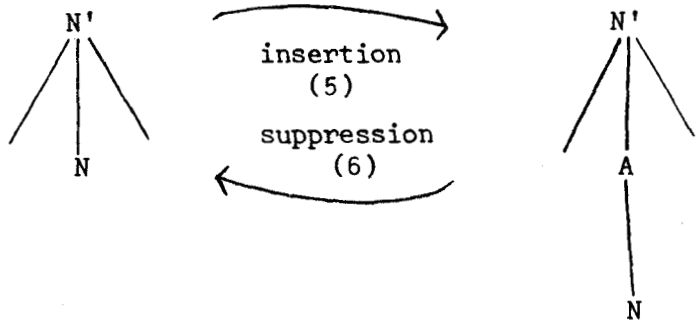
Rattachement(1)/clivage(2)



Déplacement local (3) et (4)



Insertion(5)/suppression(6)



Nous pouvons remarquer que le déplacement local n'est pas absolument nécessaire, car on peut le réaliser par clivage suivi de rattachement ; nous l'étudierons néanmoins, car il simplifie les calculs, ne nécessite pas de placement, et est plus simple à utiliser que la séquence clivage-rattachement.

Une dernière fonction à considérer est la fonction de suppression globale d'une sous-arborescence, bien qu'elle puisse être équivalente aux suppressions des noeuds et des feuilles ; la suppression d'une feuille annule alors sa création.

Les primitives ainsi définies correspondent bien à la méthodologie de construction et manipulation d'images : la construction d'objets est basée sur les notions complémentaires de regroupements et de décompositions successives : le rattachement consiste bien à introduire un

descendant dans la description d'un noeud,et l'insertion à créer un ascendant (intermédiaire ou non) à un noeud ; l'enchaînement d'une insertion (règle (5)) et de déplacements locaux (règle (3)) engendre le regroupement de noeuds en un niveau supplémentaire dans la structure.

D'autres enchaînements des fonctions sont intéressants à étudier, et influent sur le choix des noeuds paramètres, du noeud courant. Les enchaînements à simplifier impérativement sont les annulations des fonctions, c'est-à-dire les successions suivantes (données par leur numéro de règle) :

(1) suivi de (2), (2) suivi de (1)

(3) suivi de (4), (4) suivi de (3)

(5) suivi de (6), (6) suivi de (5).

Les autres enchaînements remarquables sont :

- le regroupement de frères par combinaison de (5) puis de $(3)^*$,

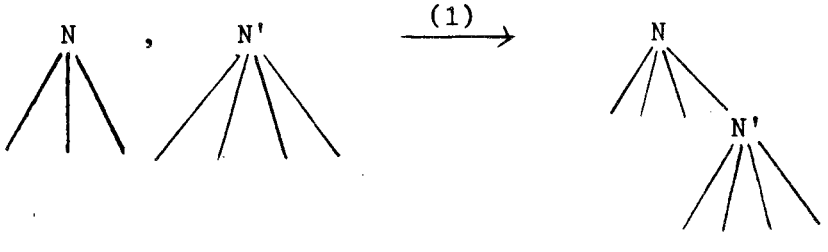
- le regroupement de racines par combinaison de (5) puis de $(1)^*$,

Lorsque les fonction nécessitent plusieurs paramètres, le choix du noeud courant vise toujours à simplifier ces enchaînements. Nous allons maintenant étudier en détail ces différentes fonctions, leur propriétés et les problèmes qu'elles posent.

III.5.2.1. Rattachement/clivage

a) *Rattachement*

Il correspond à la règle (1) suivante :



L'objet-image rattaché peut être soit une feuille, soit une racine d'arborescence, et il est visualisé dans une clôture. C'est une fonction à deux paramètres : soit à ajouter N' à la descendance de N, N est le premier paramètre : c'est le noeud courant et N' est le second paramètre à identifier. La fonction de rattachement nécessite donc une validation (cf. III.2.).

Pratiquement, rattacher un descendant N' à un noeud N comporte deux étapes : le placement de N' par rapport à N et le rattachement proprement dit. Ce placement peut précéder ou suivre la demande de rattachement, mais ce dernier n'est effectif que si le placement est effectué, i.e., si la taille et la position de N' est déterminée par rapport à l'ascendant N.

Le noeud courant après l'opération est le noeud père du noeud rattaché, ce qui permettra un enchaînement aisé des fonctions (en particulier, le rattachement suivi de clivage ou vice-versa).

Conséquences :

* Cette fonction ne permet pas de créer de racine d'arborescence, ni même de niveau intermédiaire à des niveaux existants.

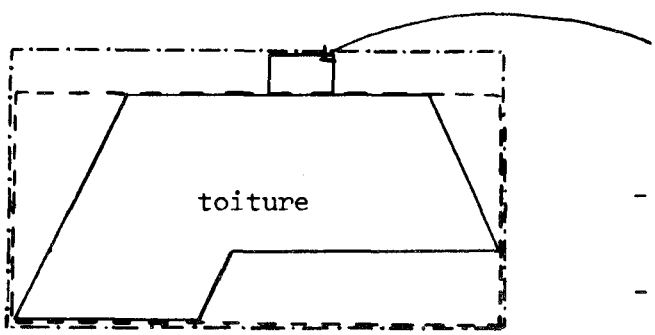
* On ne peut rattacher de cette façon qu'une racine d'arborescence, visualisée dans une clôture différente de la clôture de son futur ascendant. Le mode de fonctionnement est obligatoirement la visualisation en contexte multiple.

Action du rattachement sur l'environnement

* Le noeud où se fait le rattachement est le noeud courant, il est donc visualisé, et il n'y a pas de modification du noeud contexte.

* Les fenêtres des ascendants, peuvent quant à elles être notablement modifiées par la fonction. En effet, les fenêtres doivent rester emboîtées (la fenêtre d'un ascendant contient la fenêtre de chaque descendant). Il se peut que le descendant qu'on rattache ne soit pas inclus dans la fenêtre de l'ascendant.

Reprenons l'exemple II.1. Considérons la maison construite sans la cheminée ; on désire rattacher la cheminée au faîte du toit.



Noeud à rattacher à la structure de la toiture

- La fenêtre du toit est indiquée en -----
- La nouvelle fenêtre contient le toit et la cheminée -.-.-.-.-.-.-.-.-.-.-.-

Le problème d'agrandissement potentiel de la fenêtre de l'ascendant se répercute sur tous les ascendants du noeud où l'on rattache. Il y a donc un recalcul des fenêtres des ascendants, qui se fait de la façon suivante :

Algorithme :

- Si la fenêtre F_C du noeud considéré (au départ, le noeud rattaché) est incluse dans la fenêtre F_A de son ascendant A, il suffit de déterminer l'emplacement et la taille de F_C dans F_A.

- Si F_C n'est pas incluse dans F_A, il faut agrandir F_A en $F_{A'}$, où $F_{A'}$ est la plus petite fenêtre contenant F_A et F_C. Il en résulte une translation pour l'origine de la fenêtre de chacun des descendants.

par exemple en 2D : $F_A \begin{pmatrix} x_{min} \\ y_{min} \end{pmatrix}, \begin{pmatrix} x_{max} \\ y_{max} \end{pmatrix}$ devient $F_A : \begin{pmatrix} x'_{min} \\ y'_{min} \end{pmatrix}, \begin{pmatrix} x'_{max} \\ y'_{max} \end{pmatrix}$,

la translation à effectuer est de $\begin{pmatrix} \frac{x'_{min} + x'_{max}}{2} - \frac{x_{min} + x_{max}}{2} \\ \frac{y'_{min} + y'_{max}}{2} - \frac{y_{min} + y_{max}}{2} \end{pmatrix}$

De plus, il faut redéterminer les tailles relatives de ces fenêtres par rapport à F_A : l'accroîssement de taille (dans chaque dimension) par rapport à A est tel que si la taille de A est prise comme l'unité, la taille de A' est : 1 + x (accroîssement en x) où x est l'accroîssement relatif de la taille de A' par rapport à celle de A ; donc la taille de A' par rapport à la taille de A est un rapport a/b (où a > b). La taille de A' étant considérée comme l'unité, toutes les tailles des fenêtres des descendants de A sont dans le rapport b/a par rapport à ce qu'elles étaient avant le rattachement.

- Il reste à considérer le noeud A par rapport à son ascendant, (F_A devient F_C) et à réitérer l'algorithme.

Visualisation :

Le second problème est celui de la visualisation dans la clôture. On part du principe que tout rattachement ne peut se faire que dans la clôture contenant l'objet ascendant de celui que l'on désire rattacher : seule la partie incluse dans la clôture est prise en compte. Il est donc

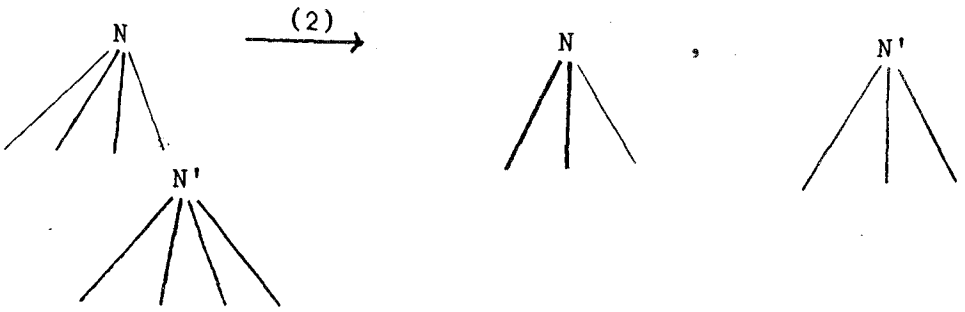
aisé de positionner la fenêtre résultante dans cette clôture pour la visualisation. Le mode de fonctionnement est obligatoirement le contexte multiple ; après le rattachement, le passage en contexte unique ne pose aucun problème, car la clôture n'a pas été modifiée par cette fonction.

* Le rattachement d'une racine d'arborescence n'entraîne pas de modification automatique de la liste des contextes visualisés du mode de fonctionnement "contexte multiple" ; en particulier, la clôture de la racine rattachée contient encore cette sous-arborescence après le rattachement. La conservation ou la non-conservation de ce contexte dans cette clôture est laissée au choix de l'utilisateur.

Note : Les différentes clôtures permettent de considérer l'écran comme un repère absolu commun à tous les objets-images ce qui facilite les déterminations des tailles des objets et les calculs de fenêtres : taille et position relatives entre-elles. Ainsi, les fenêtres ne sont pas seulement des référentiels, mais elles ont une composante taille importante.

b) *Clivage*

Il correspond à la règle (2) suivante :



Il y a, d'une part, suppression de N' dans l'arborescence de N, et conservation de l'objet-image N' indépendant de N.

Notons que si N' est l'unique descendant de N, la suppression de N' vis-à-vis de N entraîne que N n'est plus visualisable (c'est un noeud non défini par une ou des feuilles).

N n'est donc plus accessible visuellement, mais l'est encore par son nom (s'il en a un) ; il est temporairement incomplètement spécifié, et le système interdit de sauvegarder l'objet-image N tant qu'il n'a pas été défini par au moins un descendant. Lors de la demande de clivage, la détection de N' unique descendant de N provoque un dialogue avec l'usager : le noeud N n'aura plus de descendant ; faut-il le supprimer ou le conserver ? En cas de conservation, l'utilisateur devra compléter sa description.

Le clivage est une fonction à deux paramètres : le premier est le noeud père de la sous-arborescence à supprimer (c'est N sur le schéma) ; il est implicitement identifié par le noeud courant. Le second paramètre à identifier (par les fonctions d'accès) est la racine de la sous-arborescence à éliminer, celle-ci étant l'un des fils du premier paramètre (c'est N' sur le schéma) ; la validation entraîne alors le clivage du sous objet-image N'.

Effet du clivage sur l'environnement

Cette fonction a l'effet inverse du rattachement.

* Afin de rendre l'enchaînement clivage-rattachement équivalent à une action vide, dans le cas où le rattachement consiste à remettre l'objet-image considéré à sa place avant le clivage, le noeud courant après l'opération de clivage est le noeud père du noeud détaché.

* Il n'y a pas de modification du noeud contexte de l'objet-image initial.

* Alors que le rattachement à un noeud N d'un noeud N' nécessite le placement de N' par rapport à N, la fonction de clivage ne nécessite aucun placement (mais il faut éventuellement créer une clôture, fonction superflue pour le rattachement).

* Le clivage du noeud N' par rapport à son ascendant N entraîne la suppression de la fenêtre F_N, de N' auparavant incluse dans la fenêtre F_N

de N (mais F_N, n'est pas supprimée en tant que fenêtre de N'). De façon symétrique aux modifications engendrées par un rattachement, il faut donc modifier la fenêtre des noeuds ascendants de N'. Ces modifications se font dans le sens de la réduction de leur taille : elles peuvent sembler moins cruciales que les modifications d'agrandissements dûs aux rattachements. En fait, il faut absolument effectuer ces réductions, car on risquerait un accroîssement progressif des tailles de fenêtres, ce qui serait tout-à-fait gênant pour la visualisation ! (la fenêtre serait maximale, alors que la fenêtre réelle ne serait qu'un point). Les mises à jour sont symétriques de celles réalisées pour le rattachement.

Visualisation :

De même que le rattachement ne peut se faire qu'en mode de fonctionnement contexte multiple, le clivage ne peut s'exécuter qu'en contexte multiple ; la sous-arborescence de racine N' clivée du noeud N n'est plus forcément accessible visuellement ; elle l'est si au moins une clôture dans l'écran visualise un noeud contexte descendant de N' ou N' lui-même. Il en résulte un dialogue avec l'usager lui permettant soit de conserver l'accès à N' par l'une des clôtures existantes, soit de créer une clôture pour y visualiser N', soit supprimer tout accès à N'. Ce dialogue peut se réaliser, par exemple, de la façon suivante, après la validation du clivage :

(a) - Le noeud détaché est accessible ; voulez-vous une nouvelle clôture ?

(b) - Le noeud détaché n'est plus accessible ; voulez-vous une nouvelle clôture ?

* La réponse oui aux cas (a) et (b) entraîne la définition d'une clôture et active la fonction "DEFINITION CONTEXTE" qui permet de visualiser le noeud considéré dans la nouvelle clôture ; le clivage effectif se déclenche ensuite automatiquement.

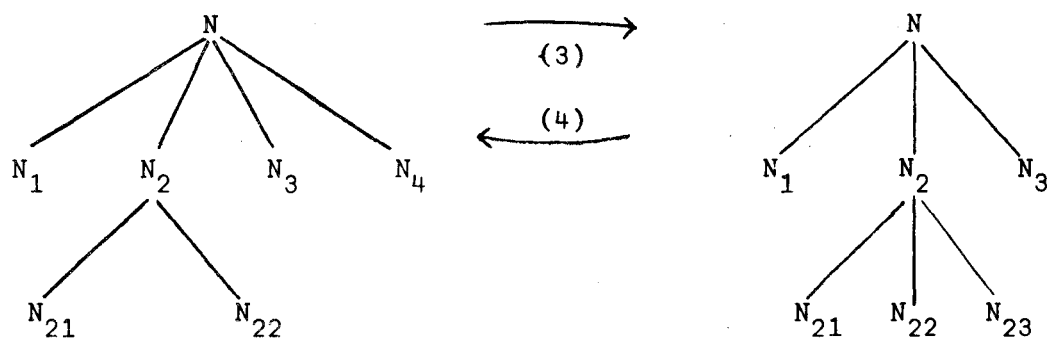
* La réponse non au cas (a) laisse inchangée la liste des contextes visualisés, et le noeud détaché reste accessible.

* La réponse non au cas (b) supprime tout accès au noeud détaché par les primitives définies en III.3.

• Le passage en contexte unique immédiatement après le clivage fournit l'image du noeud contexte correspondant au noeud courant de l'arborescence antérieure au clivage ; la clôture n'ayant pas été modifiée par le clivage, l'opération ne pose pas de problème.

III.5.2.2. Déplacement local

C'est la fonction qui permet de transformer un frère en un fils d'un noeud. Elle correspond à la règle (3) suivante ; la fonction inverse (le fils d'un noeud devient le frère) est schématisée par la règle (4).



Cette fonction n'est pas indispensable pour le système, car un déplacement local peut se réaliser par l'enchaînement d'un clivage suivi d'un rattachement. Nous l'introduisons cependant dans les fonctions de base, car elle est très utile pour les regroupements de noeuds frères (élaboration d'un niveau intermédiaire) et elle simplifie beaucoup le travail de l'utilisateur. Le noeud à déplacer est repéré par des fonctions "↦" ou "↤" ; il n'y a pas de placement à faire, alors que par clivage/rattachement, il faut passer en contexte multiple et identifier successivement les noeuds...

De plus, cette fonction restreint les mises à jour (fenêtre des ascendants jusqu'à la racine éventuellement) dûes à la fois au clivage et au rattachement, en limitant les calculs aux niveaux concernés directement par l'opération ; reprenons les noeuds de la règle (3) (ou (4)) ; nous constatons que :

- le noeud N est visualisé car N_2 et N_4 le sont ;

- la visualisation du noeud N n'est pas modifiée par la fonction ;

- la fenêtre F_N de N n'est donc pas modifiée ;

- F_{N_2} (i.e., fenêtre de N_2) $\subset F_N$ et F_{N_4} (i.e., fenêtre de N_4) $\subset F_N$.

Il en résulte les mises à jour suivantes pour la règle (3) (réciproquement règle (4)) :

- F_{N_2} devient $F_{N_2'}$, i.e., la plus petite fenêtre contenant F_{N_2} et F_{N_4}.

- Détermination de N_4 par rapport à N_2' et non plus par rapport à N (la transformation géométrique résultante $g_{N_4N_2} = g_{N_4N} \circ g^{-1}_{N_2'N}$).

Ce sont les seules mises à jour à effectuer, car la visualisation de N est inchangée. Pour ces opérations de regroupements de frères, cette fonction simplifie donc à la fois le travail de l'utilisateur et la tâche du système.

Pour les mêmes raisons d'enchaînement des fonctions (3) et (4), nous posons que

* *pour la règle (3)* ~ La fonction dépend du noeud courant (N_2) auquel l'utilisateur attribue un fils repéré parmi ses frères (N_4) à l'aide des fonctions "→" et "↓" ; le noeud courant n'est pas modifié par la fonction. En prenant ce choix de noeud courant, nous rendons simple l'enchaînement des fonctions (3), ce qui permet de regrouper facilement des frères par combinaison de (5) puis de $(3)^*$ (cf. III.5.2. Introduction).

- Le noeud contexte n'est pas modifié.

- Le mode de fonctionnement est indifférent (contexte unique ou multiple).

* *pour la règle (4)* - La fonction a un paramètre, c'est le noeud (N_4) fils du noeud courant (N_2) dont il va devenir un frère ; le noeud courant n'est pas modifié par la fonction

- Pour pouvoir appliquer cette fonction, il faut que le noeud contexte soit un ascendant du noeud courant N_2, ainsi, N_4 sera toujours visualisé lorsque la fonction est exécutée. Le contexte n'est pas modifié.

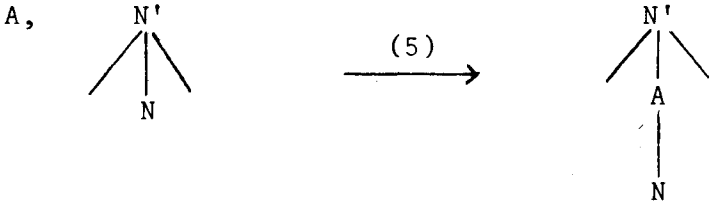
- Le mode de fonctionnement est indifférent.

Cette fonction est soumise à une restriction importante : si le noeud paramètre (N_4) est le seul descendant du noeud courant N_2, cette fonction est interdite, car elle équivaut à une suppression, et le noeud N_2 ne serait plus visualisable (cf. III.5.2.3.).

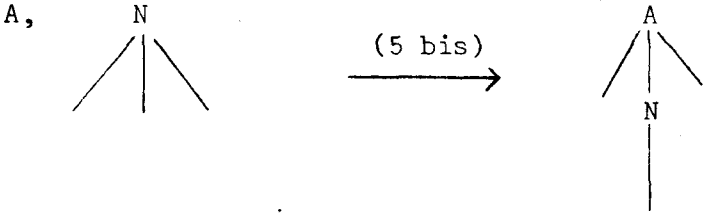
III.5.2.3. Insertion/Suppression

a) Insertion

Elle correspond à la règle (5) suivante :



si N est une racine, la règle (5 bis) est la suivante :



Cette fonction consiste à insérer un noeud ascendant A à un noeud N ; nous l'appellerons donc indifféremment "ajout ascendant" ou "insertion". Elle n'est intéressante que dans la mesure où l'utilisateur désire regrouper des noeuds, soit pour construire un objet à partir d'éléments indépendants, soit pour définir un niveau intermédiaire de description dans un objet-image. Elle se décompose en fait en deux fonctions élémentaires : d'une part, la création d'un noeud (qui n'est pas une feuille), qui est automatique, d'autre part l'insertion proprement dite, qui permet de compléter la définition du noeud inséré.

C'est une fonction à un seul paramètre : insérer un noeud qui sera père d'un noeud identifié (ce sera le noeud courant).

Pour les enchaînements de fonctions, le noeud courant après l'insertion est le noeud inséré. En effet, nous pouvons détailler l'enchaînement qui permet de regrouper des racines d'arborescences en un objet-image: C'est l'insertion d'un ascendant A à l'une de ces racines, suivie de rattachements des autres racines à A. Le choix de noeud inséré comme noeud courant assure l'aisance de cet enchaînement.

Il est important de noter que l'insertion ne demande aucun placement de la part de l'utilisateur. L'image n'est pas modifiée par cette fonction, car le noeud n'est visualisable que par son seul et unique descendant après l'insertion.

Remarque : L'insertion ne permet pas de créer de feuille d'arborescence, alors qu'elle permet de créer des racines et des niveaux intermédiaires.

Effet sur l'environnement

* Le noeud courant après l'insertion est le noeud inséré.

* Si le noeud contexte est le noeud courant N avant l'insertion, le noeud contexte après l'insertion, est le noeud inséré ; sinon,

le noeud contexte reste inchangé (c'est un ascendant du noeud inséré).

* L'ajout de l'ascendant A à un noeud N provoque la détermination d'une fenêtre F_A pour A. De façon automatique, on prend $F_A = F_N$ où F_N est la fenêtre de N ; il n'y a donc aucun problème de détermination de fenêtre, contrairement aux mises à jour nécessaires pour le rattachement.

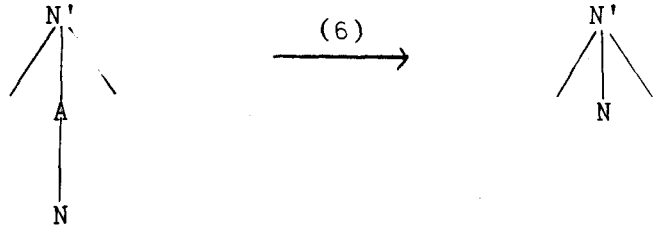
* La fenêtre F_A de A étant identique à la fenêtre F_N de N, F_N = identité (F_A) ; il n'y a pas à placer N par rapport à A. Cependant, si N', de fenêtre $F_{N'}$, était le père de N avant l'insertion, il faut que l'emplacement de F_A dans $F_{N'}$ soit identique à l'emplacement de F_N par rapport à $F_{N'}$ avant l'insertion. Il suffit de recopier (automatiquement) le placement de F_N (par rapport à $F_{N'}$) au placement de F_A (par rapport à $F_{N'}$) et de donner ensuite le placement "identité" pour F_N par rapport à F_A. Cette mise à jour du placement des fenêtres est inchangée si N est une racine ; c'est pourquoi nous avons fait ce choix de $F_A = F_N$ (et non pas $F_A = F_{N'}$).

Visualisation

Le mode de fonctionnement est indifférent pour l'insertion, il n'est pas modifié et la visualisation reste inchangée.

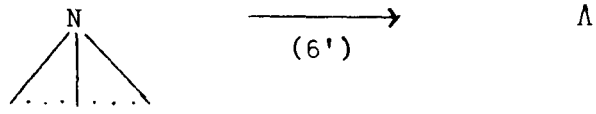
b) *Suppression*

Cette fonction est a priori la fonction inverse de la fonction d'insertion, et correspondrait donc à la règle (6) suivante :



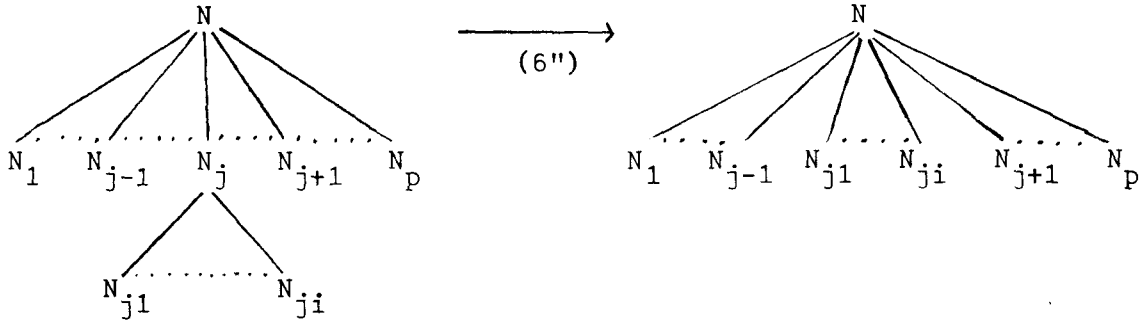
En pratique, la suppression agit rarement sur un noeud n'ayant qu'un seul descendant, car il faudrait faire une succession de déplacements locaux pour supprimer un niveau intermédiaire, et une succession de clivages pour supprimer une sous-arborescence. Nous retiendrons donc la sémantique suivante pour la fonction de suppression :

* la suppression d'une racine d'arborescence consiste à supprimer toute cette arborescence, ce qui se schématise par la règle (6')



où Λ est le mot vide.

* la suppression d'un noeud qui n'est pas racine consiste à supprimer ce noeud seul, ce qui se schématise par la règle (6")



Le noeud N_j est supprimé, mais tous ses descendants le remplacent. Cette règle est bien la généralisation de la règle (6) à l'ensemble des descendants du noeud supprimé.

Remarque : Pour suprimer une sous-arborescence A, il suffit d'enchaîner le clivage de A et sa suppression (A est alors une racine).

Etudions l'effet de la suppression selon qu'elle est ou non appliquée sur une racine.

Suppression d'une racine

- Le noeud courant, qui était la racine, est supprimé ainsi que le noeud contexte ; a fortiori, il n'y a plus de fenêtre associée à ces noeuds.

- La fonction n'a qu'un paramètre, i.e., la racine ; bien qu'il ne paraisse pas nécessaire de valider la fonction pour son exécution, nous imposons la validation afin que l'utilisateur ne détruise pas par mégarde une arborescence. Lorsque la suppression est demandée, l'image de l'arborescence auparavant visualisée est effacée de l'écran (l'effacement n'agit que sur l'image d'un objet mais n'affecte pas sa structure). L'effacement est suivi de la demande de validation, et si la fonction est validée, de la suppression complète de l'objet ; on ne peut plus y accéder visuellement, ni même par nommage, car il n'existe plus du tout.

Suppression d'un noeud qui n'est pas une racine

- Le noeud courant, qui était le noeud à supprimer, est remplacé par son premier descendant ; ainsi, l'enchaînement insertion/suppression de ce noeud est équivalent à une action vide. Notons toutefois que l'enchaînement suppression/insertion doit être suivi de déplacements locaux pour que l'action résultante soit vide (il faut reconstituer la descendance du noeud inséré, sachant que le premier descendant est seul défini après l'insertion).

- Le noeud contexte doit être un ascendant du noeud supprimé ; il n'est pas modifié par la fonction.

- La fonction n'a qu'un paramètre, le noeud à supprimer ; elle sera néanmoins validée pour plus de sécurité.

- La suppression du noeud N_j entraîne la suppression de sa fenêtre F_{N_j} mais il est aisé de connaître les positions relatives des descendants de N_j par rapport au noeud père de N_j (composition des

transformations géométriques d'un niveau par rapport au niveau père). La suppression de F_{N_j} ne modifie donc que la fenêtre des descendants directs de N_j (car les fenêtres sont emboîtées).

Quelle que soit la nature de la suppression (un noeud seul ou une arborescence), le mode de fonctionnement est indifférent et n'est pas modifié. En contexte multiple, la suppression d'une arborescence libère le contenu de la clôture directement concernée. Pour la cohérence des données, elle doit également se répercuter sur chacune des clôtures qui visualise le noeud faisant l'objet de la suppression. Les effets de bords sont les suivants : Si une clôture visualise une sous-arborescence du noeud supprimé (ou le noeud supprimé N_j), elle est libérée. Si le noeud contexte contient le noeud supprimé N_j, N_j est effectivement supprimé et si le noeud courant appartenait à la sous-arborescence de N_j (ou N_j lui-même), alors le noeud courant devient l'ancien père (N) de N_j, sinon, le noeud courant n'est pas modifié. La suppression ne nécessite évidemment aucun placement, mais affecte la visualisation des objets ; la suppression d'un noeud seul entraîne que le noeud courant et le noeud contexte ne peuvent plus être le noeud supprimé, visualisé par ses seuls descendants.

III.5.2.4. Conclusion

Les modifications que nous venons d'étudier sont les fonctions de base agissant sur la structure des objets-images. Nous avons vu que certaines (rattachement) nécessitent un placement ; nous allons donc étudier maintenant les fonctions permettant de placer (ou de déplacer) des objets-images, et les façons de les représenter dans la structure de données.

III.5.3. Modification d'un noeud

Introduction

Nous appelons modification d'un noeud toute fonction qui modifie les caractéristiques du noeud autres que les informations d'ordre structurel. Nous pouvons considérer deux types principaux dans ces modifications :

* les placements, qui fournissent la position et la taille d'un sous objet-image par rapport à son ascendant ;

* les transformations dites "graphiques" en ce sens qu'elles ont trait aux attributs spécifiquement graphiques des objets [New-Sp, Bra.].

Nous avons noté l'importance des placements dans la fonction de rattachement (III.5.2.1.). Cette transformation géométrique qu'est le placement n'est pas seulement une conséquence du rattachement ; c'est également une fonction de déplacement d'objets (i.e., modification d'un placement antérieur) qui existe pour l'utilisateur indépendamment des autres fonctions.

Cette partie se décompose donc de la façon suivante

III.5.3.1. Transformations géométriques

III.5.3.2. Représentation des transformations géométriques

III.5.3.3. Attributs graphiques

Toutes ces modifications sont liées à la représentation des objets-images sur l'écran, et sont toutes des fonctions liées à la visualisation de ces objets graphiques.

III.5.3.1. Transformations géométriques

Placer un sous objet-image S par rapport à son ascendant A, c'est définir une transformation géométrique qui, dans un certain référentiel, positionne S par rapport à A. On peut (classiquement) regrouper les fonctions par des propriétés communes.

Rappel

* Les transformations géométriques sont des affinités, dont on peut extraire un sous-ensemble qui est celui des similitudes (applications qui conservent le rapport des longueurs des objets, et le produit scalaire). De ces similitudes, on peut encore extraire un sous-ensemble qui est celui des isométries (applications qui conservent en outre les distances). L'ensemble des isométries est un groupe et comporte les translations, les rotations et les symétries. L'ensemble des similitudes comporte en outre les homothéties, mais pas les changements d'échelles (rapport de longueur différent selon les directions), qui sont des affinités.

* Nous considérons donc différemment les similitudes et les affinités, toutes appliquées sur des noeuds inclus chacun dans une fenêtre. Nous en déduirons un rôle plus précis de la fenêtre d'un objet, qui est dès à présent connue comme référentiel auquel on attribue une taille (cf. III.4. et III.5.2), et qui est utilisée à des fins de repérage d'un objet par rapport à ses ascendants.

Nous étudions dès maintenant les placements les plus simples constitués par les isométries, et les conséquences pour la définition plus précise de la fenêtre d'un noeud.

Selon que l'on considère les transformations géométriques sur les objets par rapport aux fenêtres ou sur les fenêtres (des objets) entre-elles, le placement (ou le déplacement) aura des effets différents quant à la visualisation des objets-images.

Prenons tout d'abord l'exemple d'une isométrie :

Soit un sous-objet SO d'un objet O.

SO est défini dans la fenêtre F_{SO}.

O est défini dans la fenêtre F_O.

Déplaçons par une isométrie SO dans O.

1er cas

SO est déplacé dans F_{SO}, et F_{SO} est déplacé par rapport à F_O.

Cette solution ne semble guère satisfaisante, car un déplacement voulu par l'utilisateur génèrerait systématiquement deux déplacements qu'il faudrait déterminer, et qui ne sont pas forcément uniques.

2ème cas

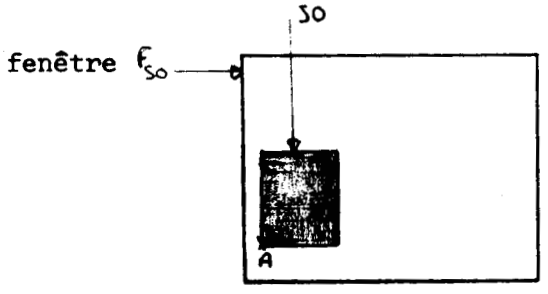
SO est déplacé dans F_{SO}, et F_{SO} reste fixe par rapport à F_O.

Cette solution est celle où

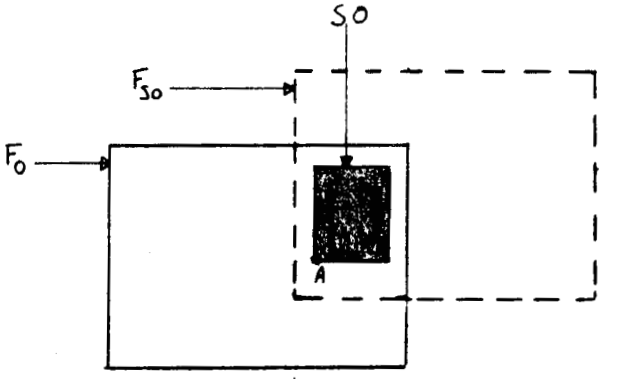
- tout objet est défini dans une fenêtre fixe définie lors de la création de l'objet ;

- toute fenêtre d'un sous-objet est repérée par une translation d'un point pris comme origine de la fenêtre par rapport à l'origine de la fenêtre de l'ascendant.

Prenons un exemple pour "voir" le résultat de la transformation.

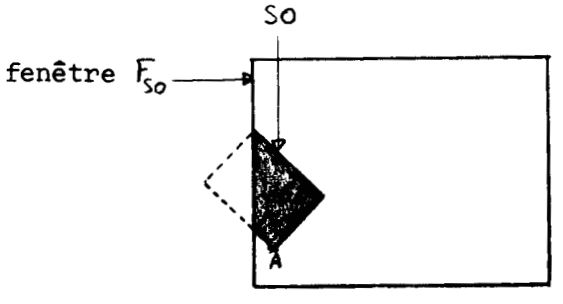


SO dans sa fenêtre F_{SO}

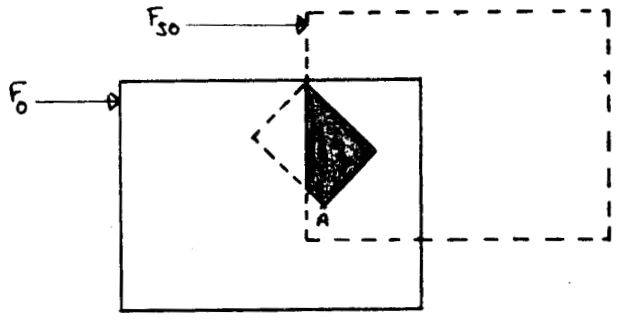


SO dans la fenêtre F_O de son ascendant O

2) Appliquons une rotation de 45° autour de A.



SO dans sa fenêtre F_{SO}



SO dans la fenêtre F_O de son ascendant O

Nous remarquons qu'il y a un coupage ("clipping") de SO dans sa fenêtre qui se répercute dans O. Ce phénomène peut apparaître pour n'importe quelle transformation, et à n'importe quel niveau dans l'arborescence, même s'il n'est pas voulu ! Pour éviter ce phénomène, il faut que l'utilisateur ait conscience de toutes les dispositions initiales des objets dans leur fenêtre, ce qui n'est pas aisé pour de nombreux objets...

3ème cas

SO reste "fixe" dans F_{SO}, et c'est F_{SO} qui est déplacé de façon adéquate.

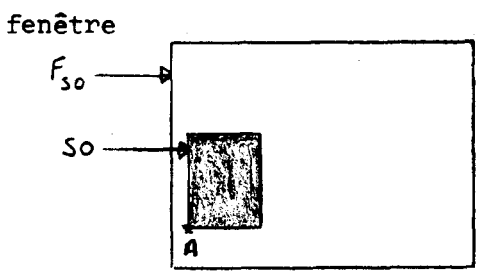
Cette solution est celle où

- tout objet est défini dans une fenêtre indépendante de l'écran réel, et les transformations appliquées aux objets sont traduites en transformations appliquées aux fenêtres des objets. La La fenêtre constituerait en quelque sorte un "écran virtuel" ;

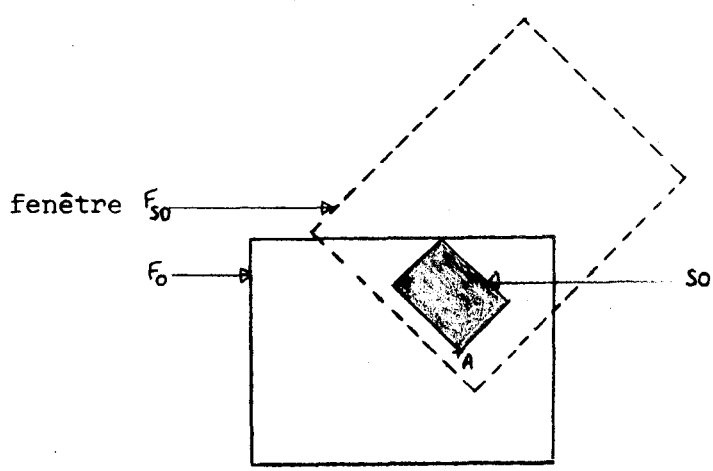
- tout objet est défini intrinsèquement dans sa fenêtre ;

- toute isométrie d'un objet-image par rapport à un ascendant est transformée en une isométrie appliquée sur la fenêtre de cet objet-image.

Reprenons l'exemple précédent du sous-objet SO de O, auquel on applique la même rotation de 45° autour de A. On obtient :



SO reste fixe dans F_{SO}



SO dans la fenêtre F_O de son ascendant O

Cette solution ne provoque pas de coupage parasite. De plus, nous pouvons remarquer que la notion ainsi définie de fenêtre liée à l'objet est indépendante de la nature de l'écran de visualisation ; nous prendrons donc comme définition de fenêtre :
C'est un référentiel dont on connaît

- le point origine du repère,
- tel que le repère est orthogonal orienté

et tel que l'objet est inclus dans un espace rectangle.

Homothétie et changement d'échelle

Ce sont les fonctions qui permettent de définir précisément la relation de taille d'un objet par rapport à sa fenêtre : la taille de la fenêtre doit elle être fixe (c'est l'objet-image qui subit la transformation dans la fenêtre), ou doit-elle être variable de façon à conserver une taille relative constante entre l'objet-image et sa fenêtre ?

Pour les mêmes raisons de coupage parasite dans le placement des objets entre-eux, nous retiendrons la solution de taille variable de la fenêtre relativement aux ascendants (i.e., la deuxième solution).

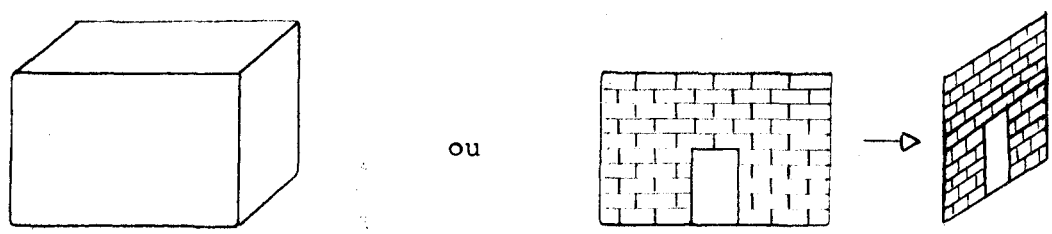
Pour l'utilisateur, l'homothétie est une fonction parfaitement naturelle : augmenter ou réduire la taille d'un objet par rapport à l'ensemble qui le contient ; les changements d'échelle sont des fonctions plus complexes : les plus utiles semblent être

1) le changement d'échelle selon les axes de la fenêtre,

2) le changement d'échelle selon les diagonales d'un quadrilatère ; ce dernier permet de construire des losanges à partir de carrés...

Nous pouvons appliquer ces deux types de fonctions sur la fenêtre d'un objet ; Il y a alors modification de l'objet inclus dans cette fenêtre, mais pas de modification intrinsèque de la définition de l'objet-image dans sa fenêtre.

Le deuxième type d'affinité appliqué sur une fenêtre peut être utilisé pour représenter des objets en perspective, comme sur les exemples ci-dessous



Conséquences

- A sa création, le rapport d'homothétie appliqué sur un objet-image est de 1 ; la fenêtre est liée à l'objet-image et correspond au cadre minimum nécessaire à la description de cet objet-image.

- La visualisation consiste à remplir la clôture de façon telle que la fenêtre y soit incluse, sans modifier les proportions de cette fenêtre.

Conclusion

Nous avons vu ce que sont les différents placements et déplacements vus par l'utilisateur, et les contraintes qu'ils imposent à la définition de fenêtre. Il ne nous reste plus qu'à déterminer la façon de représenter ces transformations dans la structure de données. C'est ce que nous nous proposons de faire maintenant.

III.5.3.2. Représentation des transformations géométriques

III.5.3.2.1. Représentation d'une transformation géométrique

Introduction

La géométrie classique fournit deux façons de représenter les transformations géométriques : l'une, naturelle, en indique le type (symétrie, homothétie, rotation...) et les paramètres ; l'autre, plus abstraite, utilise les résultats de la géométrie projective qui permet de les écrire en termes de matrices. Nous allons présenter ces deux façons et déterminer la plus adaptée aux transformations géométriques désirées pour l'éditeur graphique.

Représentation "naturelle" des transformations

Elle consiste à introduire les paramètres des transformations géométriques envisagées, qui peuvent se réduire à :

* Rotation : centre de rotation, angle de rotation.

* Translation : vecteur de translation.

* Symétrie axiale quelconque : symétrie par rapport à un axe fixe composée avec une rotation ; il suffit de savoir s'il y a symétrie ou non.

* Homothétie : centre d'homothétie, rapport d'homothétie.

* Changement d'échelle : rapports de changement d'échelle, directions.

Etudions la représentation des similitudes et des affinités :

- Les similitudes

Nous pouvons représenter toute similitude S (composée de symétries, rotations, translations, homothéties sous la forme suivante :

$S = T \circ R \circ S \circ H$ où H : homothétie
S : symétrie (vrai/faux)
R : rotation
T : translation

Il est aisé de transformer une homothétie ou une rotation de centre quelconque en une homothétie ou une rotation ayant pour centre un point déterminé suivi d'une translation ; il suffit de connaître un vecteur de translation, un angle, un indicateur de symétrie et un rapport d'homothétie pour pouvoir calculer la transformation résultante. Il y a peu de paramètres, et cette représentation est peu "volumineuse". De plus, on conserve les paramètres de chaque transformation, ce qui permet d'utiliser des résultats géométriques et de limiter les calculs.

Par exemple : $R(O, \theta_1) \circ R(O, \theta_2) = R(O, \theta_1 + \theta_2)$.

• Les affinités

Si l'on désire faire un changement d'échelle différent d'une homothétie, (i.e., si les rapports d'échelle sont différents selon les axes choisis) on ne peut plus composer les transformations de façon à obtenir une transformation résultante de la forme T o R o S o H, car les affinités ne conservent pas les angles ; on ne pourra donc utiliser la représentation "naturelle" des transformations que si l'on restreint l'ensemble des transformations géométriques de l'utilisateur aux similitudes.

Les coordonnées homogènes

Grâce aux résultats fournis par la géométrie projective, nous pouvons traiter dans un espace à (n+1) dimensions les problèmes de transformations géométriques posés dans un espace à n dimensions ; il suffit, en ayant traité le problème à (n+1) dimensions, de faire la projection dans l'espace à n dimensions [New 77].

C'est ainsi que nous pouvons écrire les transformations géométriques sous forme matricielle.

par exemple en dimension 2

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

$$\text{translation} : \begin{cases} a_1 = b_2 = 1 \\ \text{et} \\ a_2 = b_1 = 0 \end{cases} \quad \text{rotation} : \begin{cases} a_1 = b_2 \\ a_2 = -b_1 \\ a_1^2 + b_1^2 = 1 \\ a_3 = b_3 = 0 \end{cases} \quad \text{symétrie} : \begin{cases} a_1 = -b_2 \\ a_2 = b_1 \\ a_1^2 + b_1^2 = 1 \\ a_3 = b_3 = 0 \end{cases}$$

$$\text{changement d'échelle} : \begin{cases} a_2 = b_1 = 0 \\ a_3 = b_3 = 0 \end{cases}$$

Toute transformation géométrique est définie par une matrice 3 × 3 (en dimension 2) ; la composition de transformations géométriques se résume en une multiplication de matrices (idem matrices 4 × 4 pour travailler en 3D).

- cette méthode de représentation des transformations a la simplicité de la généralité ;

- elle présente l'avantage de pouvoir combiner n'importe quelles transformations géométriques, et dans n'importe quel ordre (La multiplication n'est pas commutative, mais il suffit de multiplier les matrices dans l'ordre des combinaisons des transformations).

- Elle présente néanmoins l'inconvénient de faire de nombreux calculs (multiplication de matrices) dans des cas qui peuvent être simples, par exemple , les translations.

On pourrait définir des types de matrices permettant de simplifier les calculs ; il s'agit alors de définir les types Rotation, Symétrie, Translation. Cependant, comme on perd certains renseignement du type,par exemple l'angle θ d'une rotation en représentation matricielle des transformations; on ne peut utiliser des propriétés du genre :

$R(\theta) + R(\theta') = R(\theta + \theta')$ ... avec les angles θ, θ' exprimés par des entiers.

Il faut au contraire utiliser les formules :

$$\cos(\theta + \theta') = \cos\theta \cos\theta' - \sin\theta \sin\theta'$$
$$\text{et } \sin(\theta + \theta') = \sin\theta \cos\theta' + \sin\theta' \cos\theta$$

où les sinus et cosinus sont des réels, qui doivent être exprimés avec une précision relativemnet importante ; la propagation des erreurs dues à la précision des données est équivalente ; toutefois, nous ne connaissons pas précisément la différence d'imprécision due à la propagation des erreurs dans le calsul, nous la supposerons faible.

Prenons l'exemple de la composée de deux rotations : $R(\theta) + R(\theta')$

- θ est connu avec une précision ε
- θ' est connu avec une précision ε'

nous avons en fait $R(\theta + \varepsilon) + R(\theta' + \varepsilon') = R(\theta + \theta' + \varepsilon + \varepsilon')$.

Soit $\varepsilon' = \varepsilon$ la précision obtenue est toujours inférieure ou égale à 2ε sur le calcul de $R(\theta + \theta')$

En représentation matricielle, nous calculons en fait $\cos(\theta + \theta' + \varepsilon + \varepsilon')$ et $\sin(\theta + \theta' + \varepsilon + \varepsilon')$ ce qui donne, pour le calcul de cosinus : (en prenant $\varepsilon = \varepsilon'$ pour simplifier les calculs)

$$\cos(\theta + \theta' + 2\varepsilon) \simeq \cos(\theta + \theta') \underbrace{- \left(\frac{\varepsilon^2}{2} \cos(\theta + \theta') + 2\varepsilon \sin(\theta + \theta')\right)}_{\varepsilon'}$$

car $\varepsilon \to 0$

l'erreur $\varepsilon' \in \left[\frac{\varepsilon^2}{2}, 2\varepsilon\right]$ et dépend des valeurs de θ et de θ'.

$$\sin(\theta + \theta' + 2\varepsilon) \simeq \sin(\theta + \theta') \underbrace{+ \left(2\varepsilon \cos(\theta + \theta') - \frac{\varepsilon^2}{2} \sin(\theta + \theta')\right)}_{\varepsilon''}$$

car $\varepsilon \to 0$

l'erreur $\varepsilon'' \in \left[\frac{\varepsilon^2}{2}, 2\varepsilon\right]$ et dépend des valeurs de θ et de θ'.

Remarque : Nous n'avons pas tenu compte des erreurs d'arrondi ou de troncature ; nous avons considéré qu'elles étaient négligeables, ce qui suppose une bonne précision sur les nombres réels et accroît leur encombrement.

Comparaison des méthodes et choix

* Notons d'abord que pour chacune des deux méthodes, il faut toujours transformer les rotations, symétries, homothéties quelconques en rotations, symétries, homothéties par rapport, soit à un point donné, soit à un axe donné, composé avec une translation à déterminer.

De plus, au niveau d'un noeud donné, toute transformation géométrique est déterminée relativement à la fenêtre de ce noeud. Pour déterminer la transformation géométrique depuis la racine d'une arborescence jusqu'à un noeud quelconque donné, il faudra tenir compte des changements de repère correspondants aux différentes fenêtres.

De ces deux points de vue, les deux méthodes de représentation sont similaires, bien qu'elles ne nécessitent pas la même charge de calculs. Ces considérations sont transparentes pour l'utilisateur.

* Point de vue de l'utilisateur

- La représentation "naturelle" des transformations est proche de ce que demande l'utilisateur ; elle nécessite peu de paramètres et est bien explicite. Par contre, elle interdit l'usage des affinités quelconques, et l'utilisateur ne pourra pas faire de perspective cavalières.

- La représentation par coordonnées homogènes est plus artificielle que la précédente ; elle est également plus lourde, mais elle est aussi beaucoup plus générale, et elle permet de réaliser des perspectives cavalières ; elle s'adapte plus facilement au traitement en 3D.

* Point de vue implémentation

- La représentation "naturelle" est peu encombrante (8 octets environ) car les paramètres sont des entiers ; elle nécessite relativement peu de calculs au moment de l'affichage. Cependant, elle présente l'inconvénient suivant : Il faut représenter de façon unique les composées des transformations géométriques afin de ne pas créer d'ambiguité , car les transformations géométriques ne sont pas commutatives

en général. Ceci implique une conversion supplémentaire et systématique des transformations, afin de les composer dans l'ordre T o R o S o H. Dans la pratique, les conversions à effectuer sont en général simples à calculer. Cette méthode est économe en place, ne nécessite que peu de calculs, mais très limitative quant aux possibilités d'extension souhaitées : pas d'affinité, et pas d'extension 3D.

- La méthode des coordonnées homogènes est beaucoup plus encombrante (des matrices (p+1, p+1) de réels, dont on veut une précision relativement bonne (comparée à la précision de l'écran) pour travailler dans un espace de dimension p). Elle est aussi plus lente que la méthode précédente, car on doit effectuer des produits de matrices ; pour calculer la transformation B dans un repère $(O, \vec{i}, \vec{j})$ à partir de la transformation A dans le repère $(O', \vec{i}', \vec{j}')$ on a $B = P^{-1} AP$ avec P matrice de changement de base ... donc il faut calculer P, P^{-1}... Les matrices de transformations étant assez particulières, et de dimension restreinte (3 × 3 ou 4 × 4), on peut réduire les temps de calculs par optimisation.

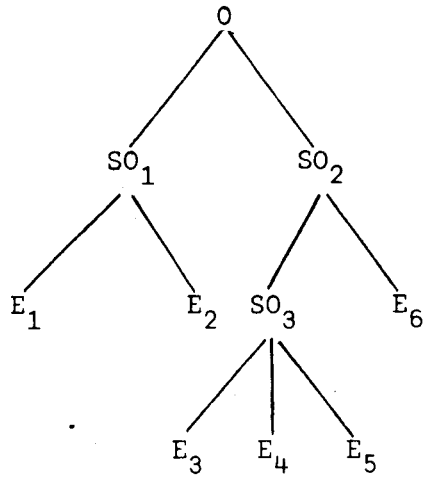
* Le choix concilie les avantages des deux méthodes, en se basant sur la constatation suivante : il est aisé de transformer la représentation naturelle (celle de l'utilisateur) en représentation matricielle (représentation interne plus générale), alors que la transformation inverse est difficile, si ce n'est même impossible dans certains cas. Les fonctions de dialogue sont réalisées de façon telle que l'utilisateur fournit naturellement les paramètres des transformations qu'il demande, et la traduction matricielle est automatique ; elle permet d'exécuter les calculs de composition de fonctions, et de conserver aisément la transformation géométrique résultante, quelle qu'elle soit (même les affinités). Pour minimiser l'encombrement et les calculs inhérents à cette méthode, nous utilisons les propriétés des matrices de la forme

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{ou} \quad \begin{pmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

dans les calculs d'inverse et de multiplication de matrices.

III.5.3.2.2. Représentation des transformations géométriques dans la structure de données

Soit un objet O :

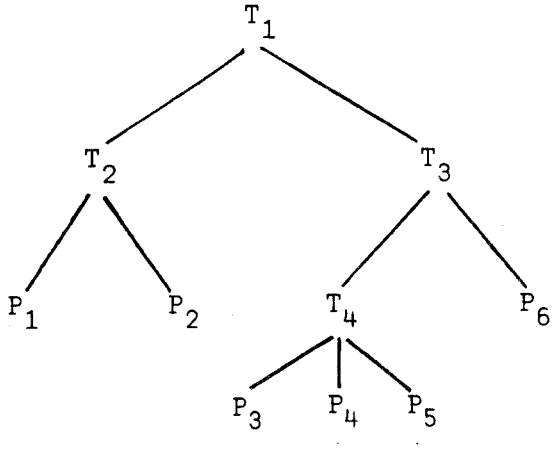


Tout objet est créé dans sa fenêtre, à un emplacement donné de l'écran, et d'une taille donnée.

Structure hiérarchique

Pour se donner la possibilité d'appliquer une transformation géométrique à n'importe quel niveau de l'arborescence, il faut conserver une trace des transformations géométriques à chaque niveau ; de plus, si l'on veut accéder au sous-objet d'un objet, il faut connaître les transformations de tous ses descendants par rapport à ce sous-objet, et non pas forcément par rapport à la racine de l'objet.

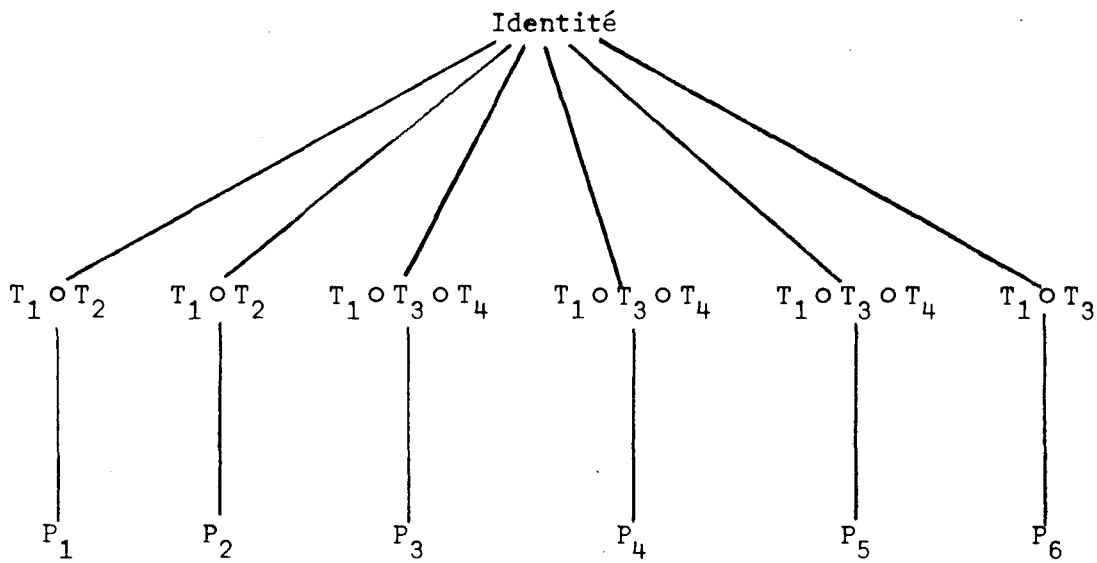
Nous pouvons alors représenter les transformations T_i des noeuds sous forme d'une "Structure hiérarchique", soit, pour l'objet O :



Structure hiérarchique simplifiée

La représentation précédente présente un inconvénient majeur : A chaque fois que l'on veut représenter un objet, il faut calculer la composée de toutes les transformations effectuées depuis la racine de l'objet jusque chaque feuille ! Ceci est long, et nuit donc à l'aspect interactif du système. [New 77]

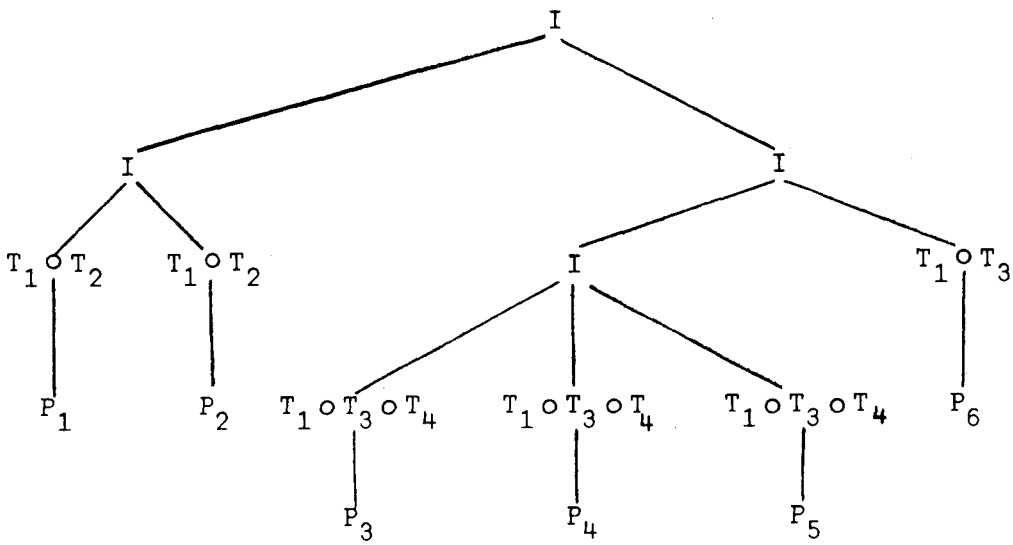
Il serait bon de conserver la transformation résultante au niveau de chaque feuille. Ceci nous suggère la représentation en "Structure hiérarchique simplifiée", soit pour l'objet O :



Structure hiérarchique canonique

En utilisant la représentation ci-dessus, nous perdons toute la structure de l'objet-image. Il n'est plus possible d'accéder à un noeud de l'arborescence.

La représentation en "Structure hiérarchique canonique" [RAUCH 79] permet de conserver une trace de la structure de l'objet :

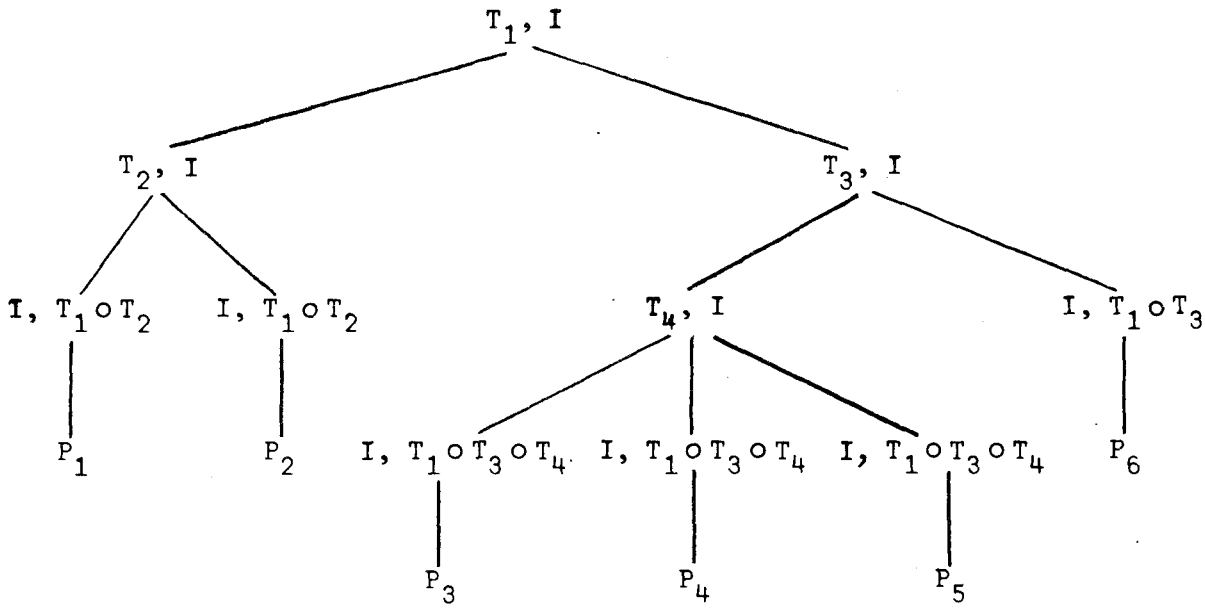


Structure hiérarchique double

La représentation précédente conserve la structure de l'objet, mais l'on n'a pas la possibilité de retrouver les transformations à effectuer pour représenter un sous-objet SO_i quelconque d'un objet O, dans la fenêtre de SO_i (et non pas de O).

Pour pallier à cela, une solution consiste à "doubler" les informations sur les transformations, en conservant d'une part la trace des transformations à chaque noeud (structure hiérarchique simple) et d'autre part, la transformation résultante.

Soit pour O :



A priori, il peut sembler gênant de devoir "doubler" les informations, mais nous pouvons remarquer que pour chaque noeud différent d'une feuille, le couple de transformations est du type (T, I) et pour chaque feuille, le couple est du type (I, T), avec I = identité, et T transformation quelconque.

Il n'est pas nécessaire de conserver l'information de la transformation Identité (élément neutre des transformations!) ; il suffit donc de savoir si le noeud est une feuille ou non pour interprêter correctemnet la transformation associée au noeud.

Le noeud de référence pour le calcul de la transformation résultante peut être, au choix de l'implémentation, soit la racine de l'objet, soit le noeud contexte, ce qui est plus utile !

Remarque : Que ce soit du point de vue de l'implémentation ou de celui de la gestion, la mise en oeuvre de ce type de structures est plus délicate que celle des autres types de représentation. Nous choisissons néanmoins cette dernière façon de représenter les transformations géométriques, car elle conserve la structure des objets, la structure des transformations, mais elle n'est pas trop pénalisante en temps d'accès et d'affichage (on connait les transformations résultantes des feuilles par rapport au noeud contexte - ou la racine-).

III.5.3.3. Attributs graphiques

Ce sont les informations permettant de calculer la couleur, l'intensité lumineuse des images. Parmi les attributs, on distingue :

- la couleur
- la luminosité
- la transparence
- la réflectance
- la texture
- l'ombrage
- le clignotement...

Mise à part la couleur, qu'il est facile de déterminer, la plupart des attributs nécessitent de nombreux calculs lors de la visualisation. Différents modèles mathématiques [Olej] peuvent être pris pour calculer ces attributs. Notre propos n'est pas d'étudier en détail ces différents modèles, mais d'en déterminer, en connaissance de leur complexité, une représentation dans la structure de données.

Représentation hiérarchique des attributs

Le principe de base conduisant à une représentation hiérarchique des attributs est le suivant : tout attribut peut s'appliquer sur un noeud quelconque d'une arborescence, et affecte la visualisation de tous ses descendants [Bradier], c'est le raisonnement déjà utilisé pour les transformations géométriques, mais dans le cas d'attributs graphiques,

la composition des attributs est plus délicate à réaliser ; nous aborderons le cas de la composition de couleurs.

De façon cohérente avec la représentation des transformations géométriques, il faut donc prévoir un mécanisme permettant d'associer au niveau de chaque noeud, un ou plusieurs attributs graphiques ; plusieurs solutions s'offrent à nous :

1) représentation de l'ensemble des attributs dans chaque noeud ;

2) représentation des attributs externe aux noeuds.

Les calculs d'attributs étant longs et complexes, en général, et pour limiter la place occupée par la description d'un objet, nous associerons chaque objet à une zone réservée aux paramètres des différents calculs d'attributs ; nous obtenons ainsi des arborescences d'attributs "parallèles" aux arborescences des objets-images ; ces calculs ne seront faits qu'à la demande de l'utilisateur.

Les modifications d'attributs seront réalisées par simples modifications des paramètres.

A ce choix nous excluons la représentation de l'attribut "couleur", car ce dernier est toujours aisément calculable, peu encombrant (il est en général codé par un nombre interprété en couleur à la visualisation, soit directement, soit à partir d'une table de fausse-couleur).

Contexte d'attribut graphique

Il semble interressant de pouvoir isoler les attributs graphiques spécifiques à un objet-image des attributs provenant du contexte de visualisation de cet objet ; Prenons l'exemple de l'attribut Couleur : un objet, lors de sa définition, possède une certaine couleur ; Selon le contexte d'attribut couleur de la visualisation, il pourra y avoir une modification de la couleur résultante, mais l'attribut propre à l'objet-image est inchangé. Par exemple : l'objet "Soleil" a pour attribut de couleur, la couleur jaune.

Si le contexte est "coucher de soleil", il faut alors établir une "fusion" avec la couleur rouge pour obtenir une couleur orangée ; la "fusion" est une fonction de mélange de couleurs, appelée par l'utilisateur ; elle n'affecte pas la couleur propre de l'objet ; la fusion est une façon de composer les couleurs, et l'on considère que les couleurs associées aux feuilles sont des couleurs propres tandis que les couleurs associées aux autres noeuds sont les contextes d'attributs de couleur ; la représentation des couleurs est donc hiérarchique intégrée à la structure des objets-images.

Conséquences pour l'utilisation des attributs graphiques

Certains attributs graphiques (comme la texture) demandent des temps de calcul longs lors de la visualisation alors que d'autres sont quasiment immédiats (dans des cas simples, la couleur par exemple). Ce déséquilibre nous conduit à deux utilisations distinctes de l'éditeur graphique :

- soit la prise en compte des informations minimales concernant les attributs : la couleur est en général la seule prise en compte ; la visualisation fournit une image relativement grossière d'un objet-image, qui est calculée rapidement : c'est, en quelque sorte, un beau "brouillon",

- soit la prise en compte de l'ensemble des attributs : la visualisation résultera de calculs longs, mais l'image sera plus fine et nuancée ! il n'est pas raisonnable d'espérer dans ce cas travailler en temps réel pour l'utilisateur ; cette visualisation est destinée à être demandée lorsque l'usager veut peaufiner l'image qu'il a créée, dont il a déjà un bon aperçu avec le "brouillon". Cette image est, en quelque sorte, le document "au propre" de l'utilisateur.

Conclusion

Des études doivent être poursuivies dans le sens de l'optimisation des modèles de calculs des attributs graphiques, de façon à obtenir en temps réel des images de plus en plus belles. Ces calculs doivent être d'autant plus rapides qu'ils s'ajoutent aux calculs de matrices de transformations géométriques, dont le nombre varie en fonction de la profondeur des sous-arborescences à visualiser. Ces études doivent s'accompagner de la définition de fonctions de dialogue simples pour l'implémentation de ces attributs ; ce ne sont pas les préoccupations essentielles actuellement, car étant donné l'état actuel de l'art dans ce domaine, ces fonctions sont difficilement implémentables sur des micro-ordinateurs.

CHAPITRE IV

DES OBJETS PARTICULIERS

CHAPITRE IV

DES OBJETS PARTICULIERS

Nous allons considérer trois "types" d'objets-images :

* Les objets-images catalogués sont les objets-images auxquels on accède par leur nom (on dira par "nommage"), qu'ils soient présents en mémoire centrale ou en mémoire de masse. L'accès par nommage est explicite (cf. II.3).

* Les objets-motifs sont des feuilles d'arborescences dont les taches correspondantes contiennent des "éléments d'images répétitifs" (par exemple, un toit est une tache bleue constituée d'ardoises).

* Les objets-récursifs sont des objets encore plus particuliers étant donné qu'ils se décrivent en utilisant leur propre description...

Nous étudions les propriétés de ces objets ainsi que leur utilisation et les contraintes de manipulations de ces objets.

IV.1. LES OBJETS-IMAGES CATALOGUES

Définition :

Ce sont les objets-images (créés en tant que sous-arborescences) de l'utilisateur auxquels il a donné un nom afin de pouvoir y accéder en utilisant ce nom comme référence.

Tous objet-image peut être catalogué. Seuls les objets-images catalogués peuvent être stockés en mémoire de masse, car l'identification d'un objet-image non visualisé ne peut être réalisée que de façon explicite.

L'accès à un objet-image catalogué visualisé peut être réalisé soit explicitement (par nommage) soit implicitement (identification visuelle) (cf. II.3).

Nous étudions dans un premier temps l'emploi de ces objets catalogués, puis les contraintes apportées tant sur la structure de ces objets-images que sur les mécanismes d'accès.

IV.1.1. Principe d'utilisation des objets catalogués, contraintes

IV.1.1.1. Principe d'utilisation

Un objet-image catalogué peut servir à la définition d'autres objets-images. Par exemple, des ordinateurs différents peuvent être réalisés avec quelques composants identiques (cartes) ; un composant ayant été décrit, il faut pouvoir l'utiliser pour chacun des ordinateurs, sans être obligé de le référencer différemment à chaque utilisation. L'exemple est schématisé dans la figure IV.1.

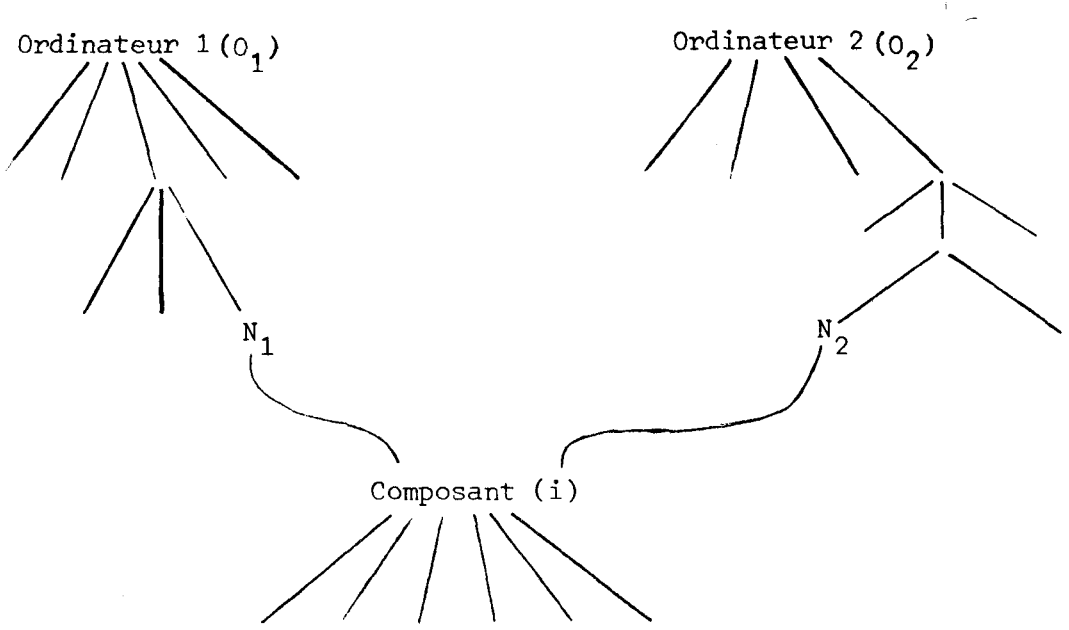


Figure IV.1. : Exemple d'utilisation d'un objet-image (i) par deux autres objets-images O_1 et O_2.

IV.1.1.2. Les contraintes

Elles sont de plusieurs ordres :

(1) Les accès doivent être contrôlés de façon à conserver la nature hiérarchique des objets-images : En effet, si l'on se réfère à la figure IV.1, il est clair que l'accès à l'objet Composant (i) interdit l'application de la primitive "↑", car elle est ambiguë : on ne sait s'il faut accéder à N_1 ou à N_2, la structure arborescente n'étant pas respectée (au plus un noeud père pour un noeud quelconque).

(2) Les modifications de l'objet catalogué composant (i) se répercutent sur tous les objets qui l'utilisent. Ceci peut être très pratique, mais parfois dangereux, ce qui justifie que l'on distingue l'accès aux objets catalogués des accès aux noeuds dans la sous-arborescence des objets.

IV.1.2. Mécanismes proposés

IV.1.2.1. Structure des objets-images nommant des objets-images catalogués

* Noeud de type "appel objet".

Considérons l'objet-image catalogué C référencé dans la description d'un objet-image O. On ne peut accéder à C lui-même par les primitives "↑", "↓", "↦", "↤", ⊕, car on ne pourrait plus remonter dans la structure à partir de C. Ceci revient à dire que le noeud N qui nomme C est considéré comme une feuille vis-à-vis de l'arborescence O à laquelle il appartient. Pour cela, "↓" sur N est sans effet, et ⊕ sur une feuille quelconque de C visualisée dans O fournit N et non la feuille de C. N agit comme une frontière qui ne peut être franchie impunément, sans que l'utilisateur s'en aperçoive ; ainsi, il ne peut pas modifier un objet-catalogué sans se rendre compte de la portée de ses modifications, répercutées sur tous les objets qui le nomment.

Le noeud N, qui nomme C, est une feuille d'un type particulier, le type "appel-objet", spécifiant que N est défini par la description de C. Il est réservé à l'établissement de la liaison entre le père P de N et l'objet-catalogué C auquel on accède par nommage. En particulier, N contient toutes les informations géométriques et graphiques relatives à O pour la description de P. Ces informations de position de O par rapport à P, ainsi que les attributs graphiques donnés relativemnt à l'objet P ne peuvent évidemment être déterminées dans C, celui-ci étant accessible par nommages provenant d'autres objets-images. Les transformations graphiques entre N et C sont toutes l'identité.

* Définitions de liens

- Nous appellons *lien arborescent* le lien qui unit deux objets O_1 et O_2 tels que, O_2 est un élément de la décomposition de O_1, O_2 est fils de O_1 au sens du chapitre III. Les fonctions "↑" et "↓" exploitent uniquement les liens arborescents.

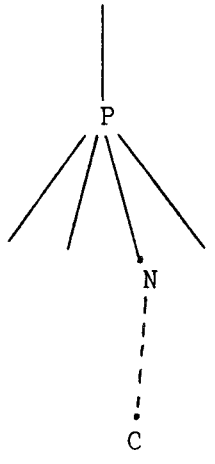
- Nous appellerons *lien par nommage* le lien qui unit deux objets O_1 et O_2 tel que O_1 nomme O_2. Ceci équivaut à dire que O_1 est du type "appel-objet" de O_2.

* Conséquences

1/ Par ce mécanisme, le noeud de type "appel-objet" est considéré comme une feuille de l'arborescence où il se trouve, vis-à-vis des primitives d'accès du chapitre III, et donc aussi pour la notion de contexte et la désignation, mais il peut être défini par un objet-image très complexe décrit préalablement éventuellement déjà utilisé, ce qui justifie les liens par nommage.

2/ La racine de l'objet-image catalogué C ne contient (évidemment) pas les informations relatives à un noeud qui l'appelle (par nommage). C est néanmoins considéré comme complètement spécifié, car tous les noeuds autres que la racine sont complètement spécifiés, et le système attribue automatiquement des valeurs par défaut aux attributs relatifs à un "père" éventuel. Par exemple, la valeur par défaut de la transformation géométrique de la racine de C par rapport à un noeud appelant quelconque est l'identité. C est donc visualisable seul, comme tout objet-image.

3/ La notion de fenêtre est adaptée à ce mécanisme de liaison entre objets. Soit le schéma suivant :



N est le noeud "appel objet" de C.
P est le père de N.
Soient F_N, F_C, F_P les fenêtres recpectives de N, C et P.

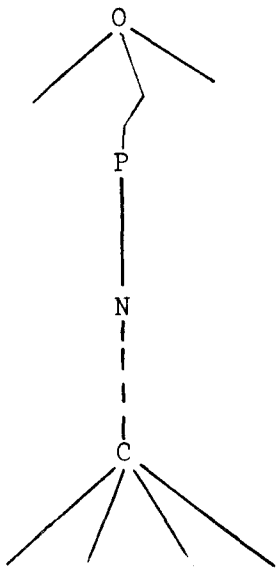
On a : - F_N est déterminée par rapport à F_P, i.e., $F_N \subset F_P$ et il existe une relation (fonction géométrique f) telle que $F_N = f(F_P)$.

- F_C est déterminée par rapport à F_N avec $F_C \subset F_N$ et F_C = identité (F_N).

Par composition on a F_C = identité ($f(F_P)$) = $f(F_P)$ et $F_C \subset F_P$.

IV.1.2.2. Fonctions d'accès

Reprenons le schéma précédent :



N nomme C

Les primitives d'accès étudiées au chapitre III.3 exploitent uniquement les liens arborescents et non les liens par nommage. En particulier, la désignation d'un point de l'image de N fournit N, car N est une feuille vis-à-vis de O.

On ne peut pas accéder à C en venant de O par les primitives normales ; de même, lorsque l'on a accès à C, on ne peut revenir à O par les primitives normales : l'accès à l'objet catalogué C se fait indépendamment de tout objet appelant O. Pour réaliser l'accès à C, l'utilisateur doit définir une clôture pour C, puis activer la fonction "DEFINITION CONTEXTE" et enfin accéder à C (paramètre de la fonction),

- Soit en donnant son nom.

- Soit en utilisant la fonction "APPEL-IMAGE" sur le noeud (forcément le noeud courant) du type "appel objet" de C.

Il ne reste plus qu'à valider la fonction "DEFINITION CONTEXTE".

IV.1.2.3. Mécanismes d'utilisation des objets catalogués

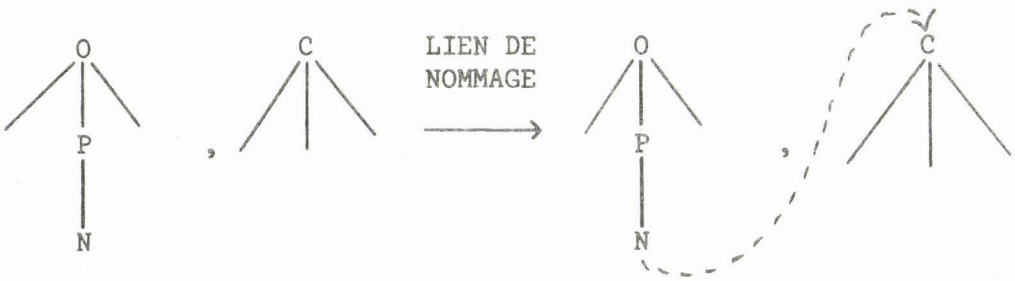
Soit un objet O dont on sait qu'un élément a déjà été décrit et stocké : c'est l'objet catalogué C. Pour construire l'objet O, plusieurs stratégies sont possibles :

1/ Copie de C dans O : C existe toujours en tant qu'objet catalogué, mais une copie est intégrée dans O ; Cette copie C' est alors une sous-arborescence de O et il n'y a pas de lien de nommage avec C'. On dit que C' est le dépliage de C dans O (au sens de la théorie des arbres).

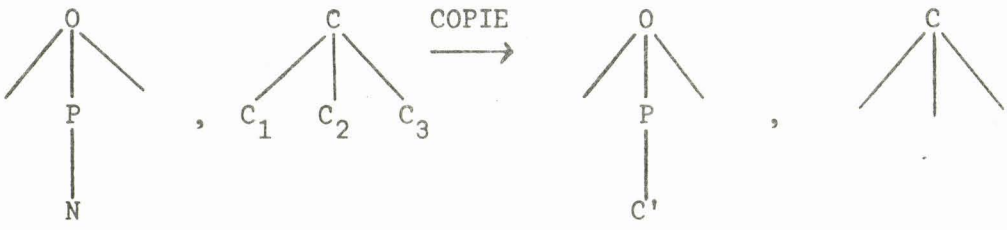
2/ Intégration de C dans O : C n'existe plus en tant qu'objet catalogué, mais uniquement en tant que sous-arborescence de O. Il n'y a pas de lien de nommage avec C, donc pas de noeud "appel-objet" pour C ; C est déplié dans O. Si d'autres objets appelaient C, l'intégration est telle qu'ils appellent alors un objet qui n'existe plus en tant que tel. L'intégration est alors équivalente à un ajout. Ce mécanisme est assez dangereux car très restrictif. Notons que toute intégration peut être réalisée par une copie puis une suppression ; Cette dernière devant de toutes façons exister (gestion de la base de données), il est préférable de limiter le nombre de fonctions dangereuses ; l'intégration ne devrait donc pas exister.

3/ Etablissement d'un lien de nommage par la création d'un objet N de type "appel-objet" pour C ; il n'y a pas de dépliage.

L'utilisation d'un objet catalogué C se fait par la création préalable du lien de nommage ; par la fonction "LIEN DE NOMMAGE", C est relié à N, et N devient le noeud courant. La fonction "COPIE" peut être reportée à un moment ultérieur quelconque et réalise la copie C' de C sur son lien de nommage ; la racine de C' remplace N, et devient le noeud courant. Nous étudions les conséquences de ces mécanismes après les avoir schématisés dans la figure IV.2.



le lien de nommage est indiqué
en pointillés.



appel objet C

C = C' mais
nom (C) ≠ nom (C')

Figure IV.2. : Illustration des mécanismes d'utilisation des objets catalogués.

Par la copie, l'objet O n'a plus de noeud "appel objet" ; C'est une simple arborescence. On peut accéder à C' et à ses composants par les primitives définies en III.3.

Toute modification de C' ne peut évidemment se répercuter que dans l'arborescence O qui contient C'. C est resté indépendant de O, donc une modification de C après avoir copié C (en C') dans O n'a aucun effet sur O ; en particulier, la suppression de C n'a aucun effet sur les copies déjà existantes de C.

Par le lien de nommage, la description de C reste extérieure à O, mais de nombreux objets peuvent utiliser C de cette façon. Dans ce cas, toute modification dans C se répercute sur l'ensemble des objets qui le nomment.

Remarque :

Lorsque l'on désire modifier, pour une seule arborescence A, un noeud auquel on accède par lien de nommage et qui n'est autre que la racine de l'objet nommé, sans modifier toutes les arbres arborescences qui l'utilisent par nommage, il faut copier l'objet catalogué correspondant dans cette arborescence A. Si le noeud à modifier est la racine de l'objet catalogué, la modification est faite sur le noeud "appel objet" lui-même, la copie n'est alors pas nécessaire.

IV.1.2.4. Conclusion sur les mécanismes d'utilisation des objets catalogués

La copie (suivie éventuellement d'une suppression) est un mécanisme qui permet de construire des arborescences sans lien de nommage, tout en utilisant des objets catalogués. Alors que l'intégration (copie puis suppression) est une opération dangereuse pour l'utilisateur, la copie est une façon sûre et pratique, mais redondante, d'utiliser les objets catalogués. Elle se justifie par la possibilité d'apporter des modifications ponctuelles et distinctes dans les différentes copies. C'est l'utilisation la plus encombrante en espace mémoire.

L'établissement de lien de nommage est une méthode beaucoup plus concise que la copie. C'est la seule méthode qui conserve des noeuds de type "appel-objet". Ces derniers étant considérés comme des feuilles, ce mécanisme permet de séparer les niveaux en groupes logiques dans lesquels l'utilisateur désire travailler. L'intérêt essentiel des liens de nommages provient de ce que la modification d'un objet catalogué entraîne la modification de tous les objets qui l'appellent. L'utilisateur ne risque donc pas d'oublier de faire les mises à jour sur ces objets "appelants", alors qu'il doit réaliser, sauf indication contraire, ces mises à jour sur tous les objets qui utilisent une copie de l'objet catalogué.

IV.2. LES OBJETS-MOTIFS

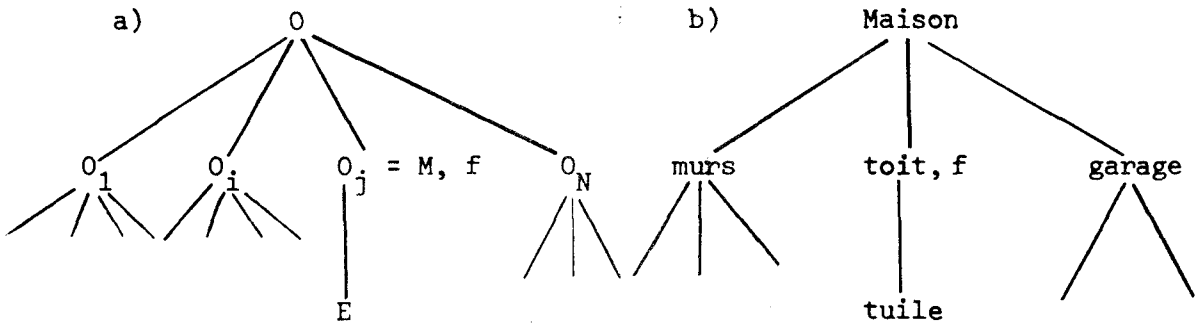
Le système graphique ainsi conçu est extrêmement contraignant lorsque l'utilisateur désire utiliser un objet-image de façon répétitive : Il est obligé de définir chaque exemplaire de cet objet-image. Par exemple, prenons la description de la toiture d'une maison. Cette toiture peut être considérée comme une **feuille**, mais si l'on veut en faire un toit de tuiles, il faut, par exemple, appeler chaque tuile pour lui attribuer un emplacement, l'objet-image tuile étant un objet-image catalogué. Cette démarche est très lourde et fastidieuse. Il semble tout-à-fait nécessaire que l'éditeur d'images fournisse les moyens de générer simplement des exemplaires répétés d'objets-images (les tuiles) qui soient contenus dans une feuille d'arborescence (la toiture). Les objets-motifs sont des objets-images un peu particulier qui visent à apporter une solution simple à ce genre de problème.

IV.2.1. Définition

Un objet-motif est une feuille d'arborescence dotée de propriétés supplémentaires : il est visualisé par une tache - en particulier cet objet a une forme déterminée - et l'intérieur de la tache contient les images des différents exemplaires d'un objet-image quelconque appelé *élément de motif de base*. Ces exemplaires sont appelés les *éléments du motif* et sont générés automatiquement par une *fonction d'occurrence*. Le motif est donc perçu comme une feuille "structurée" représentée sur l'écran de visualisation par une tache "composée". Si le calcul d'une occurrence particulière de l'élément de motif de base fournit un exemplaire non complètement inclus dans la tache du motif, il y a coupage ("clipping") de cet élément de motif par le contour de la tache du motif ; Seule sa partie intérieure au motif est visualisée : On dit que les images des éléments du motif remplissent la tache de ce motif.

La fonction d'occurrence est attachée au motif et non pas aux éléments du motif ; elle modifie les attributs graphiques ou géométriques propres à chaque élément du motif constituant le motif.

Une illustration de la structure d'un objet-motif est donnée par le schéma IV.3.



a) O_j est le motif M ayant l'élément de motif de base E et la fonction d'occurrence f.

b) Exemple concret du motif "toit" et de l'élément de motif de base "tuile".

Figure IV.3. : Exemple d'objets utilisant un motif.

IV.2.2. Création d'un motif

La fabrication d'un objet-motif est très simple : L'utilisateur décrit son motif comme une feuille "ordinaire" et achève sa description par la définition de l'élément de base du motif et de la fonction d'occurrence. Il n'a qu'un seul élément de motif à décrire, ce qui supprime l'aspect fastidieux et désagréable des descriptions répétitives et parfois longues d'objets.

IV.2.2.1. Propriétés de la fonction d'occurrence

1/ Elle peut modifier des attributs graphiques ou géométriques des exemplaires qu'elle génère. Elle peut donc se représenter au niveau du noeud motif exactement de la même façon que les transformations graphiques ou géométriques du noeud motif par rapport à son noeud père, à la différence près que cette fonction agit sur les éléments du motif dans le motif.

2/ La fonction d'occurrence doit être bornée, i.e., le nombre d'exemplaires d'éléments de motifs doit être limité, cette limite doit être fournie par l'utilisateur.

Remarques :

* Le cas particulier où f = identité, nombre occurrences = 1, et où l'élément de motif est identique à la feuille que constitue le motif (même forme, même couleur, même taille, même fenêtre) est tel que le motif est, dans ce cas, réduit à une feuille ordinaire d'arborescence, visualisée par une tache "simple" : les motifs sont bien définis de façon cohérente vis-à-vis des feuilles.

* Si f = identité, nombre occurrences = 1, et l'élément de motif de base est différent de la feuille, cet élément peut (éventuellement) être tronqué par la forme du motif lui-même ; une application serait de voir un paysage dans un cadre.

IV.2.2.2. Nature de l'élément de motif de base. Conséquences

Les informations I de l'élément de base du motif E relatives au motif M dépendent de M et sont donc indiquées dans M et non dans E. Le noeud motif M a un type particulier "objet-motif" lui permettant de contenir, les informations propres au motif, alors que l'élément de base E est un objet-image quelconque : une sous-arborescence, un objet-image catalogué, un autre objet-motif.

Si l'élément de base E du motif M est une sous-arborescence, on dit que le lien entre M et E est un *lien d'accès à l'élément de motif*. Si E est un objet-image catalogué, le lien entre M et E est un *lien de nommage de l'élément de motif*.

Si E est un motif M', on dit que M' est imbriqué dans M : Cette notion d'imbrication de motifs est très intéressante, car elle permet de simplifier les fonctions d'occurrences. Par exemple le motif "mur de briques" et la fonction d'occurrence "superposition décalée d'une demi-brique" avec une brique comme élément de base, peut se décomposer de façon plus simple en : "mur de briques" et la fonction d'occurrence, "superposition décalée d'une demi-brique" avec "ligne de briques" comme élément de base, où "ligne de briques" est un motif de fonction d'occurrence "alignement" avec une brique comme élément de base. Il y a ajout d'un noeud (ligne de briques), mais deux fonctions d'occurrence simples au lieu d'une compliquée.

Il est également intéressant de remarquer qu'un objet-motif, comme n'importe quel objet-image, peut être catalogué, par exemple le motif "toit de tuiles" peut être catalogué. L'éditeur d'image est donc puissant aussi bien au niveau de la facilité d'emploi, qu'à celui du souci constant de minimisation de l'encombrement des données graphiques en mémoire, évitant un allongement important des temps de calculs et d'accès aux objets (au moins dans le cas de fonctions d'occurrences simples, par exemple, translations, variations de couleur...).

IV.2.2.3. Conséquences sur la notion de fenêtre

1/ La fenêtre d'un objet-motif est la fenêtre de cette feuille d'arborescence.

2/ La fenêtre F_E de l'élément de motif de base E est déterminée en fonction de la fenêtre F_M de l'objet-motif M. F_E doit être incluse dans F_M (emboîtement des fenêtres). La fenêtre F_O d'une occurrence O de E est calculée à la génération de cette occurrence ; cette fenêtre (F_O) doit être éventuellement coupée par la fenêtre du motif. La fenêtre de l'occurrence visualisable est donc $F_O \cap F_M$. F_M détermine certaines limites pour M et le coupage des fenêtres des éléments du motif sur la fenêtre de M précède et restreint l'étude du coupage de O dans M à l'intérieur de $F_O \cap F_M$.

IV.2.3. Les fonctions d'accès

L'utilisateur doit concevoir l'objet-motif M comme une feuille dont l'élément de motif de base E est un élément générique. Il ne peut donc pas accéder à ce dernier par "↓" et il ne peut désigner par ● un élément dans le motif. Par ⊕, il ne peut désigner que M. Un accès particulier doit être fourni pour permettre à l'utilisateur de travailler sur l'élément de base du motif E en tant qu'objet-image. Deux fonctions permettent d'accéder à E : la fonction "ACCES ELT MOTIF" fournit l'élément de base E si E n'est pas catalogué ; la fonction "APPEL ELT MOTIF" fournit l'élément de base E s'il est catalogué. Cet élément est soit visualisé à l'emplacement que l'utilisateur lui a attribué sur l'écran (par définition d'une clôture suivie de la fonction "DEFINITION CONTEXTE" puis l'accès à l'élément de motif), soit à l'emplacement qu'il occupe dans le motif.

Remarque :

Afin d'éviter toute confusion, l'utilisateur aura tout intérêt à visualiser E dans une clôture disjointe de celle qui contient M, d'autant que toute modification de E se répercute automatiquement dans M, et que l'élément de motif de base n'est pas forcément visible dans le motif (s'il est caché par les différents exemplaires).

IV.2.4. Comparaison entre les objets-images catalogués et les objets-motifs

1/ Le noeud N qui appelle un objet-image catalogué C et le noeud M motif relié à un élément de motif E sont tous deux des noeuds terminaux de leur arborescence. N est considéré comme une feuille (ayant une forme) dont le remplissage est généré à partir d'un élément de base et d'une fonction d'occurrence.

2/ Il est clair que tout objet-motif peut être catalogué, mais la réciproque n'est pas vraie.

3/ Les liens qui unissent N à C et M à E ne permettent pas d'accéder à C (ou à E) par la fonction "↓".

pour accéder à C, il faut utiliser la fonction "APPEL-IMAGE",
pour accéder à E, il faut utiliser "APPEL ELT-MOTIF" si E est catalogué,
"ACCES ELT-MOTIF" si E n'est pas catalogué.

4/ L'utilisation des motifs et des objets catalogués est souple : On peut composer des motifs entre-eux, on peut utiliser des objets catalogués pour décrire un motif et des motifs dans la description des objets catalogués. On peut également référencer des objets catalogués dans la description d'autres objets catalogués. Nous verrons en IV.3 les conséquences de cette utilisation dans un cas particulier.

IV.2.5. Conclusion sur les objets-motifs

Les objets-motifs ne modifient pas les notions de fenêtre et de contexte déterminées pour les objets-images. Ils apparaîssent comme des feuilles structurées, et se prêtent tout-à-fait à l'ensemble des manipulations faites sur les objets-images, moyennant quelques restrictions au niveau des accès pour conserver un système cohérent et orthogonal. Nous allons maintenant étudier une utilisation très particulière des objets-images et des objets-motifs : ce sont les objets-images récursifs.

IV.3. LES OBJETS-IMAGES RECURSIFS

IV.3.1. Définition et exemple

Un objet récursif est un objet qui s'utilise lui-même dans sa définition. Quand l'objet est un objet-image, c'est un objet-image récursif ;

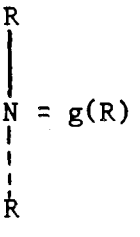
exemples d'images récursives ne sont pas rares : 'image du photographe se photographiant devant un miroir qui réfléchit l'image du photographe en train de se photographier etc..., la boîte dont le couvercle représente une vache ayant des boucles d'oreilles identiques à cette boîte et dont les couvercles...

IV.3.1.1. Description d'un objet-image récursif de l'éditeur graphique

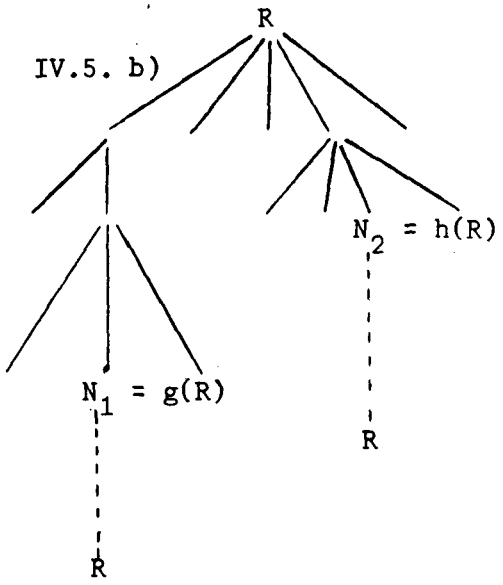
Un objet-image récursif R est un objet-image contenant éventuellement des objets-motifs, tel qu'à un endroit de sa description un noeud N fait référence à la racine de l'objet-image. N nomme R, qui est donc catalogué. Si N est décrit de la même façon que R la copie de R au noeud N, cesse la récursivité ; le nombre de dépliages ne peut être infini. De plus, la copie de R en N devient modifiable indépendamment de l'original. Il semble donc assez naturel de considérer un objet-image récursif R comme un objet-image catalogué se nommant lui-même directement ou indirectement, car le nommage permet de ne pas déplier la description de R, tout au moins pour sa mémorisation. Il est clair que le nommage ne permet d'accéder qu'au premier niveau de la récursion.

Des exemples d'objets-images récursifs simples sont donnés dans la figure IV.5.

IV.5. a)



L'objet-image récursif R est référencé dans N. N contient toutes les indications nécessaires à l'appel de R.



L'objet-image récursif peut être référencé de diverses façons dans sa description N_1 et N_2 contiennent toutes les indications nécessaires à l'appel de R.

Les liens de nommages sont indiqués en pointillés.

Figure IV.5. : Récursivité simple.

a) Un objet-image récursif minimum.

b) Exemple d'objet-image récursif.

Il apparait ainsi que la possibilité de créer des objets récursifs est une conséquence de l'"appel" d'objet catalogué ; il serait possible de contrôler les objets récursifs directs (l'objet s'appelle lui-même), et d'imposer sur eux des contraintes permettant d'assurer leur visualisation. Cependant, la récursivité croisée est plus difficile à maîtriser pour le système. Un exemple de récursivité croisée est donné dans la figure IV.6.

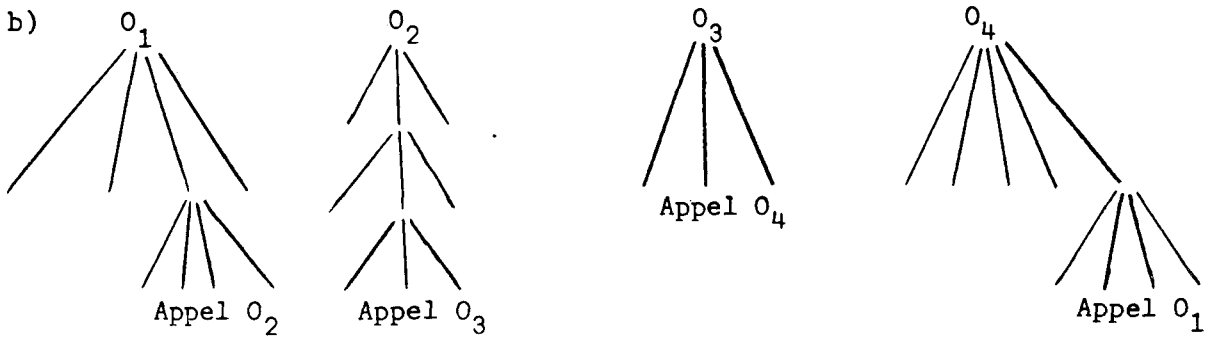


Figure IV.6. : Exemple de récursivité croisée.

a) A un niveau d'appel.

b) A trois niveaux d'appel.

Le nombre de niveaux d'appel est déterminé par l'utilisateur, ce dernier doit être vigilant afin de maîtriser lui-même la récursivité de ces objets.

IV.3.1.2. Les problèmes de description

Le premier est celui de la paramétrisation de l'objet-image récursif R : Au premier niveau de la récursion, le noeud N "appel objet-image" de R doit contenir tous les paramètres nécessaires à la visualisation du premier niveau appelé, ce dernier contient un noeud "appel objet-image" de R dont les paramètres n'ont évidemment pas pu être définis différemment au premier niveau de R. La question qui se pose est : Comment paramétrer ? et même : Comment paramétrer de façon à rendre facile la mise au point de l'objet-image ?

Les autres problèmes sont liés à la récursivité même des objets et en ce sens, sont assez classiques :

* Comment faut-il décrire l'objet-image de façon à pouvoir le visualiser ?

* Quand faut-il arrêter la description pour la visualisation de l'objet-image ?

* Quels sont les moyens d'accès à un niveau de description donné ? Utilité de ce genre d'accès...

* Comment donner à l'utilisateur les moyens de contrôler la récursivité simple (cf. figure IV.5) et la récursivité croisée (cf. figure IV.6) ?

* Comment limiter les temps de calculs pour la visualisation ?

IV.3.1.3. Contraintes de limitation des temps de calcul

Les contraintes proviennent essentiellement de ce qu'il faut limiter les temps de calculs pour la visualisation, celle-ci constituant le problème majeur posé par la récursivité.

Une première idée pour contrôler le temps de calcul consiste à imposer l'inclusion stricte des fenêtres des noeuds "appel objet" par rapport à la fenêtre de la racine de l'objet. Les fenêtres sont ainsi de plus en plus petites. Une taille de fenêtre inférieure au pixel arrête alors la récursion. Cette contrainte limite ainsi le temps de calcul pour la visualisation des objets récursifs simples. Elle ne peut être généralisée à l'ensemble des objets appelés, car c'est une condition trop contraignante. Une solution consiste à borner le temps de calcul par un "chien de garde" ou un mécanisme adapté ; en cas de non-respect, les calculs de visualisation de l'objet sont interrompus, l'objet-image étant considéré comme "non visualisable".

C'est à l'utilisateur de prendre garde à l'inclusion stricte des fenêtres, le contrôle n'est pas assuré par le système.

IV.3.2. Utilisation. Résolution des problèmes dûs à la récursivité

1/ Les objets récursifs sont des objets catalogués nommés. On ne peut donc pas accéder aux différents niveaux de récursivité de ces objets. Les niveaux, pris indépendamment les uns des autres sont intrinsèquement identiques, ce n'est donc pas gênant. Que la récursivité soit simple ou croisée, on ne peut modifier que le premier niveau des objets récursifs.

2/ Considérons la représentation des objets récursifs en mémoire : Nous avons vu que la copie (à fortiori l'intégration) supprime la récursivité (IV.3.1) ; en mémoire, les objets récursifs ne peuvent donc pas être dépliés (au sens de la théorie des arbres).

3/ Considérons la visualisation des objets récursifs. Il faut visualiser l'objet appelé R dans la fenêtre de l'objet appelant (c'est R dans le cas de la récursivité simple). Il faut alors intégrer R uniquement à ces fins de visualisation ; nous appellons cette intégration "intégration de visualisation", car elle n'enlève pas la récursivité, c'est un dépliage qui ne dépend que de la visualisation.

Plusieurs possibilités sont proposées à l'utilisateur :

* 1° Le système effectue automatiquement le dépliage des objets jusqu'à ce que la fenêtre de l'objet à visualiser soit trop petite. Il faut alors que l'utilisateur ait vérifié que les fenêtres sont strictement incluses. L'image obtenue est la plus détaillée que l'on puisse obtenir.

* 2° Le nombre de dépliages est fixé ; dans ce cas, le dernier niveau déplié de l'objet est représenté implicitement par la tache que constitue le rectangle de sa fenêtre : c'est un pavé. Cette méthode est simple à utiliser pour la récursivité simple mais elle résoud de façon arbitraire les problèmes de la récursivité croisée.

* 3° Le dépliage est "visuel" : Une procédure de choix s'applique à chaque fois qu'un objet-image est appelé ; Le choix consiste à demander la visualisation d'un niveau supplémentaire (i.e., un dépliage) ou l'arrêt des dépliages accompagné de la visualisation d'un pavé (ou d'aucune visualisation).

NOTA :

Ces possibilités peuvent être générales pour tout appel d'objet catalogué, qu'il soit intérieur à une récursivité ou non ; Ceci permettrait à l'utilisateur d'appréhender ce qui est effectivement dans l'objet-image proprement dit.

Ces possibilités sont implémentables de la façon suivante :

1° Lorsque la fenêtre d'un objet-image est réduite à un point de l'écran, le dépliage s'achève automatiquement : Le principe de dépliage est analogue au principe utilisé par Warnock (Newmann et Sproull) pour la détermination des surfaces visibles : Il est inutile de calculer des points s'ils ne sont pas visibles ; une facette est soit complètement visible, soit complètement invisible, soit indéterminée dans un certain espace rectangulaire de visualisation. Les premiers cas (triviaux) sont vite résolus, et la résolution du troisième cas se fait par dichotomie selon les axes de cet espace de visualisation. Si l'espace est réduit à un point, le processus s'arrête sur un point visible ou invisible (aléatoire ou pas). Ceci explique l'emboîtement strict des fenêtres.

2° Lorsque le nombre maximum de dépliages est atteint, le dépliage s'achève : Le nombre, donné par l'utilisateur, est automatiquement décrémenté de un à chaque appel d'objet dans un même niveau de dépliage. Ce nombre est modifiable aisément par l'utilisateur.

3° Une procédure de choix propose le dépliage : L'arrêt ou la continuation est directement commandé par l'utilisateur à chaque appel d'objet au lieu d'être commandé automatiquement ① et ②.

4° L'utilisation des objets-images récursifs est une conséquence des "appel objet" il n'y a donc pas de distinction à apporter pour les objets récursifs : ce sont des cas particuliers d'objets catalogués.

Remarque :

Lorsqu'il y a inclusion stricte des fenêtres, la différence de taille des fenêtres doit être supérieure à la précision sur les nombres exprimant ces tailles (que ce soient des entiers ou des réels).

IV.4. CONCLUSION

Les mécanismes proposés facilitent et écourtent les descriptions longues et ennuyeuses d'objets-images. Tout objet-image, une fois catalogué peut être nommé dans la description d'objets-images plus complexes ce qui rend très modulaire la conception d'images. De même, les objets répétitifs et les objets récursifs sont traités simplement, de façon très concise pour l'utilisateur interactif. La lacune principale commune à ces objets est l'incapacité de créer et d'utiliser des objets "flous" qui serviraient de modèles pour élaborer des familles d'objets-images ressemblants, mais tous différents. De tels objets étendent considérablement la puissance de l'éditeur ; les objets-images, objets-motifs, objets-images récursifs et les

objets-images catalogués ne forment qu'une petite partie "complètement définie" de ces objets "flous", partiellement définis. Nous étudions au chapitre V cette extension du système qu'engendrent ces objets appelés "objets patrons".

CHAPITRE V

LES OBJETS PATRONS

CHAPITRE V

LES OBJETS PATRONS

V.1. DEFINITION ET EXEMPLES D'UTILISATION

Selon la définition du dictionnaire, un patron est "un modèle en tissu ou en papier fort à partir duquel on taille un vêtement". Pour l'éditeur graphique, c'est véritablement un modèle pour la construction des objets-images ; un objet-patron est perçu comme un "objet-image incomplètement spécifié : "c'est un objet graphique de même nature qu'un objet-image (ou un objet-motif), mais son degré de spécification est quelconque : l'objet-patron sert d'ébauche de construction pour différents objets graphiques. Les objets-images construits autour d'un patron ne peuvent différer que par les informations non spécifiées dans le patron, ce qui permet d'obtenir des familles d'objets-images.

Il est important de noter qu'un objet-image en cours de modification ne peut pas être considéré comme un objet-patron, bien qu'il soit temporairement incomplètement spécifié : Le type "objet-patron" est donné par l'utilisateur, il signifie que l'objet est un modèle. On ne peut utiliser un objet comme modèle lorsqu'il est en phase de modification, car on ne pourrait pas savoir si l'objet identifié est l'objet avant ou après la prise en compte de cette modification.

De nombreuses applications sont fondées sur ce principe de modèle ; prenons quelques exemples d'applications : une lampe à incandescence est toujours constituée d'un support, de contacts métalliques, d'un filament incandescent et d'un verre. Pour un architecte, la conception d'un F2 doit respecter certaines normes : une cuisine, une salle de séjour, une chambre et une salle de bains doivent être prévues ; la forme, la disposition, la taille de ces éléments dépendent de l'objet que l'on veut construire.

V.2. PRINCIPE D'UTILISATION. LES PROBLEMES POSES

Créer une famille d'objets-images à partir d'un patron P, c'est donner à ces objets-images toutes les caractéristiques de P et compléter leur description indépendamment les uns des autres. Les informations de P doivent être attribuées facilement aux objets-images considérés, et la définition des informations manquantes dans P nécessite un dialogue simple et exhaustif, condition sine qua non d'utilisation des patrons.

Le problème majeur qui se pose pour l'emploi des patrons est qu'ils sont, par définition, incomplètement spécifiés, et éventuellement non visualisables. Nous proposons en V.4 une représentation sur l'écran des patrons, qui concilie les manipulations sur les patrons et leur utilisation pour la description d'objets-images.

Regardons maintenant la démarche précise de l'utilisateur qui crée un objet-image O, et désire utiliser un patron P. Pour fixer les idées, O est un immeuble et P est la description d'un type d'appartement. "A un noeud N de l'arborescence O (i.e., à un certain endroit de l'immeuble), je désire utiliser le patron P (i.e., mettre un appartement ayant les caractéristiques de P)" ; il reste à compléter N.

Cette démarche est exactement assimilable à l'utilisation d'un objet-image catalogué dans la description d'une arborescence.

Conséquence :

Seuls les objets-patrons catalogués sont utilisables pour définir des objets-images. Nous étudions quels sont les mécanismes adaptés lorsque l'on utilise des objets-patrons. Nous allons considérer tout d'abord la description d'un objet-image à l'aide d'un objet-patron, puis la description d'un objet-patron à l'aide d'un autre objet-patron.

V.3. LES MECANISMES D'UTILISATION DES OBJETS-PATRONS

V.3.1. Description d'un objet-image à l'aide d'un objet-patron

Les mécanismes définis par les objets-images catalogués sont le nommage, l'intégration et la copie. Etudions les pour ces objets incomplètement spécifiés que sont les objets-patrons. Considérons le noeud N que l'on veut décrire à l'aide d'un patron P.

* Le nommage nécessite que toutes les informations marquantes de P soient définies dans N ; ces informations portant sur le patron tout entier (et non pas sur la racine seule, comme dans le cas des objets-images catalogués), il n'est pas pensable d'accepter ce mécanisme.

* L'intégration de P suivie de la spécification de toutes les informations indéfinies de P évite la description de P ; N est décrit à l'aide de P, mais P n'existe plus en tant qu'objet-patron (cf. IV.1.2) ; Lorsque N est complètement défini, N est un objet-image ordinaire : N a perdu tout lien avec l'objet-patron qu'il a utilisé ; la description des champs indéfinis correspond à la description des noeuds temporairement indéfinis par suite d'un clivage de l'unique descendant d'un noeud que l'utilisateur désire conserver (cf. III.5.2).

* La copie P' de P suivie de la spécification des informations manquantes dans P' a le même effet quant à la description de N, mais l'objet-patron P est conservé. Le noeud N complètement défini est un objet-image ordinaire, indépendant de P.

Un objet-patron devant - normalement - servir à la description de plusieurs objets-images, son utilisation se fait implicitement et automatiquement par copie dans l'objet-image.

V.3.2. Description d'un objet-patron à l'aide d'un autre objet-patron

C'est la généralisation aux objets incomplètement spécifiés des mécanismes étudiés sur les objets-images catalogués.

Un objet O est incomplètement spécifié dès qu'un des éléments qui le composent l'est ; a fortiori, O peut faire référence à un objet incomplètement spécifié. L'intégration, la copie et le nommage peuvent donc être utilisés indifféremment pour définir aisément des objets-patrons plus complexes.

V.3.3. Conséquences

1/ Le mécanisme de nommage nécessite une attention particulière : il pourrait changer la nature de l'objet appelant (transformer un objet-image en un objet-patron). L'appel d'un objet-patron est donc distingué de l'appel d'un objet-image : le noeud appelant un objet-patron est du type "appel objet-patron". Un tel noeud ne peut se trouver dans un objet-image.

2/ Lorsque l'on construit l'objet-image O à l'aide de l'objet-patron P, tout noeud N de P de type "appel objet-patron" P' est soit remplacé automatiquement par la copie de P', celle-ci devra être complétée pour achever la description de O, soit N, de type "appel objet-patron" P', est remplacé par un "appel objet-image" O'.

V.4. LES MANIPULATIONS SUR LES OBJETS-PATRONS

Un objet-patron ne se distingue d'un objet-image que par sa spécification incomplète et son rôle de modèle. Les différences, peu nombreuses, sont :

- La définition du type "objet-patron" par rapport à celui d'objet-image.

- L'accès à ce type d'objet ; on accède à un objet-patron par la fonction "appel patron". Tout noeud de type "appel objet-patron" est équivalent à un noeud "appel objet" et il en a donc toutes les caractéristiques (cf. IV.1).

- La visualisation ; Le mécanisme proposé est la visualisation schématique (sous une forme arborescente) de la structure de l'objet-patron doublée des images des noeuds visualisables, s'ils existent. Les primitives d'accès sont inchangées et le noeud courant est mis en évidence dans le schéma de la structure et, si c'est possible, dans ce qui est visualisé du patron. De même, le noeud contexte est le noeud dont la structure est schématisée sur l'écran.

Il s'agit bien d'une généralisation des notions et des fonctions attachées aux objets-images. Mises à part les quelques différences citées plus haut, les principes utilisés dans le système restent les mêmes (même notion de fenêtre, de création, d'ajout, de suppression, d'accès au sein des objets-patrons) ; ils assurent le maintien de la cohérence des types d'objets manipulés.

V.5. LES OBJETS-PATRONS RECURSIFS

V.5.1. Définition

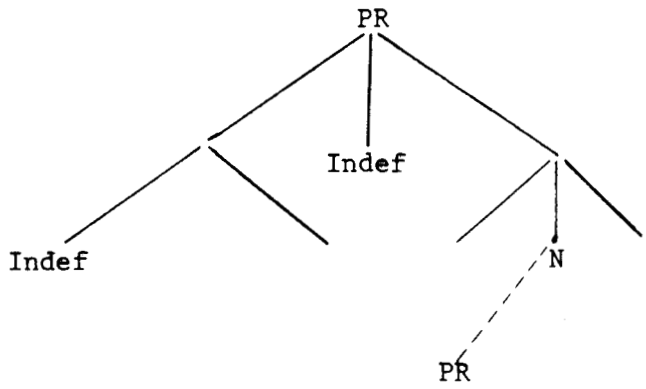
Tout comme les objets-images récursifs sont une conséquence de l'"appel objet-image" (cf. IV.3), les objets-patrons récursifs sont une conséquence de l'"appel objet-patron". Les objets-patrons récursifs servent à définir les objets-images récursifs. On en déduit que tout objet-patron récursif PR est tel que :

* PR est catalogué.

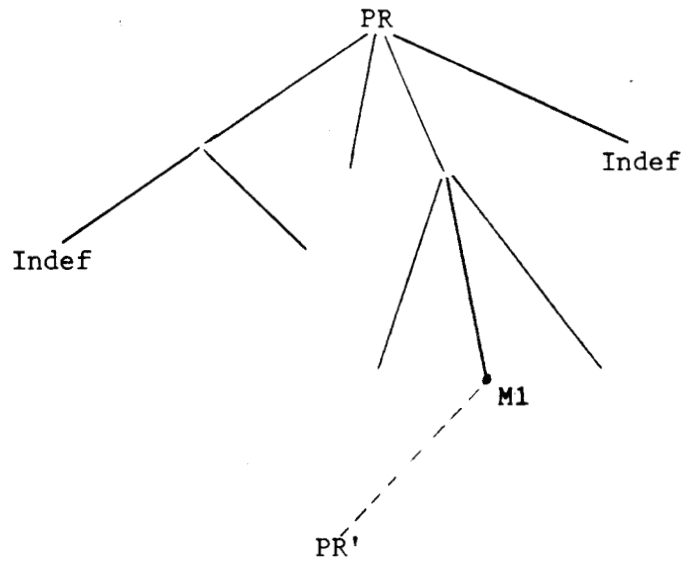
* PR doit contenir au moins un noeud N de type "Appel objet-patron". Si N est un "Appel objet-patron" PR, l'objet-image qui sera construit à l'aide de PR sera un objet récursif simple, dans les autres cas, on ne peut pas aisément détecter la récursivité croisée.

La figure V.1 représente un exemple de structure d'un objet-patron récursif simple (a) et d'objets-patrons fournissant des objets-images soumis à la récursivité croisée.

V.1.a.



V.1.b.



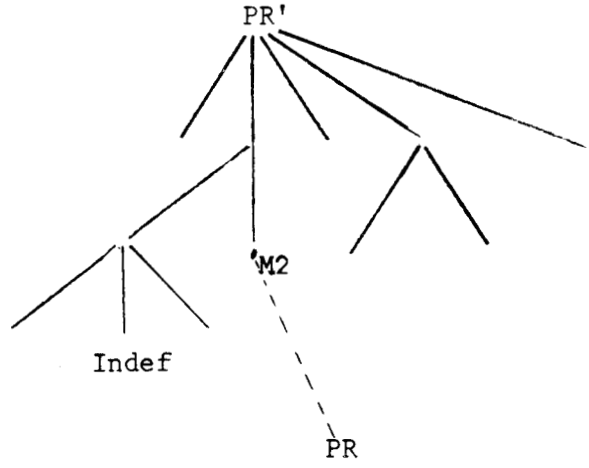


Figure V.1. : Exemples de patrons permettant de construire des objets-images récursifs (a) simple, (b) croisés.

V.5.2. Utilisation d'un objet-patron récursif

V.5.2.1. Visualisation

La visualisation d'un objet-patron récursif est réalisée comme celle de tout objet-patron, le premier niveau de récursivité peut être manipulé et visualisé de la façon décrite en (V.4). Les autres niveaux de l'objet-patron récursif pourraient être visualisés (toujours au sens de V.4) par des demandes de dépliages aux noeuds "appel objet-patron" (cf. IV.3.2), mais cela ne présente guère d'intérêt ; on ne peut l'interdire, pour des raisons de complexité en cas de récursivité croisée au niveau des objets-patrons (cf. IV.3.1).

V.5.2.2. Mécanismes d'utilisation des objets-patrons récursifs

- La description d'un objet-patron récursif est identique à celle d'un objet-patron non récursif (V.3.2). Cependant, un objet-patron ne peut servir à décrire un objet-image récursif que s'il y a nommage.

- La description d'un objet-image O à partir d'un objet-patron récursif PR fournit un objet-image récursif dès que l'utilisateur choisit l'option de remplacer l'"appel objet-patron" PR par un "appel objet-image" O, ceci ne pouvant être réalisé qu'après la détermination des éléments encore indéfinis du patron.

Nous avons vu que la copie supprimait la récursivité, l'option de nommage est donc la seule possible qui permette de décrire des objets-images récursifs à l'aide d'objets-patrons. Le nommage étant généralisé à l'ensemble des objets-patrons, on peut obtenir des objets-images récursifs simples ou croisés ; Ainsi, suivant le désir de l'utilisateur, à partir d'objets-patrons récursifs, il peut fabriquer des objets-images récursifs (simples ou croisés) ou non.

Il est clair que pour obtenir un objet-image O récursif, l'utilisateur doit définir les paramètres de l'appel objet-image O ; Le noeud "appel objet-patron" étant remplacé par l'"appel objet-image", l'objet-image récursif O construit à l'aide de l'objet-patron récursif PR est indépendant de PR. En particulier, PR peut être réutilisé.

V.6. RECAPITULATION SUR LES DIFFERENTS TYPES D'OBJETS

Les objets du système sont tous appelés des "objets-graphiques", et forment deux classes :

* Les objets complètement spécifiés (OCS), qui sont toujours visualisables en images : Ce sont les objets-images auxquels sont assimilés les objets-images catalogués, les objets-motifs et les objets-images récursifs.

* Les objets incomplètement spécifiés (OIS), qui peuvent être de même type que les précédents.

Les liens qui unissent les différents noeuds sont :

- Soit des liens de structures arborescentes.

- Soit des liens de nommage, tous les objets graphiques pouvant être catalogués.

Nous indiquons sous forme d'un tableau, selon la nature des objets-images, les liens qui peuvent unir deux noeuds père et fils, de façon à ne pas modifier la nature du noeud père.

Père / Fils	OCS	OIS
OCS	A,N	A,N
OIS	A suivi des paramètres de OIS pour le compléter	A,N

A : Lien de structure arborescente.

N : Lien de nommage.

On peut décrire de nouveaux objets-images en utilisant soit des objets-images déjà construits, soit des objets-patrons que l'on complète dans l'objet-image désiré. On peut décrire des objets-patrons en utilisant des objets-images ou des objets-patrons ; dans ce cas les liens sont quelconques.

Tous les noeuds qui établissent un lien de nommage sont de type "appel objet", c'est-à-dire :

- soit "appel objet-image" pour un objet-image ordinaire,
- soit "appel élément de motif" pour relier l'élément de motif au motif,
- Soit "appel objet-patron" pour décrire un patron en utilisant un autre objet-patron appelé par son nom,

et tout noeud qui "appelle" un objet est considéré comme une feuille vis-à-vis de ses ascendants. Les fonctions d'accès à ces objets appelés sont respectivement "APPEL-IMAGE", "APPEL ELT MOTIF" et "APPEL PATRON", ce dernier remplace les deux autres "APPELS" quand on travaille sur des objets-patrons.

Remarque :

L'accès à un élément de motif non relié par nommage est réalisé par la fonction "ACCES ELT MOTIF" (cf. IV.2), les fonctions "APPELS" sont réservées aux accès par nommage.

La proposition de structure des objets est donnée au chapitre VI sous forme d'une grammaire de langage.

V.7. CONCLUSION

Les objets-patrons sont la généralisation "incomplètement spécifiée" de l'ensemble des objets du système. En particulier, un objet-patron complètement spécifié est un objet-image. L'aspect fonctionnel des objets-patrons est très riche en possibilités offertes à l'utilisateur. Cependant, un travail important subsiste en ce qui concerne la définition d'un dialogue simple pour l'utilisation des objets-patrons dans la création d'objets-images ; ce dialogue doit s'appuyer sur une visualisation adaptée aux objets-patrons utilisés.

De façon générale, le nombre de primitives de bases est assez restreint, et la plupart des fonctions d'accès n'est pas paramétrée, ce qui rend leur utilisation plus souple.

Les mécanismes d'élaboration d'objets correspondent aux structures logiques des objets ; ceci confère au système une puissance importante dans la simplicite de fonctionnement.

CHAPITRE VI

FORMALISATION

CHAPITRE VI

FORMALISATION

VI.1. INTRODUCTION

De nombreux problèmes cruciaux en informatique (fiabilité, correction des implémentations, modularité,...) sont tributaires des difficultés rencontrées pour décrire de façon à la fois rigoureuse et réaliste la sémantique des langages de programmation. De puissants outils ont été développés dans ce sens : la sémantique algébrique (Nivat [1], Kott [2], Boudol [3], Guessarian [4]), les types abstraits (Goguen et al. [5]), les attributs sémantiques (Engelfriet [6], Courcelle [8]). Exploitant ces idées, des logiciels ont été réalisés. Citons par exemple : Perluette (M.C. Gaudel) pour les types abstraits, Delta (Lohro) pour les attributs, PPSI (D. Begay) pour la sémantique algébrique.

Les transducteurs d'arbres (Dauchet [9], Guessarian [4], Engelfriet [7], Lilin [10]) sont également connus comme pouvant décrire la sémantique des langages de programmation (Dauchet, Engelfriet [7].

Nous utilisons ici cet outil pour définir la sémantique de notre langage d'édition d'images. Cette démarche va dans le sens du murissement des logiciels graphiques où, au delà d'un réel savoir faire, les outils de codification sont encore rares.

Les objets graphiques utilisés sont ceux décrits dans les chapitres précédents : des objets-images, des objets-patrons et des objets-motifs, ces derniers formant un type particulier d'objets-images. Afin de donner au lecteur une vue synthétique des caractéristiques de ces objets et des relations qui existent entre-eux, nous décrivons en VI.2, sous une forme assimilable aux grammaires de langages, la syntaxe de l'ensemble de ces objets graphiques.

Considérons un alphabet gradué Σ, i.e. tel que pour tout entier $k \geq 0$ un sous-ensemble Σ_k de Σ est défini tel que Σ_k est non vide seulement pour un nombre fini de valeurs de k. Un élément de Σ_k est dit de degré k, et peut être assimilé à un noeud d'arborescence ayant k descendants. L'opération de linéarisation des arbres associe à tout arbre construit sur Σ un mot du langage dénoté $T(\Sigma)$ sur l'alphabet $\Sigma \cup \{(,)\}$ défini par :

1 - si $\sigma \in \Sigma_0$ alors $\sigma \in T(\Sigma)$

2 - si $k \geq 1$, $\sigma \in \Sigma_k$ et $t_1, \ldots, t_k \in T_\Sigma$ alors $\sigma(t_1, \ldots, t_k) \in T(\Sigma)$

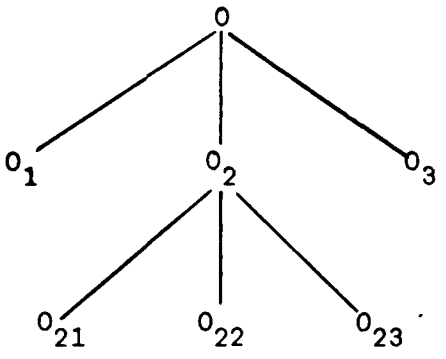
Pour pouvoir définir de façon un peu formelle des transformations qui sont à appliquer localement sur les arbres, deux types de notations s'avèrent nécessaires :

1° Une notation pour représenter une configuration locale d'arbres sans expliciter certains de ses sous-arbres, mais en les remplaçant par des variables de l'ensemble $X = \{x_1, x_2, x_3, \ldots\}$. En considérant les éléments de X comme étant de degré 0, une telle configuration locale avec p variables est donc représentée par un mot de $T(\Sigma \cup \{x_1, \ldots, x_p\})$ noté encore $T(\Sigma)_p$. Si $\alpha \in T(\Sigma)_p$ et $t_1, \ldots, t_p \in T(\Sigma)$ nous noterons $\alpha \cdot (t_1, \ldots, t_p)$ l'arbre $\in T(\Sigma)$ obtenu en remplaçant pour tout i, les occurrences de x_i dans α par t_i.

2° Une notation pour repérer certains noeuds de l'arbre où doit se faire la transformation. On repèrera un tel noeud n en "insérant" dans la linéarisation de l'arbre un noeud q entre n et son père. En considérant un alphabet gradué Q où chaque élément est de degré 1, un arbre pour lequel on repère certains noeuds sera représenté par un mot de $T(\Sigma \cup Q)$. Cette méthode permettra de repérer formellement par exemple le noeud contexte et le noeud courant tels qu'ils ont été définis au chapitre III.

Exemple :

Considérons l'arbre suivant :



qui se représente $0(0_1\ 0_2(0_{21}\ 0_{22}\ 0_{23})\ 0_3)$.

Si on suppose que 0_2 est noeud contexte et 0_{22} noeud courant, nous aurons $0(0_1\ \underline{ct}\ (0_2(0_{21}\ \underline{co}\ (0_{22})\ 0_{23})\ 0_3)$.

L'information locale que l'on peut isoler, pour définir une transformation par exemple, serait $0(x_1\ \underline{ct}\ (x_2)\ x_3)$ qui précise que le noeud contexte est le $2^{ème}$ parmi les trois descendants de 0, mais ne précise pas la structure de ces descendants eux-mêmes. L'arbre peut donc se noter également

$$0(x_1\ \underline{ct}\ (x_2)\ x_3) \cdot (0_1,\ 0_2(0_{21}\ \underline{co}\ (0_{22})\ 0_{23}),\ 0_3)$$

pour mettre en évidence cette structure locale.

Nous pouvons maintenant définir un transducteur d'arbre comme un 5-uple $M = <\Sigma, Q, G, \Gamma, R>$ où :

Σ est l'alphabet gradué des noeuds

Q est un alphabet gradué des repères (tous de degré 1)

G est un alphabet des états

Γ est un alphabet de pile

R est un ensemble fini de règles de la forme :

$$(y, \alpha, g) \rightarrow (z, \beta, g')$$

avec $y \in \Gamma$, $z \in \Gamma^*$,

$\alpha, \beta \in T(\Sigma \cup Q)_p$

$g, g' \in G$.

Intuitivement, une telle règle peut s'appliquer si

1) le sommet de pile est v,

2) l'arbre contient la configuration locale α,

3) le transducteur est dans l'état g.

Lorsqu'on l'applique, il s'ensuit une modification de la pile, et localement de l'arbre (remplacé par β) et le transducteur passe dans l'état g'. On peut interprêter l'état g comme une sorte de garde qui valide la règle.

Le transducteur induit une relation de $\Gamma^* \times T(\Sigma \cup Q) \times G$ dans $\Gamma^* \times T(\Sigma \cup Q) \times G$ notée $\Rightarrow$ telle que pour $u \in \Gamma^*$, $\varphi, \psi \in [\Sigma \cup Q \cup \{(,)\}]^*$, $t_1, \ldots, t_p \in T(\Sigma \cup Q)$ et $(y, \alpha, g) \rightarrow (z, \beta, g') \in R$ alors
$(u\, y, \varphi\, \alpha \cdot (t_1, \ldots, t_p)\, \psi, g) \Rightarrow (u\, z, \varphi\, \beta \cdot (t_1, \ldots, t_p)\, \psi, g')$.

On notera $\overset{*}{\Rightarrow}$ la fermeture transitive de cette relation.

Nous allons utiliser ce formalisme pour définir avec précision les fonctions étudiées dans le chapitre III. Certaines fonctions peuvent s'écrire par une seule règle. D'autres par contre demandent plusieurs règles qui doivent s'enchaîner dans un certain ordre, qui est contrôlé par l'ensemble des états G. Nous distinguerons un état particulier noté $\emptyset$ dans

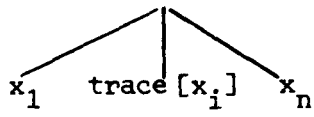
lequel sera le transducteur lorsqu'il est en attente d'une nouvelle fonction (état neutre). Ceci induit une relation de $\Gamma^* \times T(\Sigma \cup Q)$ dans $\Gamma^* \times T(\Sigma \cup Q)$:

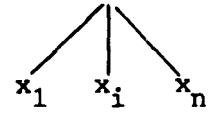
$$(v, t) \Rightarrow (v', t')$$

si $(v, t, \emptyset) \overset{*}{\Rightarrow} (v', t', \emptyset)$ sans passer par l'état $\emptyset$ lors d'une étape intermédiaire. C'est en fait cette relation qui a été définie informellement dans le chapitre III.

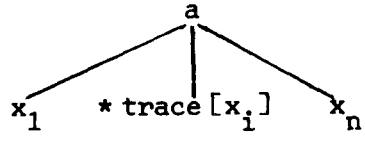
Pour simplifier la présentation des règles formelles et en diminuer le nombre, nous représentons le contenu complet de la pile dans les règles et non le sommet seul, en reportant dans le texte, lorsque c'est nécessaire, la modification sur la pile. De plus nous supposons que pour toute fonction graphique f, il existe un état $g_f \in G$ noté plus simplement f, et une règle $(v, \Lambda, \emptyset) \rightarrow (N, \Lambda, f)$ qui a pour but de déclencher la fonction f. Enfin le noeud contexte sera repéré par *, le noeud courant par trace, courant, ou mode, suivant que le mode joue un rôle ou non dans la règle.

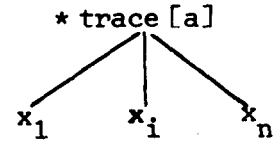
Les règles pour la fonction "↑" en mode trace s'écrivent :

r_1 : (v, 

, ↑) → (vi, 

, ∅)

r_2 : (v, 

, ↑) → (vi, 

, ∅)

Trace [noeud] indique que noeud est le noeud courant.

v est le contenu de la pile ; la caractérisation vi indique que l'on a empilé i.

La règle r_1 ne modifie pas l'* noeud, c'est pourquoi il n'est pas indiqué. La règle r_2 change l'* noeud, c'est-à-dire, change de noeud visualisé : il y a élargissement de contexte si le noeud sur lequel s'applique la fonction est un noeud contexte.

Le troisième champ indique que l'on est dans l'état ↑ de l'automate des fonctions. Les règles r_1 et r_2 ne sont applicables que si la fonction demandée est la fonction ↑.

Lorsque le troisième champ est ∅, c'est l'état neutre.

Les niveaux de formalisme

La formalisation de la description des fonctions est réalisée aux différents niveaux suivants : la syntaxe et la sémantique.

La syntaxe : c'est elle qui codifie la communication homme/machine.

La sémantique : La sémantique a pour but d'écrire l'effet d'une instruction formelle sur l'état de l'environnement (état des registres, valeur des objets manipulés). Plus précisément, nous donnons ici la sémantique de notre langage sous deux formes :

* l'aspect formel, plus particulièrement destiné à l'implémenteur ; l'effet de chaque instruction est ici décrit par les règles d'un transducteur d'arbres.

* En commentaire, nous donnons une description informelle de la sémantique destinée particulièrement à l'utilisateur.

Justification du formalisme choisi

Ce formalisme est utilisé pour plusieurs raisons essentielles :

* Il sert de base pour l'implémentation ; en effet, implémenter le langage décrit de cette façon est équivalent à simuler le transducteur défini par les règles de la sémantique concrète.

* Le langage et sa sémantique sont parfaitement définis, on peut décrire et prouver des procédures.

* Etendre le langage consiste simplement à introduire de nouvelles règles.

* On peut modifier le langage sans problème en modifiant les règles.

Nous présentons un échantillon de règles en VI.4 correspondant aux fonctions d'accès, mais elles ne forment qu'une petite partie de l'ensemble des fonctions du langage, les principales classes de fonctions étant les fonctions d'accès, de création, de modification de structure ou de noeuds, de visualisation, de chargement et de sauvegarde des objets graphiques.

En VI.5, nous présentons la preuve du transducteur, c'est-à-dire, la preuve que les règles décrites pour les différentes fonctions leur confèrent les caractéristiques désirées.

VI.2. DESCRIPTION FORMELLE DES OBJETS UTILISES DANS LE SYSTEME GRAPHIQUE

VI.2.1. Présentation de la grammaire

Une grammaire comparable aux grammaires de description de langages est utilisée. Les objets considérés sont, soit complètement spécifiés donc définis, soit incomplètement spécifiés, i.e. une information (au moins) est indéfinie.

Les règles de dérivation sur ces deux types d'objets sont presque identiques, et la plupart des distinctions entre les règles sur les différents types d'objets n'apparaissent que dans les règles qui ont des noms terminaux en membre droit.

L'idée est alors d'utiliser une grammaire à deux niveaux afin de n'avoir qu'une règle de réécriture à la place des deux règles respectives correspondant aux deux types d'objets. Le premier niveau de la grammaire ne contient qu'une seule règle : α :: = IMAGE|PATRON ce qui se lit : α se réécrit soit comme IMAGE (i.e. soit complètement spécifié, soit susceptible d'être défini avec des valeurs par défaut), soit comme PATRON (i.e. incomplètement spécifié).

Les notations utilisées au second niveau de la grammaire sont les suivantes :

A := B Règle de réécriture, le membre gauche A "se réécrit comme" le membre droit B.

C|D Ou exclusif : ce symbole n'existe qu'en membre droit. L'expression A := C|D signifie : A se réécrit soit comme C, soit comme D.

E[F] Entre crochets sont indiqués les attributs (F) liés à E.

G ; H Le ";" signifie "suivi de".

I, J la "," signifie "a pour frère dans l'arborescence". J est le frère du noeud I.

K(L) Entre parenthèses sont indiqués les descendants (L) du noeud (K) c'est la notation parenthésée des arbres.

& OBJET Référence de l'"OBJET" par son nom dans la bibliothèque.

M(OBJET &) Définition d'un lien entre un noeud (M) et son descendant ; le lien est établi à partir du nom de ce descendant, celui-ci étant référencé en bibliothèque.

Ce lien est assimilable à un appel de procédure dans les descriptions de langages.

Λ représente le mot vide.

Indef signifie indéfini, Indef est un nom terminal.

Les mots terminaux sont en lettres minuscules.

Les mots non terminaux sont en lettres majuscules.

VI.2.2. Description syntaxique des objets

1° niveau de grammaire

α ::= IMAGE|PATRON

2° niveau de grammaire

OBJET-CATALOGUE := & OBJET-IMAGE|& OBJET-PATRON
OBJET-GRAPHIQUE := OBJET-IMAGE|OBJET-PATRON
OBJET-α := NOEUD-α (LISTFILS-α)|FEUILLE-α
LISTFILS-α := OBJET-α, LISTFILS-α|OBJET-α
NOEUD-α : = NOM-α [ATTRIBUT-α]
FEUILLE-α := FEUILLE-SIMPLE-α|OBJET-MOTIF-α|APPEL-OBJET-α
FEUILLE-SIMPLE-α := NOM-α [ATTRIBUT-α ; FORME-α]
OBJET-MOTIF-α := OBJET-MOTIF-SIMPLE-α|MOTIF-APPEL-ELT-MOTIF-α
OBJET-MOTIF-SIMPLE-α := FEUILLE-SIMPLE-α [FONCTION-OCCURRENCE-α] ELT-BASE-MOTIF-α
MOTIF-APPEL-ELF-MOTIF-α := FEUILLE-SIMPLE-α [FONCTION-OCCURRENCE-α] REF-OBJET-α
APPEL-OBJET-α := NOEUD-α (REF-OBJET-α)
ELT-BASE-MOTIF-IMAGE := OBJET-IMAGE
ELT-BASE-MOTIF-PATRON := OBJET-GRAPHIQUE

REF-OBJET-IMAGE := identificateur d'objet-image catalogué
REF-OBJET-PATRON := identificateur d'objet catalogué

ATTRIBUT-α := EMPLACEMENT-α ; CADRAGE-α ; ATTRIBUT-PAR-DEFAUT-α
ATTRIBUT-PAR-DEFAUT-α := Λ|TRANSF-GEOM-α|ATTRIB-GRAPH-α|
TRANSF-GEOM-α ; ATTRIB-GRAPH-α
TRANSF-GEOM-α := AFFINITE-α|NON-AFFINITE-α
ATTRIB-GRAPH-α := TEXTURE-α ; ATTRIB-GRAPH-SANS-TEXT-α|ATTRIB-GRAPH-SANS-TEXT-α
ATTRIB-GRAPH-SANS-TEXT-α := ASPECT-α ; ATTRIB-GRAPH-SANS-ASP-α|ATTRIB-GRAPH-SANS-ASP-α
ATTRIB-GRAPH-SANS-ASP-α := TRANSPARENCE-α ; ATTRIB-GRAPH-SANS-TRANSP-α|
ATTRIB-GRAPH-SANS-TRANSP-α
ATTRIB-GRAPH-SANS-TRANSP-α := REFLECTANCE-α ; ATTRIB-GRAPH-SANS-REFLEC-α|
ATTRIB-GRAPH-SANS-REFLEC-α
ATTRIB-GRAPH-SANS-REFLEC-α := ECLAIREMENT-α ; COULEUR-α|COULEUR-α
ECLAIREMENT-α := DIFFUS-α|AUTRE-PONCTUEL-α|DIFFUS-α ; AUTRE-PONCTUEL-α
AUTRE-PONCTUEL-α := PONCTUEL-α|PONCTUEL-α ; AUTRE-PONCTUEL-α
FORME-α := FORME-CREUSE-α|FORME-PLEINE-α

FONCTION-OCCURRENCE-α := ATTRIBUT-PAR-DEFAUT-α

NOM-PATRON := NOM-IMAGE|Indef
NOM-IMAGE := nom

EMPLACEMENT-PATRON := EMPLACEMENT-IMAGE|Indef
EMPLACEMENT-IMAGE := emplacement du centre de la fenêtre|origine de la
fenêtre de l'ascendant

CADRAGE-PATRON := CADRAGE-IMAGE|Indef
CADRAGE-IMAGE := taille de la fenêtre par rapport à la taille de la fenêtre
de l'ascendant|1 (1 <=> taille de la fenêtre = taille de
la fenêtre de l'ascendant)

AFFINITE-PATRON := AFFINITE-IMAGE|Indef
AFFINITE-IMAGE := matrice de transformation en coordonnées homogène|matrice
de l'identité

NON-AFFINITE-PATRON := NON-AFINITE-IMAGE|Indef
NON-AFFINITE-IMAGE := expression laissée au choix de l'implémenteur|Λ

TEXTURE-PATRON := TEXTURE-IMAGE|Indef
TEXTURE-IMAGE := expression au choix de l'implémenteur|Λ

ASPECT-PATRON := ASPECT-IMAGE|Indef
ASPECT-IMAGE := expression au choix de l'implémenteur|Λ

TRANSPARENCE-PATRON := TRANSPARENCE-IMAGE|Indef
TRANSPARENCE-IMAGE := expression au choix de l'implémenteur|Λ

REFLECTANCE-PATRON := REFLECTANCE-IMAGE|Indef
REFLECTANCE-IMAGE := expression au choix de l'implémenteur|Λ

DIFFUS-PATRON := DIFFUS-IMAGE|Indef
DIFFUS-IMAGE := expression au choix de l'implémenteur|Λ

PONCTUEL-PATRON := PONCTUEL-IMAGE|Indef
PONCTUEL-IMAGE := expression au choix de l'implémenteur|Λ

COULEUR-PATRON := COULEUR-IMAGE|Indef
COULEUR-IMAGE := {rouge, vert, bleu}|noir

FORME-PLEINE-PATRON := FORME-PLEINE-IMAGE|Indef
FORME-PLEINE-IMAGE := Carré plein|Disque|Tonneau plein|Polygone plein|AUTRE-PLEIN

AUTRE-PLEIN := PLEIN|PLEIN ; AUTRE-PLEIN
PLEIN := forme nouvelle au choix de l'implémenteur|Λ

FORME-CREUSE-PATRON := FORME-CREUSE-IMAGE|Indef
FORME-CREUSE-IMAGE := Contour carré|Cercle|Contour tonneau|Contour polygone|
AUTRE-CONTOUR

AUTRE-CONTOUR := CONTOUR|CONTOUR ; AUTRE-CONTOUR
CONTOUR := forme nouvelle au choix de l'implémenteur|Λ

VI.4. LES PRIMITIVES LIEES AUX CONTEXTES DE VISUALISATION. VALIDATION ET INVALIDATION. PRIMITIVES D'ACCES

VI.4.1. Les primitives liées aux contextes

Syntaxe DEFINITION CONTEXTE (η)

Sémantique $(v, \text{mode } [x], Def) \rightarrow (v\ \dot{x}, * \text{ mode } [\eta], \emptyset v)$

Commentaire Quand l'utilisateur désire visualiser un noeud dans une fenêtre de visualisation qu'il vient de créer, il définit un contexte dans cette fenêtre de visualisation en appelant la fonction "DEFINITION CONTEXTE" puis en désignant le noeud η (visualisé dans une autre fenêtre de visualisation) qu'il veut visualiser. La désignation de η doit être suivie de la fonction "VALIDATION" qui fixe le noeud courant comme noeud paramètre pour l'activation de la primitive. Si le noeud choisi est le noeud courant, la seule validation suffit. Si la primitive "DEFINITION CONTEXTE" (η) est suivie de la fonction "INVALIDATION", la primitive est réduite à la fonction identité (retour à l'état antérieur à l'appel de la primitive).

Lorsque la fonction est effective (i.e. validée), le noeud courant est le noeud contexte.

Syntaxe REDUCTION CONTEXTE

Sémantique $(v, * [a], Red) \rightarrow (v, a, \overline{Red})$
$(v, \text{mode } [\eta], \overline{Red}) \rightarrow (\Lambda, * \text{ mode } [\eta], \emptyset)$

Commentaire La réduction du contexte de visualisation s'applique sur le noeud courant, qui est un descendant du noeud contexte précédent. L'appel de cette fonction provoque la perte des informations relatives au cheminement effectué pour accéder au noeud courant (et ceci quel que soit le mode de parcours).

Syntaxe CONTEXTE ECRAN

Commentaire Cette fonction ne modifie aucun état des objets, mis à part le fait qu'un objet est ou n'est pas visualisé dans une clôture. Le mode de parcours, le noeud courant et le noeud contexte de chaque objet visualisable sont inchangés par cette fonction.
A un instant donné, l'utilisateur travaille sur un noeud dans une et une seule clôture, c'est le noeud courant de l'utilisateur, même si celui-ci est en contexte écran.

Syntaxe CONTEXTE UNIQUE

Commentaire C'est également une fonction de pure visualisation ; le contexte dans lequel travaille l'utilisateur au moment où il active la primitive "CONTEXTE-UNIQUE" est seul représenté sur l'écran ; tout se passe comme si la clotûre est agrandie à la taille de l'écran, et les autres objets sont "extérieurs" à cette clôture.
Le contexte visualisé est représenté sur l'écran de façon à occuper la place maximale sur l'écran sans modifier les proportions du noeud à visualiser.

Syntaxe CHANGEMENT CONTEXTE

Sémantique (v, mode [a], CHCTC) → (Λ, mode [η], ∅)
a est le noeud courant du contexte initial
η est le noeud courant du contexte désigné

Commentaire C'est la fonction qui permet de passer d'une clôture dans une autre (en contexte écran) sans modifier les noeuds courants des contextes correspondants.

VI.4.2. Validation-Invalidation

∀f, fonction paramétrée par un (ou plusieurs) noeud(s) désigné(s) à l'aide d'une quelconque primitive de désignation, f doit être suivie, soit de la fonction "VALIDATION", soit de la fonction "INVALIDATION".

. La validation rend effective la fonction f. La formalisation de telles fonctions est donnée indépendamment de la validation ou l'invalidation ; c'est le cas de "DEFINITION CONTEXTE" (VI.3.1) et de "PREMIER NOEUD COMMUN" (VI.4.3).

Syntaxe VALIDATION

Sémantique $(v, \text{mode } [\eta], \emptyset v) \rightarrow (\Lambda, \text{mode } [\eta], \emptyset)$

Syntaxe INVALIDATION

Sémantique $(v, \text{mode } [\eta], \emptyset v) \rightarrow (v, \text{mode } [\eta], \text{Inv-f})$

$(vs, \text{mode } [\eta], \text{Inv-f}) \rightarrow (v, \text{mode } [\eta], \text{Inv-f}) \qquad s \neq \dot{x}$

$(v\dot{x}, \text{mode } [\eta], \text{Inv-f}) \rightarrow (v\dot{x}, \eta, \text{Inv-f}_1)$

$(v\dot{x}, x, \text{Inv-f}_1) \rightarrow (v, \text{mode } [x], \emptyset)$

Commentaire La validation intervenant après la désignation d'un noeud η, ce noeud devient noeud courant départ d'un chemin de parcours, tout comme les noeuds désignés.

L'invalidation permet de revenir exactement dans l'état précédant la fonction invalidée.

VI.4.3. Ecriture des fonctions liées aux accès

VI.4.3.1. Les fonctions de mode de parcours

Syntaxe Ce sont les fonctions de changement de mode : MODE TRACE et MODE COURANT

Sémantique $(v, \text{mode } [\eta], CHM) \rightarrow (w, \text{mode1 } [\eta], \emptyset)$

$w =$ si mode = mode1 alors v
sinon Λ

Commentaire : Le changement de mode est l'identité si le mode demandé est déjà le mode de parcours avant l'appel de la fonction. S'il y a véritable changement de mode (du mode courant au mode trace ou vice-versa), tout le cheminement réalisé avant l'accès au noeud courant est perdu (ce qui se traduit par la pile vide). Le noeud courant et le contexte sont inchangés par la fonction.

VI.4.3.2. Les primitives de base

VI.4.3.2.1. Désignation

Tout noeud désigné est le point de départ d'un chemin de parcours, quelle que soit la fonction de désignation utilisée.

Syntaxe Il y a deux primitives de base pour la désignation

- ⊕ provoque la désignation d'une feuille visualisée
- DESIGN-nom de l'objet est la désignation par son nom d'un noeud quelconque visualisé. C'est la primitive de base la plus générale pour la désignation de base.

Sémantique

Désignation de base : (v, mode [a], DES) → (Λ, mode [η], ∅)
si désignation de base = ● alors η est une feuille

Commentaire La désignation d'un noeud visualisé η ne modifie jamais le contexte, en échange, il peut y avoir modification du noeud courant. La désignation redéfinit toujours un nouveau chemin de parcours ; tout noeud visualisé dans un même contexte est désignable par la "désignation de base" utilisant le nom du noeud ; la fonction ● ne permet que la désignation d'une feuille, l'accès aux autres noeuds s'effectue à l'aide des primitives de parcours.

Remarque :

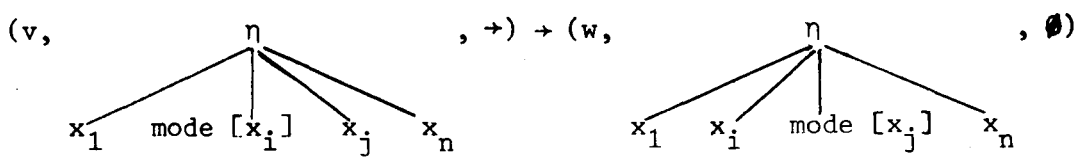
La désignation d'objets non visualisés ne se fait que par leur nom et n'entre pas dans le cadre des fonctions formalisées.

VI.4.3.2.2. Les primitives de base du parcours

* les fonctions de parcours latéral : "DROITE" ou "↳"
et : "GAUCHE" ou "↰" qui réalisent
les accès aux frères du noeud courant.

Syntaxe DROITE ou →

Sémantique



avec n = nombre de noeuds descendant du même père η, où η est aussi le père du noeud courant, on a : ∀i < n, j = i+1, et i = n ⇒ j = 1

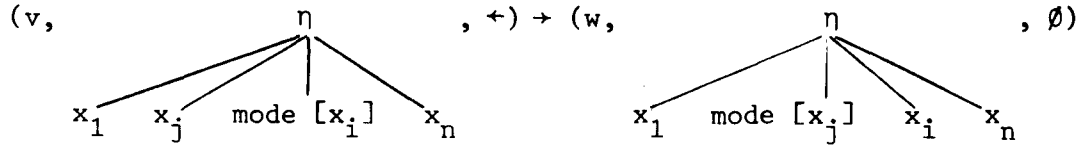
```
si n = 1 alors w = v
         sinon si (v = s ←1) ou (v = s →n-1)
                  alors w = s
                  sinon si v = s ↑i alors w = s →i+1
                                    sinon si v = s ↑i alors w = s ←i-1
                                                      sinon w = v ↑1
                                co v = s i ou v = Λ ou (v = s # et mode = trace) co
```

Commentaire La fonction "↦" délivre le premier frère droit du noeud courant ; s'il n'y en n'a plus, elle délivre le dernier frère gauche : les frères forment un anneau. Lorsque la fonction "↦" boucle l'anneau (i.e. retour au noeud de départ après avoir examiné consécutivement tous les frères), le cycle obtenu n'est pas conservé, et tout se passe comme si l'on avait appliqué la fonction "identité" sur le noeud de départ de la boucle. En particulier, si un noeud n'a pas de frère, "↦" est l'identité sur ce noeud (boucle de 1 élément).

La fonction "↦" ne modifie jamais le contexte, mais elle ne peut pas s'appliquer sur un noeud contexte, i.e. dont le(s) frère(s) n'est (ne sont) pas visualisé(s)

Syntaxe GAUCHE ou ←

Sémantique



avec n = nombre de noeuds descendant du même père η, où η est aussi le père du noeud courant, on a : ∀i > 1, j = i-1, et i = 1 ⇒ j = n

```
si n = 1 alors w = v
        sinon si (v = s →1) ou (v = s ←n-1)
                alors w = s
                sinon si v = s ←i alors w = s ←i+1
                                sinon si v = s →i alors w = s →i-1
                                                sinon si mode = courant
                                                      alors w = v →n-1
                                                      sinon w = v ←1
```

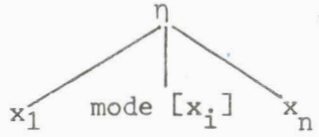
Commentaire La fonction "↤" délivre le premier frère gauche du noeud courant ; s'il n'y en n'a plus, elle délivre le dernier frère droit : les frères forment un anneau. Lorsque la fonction "↤" boucle cet anneau (i.e. retour au noeud de départ après avoir examiné consécutivement tous les frères), le cycle obtenu n'est pas conservé, et tout se passe comme si l'on avait appliqué la fonction "identité" sur le noeud de départ de la boucle. En particulier, si un noeud n'a pas de frère, "↤" est l'identité sur ce noeud (boucle de 1 élément).

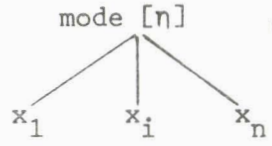
* La fonction "↤" ne modifie jamais le contexte, mais elle ne peut pas s'appliquer sur un noeud contexte (même principe que pour la fonction "↦").

Les fonctions de parcours ascendant et descendant :
"HAUT" ou "↑" permet d'accéder au noeud père du noeud courant
"BAS" ou "↓" permet d'accéder à un noeud fils du noeud courant.

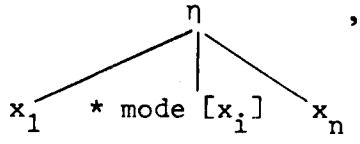
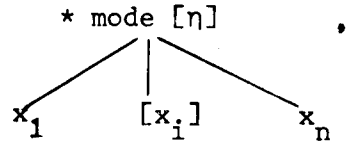
Syntaxe HAUT ou ↑

Sémantique

(1) (v, 

, ↑) → (w, 

, ∅)

(2) $(v,$  $, \uparrow) \rightarrow (w,$  $, \emptyset)$

si mode = trace alors si v = s # alors w = s
sinon w = v i

si mode = courant alors si v = s $\overrightarrow{j}$ ou v = s $\overleftarrow{j}$ avec $0 < j \leq n-1$
alors si i = 1 alors w = Λ
sinon w = i
sinon si v = Λ et i = 1
alors w = Λ
sinon w = vi

quant $x_i \neq$ racine, on considère que le noeud d'où l'on vient est le i° fils du noeud auquel on accède.

Commentaire Quel que soit le mode de parcours, la fonction "↑" appliquée sur une racine est l'identité. Dans tous les autres cas, la fonction "↑" fournit le père du noeud courant x_i sur lequel est appliquée la fonction.

Si x_i est un noeud contexte, ce dernier est automatiquement élargi à η. La fonction "↑" est la seule qui provoque un élargissement de contexte.

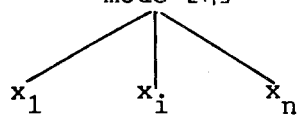
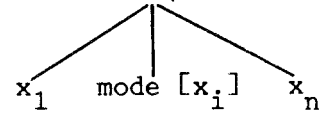
Dans le mode trace, le noeud d'où l'on vient est mémorisé de façon à pouvoir y retourner facilement par la fonction "Retour". Un cas particulier : Si le noeud x_i d'où l'on vient suit immédiatement une action "↓" appliquée sur le père η de x_i, la fonction "↑" va refournir η et l'action résultante de "↓" suivie de "↑" est l'identité.

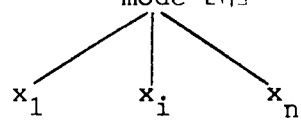
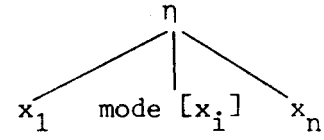
Dans le mode courant, si le noeud x_i sur lequel s'applique la fonction "↑" n'appartient pas au chemin courant, (i.e. le noeud est un frère de l'un des noeuds du chemin courant), l'activation de "↑" provoque automatiquement la définition d'un nouveau chemin courant qui relie le noeud courant x_i à la racine. Si le noeud x_i appartient au chemin courant, on le mémorise pour pouvoir

éventuellement y revenir par la fonction "↓".

Syntaxe "BAS" ou "↓"

Sémantique

(1) $(v,$  $, \downarrow) \rightarrow (w,$  $, \emptyset)$

(2) $(v,$  $, \downarrow) \rightarrow (w,$  $, \emptyset)$

si mode = trace alors i = 1 et si v = s 1 alors w = s
sinon w = v #

si mode = courant alors si $v = s \overrightarrow{j}$ ou $v = s \overleftarrow{j}$ ou v = Λ
alors i = 1 et w = Λ (on a $x_i = x_1$)

sinon co v = s i et x_i est le $i^{ième}$ fils de η co
w = s

Commentaire La fonction "↓" ne modifie jamais le contexte.

Si le noeud courant est une feuille, la fonction est l'identité quel que soit le mode. Sinon, la fonction ne fournit pas le même noeud selon le mode de parcours.

Dans le mode trace, la fonction "↓" permet d'accéder au premier fils x_1 du noeud η sur lequel on est positionné. Si le noeud η d'où l'on vient suit immédiatement une action "↑" appliquée sur ce fils x_1 de η, la

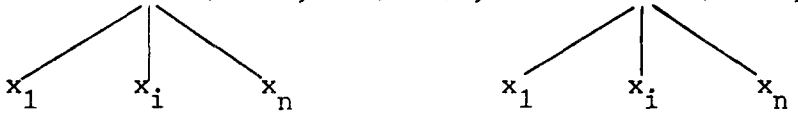
fonction "↓" va refournir x_1 et l'action résultante de "↑" suivi de "↓" (dans ce cas très précis) est l'identité.

Dans le mode courant, si le noeud courant η et un de ses descendants x_i appartiennent au chemin courant, la fonction "↓" fournit ce descendant x_i et le chemin courant n'est pas modifié. Dans tous les autres cas, le noeud auquel on accède est le premier fils x_1 du noeud courant initial η, et un nouveau chemin courant (depuis le noeud x_1 auquel on a accédé jusqu'à la racine) remplace le précédent ; dans le cas où le noeud courant appartenait au chemin courant, le nouveau chemin est l'extension du précédent.

Syntaxe "RETOUR"

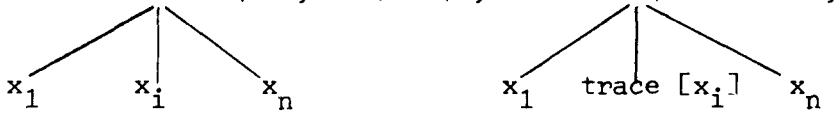
Sémantique

(1) $(\Lambda,$ trace [η] $, Ret) \rightarrow (\Lambda,$ trace [η] $, \emptyset)$

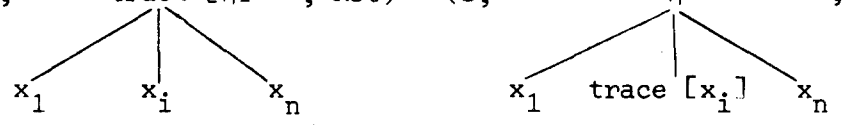


c'est le retour après une désignation ou un changement de mode.

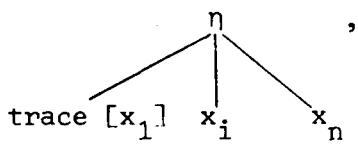
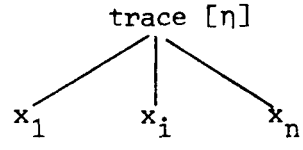
(2) $(si,$ * trace [η] $, Ret) \rightarrow (s,$ * η $, \emptyset)$



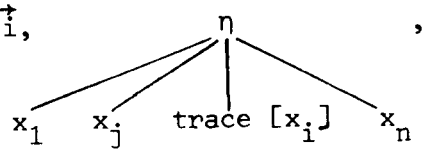
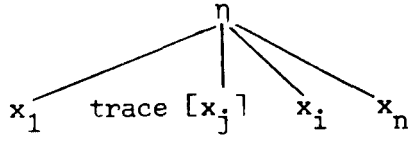
(3) $(si,$ trace [η] $, Ret) \rightarrow (s,$ η $, \emptyset)$



c'est le retour après une fonction "↑" (non réduite à l'identité). Le contexte est inchangé par cette fonction

(4) $(s\ \#\ ,$  $,\ Ret) \rightarrow (s,$  $,\ \emptyset)$

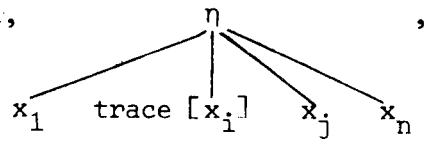
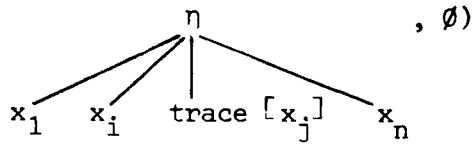
c'est le retour après une fonction "↓" (non réduite à l'identité) ; le contexte est ascendant du noeud sur lequel s'applique la fonction.

(5) $(\overrightarrow{si},$  $,\ Ret) \rightarrow (w,$  $,\ \emptyset)$

$$\forall i > 1,\ j = i-1,\ \text{et}\ i = 1 \Rightarrow j = n$$

$$\underline{\text{si}}\ i = 1\ \underline{\text{alors}}\ w = s$$
$$\underline{\text{sinon}}\ w = s\ \overrightarrow{i-1}$$

c'est le retour après la fonction "↦" ; il n'y a pas de retour sur les cycles, car ils sont éliminés automatiquement lors de l'appel de la fonction "↦".

(6) $(\overleftarrow{si},$  $,\ Ret) \rightarrow (w,$  $,\ \emptyset)$

$$\forall i < n,\ j = i+1,\ \text{et}\ i = n \Rightarrow j = 1$$

$$\underline{\text{si}}\ i = 1\ \underline{\text{alors}}\ w = s$$
$$\underline{\text{sinon}}\ w = s\ \overleftarrow{i-1}$$

c'est le retour après la fonction "↤" ; il n'y a pas de retour sur les cycles, car ils sont éliminés automatiquement lors de l'appel de la fonction "↤".

Commentaire La fonction retour annule la fonction précédente. Cette fonction est toujours la dernière dont les cycles sur les frères ont été annulés en temps utile. Les noeuds par lesquels on a accédé au noeud courant depuis la désignation ont été conservés (c'est la trace), et l'on peut donc faire "marche arrière" en pas à pas.

La fonction de retour n'engendre jamais d'élargissement de contexte. En particulier, on ne peut accéder à un noeud qui n'est plus visualisé à la suite d'une réduction de contexte. Lorsqu'il y a réduction de contexte sur un noeud N, la trace des noeuds auxquels on a accédé avant d'arriver à N est perdue. La fonction retour ne permet donc pas de revenir sur un noeud auquel on a accédé avant une réduction de contexte.

VI.4.3.3. Une primitive d'édition

Nous considérons la fonction d'identification d'un noeud par premier noeud commun.

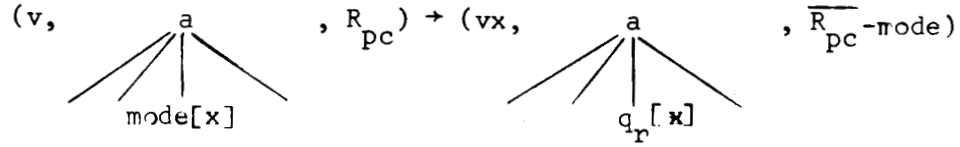
Syntaxe PREMIER NOEUD COMMUN (λ)

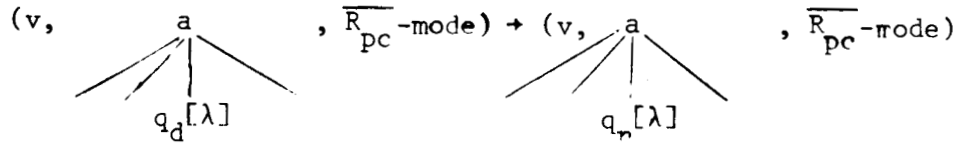
Sémantique La sémantique de cette fonction d'édition est définie sous forme d'un ensemble non déterministe de règle. Cependant, le lecteur pourra vérifier que pour un arbre donné, le noeud courant final existe et est unique.

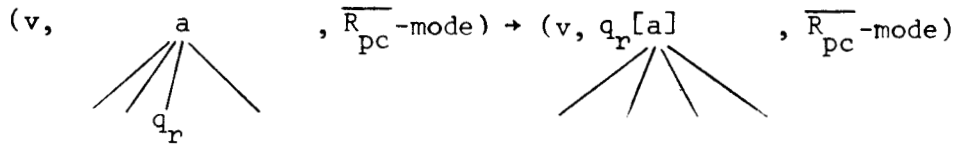
La fonction identifie le premier noeud qui est commun au noeud courant repéré initialement par "mode" et à une feuille λ, dite noeud désigné et repéré initialement par q_d.

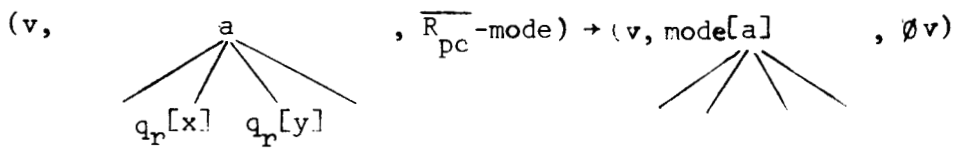
q_r est utilisé pour repérer des noeuds au cours de la recherche.

1er cas : le noeud désigné n'est pas descendant du noeud courant.

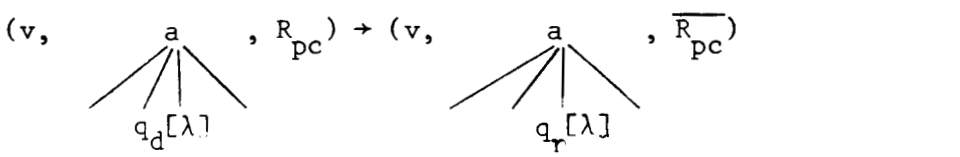


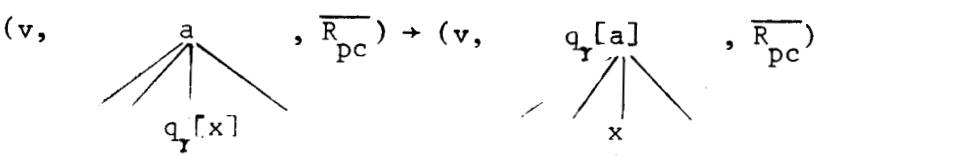


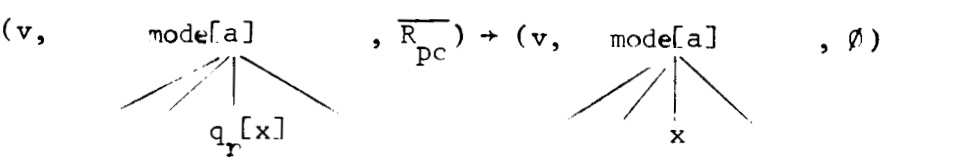




2ème cas : le noeud désigné est descendant du noeud courant.







3ème cas : le noeud désigné est le noeud courant.

$(v, mode, q_d[a], R_{pc}) \rightarrow (v, mode[a], \emptyset)$

Commentaire Cette fonction détermine le premier noeud commun au noeud courant et à un autre noeud qui est désigné. La fonction n'est effective que si la fonction est validée ; dans ce cas, le noeud courant déterminé est considéré comme un noeud désigné vis-à-vis d'un cheminement partant de ce noeud.

La fonction de "VALIDATION" fixe le noeud courant comme noeud paramétre pour l'activation de la primitive.

Si la primitive "PREMIER NOEUD COMMUN" (λ) est suivie de la fonction "INVALIDATION", la primitive est réduite à la fonction identité (retour à l'état antérieur à l'appel de la primitive).

Lorsque la primitive "PREMIER-NOEUD-COMMUN" est validée (1^{er} cas) nous dirons qu'elle est effective ; si nous sommes dans les cas (2) ou (3), la validation n'est pas nécessaire.

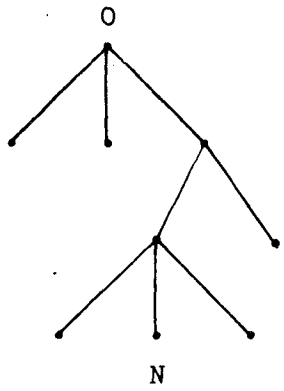
VI.5. PROPRIETES DES ACCES

Soit à montrer que les hypothèses de définition des primitives d'accès dans les différents modes de parcours sont bien vérifiées. Ces primitives faisant intervenir l'automate des fonctions de façon triviale, nous ne considérons pas ici ce troisième champ des triplets de la formalisation ; nous n'étudions que le comportement des couples (pile, arbre) $\in \Gamma^* \times T(\Sigma \cup Q)$. Le lecteur pourra vérifier que quels que soient $(v, t) \in \Gamma^* \times T(\Sigma \cup Q)$, et la fonction graphique $f \in F = (\rightarrow, \leftarrow, \uparrow, \downarrow)$, l'application des règles définies pour cette fonction fournit un unique $(v', t') \in \Gamma^* \times T(\Sigma \cup Q)$, tel que $(v, t) \Rightarrow (v', t')$, que nous noterons $f(v, t) = (v', t')$.

VI.5.1. Mode courant

Dans ce mode, nous avons défini la notion de chemin courant, ainsi que les effets des opérations sur le chemin courant. Nous allons tout d'abord montrer que le contenu de la pile définit ce chemin courant.

Un chemin d'un arbre peut être représenté par un mot de $(1, 2, 3, \ldots, p)^*$ où chaque symbole i de droite à gauche définit le noeud suivant comme étant le $i^{ième}$ fils du noeud précédent (on part de la racine). Par exemple 213 dénote le chemin qui va de N à la racine dans l'arbre et que nous noterons ch(N).

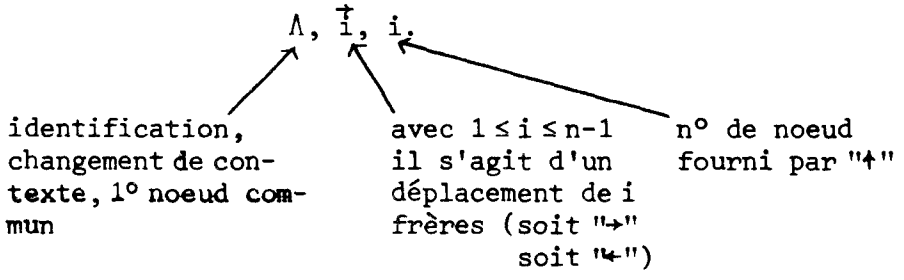


Considérons maintenant un couple (v, t) état du transducteur en mode courant, et définissons $\varphi(v, t) \in (1..p)^*$ comme :

si $v \in (1..p)^*$ alors $\varphi(v, t) = v$ ch(noeud courant)
si $v = u \overset{+}{i}$ avec $u \in (1..p)^*$ alors
$\varphi(v, t) = u$ ch($i^{ième}$ frère gauche du noeud courant)

Le lecteur pourra vérifier, en analysant les modifications de la pile effectuées par chaque règle, que nous avons les deux propriétés suivantes, ce qui montre la consistance de la définition de φ.

1. les symboles possibles contenus dans la pile sont :



2. le symbole de la forme $\vec{i}$ ne peut se trouver qu'en sommet de pile

Pour cela, il suffit de montrer que l'on n'empile jamais un symbole du type $\vec{i}$ ou $\overleftarrow{i}$ si le sommet de pile est déjà de ce type. Les seules règles qui ont cet effet sont relatives aux fonctions $\rightarrow$ et $\leftarrow$. L'analyse des règles correspondantes montre alors que le seul cas où l'on empile quelque chose, c'est précisément lorsque le sommet n'est pas de type $\vec{k}$ ou $\overleftarrow{k}$. Les autres cas modifient le sommet, dépilent le sommet, ou le laissent inchangé.

Lemme 1 : *Le chemin courant est* $1^r \varphi(v, t)$ avec $r \geq 0$.

Il suffit en fait de montrer que cette propriété est conservée par les fonctions $\rightarrow$, $\leftarrow$, $\uparrow$, $\downarrow$. Notons cc le chemin courant, et nc le noeud courant avant application de la fonction f, et f(cc) et f(nc) après application de la fonction f et (v', t') = f(v, t).

(1) cas de la fonction "$\hookrightarrow$"

Nous avons d'une part $\rightarrow$(cc) = cc = 1^{er} $\varphi(v, t)$
si $v \in (1..p)^*$ alors $\varphi(v, t) = u$ ch(nc)
si nc est seul descendant (pas de frère) alors nc = $\rightarrow$(nc)
et v' = v entraîne $\varphi(v', t') = \varphi(v, t)$
sinon $v' = v \vec{1}$ et nc est le 1^{er} frère gauche de $\rightarrow$(nc).

si $v = u \vec{i}$ avec $u \in (1..p)^*$ alors $\varphi(v, t) = u$ ch($i^{ième}$ frère gauche de nc)
si $i \neq n-1$ alors $v = u \vec{i+1}$ et le $i^{ième}$ frère gauche de nc est aussi le $(i+1)^{ième}$ frère gauche de $\rightarrow$(nc).

si $i = n-1$ alors $v = u$ et $\rightarrow$(nc) est le $(n-1)^{ième}$ frère gauche de $\rightarrow$(nc).

Dans chaque cas ceci implique $\varphi(v', t') = \varphi(v, t)$.

(2) le cas de la fonction "$\leftarrow$" est analogue au cas précédent.

(3) cas de la fonction "$\uparrow$". Supposons que nc soit le $j^{ème}$ fils de son père qui devient noeud courant. i.e., ch(nc) = j ch($\uparrow$(nc)).

si $v \in (1..p)^*$ alors $cc = 1^r$ $\varphi(v, t) = 1^r$ v ch(nc)
montre que nc est sur cc et par conséquent $\uparrow(cc) = cc$.

si $v = \Lambda$ et $j = 1$ alors $v' = \Lambda$ et il s'ensuit :
$\uparrow(cc) = cc = 1^r$ ch(nc) = 1^{r+1} ch($\uparrow$(nc)) = $1^{r+1}\varphi(v', t')$

si $v \neq \Lambda$ ou $j \neq 1$ alors $v' = vj$ et il s'ensuit :
$\uparrow(cc) = cc = 1^r$ v ch(nc) = 1^r vj ch($\uparrow$(nc)) =
1^r v' ch($\uparrow$(nc)) = $1^r\varphi(v', t')$

si $v = u\ \vec{i}$ avec $u \in (1..n)^*$ alors
$cc = 1^r$ ch(v, t) = 1^r u ch(1^{er} frère gauche de nc)
comme nc $\neq 1^{er}$ frère gauche de nc, il s'ensuit que nc n'appartient pas à cc et $\uparrow(cc)$ = ch(nc). De plus v' = j. Il s'ensuit $\uparrow(cc)$ = ch(v', t').

(4) cas de la fonction "$\downarrow$".

si $v = \Lambda$ alors $cc = 1^r$ ch(nc). Dans ce cas nc appartient à cc et quelle que soit la valeur de r, $\downarrow$(nc) est le 1^{er} fils de nc, i.e. ch($\downarrow$(nc)) = 1 ch(nc). Si $r > 0$ alors $\downarrow(cc) = cc = 1^{r-1}$ ch($\downarrow$(nc)) = ch(v', t'). Si $r = 0$ alors $\downarrow(cc) = 1\ cc$ = ch($\downarrow$(nc)) = ch(v', t').

si $v = u\ i$ avec $u \in (1..p)^*$ et $i \in 1..p$ alors
$cc = 1^r$ u i ch(nc). Dans ce cas nc appartient à cc ainsi que son 1^{er} fils qui devient donc $\downarrow$(nc). Il s'ensuit que cc ne change pas. De plus v' = u et donc
$\varphi(v', t')$ = u ch($\downarrow$(nc)) = u i ch(nc) = $\varphi(v, t)$

si $v = u \overset{\uparrow}{1}$ avec $u \in (1..p)^*$ comme pour la fonction "$\uparrow$", nc n'appartient pas à cc et $\downarrow(nc)$ est le 1^{er} fils de nc et $\downarrow(cc) = ch(\downarrow(nc))$. De $v' = \Lambda$ il s'ensuit $\varphi(v', t') = \downarrow(cc)$.

C.Q.D.F.

Il s'agit de montrer maintenant que les règles sont bien telles que les quatre successions de fonctions s'annulent deux à deux sauf dans certains cas, c'est-à-dire :

$(\rightarrow, \leftarrow) \vdash \Lambda$
$(\leftarrow, \rightarrow) \vdash \Lambda$
$(\uparrow, \downarrow) \vdash \Lambda$
$(\downarrow, \uparrow) \vdash \Lambda$

Nous noterons $\hat{f}$ la fonction opposée correspondant à f au sens précédent.

Théorème : $\forall (v, t) \in \Gamma^* \times T(\Sigma \cup Q)$, $\forall f \in F$ *si* $f(cc) = 1^*$ cc, *c'est-à-dire le chemin courant n'est pas perdu (cas* $\uparrow$, $\downarrow$ *sur* nc $\notin$ cc*) alors* $(v, t) \overset{f}{\Longrightarrow} (v', t') \overset{\hat{f}}{\Longrightarrow} (v, t)$ *c'est-à-dire* $\hat{f}$ *remet dans l'état précédent. Dans les autres cas,* $(v, t) \overset{f}{\Longrightarrow} (v', t') \overset{\hat{f}}{\Longrightarrow} (\Lambda, t)$ *c'est-à-dire seul le noeud courant est retrouvé. (Nous ne nous intéressons pas au noeud contexte qui peut changer en cas d'élargissement par* f *ou* $\hat{f}$*).*

La propriété est évidente pour $\rightarrow$ et $\leftarrow$ car $\leftarrow(\rightarrow(nc)) = nc$, et de l'étude de la pile dans le lemme précédent pour ces fonctions on peut déduire la propriété.

Etudions $(v, t) \overset{\uparrow}{\Longrightarrow} (v', t') \overset{\downarrow}{\Longrightarrow} (v'', t'')$. $\uparrow(nc)$ est le père de nc. Supposons que nc soit $j^{ème}$ fils de son père. Si nc appartient à cc alors $v' = vj$, donc $\downarrow(nc')$ sera le $j^{ème}$ fils de nc' d'où $t'' = t$. De plus $v'' = v$. Si par contre nc n'appartient pas à cc alors $v = u \overset{\uparrow}{1}$ et $v' = j$. Il s'ensuit que $\downarrow(nc')$ sera aussi le $j^{ème}$ fils de nc', et $t'' = t$ mais par contre $v'' = \Lambda$.

Etudions $(v, t) \overset{\downarrow}{\Longrightarrow} (v', t') \overset{\uparrow}{\Longrightarrow} (v'', t'')$. Si nc appartient au chemin courant alors $v \in (1..p)^*$. Dans ce cas, si $v = \Lambda$ alors $\downarrow(nc)$ est le 1^{er} fils de nc et $v' = \Lambda$, il s'ensuit que $v'' = \Lambda$, de plus $\uparrow(\downarrow(nc)) = nc$. Si $v = uj$ alors $\downarrow(nc)$ est le $j^{ème}$ fils de nc, $v' = u$; il s'ensuit que $v'' = uj$ et $\uparrow(\downarrow(nc) = nc$. Lorsque nc n'appartient pas au chemin courant, alors v est de la forme $u\,\vec{i}$, donc $\downarrow(nc)$ est le 1^{er} fils de nc et $v' = \Lambda$; il s'ensuit que $v'' = \Lambda$ et $\uparrow(\downarrow(nc) = nc$.

C.Q.D.F.

VI.5.2. Mode trace

Dans ce mode nous avons défini la notion de trace réduite comme suite des noeuds par lesquels on est passé depuis le noeud initial, en éliminant les cycles horizontaux et verticaux. La suite des noeuds par lesquels on passe peut se représenter par la suite des fonctions de F, partant d'un noeud initial ni. Une telle suite de fonctions peut se dénoter par un mot de F^*. Nous supposerons le noeud initial implicite. La trace est donc $w \in F^*$. Il en va de même de la trace réduite que nous noterons $\overline{w}$. On notera que l'opération de réduction dépend du noeud initial. La notion d'élimination de cycles horizontaux ou verticaux peut se définir par récurrence de la façon suivante :

- $\overline{\Lambda} = \Lambda$

  cycles horizontaux :

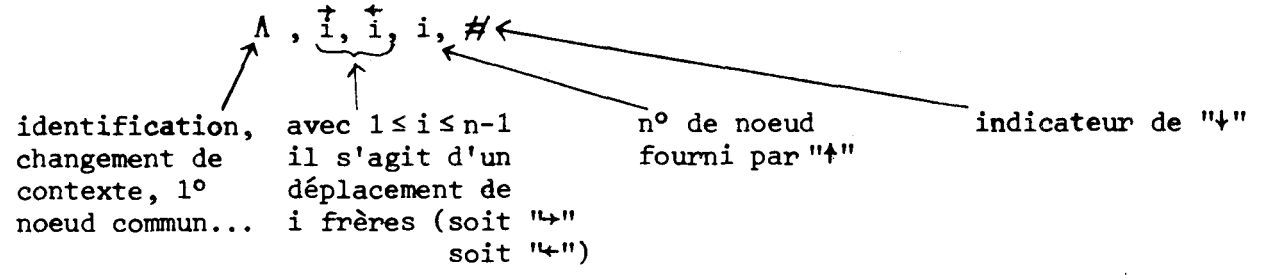
  - $\overline{\overline{w}\rightarrow}$: si $\overline{w} = u \rightarrow^{n-1}$ (où n est le nombre de descendants du père du noeud courant) ou $\overline{w} = u \leftarrow$
    alors $\overline{\overline{w}\rightarrow} = u$
    sinon $\overline{\overline{w}\rightarrow} = \overline{w}\rightarrow$

  - $\overline{\overline{w}\leftarrow}$: si $\overline{w} = u \leftarrow^{n-1}$ ou $\overline{w} = u \rightarrow$ alors $\overline{\overline{w}\leftarrow} = u$
    sinon $\overline{\overline{w}\leftarrow} = \overline{w}\leftarrow$

  cycles verticaux :

  - $\overline{\overline{w}\uparrow}$: si $\overline{w} = u \downarrow$ alors $\overline{\overline{w}\uparrow} = u$ sinon $\overline{\overline{w}\uparrow} = \overline{w}\uparrow$
  - $\overline{\overline{w}\downarrow}$: si $\overline{w} = u \uparrow$ et le noeud courant est le 1^{er} fils de son père
    alors $\overline{\overline{w}\downarrow} = u$ sinon $\overline{\overline{w}\downarrow} = \overline{w}\downarrow$.

Nous allons d'abord montrer que le contenu de la pile définit la trace réduite. Pour cela les symboles possibles contenus dans la pile sont :



définissons $\psi(\vec{i}) = \rightarrow^i$

$\psi(\overleftarrow{i}) = \leftarrow^i$

$\psi(i) = \uparrow$

$\psi(\#) = \downarrow$

Cette application $\psi : \Gamma \to F^*$ est ensuite étendue de façon naturelle à $\psi : \Gamma^* \to F^*$.

On pourra noter que si v est un contenu de la pile, et u un sous-mot de longueur 2 de v, alors u $\notin$ ($\vec{i}\,\vec{i}$, $\overleftarrow{i}\,\vec{i}$, $\vec{i}\,\overleftarrow{i}$, $\overleftarrow{i}\,\overleftarrow{i}$). Il s'ensuit en particulier que si $v = b\,\vec{i}$ ou $v = b\,\overleftarrow{i}$, alors $\psi(b)$ ne se termine ni par $\rightarrow$ ni par $\leftarrow$.

Lemme 2 : *Soit* $w \in F^*$ *une suite d'actions appliquée sur un couple* (Λ, t). *Si* $(v_1, t_1) = w(\Lambda, t)$ *alors* $\overline{w} = \psi(v_1)$ *i.e. la trace réduite est* $\psi(v_1)$.

Montrons-le par récurrence sur la longueur de w.

C'est évident pour $w = \Lambda$.

Soit $w = w_1 f$. Notons $(v_1, t_1) = w_1(\Lambda, t)$ et $(v_2, t_2) = f(v_1, t_1)$ par hypothèse de récurrence, $\overline{w}_1 = \psi(v_1)$.

(1) cas $f = \rightarrow$

si $v_1 = b\ \overrightarrow{n-1}$ alors $\overline{w}_1 = \psi(b) \rightarrow^{n-1}$ et $\overline{w\rightarrow} = \psi(b)$ et $v_2 = b$

si $v_1 = b\ \overleftarrow{1}$ alors $\overline{w}_1 = \psi(b) \leftarrow$ et $\overline{w\rightarrow} = \psi(b)$ et $v_2 = b$

si $v_1 = b\ \overleftarrow{i}$ alors $\overline{w}_1 = \psi(b) \leftarrow^{i}$ et $\overline{w\rightarrow} = \psi(b) \leftarrow^{i-1}$ et $v_2 = b\ \overleftarrow{i-1}$

si $v_1 = b\ \overrightarrow{i}$ alors $\overline{w}_1 = \psi(b) \rightarrow^{i}$ et $\overline{w\rightarrow} = \psi(b) \rightarrow^{i+1}$ et $v_2 = b\ \overrightarrow{i+1}$

sinon $v_2 = v_1\ \overrightarrow{1}$ et $\overline{w\rightarrow} = \psi(v_1)\rightarrow$

(2) cas $f = \leftarrow$ similaire à ci-dessus.

(3) cas $f = \uparrow$

si $v_1 = s\ \#$ alors $\overline{w}_1 = \psi(s) \downarrow$ et $\overline{w\uparrow} = \psi(s)$ et $v_2 = s$
sinon $\overline{w\uparrow} = \psi(v_1) \uparrow$ et $v_2 = v_1\ j$ pour un certain j

(4) cas $f = \downarrow$

si $v_1 = s\ 1$ alors $\overline{w}_1 = \psi(s) \uparrow$ et le noeud courant était le 1^{er} fils de son père donc $\overline{w\downarrow} = \psi(s)$ et $v_2 = s$.
sinon $\overline{w\downarrow} = \psi(v_1) \downarrow$ et $v_2 = v_1\ \#$

C.Q.D.F.

Lemme 3 : Si $f \in F$ est la dernière action de la trace réduite i.e. $\overline{w} = a\ f$ alors $\hat{f}$ annule cette action i.e. $w\ \hat{f}(v, t) = a(v, t)$ *sauf si $f = \uparrow$ appliquée sur un noeud courant qui n'est pas le 1^{er} fils de son père.*

Soit $(v'_1, t'_1) = a(v, t)$ et $(v_2, t_2) = f(v_1, t_1)$ et $(v_3, t_3) = \hat{f}(v_2, t_2)$. Il s'ensuit que $a = \psi(v_1)$ et $a\ f = \psi(v_2)$.

(1) cas $f = \rightarrow$. Nous avons $v_2 = b\ \overrightarrow{i}$. Il est évident que $t_3 = t_1$.
si $v_2 = b\ \overrightarrow{1}$ alors $v_1 = b$ car a f est réduit. De plus l'action de $\leftarrow$ entraîne $v_3 = b$

si $v_2 = b\ \overrightarrow{i}$ avec $i \neq 1$ alors $v_1 = b\ \overrightarrow{i-1}$ et l'action de $\leftarrow$ entraîne $v_3 = b\ \overrightarrow{i-1}$

(2) cas f = $\leftarrow$ le cas est similaire à $\rightarrow$.

(3) cas f = $\downarrow$. Nous avons $v_2 = v_1 \#$. Il est évident que $t_3 = t_1$ et l'action de $\uparrow$ entraîne alors $v_3 = v_1$.

(4) cas f = $\uparrow$. Nous avons $v_2 = v_1\ j$ et le noeud courant de t_1 est le $j^{ème}$ fils de celui de t_2. Si j = 1 alors $v_3 = v_1$ et $t_3 = t_1$.

Notons que $j \neq 1$ entraîne $v_3 = v_2 \#$ et le noeud courant de t_3 est le 1^{er} fils de celui de t_2.

Corollaire 4 : *L'action de* $w \in F^*$ *sur un couple* (Λ, t) *est la même que l'action de la trace réduite* $\overline{w}$, *i.e.* $w(\Lambda, t) = \overline{w}(\Lambda, t)$.

Théorème : *La fonction retour annule la dernière action de la trace réduite, i.e. si* $\overline{w} = a\ f$ *alors* **retour**$(w(\Lambda, t)) = a(\Lambda, t)$.

Notons $(v_1, t_1) = a(\Lambda, t)$ et $(v_2, t_2) = f(v_1, t_1) = w(\Lambda, t)$.

Nous avons $\overline{w} = \psi(v_2)$.

(1) cas f = $\rightarrow$. Nous avons $v_2 = b\ \vec{i}$ et la fonction retour est identique à $\leftarrow$.

(2) cas f = $\leftarrow$. Nous avons $v_2 = b\ \overleftarrow{i}$ et la fonction retour est identique à $\rightarrow$.

(3) cas f = $\uparrow$. Nous avons $v_2 = v_1\ i$, et quel que soit i, $v_3 = v_1$ et le noeud courant devient le $i^{ème}$ fils du précédent noeud courant i.e. le noeud courant de t_1 et $t_3 = t_1$.

(4) cas f = $\downarrow$. Nous avons $v_2 = v_1 \#$ et la fonction retour est identique à $\uparrow$.

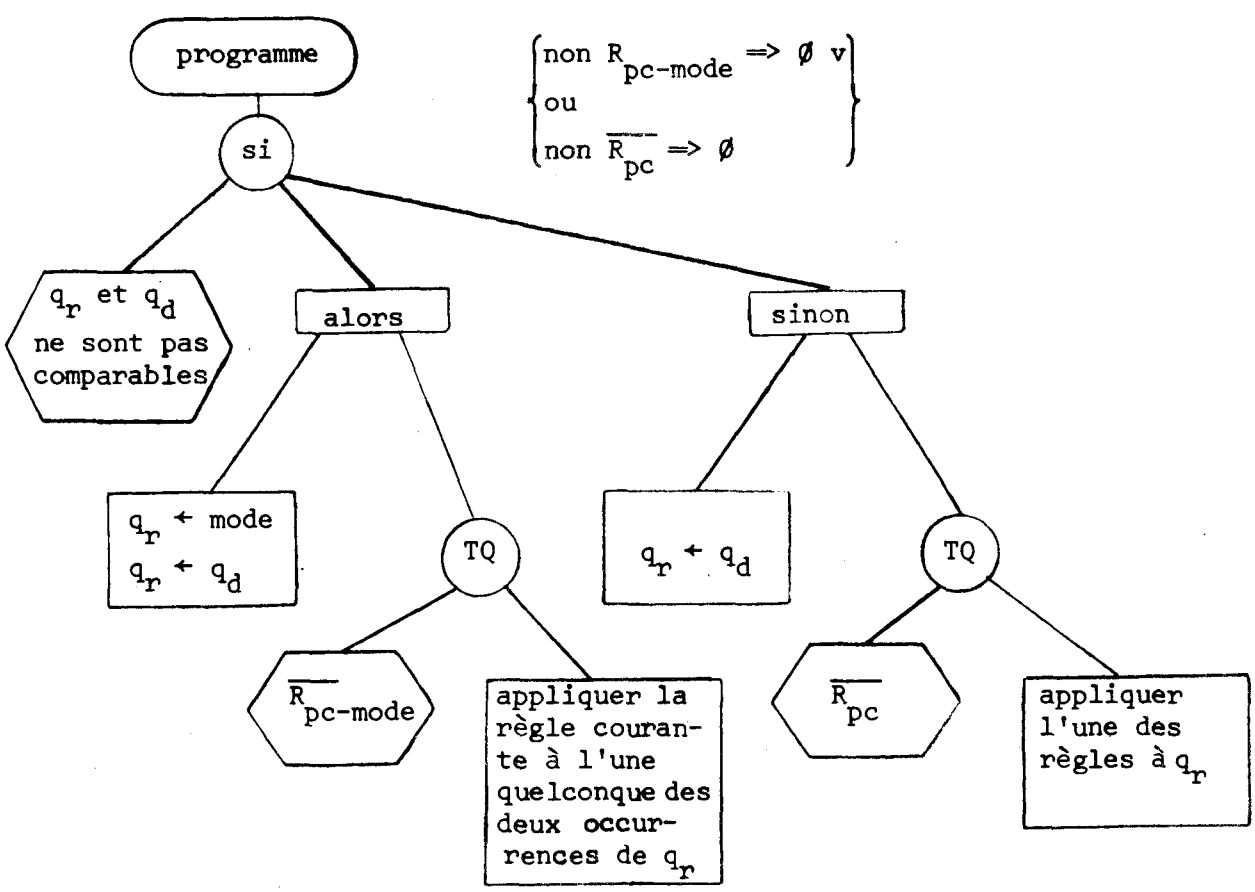
C.Q.F.D.

VI.6. EXEMPLE D'IMPLEMENTATION

Quand on définit un langage, il est raisonnable de le spécifier en propriétés logiquement cohérentes, hors de tout souci d'implémentation, à charge pour l'implémenteur de jouer de variations correctes selon les circonstances matérielles.

Ainsi, nous définissons le "1er NOEUD COMMUN" par des règles logiquement parlantes, mais qui, implémentées comme telles, engendrent un comportement non déterministe. Nous donnons un choix - il y en a bien d'autres - déterministe et efficace d'implémentation dans le même formalisme, ce qui nous permet de prouver cette implémentation.

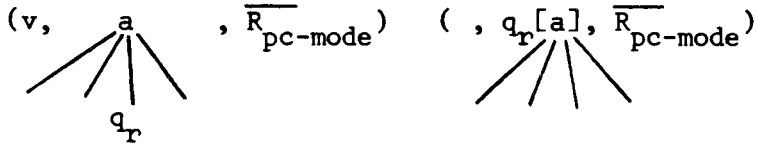
Le transducteur non déterministe de la fonction "1er NOEUD COMMUN" est donné en VI.4.3.3. et il lui correspond le programme non déterministe suivant :



Lorsque le programme s'arrête, le noeud obtenu est le noeud désiré. Le programme peut ne pas s'arrêter, ou être bloqué (non déterministe).

Dans le premier cas, mode et q_d ne sont pas ascendants l'un de l'autre. Le noeud N cherché est tel que N est le $i^{ème}$ ascendant de x_{mode} et le $j^{ième}$ de λ_{q_d} et tel que le $(i-1)^{ième}$ ascendant de x_{mode} n'est pas ascendant de λ_{q_d}. Idem pour le $(j-1)^{ième}$ ascendant de λ_{q_d} non-ascendant de x_{mode}.

On peut montrer aisément que la règle



appliquée i fois sur q_r provenant de x_{mode} et j fois sur q_r provenant de λ_{q_d} conduit au résultat.

Le deuxième cas se démontre de façon similaire.

Exemple d'implémentation

Pour ce transducteur non déterministe, il existe un programme non déterministe indépendant de l'implémentation. Nous allons montrer l'équivalence de ce transducteur (et du programme) avec un transducteur et le programme déterministe pour un exemple d'implémentation. Il existe des méthodes théoriques générales pour montrer l'équivalence de certaines classes de transducteurs, néanmoins, nous faisons ici une preuve artisanale à titre d'illustration.

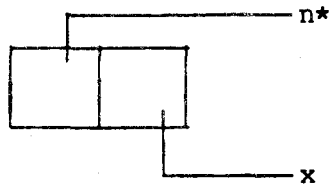
Supposons que les noeuds des arborescences soient représentés par lien horizontal, lien vertical et lien vertical inverse (Pair et Gaudel). On rappelle que le lien horizontal relie les frères entre-eux, le lien vertical relie le père à son premier fils, et le lien vertical inverse relie le fils à son père.

Un noeud est représenté par :

LH	LV	LV^{-1}	informations du noeud

De plus, le contexte est représenté par un couple : le noeud contexte, i.e., celui qui est visualisé noté "nc" ou "n*", et le noeud courant sur lequel est pointé l'utilisateur, il est noté x.

Le contexte est représenté par :



Des transducteurs déterministes permettent de décrire la recherche du premier noeud commun ; un transducteur est l'écriture abstraite d'un programme déterministe ; nous en présentons un et prouvons le programme déterministe correspondant par l'équivalence avec le programme non déterministe.

Dans l'arbre, les repères des noeuds ont la signification suivante :

- # noeud courant (abréviation de mode)
- ! noeud marqué entre noeud courant et noeud contexte
- * noeud contexte
- ? noeud désigné ou noeud ascendant du noeud désigné (λ)

Ce sont les gardes qui assurent le contrôle du programme.

Le transducteur s'écrit

$r_1 : (\#[*]\ ?\ x, \emptyset) \rightarrow (\#[*]\ x, \emptyset)$

$r_2 : (?\ n, \emptyset) \rightarrow (!\ ?\ n, q_1)$

$r_3 : (\#[*]\ !\ n, q_1) \rightarrow (\#[*]\ !\ n, q_2)$

$r_4 : (*\ !\ n, q_1) \rightarrow (*\ !\ n, q_4)$

$r_5 : \left(\begin{matrix}[\#][*]\ n \\ !\ [?]\ z\end{matrix}, q_1\right) \rightarrow \left(\begin{matrix}[\#][*]\ !\ n \\ !\ [?]\ z\end{matrix}, q_1\right)$

$r_6 : (!\ ?\ n, q_2) \rightarrow (?\ n, q_3)$

$r_7 : (\#[*]\ ?\ n, q_3) \rightarrow (\#[*]\ n, \emptyset)$

$r_8 : \left(\begin{matrix}[\#][*]\ !\ n \\ ?\ x\end{matrix}, q_3\right) \rightarrow \left(\begin{matrix}[\#][*]\ ?\ n \\ x\end{matrix}, q_3\right)$

$r_9 : (\#[*]\ !\ n, q_4) \rightarrow (\#[*]\ !\ n, q_5)$

$r_{10} : \left(\begin{matrix}[*][!]\ n \\ \#\ x\end{matrix}, q_4\right) \rightarrow \left(\begin{matrix}\#[*][!]\ n \\ x\end{matrix}, q_4\right)$

$r_{11} : (!\ ?\ n, q_5) \rightarrow (?\ n, q_6)$

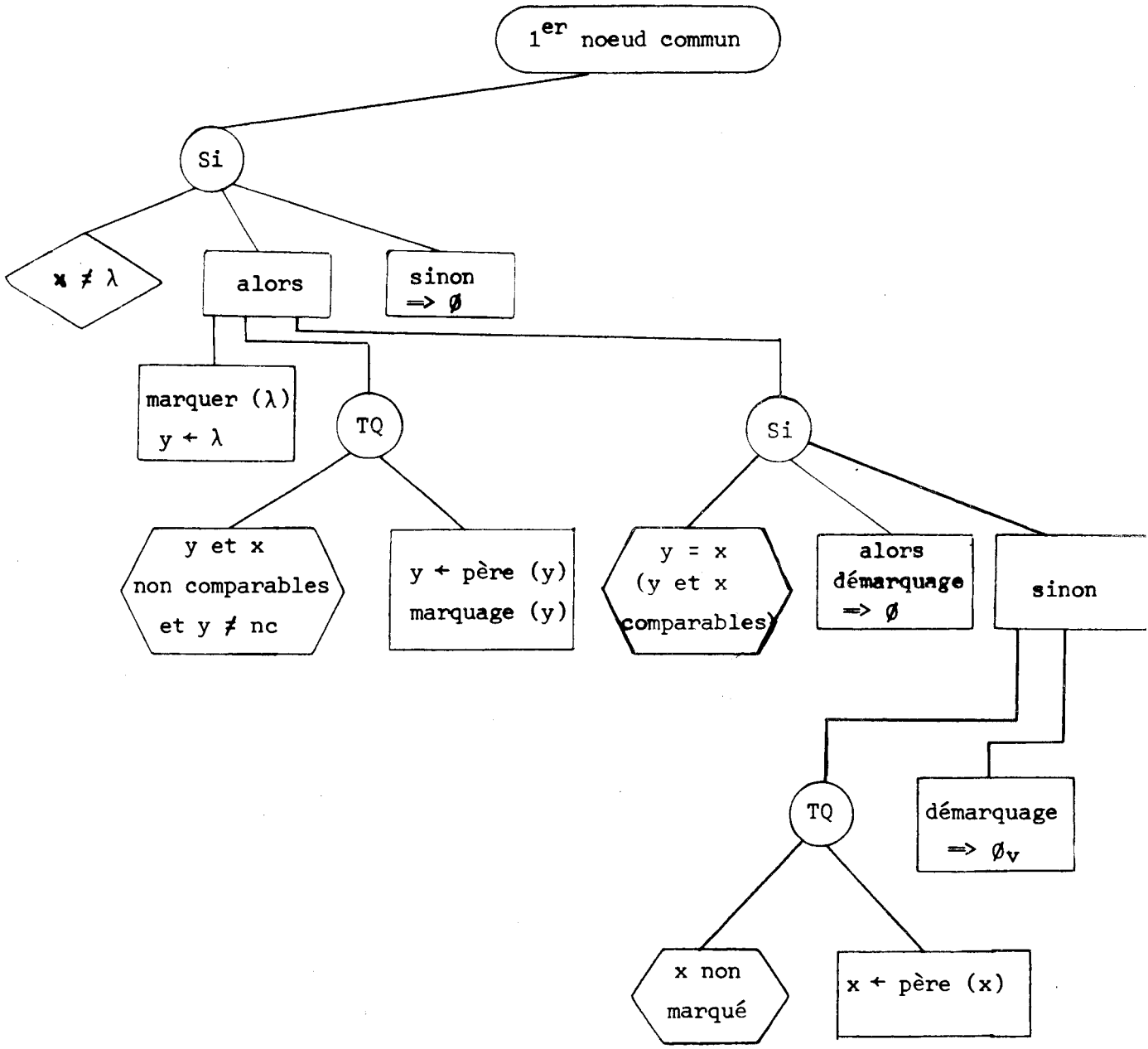
$r_{12} : ([\#]\ *\ ?\ n, q_6) \rightarrow ([\#]\ *\ n, \emptyset v)$

$r_{13} : \left(\begin{matrix}[\#][*]\ !\ n \\ ?\ x\end{matrix}, q_6\right) \rightarrow \left(\begin{matrix}[\#][*]\ ?\ n \\ x\end{matrix}, q_6\right)$

Un programme correspondant s'écrit

```
si λ ≠ x
alors marquer (λ) ; y ← λ
      tant que y ≠ x
            et y ≠ ne
      faire
            y ← père (y)
            marquer (y)
      fait
      si y = x
      alors démarquer (λ) ; y ← λ
            tant que y <> x
            faire y ← père (y)
                  démarquer (y)
            fait
      sinon co noeud cherché ≠ x co
            tant que x non marqué
            faire x ← père (x)
            fait
            démarquer (λ) ; y ← λ
            tant que y <> ne
            faire y ← père (y)
                  démarquer (y)
            fait
      fsi
fsi
```

Ce programme est déduit du programme écrit sous forme d'arbre programmatique :



Si on suppose que l'algorithme non déterministe exhibe la solution unique, il suffit de montrer que le noeud trouvé par l'algorithme déterministe est solution de l'algorithme non déterministe.

si I Partant de l'algorithme déterministe, chaque fois qu'on applique r_{5d} (sur y), on applique r_{3nd}, puis il suffit d'appliquer r_{3nd} sur x tant que cela est possible et enfin r_{4nd}.

si II Chaque fois qu'on applique r_{5d} (sur y), on applique r_{6nd}, il suffit d'appliquer r_{7nd} pour obtenir le résultat.

si III Le noeud désigné est le noeud courant, la règle r_{1d} est identique à la règle r_{8nd}.

L'algorithme déterministe correspond bien à une façon particulière d'appliquer l'algorithme non déterministe ; cette façon remettant le système dans l'état $\emptyset v$ (cas I) ou $\emptyset$ (cas II et III), l'algorithme fournit bien la solution cherchée.

A partir du même algorithme non déterministe, on pourrait définir d'autres programmess déterministes ; nous en donnons un autre exemple en laissant au lecteur la démonstration de l'équivalence (on notera que l'équivalence n'est pas tout-à-fait parfaite, car le programme s'achève sur un état $\emptyset v$ dans tous les cas (et non pas $\emptyset$ dans les cas II et III).

Le transducteur s'écrit	Un programme correspondant s'écrit
$r_1 : (\#\,[*]\,[?]\,x, \emptyset) \rightarrow (\#\,!\,[*][?]\,x, q_1)$	$y \leftarrow x$; marquer (y)
$r_2 : ([\#]\,!\,*\,[?]\,n, q_1) \rightarrow ([\#]\,![*][?]\,n, q_2)$	tant que y <> ne
	faire
$r_3 : \left(\begin{array}{c}[*][?]\,n \\ [\#]\,!\,[?]\,z\end{array}, q_1\right) \rightarrow \left(\begin{array}{c}![*][?]\,n \\ [\#]\,!\,[?]\,z\end{array}, q_1\right)$	$y \leftarrow$ père (y)
	marquer (y)
	fait
$r_4 : ([\#]\,!\,[*]\,?\,n, q_2) \rightarrow ([\#]\,!\,[*]\,?\,n, q_3)$	tant que λ non marqué
	faire
$r_5 : \left(\begin{array}{c}[\#]\,!\,[*]\,n \\ ?\,z\end{array}, q_2\right) \rightarrow \left(\begin{array}{c}[\#]\,!\,[*]\,?\,n \\ z\end{array}, q_2\right)$	$\lambda \leftarrow$ père (λ)
	fait
$r_6 : (\#\,!\,*\,[?]\,n, q_3) \rightarrow (*\,[?]\,n, q_4)$	tant que x <> ne
	faire
$r_7 : \left(\begin{array}{c}!\,[*]\,[?]\,n \\ \#\,!\,[?]\,x\end{array}, q_3\right) \rightarrow \left(\begin{array}{c}\#\,!\,[*][?]\,n \\ [?]\,x\end{array}, q_3\right)$	démarquer (x)
	x $\leftarrow$ père (x)
	fait
	démarquer (x)
$r_8 : ([*]\,?\,n, q_4) \rightarrow (\#\,[*]\,n, \emptyset v)$	x $\leftarrow \lambda$

Conclusion

Les transducteurs que nous avons définis sont indépendants du choix de l'implémentation, et ils se traduisent aisément en termes de transducteurs déterministes dépendant de l'implémentation souhaitée. Ces derniers sont quasiment immédiatement traduisibles en programmes. Ces outils forment ainsi une aide considérable à la conception de programmes, et assurent une grande modularité des fonctions et donc une fiabilité accrue, car la complexité s'en trouve réduite de façon importante.

Un travail important de définition de l'ensemble de toutes les fonctions prévues pour l'éditeur graphique reste à faire. Il s'agit des fonctions de création, de modifications de structure, des transformations géométriques, des fonctions d'accès aux objets catalogués, des fonctions de visualisation des objets-patrons... Ces travaux seront étudiés selon les principes de base que nous avons présentés ici, principes assez simples pour mener à bien efficacement l'étude proposée.

CONCLUSION

Nous avons défini un logiciel visant à satisfaire au maximum l'utilisateur, en nous basant sur les concepts d'élaboration et de manipulation d'images qui nous semblent les plus naturels. Par ses critères ergonomiques, un tel éditeur pourrait être utilisable facilement par des non-spécialistes en informatique.

De façon à préciser ce logiciel et ôter toute ambiguïté dans sa définition, nous avons proposé un début de formalisation qui soit un guide rigoureux pour l'implémenteur et pour l'usager.

Les études doivent maintenant se poursuivre dans la formalisation de l'ensemble des fonctions : elle se réalise de façon analogue à celle détaillée dans le chapitre VI, et n'est guère plus complexe : certaines fonctions comme le déplacement local, l'insertion, la suppression s'écrivent à l'aide de règles n'utilisant qu'un seul arbre (comme pour les fonctions d'accès), tandis que d'autres comme le rattachement et le clivage s'écrivent à l'aide de règles similaires, qui s'appliquent sur une forêt (et non plus sur un seul arbre).

Des fonctions complémentaires liées à la gestion des objets (sauvegarde externe, gestion des données en mémoire centrale...) doivent également être étudiées et formalisées. Il en résultera la détermination d'un langage pour l'implémentation de l'éditeur d'images. L'expérimentation permettra alors de vérifier la concordance avec les concepts initiaux, et d'étendre éventuellement le logiciel.

TABLE DES MATIERES

BIBLIOGRAPHIE

BAECKER, *"Towards an effective characterization of graphical interaction"*. Methodology of Interaction, North Holland (1980), pp. 127-148.

BEAUCOURT et al., *"LGIVD : Logiciel graphique industriel de visualisation et dialogue"*. Actes du Congrès AFCET, (nov. 1980).

BERGERON, *"A graphics programming methodology"*. Cours d'informatique graphique à Nantes,(1-12 Décembre 1980).

BERTIN, *"La graphique et le traitement graphique de l'information"*. Flammarion (1977).

BORRON, *"Classification et sélection de documents"*. Projet pilote KAYAK. Bulletin de liaison de la recherche en informatique et automatique n° 69, (1981).

BOUDOL et KOTT, *"Recursion induction principle revisited"*. Theory Computer Science (1981). [3]

BOUTRY, *"Structure de documents, langages de requête et de manipulation pour la machine de classement TREFLE"*. Actes des journées sur la burotique (Mars 1981), Agence de l'Informatique.

BRADIER, *"Elements pour la représentation de données graphiques dans un contexte interactif"*. Mémoire D.E.A. (1981).

CROCUS, *"Systèmes d'exploitation des ordinateurs"*. Dunod. Phase spécialité informatique (1975).

COURCELLE, *"Attribute grammars : theory and applications "On formalization of programming concepts"*. Lecture Note in Computer Science, Springer Verlag (1981). [8]

DAUCHET, *"Transductions de forêts. Bimorphismes de magmoïdes"*. Thèse d'Etat, Université de Lille (1977). [9]

ENCARNACAO, *"A recommendation on methodology in Computer Graphics"*. (1976).

ENGELFRIET, *"Some open questions and recent results on tree transtransducers and tree languages"*. Symp. on formal languages. Théory. Santa Barbara, California Academic Press, (1979). [6]

ENGELFRIET, *"Bottom up and Top-Down tree transformation a comparison"*. Math-System Theory, vol. 9, (1975), pp. 198-231. [6 bis]

ENGELFRIET, *Tree transducers and syntax directed semantics"*. 7ème CAAP, Lille 1982 (à paraître). [7]

FEINER, VAN DAM, *"Interactive graphical documents"*. International workshop on office information systems, (Oct. 1981), pp. 110-115.

FITZGERALD et al., *"GRIN : interactive graphics for modeling solids"*. IBM Journal of Research and Development, vol. 25, n° 4, (July 1981), pp. 281-294.

FOLEY, WENNER, *"Core System Implementation"*. Computer Graphics, vol. 15, n° 3, (1981).

GOGUEN et al., *"Initial algebra semantics and continuous algebras"*. JACM 24 (1977), pp. 68-95. [5]

GRAVE, *"Etude d'un noyau de système de synthèse d'images. Application à la visualisation de scènes tridimensionnelles"*. Thèse de Docteur-Ingénieur, (1980).

GREEN, *"A methodology for the specification of graphical user interface"*. Computer Graphics, vol. 15, n° 3, (1981).

GSPC, *"Graphics Standards Planning Commitee (ACM/SIGGRAPH) Second status report"*. Siggraph proceedings (1979).

GUEDJ, *"Methodology of interaction"*. North Holland (1980).

GUESSARIAN, *"About stack tree automata"*. 7ème CAAP, Lille 1982. (à paraître). [4]

HAGEN, *"The intermediate language for pictures"*. Proceedings IFIP. Congress, pp. 173-178, (1977).

HARDWICK, *"Graphical data structures"*. Computer Graphics, vol. 15, n° 4, (1981).

HERZOG, *"Functional specification for a core graphics system"*. Ecole d'été d'informatique, (1979).

KILGOUR, *"A hierarchical model of a graphics system"*. Computer Graphics, (1980), pp. 35-47.

KOTT, *About transformation system : "a theoretical study" in "program transformations""*. R. Robinet, Ed. Dunod, (1978), pp. 232-247. [2]

LAKIN, *"A structure from manipulation for text-graphic objects"*. ACM-SIGGRAPH (1980).

LEDUC-LEBALLEUR, *"Conception et réalisation d'un logiciel graphique de base indépendant de son contexte. Application au logiciel GRIGRI"*. Thèse de Docteur-Ingénieur, (1977).

LILLIN, *"Transducteurs finis d'arbres et tests d'égalité"*. RAIRO. Thoretical Informatics, vol. 15, n° 3, (1981), pp. 213-232. [10]

LUCAS, *"Contribution à l'étude des techniques de communication graphique avec un ordinateur. Eléments de base des logiciels graphiques interactifs"*. Thèse d'Etat. Université de Grenoble, (1977).

LUCAS, *"Some aspects of graphical Man Machine Communication"*. Cours d'informatique graphique. Nantes, (Déc. 1980).

MALGREEN, SHAW, *"Graphical transformations and hierarchic picture structures"*. Computer Graphics & Image processing, vol. 8, (1978), pp. 237-278.

MARTINEZ, *"An approach to the modelling and display of Landscapes"*. Proceedings of Eurographics, Bologna, Italy (1979).

MARTINEZ, *"CLOVIS : complexe logiciel pour la visualisation interactive structurée"*. Actes du congrès de l'AFCET, (Nov. 1980).

MERIAUX, *"Etude et réalisation d'un terminal graphique couleur tridimentionnel fonctionnant par taches (STYX)"*. Thèse de Docteur-Ingénieur, Université de Lille I, (1979).

MORVAN-LUCAS, *"Images et ordinateur. Introduction à l'infographie interactive"*. Larousse. Collection Sciences Humaines, Série Informatique (1976).

NAFFAH et al., *"La production de documents dans le projet KAYAK"*, Actes des journées sur la burotique (Mars 1981), Agence de l'Informatique.

NAFFAH, *"Le projet Pilote KAYAK, objectifs, axes de recherche et conduite"*. Les services burotiques : architecture et réalisation KAYAK. Actes des journées sur la burotique (Mars 1981), Agence de l'Informatique.

NAFFAH, *"Editing multitype document"*. International workshop on Office information systems, (Oct. 1981), pp. 116-131.

NEWMAN, *"High level programming for remote terminals"*. Pertinent concepts in Computer graphics. University of Illinois Press, pp. 200-223, (1969).

NEWMAN, SPROULL, *"Principles of interactive computer graphics"*. Mac Graw-Hill, 2nd Edition, (1979).

NEWMAN, *"Some notes on User Interface Design"*. Methodology of Interaction. North Holland (1980).

NIVAT, *"On the interpretation of recursive polyadic program schemes"*. Instituto Nazionale di Alta Mathematica, Symposia Mathematica, vol. XV, pp. 255-281, (1975). [9]

OLEJNIK, *"Realisme dans les images générées par ordinateur : ombrage, texture, transparence, ombres portées"*. Mémoire D.E.A. (Sept. 1982).

PAIR ET GAUDEL, *"Les structures de données et leur représentation en mémoire"*. INRIA, (1977).

RAUCH, GRIMONPREZ, *"Présentation du logiciel STYX"*. Journées AFCET, (Mai 1979).

RUBIN, WHITTED, *"A 3-dimensional Representation for fast rendering of complex scenes"*. Bell laboratories. ACM/SIGGRAPH, (1980).

SAJANIEMI, *"Some difficulties with abstract and concrete syntax"*. Bit bind 17, (1977).

SCHEURER et al., *"Le buroviseur, poste de travail de demain"*. Actes des journées sur la burotique (Mars 1981), Agence de l'Informatique.

SHEPHERD, *"Experimental page makeup of text with graphics on a raster print"*. pp 345-355, IBM Systems Journal, vol. 19, n° 3, (1980).

STAUDHAMMER, *"Interactive Color Graphics Proposal"*. North Carolina State. University (August 1982).

VAN DAM, GURWITZ, THORNE, *"BUMPS : A program for Animating Projections 1980"*. ACM /SIGGRAPH, (1982).

VAN DEN BOS, *"High-Level Graphics Input tools and their Sémantics"*. pp. 159-170. Methodology of Interaction, North Holland, (1980).

VOIROL, *"Editeur graphique GRED"*. Actes du congrès de l'AFCET, (1981).

WARNOCK, *"A Hidden surface algorithm for computer generated halftone pictures"*. TR 4-15, Computer Science Department, University of Utah (1969).

WILLIAMS, *"On the application of relational data structures in Computer Graphics"*. Information Processing, North Holland, (1974).