Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion

Claude Delobel

▶ **To cite this version:**

Claude Delobel. Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion. Interface homme-machine [cs.HC]. Université Joseph-Fourier - Grenoble I, 1973. Français. NNT : . tel-00010408

HAL Id: tel-00010408
https://theses.hal.science/tel-00010408

Submitted on 5 Oct 2005

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

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

THESE

présentée à

l'université scientifique et médicale de grenoble

pour obtenir

le grade de docteur ès-sciences

par

Claude Delobel

---

contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion

---

Thèse soutenue le 17 octobre 1973 devant la commission d'examen

Président: *J. Kuntzmann*

Rapporteur: *J. Arsac*

Examinateurs: *L. Bolliet*

A. Hocquenghem

C. Pair

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BERNARD Alain	Mathématiques Pures
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Jean	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique

Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physilogie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale

MM.	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	HANO JUN-ICHI	Mathématiques Pures
	STEPHENS Michaël	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

MM.	BEAUDOING André	Pédiatrie
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des fluides
	DEPORTES Charles	Chimie minérale
	GAUTHIER Yves	Sciences biologiques
	GAVEND Michel	Pharmacologie
	GERMAIN Jean-Pierre	Mécanique
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	DE ROUGEMONT Jacques	Neurochirurgie
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LUU DUC Cuong	Chimie organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)

MM.	MAYNARD Roger	Physique du solide
	MICHOULIER Jean	Physique (IUT A)
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du solide
	PHELIP Xavier	Rhumatologie
Mle	RIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marquerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VAN CUTSEM Bernard	Mathématiques appliquées
	VEILLON Gérard	Mathématiques appliquées (INP)
	VIALON Pierre	Géologie
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM.	BOUDOURIS Georges	Radioélectricité
	CHEEKE John	Thermodynamique
	GOLDSCHMIDT Hubert	Mathématiques
	SIDNEY STUARD	Mathématiques Pures
	YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physilogie
Mme	RENAUDET Jacqueline	Microbiologie

Fait le 30 mai 1972.

Que les personnes qui m'ont conseillé, orienté ma réflexion, et contribué à l'aboutissement de ce travail, trouvent ici l'expression de ma reconnaissance et de ma gratitude :

J.R. ABRIAL, Ingénieur au Laboratoire d'Informatique de l'Université de Grenoble

J. ARSAC, Professeur à PARIS VI

C. BENZAKEN, Maître de Conférences à l'Université de Grenoble

L. BOLLIET, Professeur à l'Université de Grenoble

R.G. CASEY, I.B.M., San José Research Laboratory

N. GASTINEL, Professeur à l'Université de Grenoble

P. GUERIN, Directeur du C.N.A.M.

A. HOCQUENGHEM, Professeur au C.N.A.M.

J. KUNTZMANN, Professeur à l'Université de Grenoble

M. LEONARD, Ingénieur au Laboratoire d'Informatique de l'Université de Grenoble

P. MOREAU, Chef du Service Développement Scientifique d'I.B.M. PARIS

C. PAIR, Professeur à l'Université de Nancy II

E. PICHAT, Maître de Conférences au C.N.A.M.

Enfin, je tiens à remercier les personnes qui ont participé à la réalisation matérielle de cette thèse.

C. DELOBEL

PREMIERE PARTIE

DEUXIEME PARTIE

B I B L I O G R A P H I E

[1] CODASYL

"An information algebra"

Comm ACM 5, 4, pp 190-204, (1962)

[2] BATLER, FAIRBROTHER, GATTO

"Autosate", Rand Memorandum 3976 PR

(février 1964)

[3] F. PECCOUD

"Modsin", Contrat DGRST 65 FR 291

(juillet 1967)

[4] C. DELOBEL, D. GROUCHKO

"L'analyse des systèmes d'informations"

Rapport technique ASI/1-3/67, Centre Inter-Armées de Recherche Opérationnelle (mars 1967)

[5] Y. ARCHER, C. DELOBEL, F. PECCOUD

"Méthode générale d'analyse", Contrat DGRST 67 01 015

(juillet 1969)

[6] J.W. FORRESTER

"Industrial Dynamic"

Wiley an Son, 1962

[7] D.C. CHILDS

"Feasibility of a Set theoretic Data Structure"

Proc. IFIP, Congrès 1968, vol. 1, pp 420-430

[8] CODASYL

"A survey of generalized data management systems"

ACM, New York (1971)

[9] CODASYL

"Feature analysis of generalized data base management systems"

ACM, New York (1972)

[10] CODASYL

"Introduction to 'Feature Analysis of generalized data base management systems' "

CACM, 14,5 (may 1971)

[11] LEVIEN, MARON

"A computer system for inference execution and data retrieval"
Comm ACM 10, 11, pp 715-721 (novembre 1967)

[12] G. MEALY

"Another look at data"
Proc AFIPS,1967, FJCC, vol 31, pp 525-534

[13] A.J. STRNAD

"The relational approach to the management of data bases"
Proc. IFIP, 1971

[14] E.F. CODD

"A relational model of data for large shared data banks"
Comm ACM 13, 6, pp 377-386 (juin 1970)

[15] J.R. ABRIAL

"Projet Socrate, cours d'architecture des systèmes"
Alpe d'Huez (décembre 1972)

[16] J. BOITTIEAUX-ZIDANI

"Etude mathématique des relations d'un ensemble de notions"
Contrat DGRST 65 FR 201 (juillet 1967)

[17] J. KUNTZMANN, C. BENZAKEN

"Théorie des relations"
en cours de publication, chapitres 2 et 3

[18] A. BOUCHER

"Etude combinatoire des ordonnés finis"
Thèse de Doctorat ès Sciences, Grenoble, (mai 1971)

[19] C. DELOBEL, R.G. CASEY

"Decomposition of a data base and the theory of boolean switching functions"
IBM RJ 1016 (avril 1972)

[20] E. PICHAT

"Contribution à l'algorithmique non numérique dans les ensembles ordonnés"
Thèse de Doctorat ès Sciences, Grenoble (octobre 1970)

[21] E.F. CODD

"A relational algebra for data base system"

IBM Research, Technical report

[22] M.E. SENKO, E.B. ALTMAN, M. ASTRAHAM, P.L. FEDHER

"Data Structure and accessing in data systems"

IBM Systems Journal, 12, 1, 1973

[23] E.F. CODD

"A data base sublanguage formed on the relational calculus"

Proc. 1971, ACM SIGFIDET, Workshop on data description, access and control

[24] E.F. CODD

"Relational completeness of data bases sublanguages"

IBM Research, RJ 987 (march 1972)

[25] W. ASCH, E.H. SIBLEY

"An interpretive associative processor with deductive capabilities"

Proc. ACM 23rd Nat. Conf. (1968)

[26] E.F. CODD

"Further normalization of a data base relational model"

IBM Research report, RJ 909 (august 1971)

[27] R.E. MILLER

"Switching theory"

Vol 1, Wiley, New-York, 1965

[28] I.R.E.P.

"Thesaurus de l'économie et de l'énergie"

Grenoble, 1973

[29] J.A. FELDMAN, P.D. ROVNER

"An Algol base associative language"

Communication ACM, vol 12, n° 8, Août 1969, pp 439-449

[30] C. HEWITT

"PLANNER : A language for manipulations models and proving theorems in a robot"

M.I.T., Ph D, AI Technical report 258

[31] V.Y. LUM

"Multi-attribute retrieval with combined indexes"

CACM 13, 11 (1970) pp 660-665

[32] D.E. KNUTH

"The art of computer programming"

Vol.3, Sorting and Searching

[33] AFCET

"Algol 68"

édité par P. BACCHUS, C. PAIR, D. PECCOUD

[34] J. EARLEY

"Toward an understanding of data structures"

Université de Berkeley, février 1970

[35] C. DELOBEL

"Aspects théoriques sur la structure de l'information dans une base de données"

R.I.R.O., 5ème année, B-3, 1971, pp 37-64

[36] C. DELOBEL

"A theory about data in an information System"

IBM Research Report, RJ 964 (janvier 1972)

[37] C. DELOBEL, J. RISSANEN

"Decomposition of files, a basis for data storage and retrieval"

IBM Report, San José, (décembre 1971)

[38] IMS

"Information Management System"

IBM Form. NO GH 20-0765

[39] M. TREHEL

"Etude des treillis de catalogues"

R.I.R.O., 1ère année, n° 4, 1967, pp 39-49

[40] M. TREHEL

"Catalogues, treillis de catalogues, génération et décomposition"

Mathématiques et Sciences Humaires, n° 40, 1972

AVANT-PROPOS

Le concept de système d'informations d'entreprise est encore flou et il consiste essentiellement, dans l'état actuel de nos connaissances, en un ensemble de méthodes et d'outils manuels ou automatisés, permettant à l'utilisateur de contrôler l'activité économique de l'entreprise. Le développement de l'informatique a entraîné le développement d'outils automatisés, parmi lesquels on peut citer, les programmes spécifiques d'une application de gestion : paie, production, comptabilité, etc..., les programmes de définition, création et mise-à-jour d'une base de données. Parallèlement on a assisté à la création de méthodes d'analyse dont le but originel était d'aider l'utilisateur dans la définition et la construction du système d'informations, mais les faits ont montré que la plupart de ces méthodes ont consisté en une aide à la maintenance des programmes.

Une caractéristique commune à tous ces outils, c'est que chacun peut être considéré comme un évènement isolé ayant sa propre terminologie. Ceci a pour conséquences :

- qu'il est difficile de comparer ces outils entre eux,
- de rendre plus difficile les progrès nouveaux dans la conception de nouveaux systèmes.

De plus, il est pratiquement impossible de mesurer quantativement la valeur du système d'informations; la mesure devant se faire à deux stades, le premier concernant les performances techniques du système, le second, beaucoup plus difficile, ayant trait à l'évaluation globale du système d'informations.

Notre but est d'approfondir cette notion de système d'informations en présentant un modèle pouvant être aussi bien utilisé dans la description d'un système d'informations que dans la construction d'une base données.

CHAPITRE 1

INTRODUCTION

Le problème de l'étude des systèmes d'informations d'entreprise a relevé jusqu'à maintenant d'approche plus expérimentale que formelle. Les raisons de cette situation tiennent essentiellement au fait qu'il est difficile d'avoir simultanément un modèle qui représente le système de fonctionnement de l'unité économique et le système d'informations.

Les premiers outils dont l'utilisateur a disposé pour l'analyse d'un système d'informations ont été les graphes de circulation de documents utilisés par les organisateurs. Ces outils, du point de vue de l'information, présentent plusieurs inconvénients :

- ils décrivent seulement la circulation des documents et non pas celle de l'information,
- ils n'expriment pas les relations entre informations,
- ils ne montrent pas qu'elles sont les opérations de traitement qui sont effectuées sur ces documents, mis à part des traitement élémentaires tels que : transcription, report, duplication, contrôle,
- il est difficile de reconstruire le modèle de l'unité économique à partir du graphe de circulation des documents.

La première tentative de formalisation des traitements a été proposé par le comité CODASYL [1] avec une algèbre de l'information. D'un autre côté plusieurs modèles automatisés ont été construits à partir de la notion de graphe de circulation de documents, on peut citer AUTOSATE [2] et MODSIN [3]auquel nous avons participé [4] , [5].

A l'opposé des graphes de circulation de documents les travaux de J.W. FORRESTER avec "Industrial Dynamic" [6] portent sur la construction de modèle d'entreprise où le réseau d'informations joue un rôle important mais qui est considéré de façon trop synthétique.

Le problème fondamental dans la représentation d'un système d'informations est de choisir un modèle exprimant les relations entre informations. L'article de CHILDS [7], et les propositions du comité CODASYL [8], [9], [10] et le projet UIS [22] présentent une structure logique fondée sur les ensembles. LEVIEN et MARON [11], MEALY [12], STRNAD [13], CODD [14]et ABRIAL [15] ont utilisé le concept mathématique de relations. Notre approche est voisine de celle de CODD et utilisera le concept de collection de données que nous développerons à partir du travail mathématique de J. BOITTIEAUX [16] et, plus récemment, de la théorie des relations (cf. KUNTZMANN et BENZAKEN [17]).

Les essais de formalisation pour présenter les propriétés de l'information sont peu nombreux et les articles référencés ci-dessus constituent probablement les tentatives actuelles dans ce domaine avec [35], [36], [37].

La principale difficulté dans cet essai de formalisation est de définir les concepts de base en des termes mathématiques précis et rigoureux. En effet, l'état actuel de la formalisation des connaissances dans ce domaine est parfois éloigné des faits élémentaires que nous pouvons observer. Cette difficulté nous l'avons ressentie nous-mêmes, il en résultera que notre présentation sera dans certains cas formels, et dans d'autres cas décrira les faits tels qu'ils peuvent être observés.

Nous souhaitons que le lecteur comprenne que les formalisations mathématiques proposées ne sont pas une fin en soi mais peuvent servir comme une étape vers de meilleures définitions de la notion de système d'informations.

Pour mieux situer le cadre général de notre travail et les résultats auxquels nous sommes parvenus, nous établirons un rapprochement avec les différentes étapes de la mise en oeuvre d'un système d'informations.

Le processus de conception d'un système de l'information peut être divisé en plusieurs phases comme l'indique la figure 1.1. :

- l'application et son environnement,
- la structure de l'information,
- l'organisation logique des données,
- l'organisation physique des données,
- la structure de la mémoire.

Ce processus suppose que l'on se place dans le cadre de moyens automatisés, en cas de moyens manuels, certaines étapes n'existent pas, d'autres sont très simplifiées.

A la périphérie, l'application et son environnement qui vont conditionner toutes les autres étapes, au centre la mémoire, organe physique permettant de stocker des grandes masses de données, comment peut-on passer de l'application à la mémoire ? Le cheminement est complexe et chaque étape est caractérisée par un certain nombre d'opérations que nous allons décrire.

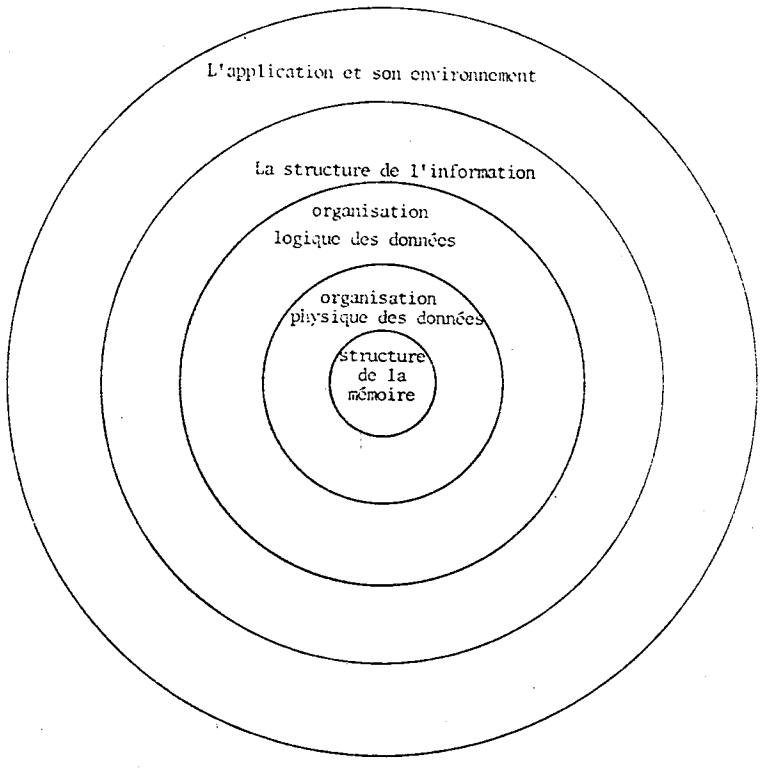


FIGURE 1.1. : Les différentes étapes dans la conception d'un système d'informations

L'application et son environnement

Le point de départ pour décrire l'application sera la notion de collection de données (chapitre 2) qui exprimera les relations entre les informations du système. Disons brièvement qu'une collection de données est un ensemble d'entités de même nature. Chaque entité est un ensemble de données qui peut représenter par exemple des faits, des objets concrets ou abstraits, des associations, des évènements. A partir de ce concept on définira un ensemble d'opérations permettant de transformer des collections de données en de nouvelles collections de données, parmi l'ensemble de ces opérations on peut citer : la composition normale, la R-composition normale, la mise-à-jour (chapitres 3 et 4). A partir de ces opérations fondamentales on obtiendra par extension un langage de manipulation de collection de données (chapitre 5). Le but de ce langage est d'exprimer les traitements de l'information sous une forme globale de telle sorte que les notions d'ordre et d'entrée/sortie n'alourdissent pas la description du système d'informations.

Avec le chapitre 6 nous proposerons un modèle pour la description du système d'informations qui répond aux objectifs suivants :
- possibilité d'exprimer les relations entre les informations,
- description des traitements à l'aide d'un ensemble d'opérateurs,
- description du modèle économique supportant l'application.

La structure de l'information

La structure de l'information est définie comme une propriété intrinsèque de l'information dans une application, indépendante des supports et des méthodes qui sont utilisés pour représenter et enregistrer cette information.

Ces propriétés sont nombreuses mais deux catégories retiendront plus particulièrement notre attention : la catégorie des relations fonctionnelles, la catégorie des propriétés qui s'expriment à l'aide de l'opérateur de composition normale.

Ces deux propriétés sont fondamentales car nous montrerons qu'elles conditionnent l'organisation logique des données.

Les chapitres 7 et 8 seront consacrés à l'étude des propriétés des relations fonctionnelles. Le principal résultat auquel nous sommes parvenus a été de relier la classification expérimentale des relations de E.F. CODD à des propriétés spécifiques de l'espace engendré par un ensemble de relations fonctionnelles.

Une autre propriété est que deux ensembles distincts de relations fonctionnelles peuvent en fait décrire la même application. C'est ainsi que dans le chapitre 9 nous développons le concept de couverture minimale qui correspond à un noyau non redondant de relations fonctionnelles. La recherche du noyau se fera à partir d'une analogie entre une classe de fonctions booléennes et les relations fonctionnelles, cette propriété avait été signalée par J. BOITTIEAUX et BOUCHER [18]. La démonstration que nous donnerons de cette analogie débouchera sur un algorithme de construction [19].

Un des buts d'un système d'informations est de restituer ou de générer de nouvelles informations à partir de celles qui sont stockées. Nous pensons qu'il existe un lien entre la possibilité du système à créer de nouvelles informations et les relations fonctionnelles. Cette faculté sera exprimée par un indice de mesure (chapitre 10).

L'organisation logique des données

La modélisation d'un système d'informations est fondée sur la notion de collection de données. Dans la pratique, la représentation d'un système nécessite une famille de collections de données pour traiter un ensemble d'applications. Plus précisément nous désignerons par le mot "requête" une opération qui consiste à produire une nouvelle collection de données à partir d'une famille de collections de données.

A partir de là, le problème qui se pose est de savoir d'une part, si la connaissance que l'on peut avoir au sujet de la structure de l'information peut aider l'exécution de la requête et peut conditionner la famille de collections de données et d'autre part, si deux familles de collections de données distinctes peuvent apporter la même réponse à une requête.

Les problèmes qui sont évoqués ci-dessus sont abordés au cours des chapitre 11 et 12 en traitant du problème de la décomposition d'une collection de données et de ses applications. Nous montrerons qu'il peut exister plusieurs familles de collections de données pour traiter une même requête ce qui nous amènera à considérer que l'organisation logique des données est composée de deux parties, le niveau primaire qui est la représentation originelle de l'utilisateur, le niveau secondaire conçu en fonction des requêtes de l'utilisateur. De plus dans le chapitre 12, nous indiquerons le rapport qu'il existe entre les organisations hiérarchisées de données et une classe de décompositions: les décompositions arborescentes.

L'organisation physique des données

Avec cette étape nous faisons un pas supplémentaire vers la mémoire et la façon dont sont enregistrées les données. Pour cela,il est nécessaire de tenir compte à la fois des systèmes d'exploitation et de la structure des mémoires.

Dans notre travail ce point ne sera pas abordé; nous indiquerons seulement les prolongements qui pourraient être faits à partir de l'organisation logique des données.

Evolution d'un système d'informations

Un système d'informations est un phénomène qui évolue avec le temps par l'introduction de nouvelles données, par la mise à jour de certaines valeurs, par l'introduction de nouvelles relations entre informations. Il est donc important qu'un système informations puisse subir aisément des restructurations et que les opérations de mise à jour ne détruisent pas les propriétés intrinsèques de l'application.

Au cours du chapitre 13 nous montrerons le rôle important joué par le noyau des relations fonctionnelles dans le problème de l'intégrité du système d'informations et nous donnerons un algorithme permettant de préserver l'intégrité du système au cours d'une opération de mise à jour.

Quant au problème de la restructuration, il est beaucoup plus complexe, nous l'illustrerons à l'aide d'exemples pour montrer que des modifications dans la structure de l'information peuvent avoir des conséquences sur l'organisation logique des données.

En conclusion, le principal objectif de notre travail a été d'approfondir le concept de système d'informations, (le schéma de la figure 1.2 résume l'articulation des différents chapitres) en présentant un modèle de description et en étudiant plus particulièrement ce que nous appelons la structure de l'information. Les prolongements de notre travail peuvent se situer dans deux voies différentes :

- la première se situant dans la perspective de l'analyse de la dynamique d'un système d'informations et du développement de méthodes d'analyse fondé sur des éléments rationnels,

- la seconde portant sur la mise en oeuvre des concepts développés dans un système de base de données et plus particulièrement à partir des concepts d'indépendance des données et de préservation de l'intégrité de la base de données.

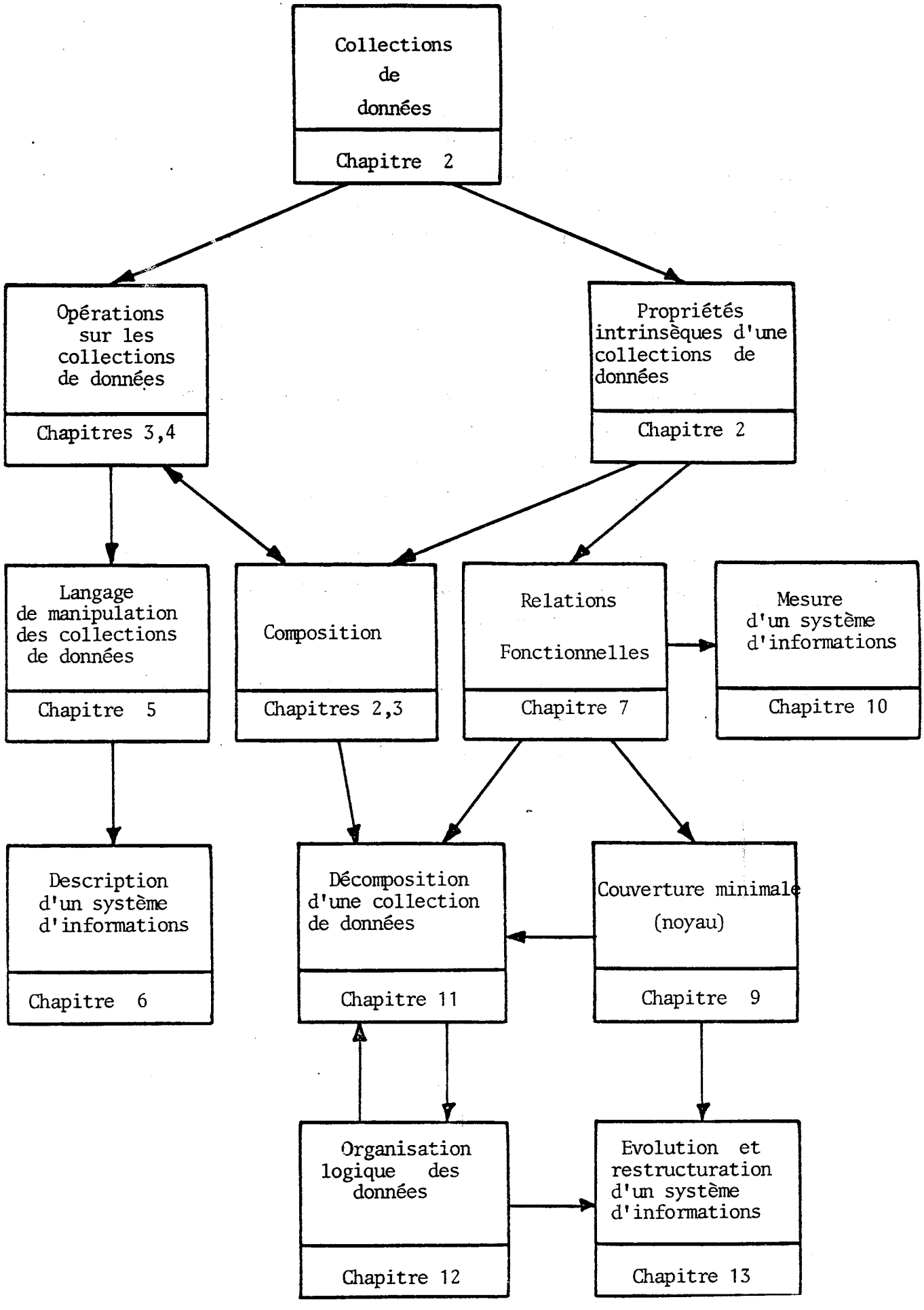


FIGURE 1.2. : ARTICULATION DES CHAPITRES

CHAPITRE 2

LES RELATIONS ET LES COLLECTIONS DE DONNEES

Le but de ce chapitre est d'introduire les principaux concepts qui seront utilisés au cours de notre travail. La notion fondamentale est celle de collection de données définie à partir de celle de relation n-aire. La collection de données est la représentation abstraite d'un fichier utilisé dans la construction d'un système d'informations qui permettra de définir un modèle de système d'informations.

Une caractéristique fondamentale des systèmes d'informations est celle d'invariant. Un invariant est une propriété intrinsèque du système d'information qui doit être vérifiée tout au long de la vie du système d'informations. Il existe de nombreux invariants, une famille retiendra particulièrement notre attention par les répercussions qu'elle aura dans la conception du système d'informations, la famille des relations fonctionnelles.

2. LES RELATIONS ET LES COLLECTIONS DE DONNEES

2.1. Les relations n-aires

2.1.1. Définition : Champ élémentaire

Nous appellerons champ élémentaire X une caractéristique d'un fait ou d'un objet suivant lequel on désire enregistrer des informations. Un champ élémentaire est un ensemble de valeurs, appelées des données, et pour exprimer que x est une donnée appartenant à X nous écrirons :

$$x \in X$$

la cardinalité de X sera notée :

$$\text{card } X \text{ ou } |X|$$

L'ensemble des champs élémentaires sera désigné par $\mathfrak{X}$. On fera l'hypothèse que tout champ élémentaire peut contenir trois données particulières :

- la valeur blanche ou inconnue notée "b"
- la valeur impossible notée "@"
- la valeur nulle notée "NUL"

Exemple 2.1.

NOM, SEXE, DEPARTEMENT sont des champs élémentaires :

NOM = {*dupont, durant, nimbus*}

SEXE = {*f, m*}

DEPARTEMENT = {*01, 02, 03, ..., 95*}

2.1.2. Définition : Champ composé

Il arrive fréquemment que les caractéristiques d'un fait ou d'un objet représentent l'association de plusieurs notions, c'est par exemple, le cas d'une date ou d'une adresse. D'une façon plus générale nous définirons un champ composé X comme une liste **finie** et ordonnée de champs élémentaires non forcément distincts. Nous noterons :

X = (U,V,T) où U,V,T sont des champs élémentaires et un élément $x \in X$ sera noté :

x = (u,v,t) où $u \in U$, $v \in V$, $t \in T$.

Notation : *Les lettres majuscules désigneront les champs et les lettres minuscules les données.*

Exemple 2.2.

ADRESSE = (NOM, NUMERO, RUE, VILLE, DEPARTEMENT)
DATE-FRANCAISE = (JOUR, MOIS, AN)
DATE-ANGLAISE = (AN, MOIS, JOUR)

2.1.3. Définition : Relation n-aire

On définira une relation n-aire à partir de :

- un ensemble de champs élémentaires,
- une liste ordonnée $X = (X_1, X_2, \ldots, X_n)$ de n champs élémentaires non forcément distincts,

comme étant une partie de $X_1 \times X_2 \times \ldots \times X_n$ que nous noterons :

$$[X_1, X_2, \ldots, X_n]\ R$$

ou encore de manière plus condensée :

$$[X]\ R\ .$$

La liste X sera appelée l'espace des champs de la relation R. Nous pouvons interpréter une relation n-aire par un autre modèle équivalent donnant un autre aspect de la notion de relation.

Etant donné un n-uplet $x = (x_1, x_2, \ldots, x_n)$ où $x \in X_1 \times X_2 \times \ldots \times X_n$ et un prédicat R dont l'évaluation sera notée :

$$[(x_1, x_2, \ldots, x_n)]\ R$$

alors :

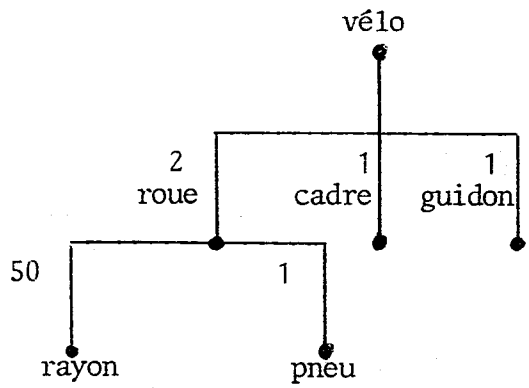
$$[X]\ R \triangleq \{(x_1, x_2, \ldots, x_n) : [(x_1, x_2, \ldots, x_n)]\ R = \text{'vrai'}\}$$

Exemple 2.3.

Etant donné deux champs élémentaires PIECE et ENTIER et le prédicat, permettant de construire une relation ternaire sur la liste ordonnée (PIECE, PIECE, ENTIER), exprimée par :

[(x,y,n)]R = 'vrai' si x ε PIECE comprend pour sa fabrication n fois (n ε ENTIER) la pièce y ε PIECE.

La figure 2.1. représente la composition d'un vélo et la relation ternaire associée.



PIECE	PIECE	ENTIER
vélo	roue	2
vélo	cadre	1
vélo	guidon	1
roue	rayon	50
roue	pneu	1

FIGURE 2.1

2.2. Les collections de données

2.2.1. Définition : Constituant élémentaire

L'exemple 2.3 nous a montré que le champ élémentaire PIECE intervenait deux fois dans la relation, ce qui signifie qu'il ne joue pas le même rôle dans la première position de la relation que dans la seconde position. Par exemple, la pièce "roue" est reliée à "vélo" en tant que composant, tandis que "roue" est composée de "pneu". En conséquence pour éviter aucune ambiguïté dans la description de la relation il est nécessaire de dénommer les différents rôles d'un même champ dans une relation.

Ce nouveau concept sera appelé un constituant élémentaire.

C'est ainsi qu'au champ élémentaire PIECE on associera deux constituants élémentaires : COMPOSE, COMPOSANT, tandis qu'au champ élémentaire ENTIER il ne serait pas nécessaire de préciser un nom de constituant puisque ce champ ne joue qu'un seul rôle dans la relation. Toutefois, par souci de précision, on conviendra de donner un nom de constituant, en l'occurence ici : QUANTITE.

Le paragraphe suivant a pour but de définir formellement la notion de collection de données et de constituant.

2.2.2. Définition : Collection de données

On appelle collection de données sur p champs élémentaires $X_1, X_2, \ldots, X_p$ un couple formé de :

(a) un ensemble E représentant par exemple des objets concrets ou abstraits des évènements. On peut dire que E est un ensemble d'éléments du monde réel,

(b) un ensemble $\mathcal{A}$ de n applications distinctes $A_i : E \rightarrow X_i \quad i = 1,2,\ldots,n$

ayant la propriété que pour tous éléments e, e' ε E, les n-uplets $(A_1(e), A_2(e), \ldots, A_n(e))$ et $(A_1(e'), A_2(e'), \ldots, A_n(e'))$ sont distincts.

Les applications A_i sont des fonctions d'accès qui permettent de définir les propriétés des éléments de E. En Algol 68 [33](page 139) les applications seraient appelées des sélecteurs d'une structure. De même cette définition est à rapprocher de la notion de "V-graphe" présentée dans [34]. Cette définition correspond à la représentation graphique de la figure 2.2.

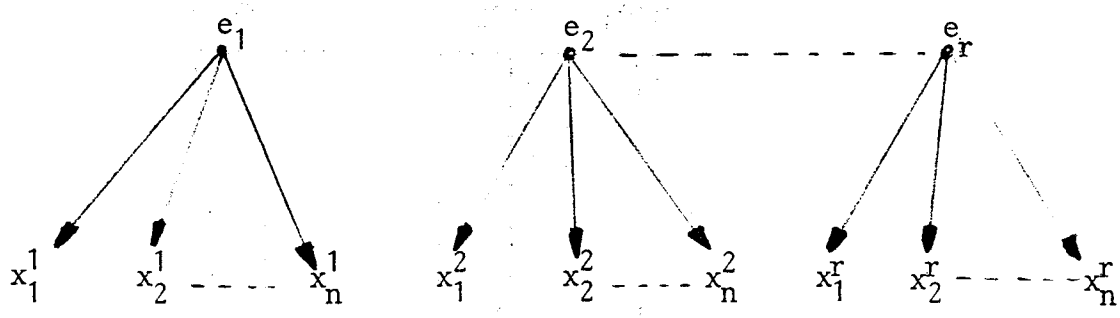


FIGURE 2.2.

La notion d'application introduite pour définir une collection de données est de même nature que la notion de constituant élémentaire présentée au paragraphe précédent qui avait pour but de préciser le rôle d'un champ. En conséquence, nous confondrons dorénavant ces deux notions et nous parlerons toujours de constituant élémentaire, sans toutefois oublier ce qu'elle représente.

On peut interpréter la définition de collections de données à l'aide d'un modèle équivalent fondé sur la notion d'entité.

On appellera entité l'ensemble des éléments de la forme :

$\{<A_i : x_i>\}$ où $x_i = A_i(e)$ pour $i = 1,2,\ldots,n$ $e \in E$

conventionnellement on écrira $<A:x> = \{<A_i : x_i>\}$ où

$A = (A_1, A_2,\ldots,A_n)$

$x = (x_1, x_2,\ldots,x_n)$

Une collection de données étant alors un ensemble d'entités. Sous cette forme, nous conviendrons de représenter une collection de données par un tableau figure 2.3 où l'ordre des colonnes n'a aucune importance. On notera alors la collection de données par :

$$[A_1/X_1, A_2/X_2,\ldots,A_n/X_n] \quad P$$

en indiquant d'une part le nom des constituants et le nom des champs.

A_1/X_1	A_2/X_2	.....	A_n/X_n
x_1^1	x_2^1		x_n^1
x_1^2	x_2^2		x_n^2
⋮	⋮		⋮
x_1^k	x_2^k		x_n^k

FIGURE 2.3. : Représentation d'une collection de données

REMARQUES :

(a) Supposons que X_1 soit l'ensemble des entités d'une autre collection de données définie sur la liste des champs $(Y_1,Y_2,\ldots,Y_p)$ et de constituants $(B_1,B_2,\ldots,B_p)$. On peut construire une nouvelle collection de données sur la liste des champs :

$$(Y_1,Y_2,\ldots,Y_p,\ X_2,\ldots,X_n)$$

et à partir de l'ensemble E en considérant les applications :

$$B_1 \circ A_1,\ B_2 \circ A_1,\ \ldots,\ B_p \circ A_1,\ A_2,\ldots,A_n$$

Dans la pratique on considère souvent que les collections définies sur $X_1,X_2,\ldots,X_n$ et $Y_1,Y_2,\ldots,Y_p,X_2,\ldots,X_n$ sont les mêmes. C'est ce qui se passe quand on considère une collection de données définies sur les champs NOM, ADRESSE et que ADRESSE est une notion composée de RUE, VILLE, DEPARTEMENT.

(b) Dans la pratique on donne souvent le même nom au constituant et au champ, ceci se justifie lorsque la collection de données est construite sur n champs distincts $X_1,\ X_2,\ldots,\ X_n$. Par abus de langage et pour simplifier l'écriture nous définirons seulement la collection de données par ses constituants:

$$[A_1,\ A_2,\ldots,A_n]\ P.$$

Exemple 2.4 :

La collection de données associée à l'exemple 2.3 pourrait se représenter par le tableau ci-dessous où les constituants sont : COMPOSANT, COMPOSE, QUANTITE

COMPOSANT/PIECE	*COMPOSE/PIECE*	*QUANTITE/ENTIER*
roue	*vélo*	*2*
cadre	*vélo*	*1*
guidon	*vélo*	*1*
rayon	*roue*	*50*
pneu	*roue*	*1*

que le tableau se présente suivant l'ordre COMPOSANT, COMPOSE, QUANTITE ou QUANTITE, COMPOSE, COMPOSANT, il possède toujours la même signification.

2.2.3. Système de collection de données

Un système d'informations est une représentation abstraite de faits et d'objets exprimant les relations entre ces faits et ces objets. Le problème de trouver un ensemble cohérent de collections de données répondant à une application sera abordée au cours du chapitre 11 : décomposition d'une collection, en donnant à l'utilisateur des règles logiques permettant de le résoudre.

Dans l'immédiat nous définirons d'abord une famille de collection de données et un système de collections.

Une famille de collections de données sera définie comme un ensemble fini de collections de données $\{P_1, P_2, \ldots, P_k\}$. Chaque collection P_i étant définie par :

- les champs de base $(X_1^i, X_2^i, \ldots, X_{P_i}^i)$,
- les constituants $(A_1^i, A_2^i, \ldots, A_{n_i}^i)$,
- l'ensemble E_i.

On peut alors envisager l'ensemble $\mathcal{F}$ dont les éléments sont des familles de collections de données construites sur les mêmes champs et les mêmes constituants. On définira alors un système de collections de données comme une partie de $\mathcal{F}$. Cette partie est en général définie à partir d'un prédicat $\mathcal{R}$ qui représente un invariant du système d'informations, nous noterons alors $(\mathcal{F}, \mathcal{R})$ le système de collection de données.

En pratique, le prédicat $\mathcal{R}$ est décrit à l'aide du langage naturel et de notations mathématiques. D'une façon générale $\mathcal{R}$ peut être une propriété globale de la collection de données ou bien une propriété vérifiée par les entités. Il faut remarquer que si $\mathcal{R}$ est toujours vrai alors il n'y a aucune restrictions sur le choix des éléments de $\mathcal{F}$.

Nous allons donner différents exemples d'invariant qu'il est possible de rencontrer.

Exemple 2.5.

Soit un système constitué par une suite de collections de données T_i définies sur les constituants élémentaires (F,P,V,D) qui représentent respectivement les concepts de fournisseur, pièce, ville du fournisseur, description de la pièce. Une entité (<F:f>,<P:p>,<V:v>,<D:d>) appartient à T_i si et seulement si le fournisseur f à partir de la ville v fournit la pièce p avec la description d à l'instant i.

Pour les indices 0 et 1, T_0 et T_1 pourraient être les suivants :

T_0

F	P	V	D
f_1	p_1	v_1	bleue
f_1	p_2	v_1	vert
f_1	p_3	v_1	rouge
f_2	p_1	v_2	bleue
f_2	p_3	v_2	rouge

T_1

F	P	V	D
f_1	p_1	v_3	bleue
f_1	p_2	v_3	vert
f_1	p_3	v_3	rouge
f_2	p_4	v_2	noire

La propriété $\mathcal{R}$ vérifiée par T_i pour $\forall i$ consiste :

- *à une pièce ne correspond qu'une couleur (c'est-à-dire que si p_1 est bleue, il n'y a pas d'autres entités ayant pour p_1 une couleur différente)*
- *à un fournisseur ne correspond qu'une ville.*

De tels invariants seront appelés des relations fonctionnelles entre les constituants et nous noterons $P \rightarrow D$ et $F \rightarrow V$.

Exemple 2.6.

Soit un système de collections de données constitué par une suite de collections T_i définies sur les constituants (C,V,M) qui désignent respectivement les notions de client, vendeur et modèle d'ordinateur. Une entité (<C:c>,<V:v>,<M:m>) appartient à T_i si le vendeur v est susceptible de vendre le modèle m au client c.

L'invariant $\mathcal{R}$ est : chaque client est toujours - dans toutes les collections de données T_i - capable d'acheter aux vendeurs qui lui sont attachés n'importe quel modèle.

Si on suppose que pour l'indice zéro il y ait juste un client deux vendeurs et un modèle, tandis que pour l'indice 1, deux clients, deux vendeurs, deux modèles, alors T_0 et T_1 pourront être représentées par les tables de valeurs suivantes :

T_0

C	V	M
c_1	v_1	m_1
c_1	v_2	m_1

T_1

C	V	M
c_1	v_1	m_1
c_1	v_1	m_1
c_1	v_2	m_1
c_1	v_2	m_2
c_2	v_1	m_1
c_2	v_1	m_2

Exemple 2.7. :

Etant donné un système de collections constitué par une suite de collections T_i définies sur les constituants élémentaires (NOM, NOM-DE-JEUNE-FILLE, AGE, SEXE, STATUT-MILITAIRE, ETAT-CIVIL, CODE-RETRAITE)

La propriété $\mathcal{R}$ pourra s'exprimer sur toutes les entités de T_i et pour $\forall i$ par :

AGE < 21 implique CODE-RETRAITE = @

SEXE = féminin implique STATUT-MILAIRE = @

ETAT-CIVIL = marié et SEXE = féminin implique NOM-DE-JEUNE-FILLE ≠ @

Exemple 2.8.

Finalement, considérons un système $(\mathcal{F},\mathcal{R})$ où un élément de $\mathcal{F}$ est une famille de collections de données T_1 et T_2 respectivement définies sur les constituants (N,E) qui désignent respectivement le nom et le nom d'enfant, et sur les constituants (N,Q) où N a la même signification que dans T_1 et Q représente la qualification de la personne.

L'invariant $\mathcal{R}$ consiste en :

- *le nom de l'enfant détermine le nom de la personne,*
- *le nom de la personne détermine la qualification,*
- *les personnes apparaissant dans les colonnes N de T_1 et T_2 sont les mêmes.*

Nous verrons ultérieurement qu'il sera possible de donner du système $(\mathcal{F},\mathcal{R})$ une autre représentation équivalente $(\mathcal{F}',\mathcal{R}')$ où les élémentsde $\mathcal{F}'$ sont des familles construites sur une seule collection de données dont les constituants sont (N,E,Q).

Les propriétés $\mathcal{R}$ sont des propriétés intrinsèques possèdées par les collections de données du système d'informations. Les exemples donnés ci-dessus ne sont pas limitatifs et le lecteur pourra trouver de

nombreux autres exemples. Toutefois, il est intéressant de remarquer que si un système de collections de données est déclaré posséder la propriété $\mathcal{R}$ *alors cette déclaration est un moyen de contrôler, si l'addition, la suppression et la mise-à-jour d'entités modifie la propriété* $\mathcal{R}$.

2.3. Les liens entre constituants élémentaires

2.3.1. Classification

Quand on examine les liens qui existent entre les valeurs x,y appartenant respectivement à des champs X,Y, on rencontre les trois cas suivants :

- à une valeur $x \in X$ correspond plusieurs valeurs $y \in Y$ et inversement,
- à une valeur de x correspond une seule valeur y,
- à une valeur de x correspond une seule valeur y et inversement.

Pour illustrer le premier cas prenons l'exemple 2.8., le constituant nom d'enfant est relié à celui de qualification par l'intermédiaire du nom. Les relations qui nous préoccupent sont par nature directes, par opposé à l'exemple ci-dessus qui présente un lien indirect.

L'étude qui va suivre va étudier plus spécifiquement les deux autres cas.

2.3.2. Les relations fonctionnelles

La définition du concept de relation fonctionnelle nécessite l'introduction de la notion de projection d'une collection de données.

Définition : Projection d'une collection de données [X]P

Si E désigne une sous-liste de X, alors il est possible d'écrire X = (E,F) on définira alors la projection de P sur E par :

$$[E]P \triangleq \{<E:e>: \exists <F:f>, \ (<E:e>,<F:f>) \in P\}$$

L'application qui à P fait correspondre [E]P sera appelée une projection que nous noterons Π_E : P → [E]P. D'une façon plus générale on notera par :

$\Pi_{E,F}$: [E]P → [F]P si F est une sous-liste de E.

Il est facile de vérifier que $\Pi_F = \Pi_{EF}$ o Π_E où le symbole o désigne la composition de deux fonctions.

Exemple 2.9.

La collection de données ci-dessous définie sur les constituants : professeur (P), matière enseignée (T), classe (Y), salle (N), heure (H) représente un emploi du temps.

P	*T*	*Y*	*N*	*H*
durand	*math*	*4*	*2B*	*6*
dupont	*français*	*2*	*1A*	*3*
dupont	*français*	*1*	*3C*	*5*
nimbus	*français*	*2*	*3C*	*10*
dupont	*français*	*2*	*1A*	*7*
nimbus	*français*	*3*	*2A*	*3*
durand	*math*	*5*	*1A*	*5*

La projection de cette collection sur les constituants P et T donne :

P	*T*
durand	*math*
dupont	*français*
nimbus	*français*

Définition : Relation fonctionnelle

Etant donné une suite de collections $[X]P_1$, $[X]P_2$,... $[X]P_i$ et si E et G sont deux sous-listes de X non nécessairement disjointes, nous dirons que G est relié fonctionnellement à E si : pour k = 1,2,...,i il existe une application ℓ_k : $[E]P_k \rightarrow [G]P_k$ vérifiant :

$\Pi_G(k) = \ell_k$ o $\Pi_E(k)$, le diagramme de la figure 2.4., illustre cette propriété.

Alors la relation fonctionnelle sera notée :

ℓ : E → G

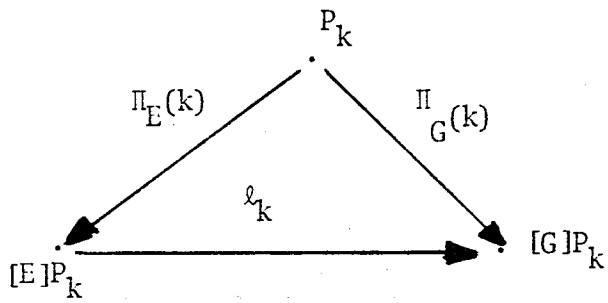


FIGURE 2.4.

Il faut remarquer que la notation ℓ : E → G représente en fait l'ensemble des applications $\ell = \{\ell_k : \ell_k : [E]P_k \rightarrow [G]P_k, k = 1,...i\}$

Exemple 2.10.

A partir de la collection de données définie dans l'exemple 2.9., on a par exemple les relations fonctionnelles suivantes :

ℓ_1 : $P \rightarrow T$
ℓ_2 : $P,H,X \rightarrow N$

Il faut aussi remarquer que nous avons une application de P,Y dans N, mais cette application ne revêt pas le caractère de permanence en fonction du temps, car un professeur avec une classe peut faire cours dans des salles différentes.

Dans le chapitre 7 nous reviendrons de façon détaillée sur les relations fonctionnelles.

Définition : Relation fonctionnelle élémentaire

Soit ℓ : E → G une relation fonctionnelle, nous dirons qu'elle est élémentaire s'il n'existe pas ℓ' : E' → G où E' ⊂ E (le symbole ⊂ doit être interprété comme E' est une sous liste de E).

Exemple 2.11.

P,H,Y → N n'est pas une relation fonctionnelle élémentaire car P,H → N est une relation fonctionnelle. Cette dernière est élémentaire car nous n'avons pas P → N ou H → N.

Nous avons vu ci-dessus que si F ⊂ E ⊂ X où X est l'espace de définition de la collection P le diagramme (figure 2.5.) vérifie la propriété :

$$\Pi_F(k) = \Pi_{EF}(k) \circ \Pi_E(k)$$

pour ∀ k.

Il en résulte que l'on peut écrire ℓ : E → F. Nous dirons que cette relation fonctionnelle est triviale car elle ne peut pas être une relation fonctionnelle élémentaire.

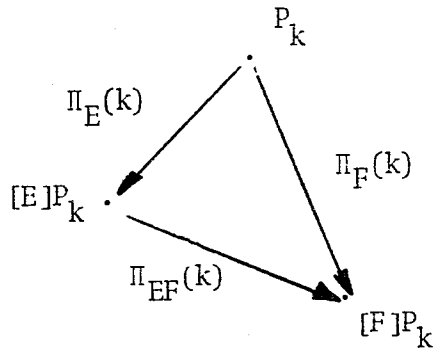


FIGURE 2.5.

2.3.3. Relation fonctionnelle et partition d'une collection de données

Définition : Partition d'une collection de données

Etant donné une collection P définie sur un espace de constituants X et E ⊂ X, nous définirons la relation binaire ω_E par :

$$\langle X:x \rangle \omega_E \langle X:x' \rangle \quad \text{si} \quad \Pi_E(\langle X:x \rangle) = \Pi_E(\langle X:x' \rangle)$$

Cette relation est symétrique, reflexive, transitive, elle est donc une relation d'équivalence qui définit sur P un ensemble de classes d'équivalence que nous noterons P/ω_E

Proposition 2.1. :

Une condition nécessaire et suffisante pour que $\ell : E \to F$ est que pour $k = 1,2,\ldots,i$: $P_k/\omega_E \equiv P_k/\omega_{E,F}$

Désignons par $[<X:x>]_e$ une classe d'équivalence telle que :

$$\Pi_E(<X:x>) = <E:e>$$

(a) si $E \to F$ alors $P_k/\omega_E \equiv P_k/\omega_{E,F}$

Il est immédiat que les classes d'équivalence des partitions ω_E et $\omega_{E,F}$ vérifient :

$$[<X:x>]_{ef} \subset [<X:x>]_e$$

D'autre part si $E \to F$ on en déduit que :

$$[<X:x>]_e \subset [<X:x>]_{ef}$$

d'où comme conclusion

$$[<X:x>]_e \equiv [<X:x>]_{ef}$$

(b) Inversement si $P_k/\omega_E \equiv P_k/\omega_{E,F}$ cela implique que les classes d'équivalence vérifient :

$$[<X:x>]_e \equiv [<X:x>]_{ef}$$

Cette identité montre qu'à une valeur e ne correspond qu'une valeur f et comme cette propriété est vraie pour toutes les collections P_k, $k = 1,2,\ldots,i$ on a bien $E \to F$.

REMARQUE :

Si théoriquement la proposition2.1 permet de détecter une relation fonctionnelle en effectuant des partitions il faut bien reconnaître que le coût de mise en oeuvre d'un tel procédé serait onéreux.

2.3.4. Constituants discriminatoires

Définition :

Etant donné une collection de données P d'espace de définition X, nous dirons que $E \subset X$ est un ensemble discriminatoire pour P si il existe une relation fonctionnelle $\ell : E \rightarrow X$.

E sera un index si la relation fonctionnelle est élémentaire.

Exemple 2.12.

P,H et H,Y sont des index de la collection de données définie dans l'exemple 2.9.

Il est important de faire la distinction entre un index primaire et un index, Une collection de données peut avoir plusieurs index, mais ne possèdera qu'un index primaire qui sera un choix du concepteur du système d'informations lorsqu'il définira la structure physique associée à la collection P et la façon d'accèder à cette collection.

CHAPITRE 3

LES OPERATIONS FONDAMENTALES SUR LES COLLECTIONS DE DONNEES

La notion de collection de données est intéressante dans la mesure ou d'une part elle représente un système d'informations et que d'autre part il est possible de transformer des collections de données en de nouvelles collections de données à l'aide d'opérateur.

Ce chapitre a pour but de définir un ensemble d'opérateurs : produit, projection, composition, composition normale, R-composition et d'étudier les propriétés mathématiques de ces opérateurs.

D'autre part, le problème de la décomposition d'une collection sera introduit et les conditions dans lesquelles cette décomposition peut être effectuée. On montrera comment le problème de la décomposition est relié à la notion de relation fonctionnelle.

3. LES OPERATIONS FONDAMENTALES SUR LES COLLECTIONS DE DONNEES

3.1. L'union, l'intersection, la différence

Nous considérons que les opérations ensemblistes d'union (U), d'intersection ($\cap$) et de différence (-) de deux collections de données P et Q ne pourront être définies que si les collections P et Q vérifient la propriété suivante : l'ensemble des champs de P est identique à l'ensemble des champs de Q.

3.2. Produit de deux collections de données

Etant donné deux collections [X]P et [Y]Q* tel que X . Y = Ø Ø désignant une liste vide, alors on définira :

$$P, Q \triangleq \{<X:x>, <Y:y> : <X:x> \in P \text{ et } <Y:y> \in Q\}$$

L'espace de définition de P,Q est (X,Y) et l'opérateur de produit est noté ",". Du fait qu'une entité est un ensemble d'élément on a : <X:x>, <Y:y> = <Y:y>, <X:x> soit P,Q = Q,P

REMARQUE :

Si les espaces de définition X et Y ne vérifient pas X . Y = Ø alors la notion de produit sera remplacée par celle de composition.

(*) Dans ce chapitre nous confondrons souvent la notion de champ et de constituant comme nous l'avons indiqué dans la remarque (b) du paragraphe 2.2.2., page 2.8. Nous utiliserons alors l'expression : étant donné une collection de données [X]P d'espace de définition X, X représentant à la fois la liste des constituants et des champs de la collection et dans ce cas les champs composants la liste X sont tous distincts.

3.3. Projection d'une collection de données à partir d'une sous-entité

La notion de projection a été définie au paragraphe 2.3.2., ici nous allons définir une variante.

Si P est une collection de données d'espace de définition X = (E,F,G) et <E:e> une sous-entité appartenant à [E]P, alors on notera par :

$$[<E:e>,F]P \triangleq \{<F:f> : (<E:e>, <F:f>) \in [E,F]P\}$$

Cette opération associe à une valeur e, l'ensemble des valeurs f qui y sont attachées dans la collection P.

Exemple 3.1.

Soit la collection P définie sur les constituants (NOM, NOM-ENFANT)

NOM	*NOM-ENFANT*
dupont	*claude*
dupont	*philippe*
durant	*isabelle*
durant	*frédéric*
durant	*marie*
nimbus	*phillis*

[<NOM : dupont> , NOM-ENFANT]P définit une collection de données d'espace NOM-ENFANT :

NOM-ENFANT
claude
philippe.

Nous verrons au cours du chapitre 5 : langage de manipulation de collections de données, comment il sera possible d'exprimer des propriétés que doit vérifier la collection [<E:e>,F]P. La plupart de ces propriétés s'exprimeront à l'aide :

- *d'expressions logiques,*
- *des quantifieurs $\exists$ (existentiel), $\forall$ (universel),*
- *de la cardinalité,*
- *des opérations élémentaires sur les ensembles.*

3.4. Composition de deux collections de données

3.4.1. Définition

Etant donné deux collections de données P et Q, dont les espaces de constituants sont respectivement X et Y, vérifiant les propriétés suivantes :

- (a) $Z = X \cdot Y \neq \emptyset$
- (b) $[Z]P = [Z]Q$ (cette collection $[Z]P$ ou $[Z]Q$ sera appelée le support de la composition).

Sous ces conditions nous dirons que la collection de données T d'espace X,Y est un composé strict de P et Q, si les projections de T successivement sur les espaces X et Y redonnent les collections P et Q, c'est-à-dire si :

$$[X]T = P$$
$$[Y]T = Q$$

Exemple 3.2.

P	
A	*B*
a_1	b_1
a_1	b_2
a_2	b_2

Q	
A	*C*
a_1	c_1
a_1	c_2
a_2	c_1
a_2	c_3

T		
A	*B*	*C*
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_2	c_1
a_2	b_2	c_3

T'		
A	*B*	*C*
a_1	b_1	c_1
a_1	b_2	c_2
a_2	b_2	c_1
a_2	b_2	c_3

Les collections T et T' sont des composés de P et Q, ce qui montre que le composé de P et Q n'est pas unique. Nous désignerons par $\mathcal{C}(P,Q)$ la famille des composés de P et Q.

3.4.2. Composition normale

Avec les notations du paragraphe 3.4.1., on peut écrire $X = (Z,A)$ et $Y = (Z,B)$ tels que $Z \cdot A = \emptyset$, $E \cdot B = \emptyset$ et $A \cdot B = \emptyset$.

On appellera la composition normale stricte de P et Q et l'opérateur correspondant sera noté "$*$ " l'opérations :

$$P * Q \triangleq \{(<A:a>, <Z:z>, <B:b>) : (<A:a>, <Z:z>) \in P \text{ et } (<B:b>, <Z:z>) \in Q\}$$

Il est évident que $P * Q \in \mathcal{C}(P,Q)$, on peut même écrire cette expression sous la forme suivante :

$$P * Q = \bigcup_{<Z:z> \in [Z]P} <Z:z>, [<Z:z>,A]P, [<Z:z>,B]\ Q$$

Cette expression exprime la composition normale de P et Q comme l'union d'un produit de collection de données. La collection T de l'exemple 3.2., est la composition normale de $P * Q$.

On peut facilement vérifier les propriétés :

- (a) si $T \in \mathcal{C}(P,Q)$ alors $T \subset P * Q$
- (b) si U et $V \in \mathcal{C}(P,Q)$ alors $U \cup V \in \mathcal{C}(P,Q)$

La propriété (a) est aisée à prouver. Considérons une entité <W:w> appartenant à T et de la forme <W:w> = (<A:a>, <Z:z>, <B:b>). Puisque cette entité appartient à T elle vérifie :

$$(<A:a>, <Z:z>) \in P$$

$$(<B:b>, <Z:z>) \in Q$$

donc $<W:w> \in <Z:z>, [<Z,z>, A]P, [<Z:z>, B]Q$

cette dernière expression, après rapprochement avec l'équation définissant la composition normale, assure que $<W:w> \in P * Q$.

La propriété (b) est vraie car si $T = U \cup V$ alors on peut écrire sachant que l'opérateur de projection est distributif par rapport à l'union :

$$\Pi_X(T) = \Pi_X(U \cup V) = \Pi_X(U) \cup \Pi_X(V)$$

$$= P \cup P = P$$

de même $\Pi_Y(T) = Q$

Le diagramme de la figure 3.1 résume ces propriétés, où i est l'injection de T dans P * Q.

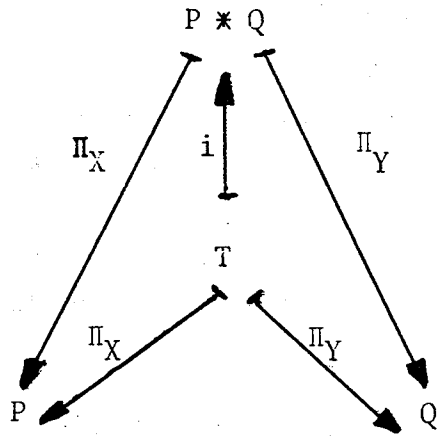


FIGURE 3.1.

Il résulte des propriétés énoncées ci-dessus que la famille $\mathcal{C}(P,Q)$ est un sup-demi-treillis par rapport à l'ordre d'inclusion. Les éléments minimaux répondent au problème : étant donné deux collections de données P et Q quelle est la plus petite collection de données T qui recouvre P et Q? Il est facile de remarquer que le nombre d'entités de la collection T est donné par la formule :

$$\text{card}(T) = \sum_{\langle Z:z\rangle \in [Z]P} \text{Max}\ \{\text{card}([\langle Z:z\rangle, A]P),\ \text{card}([\langle Z:z\rangle, B]Q)\}$$

Dans l'exemple 2.10 la collection de données T' est une collection minorante.

3.4.3. Extension de la définition de composition normale

Dans un système de collections de données, la propriété [Z]P = [Z]Q n'est pas toujours vérifiée, par suite d'opérations d'addition et de suppression d'entités portant sur les collections de données P et Q. Nous étendrons donc la définition donnée en 3.4.2 à l'aide de l'expression :

$$P * Q = \bigcup_{<Z:z> \in [Z]P \cap [Z]Q} <Z:z>,[<Z:z>, A]P,[<Z:z>, B]Q$$

Lorsqu'on utilise cette expression on a :

$$[Z,A](P * Q) \subseteq P$$
$$[Z,B](P * Q) \subseteq Q$$

ce qui signifie qu'il n'est pas possible de retrouver dans P * Q toutes les informations contenues dans les collections de données P et Q.

Si on avait voulu définir une opération qui préserve l'information il aurait été nécessaire d'introduire la notion de données ayant la valeur "b"

$$P \hat{*} Q = \bigcup_{<Z:z> \in [Z]P \cap [Z]Q} <Z:z>, [<Z:z>, A] P, [<Z:z>, B]Q$$

avec :

- si la collection [<Z:z>, A]P est vide on écrira <A:'b'>
- si la collection [<Z:z>, B]Q est vide on écrira <B:'b'>

Cette opération particulière de composition normale $\hat{*}$ possède le grave inconvénient que si on effectue des compositions successives, les compositions pourront s'effectuer sur des données blanches ce qui ne manquera pas d'introduire des données ambigües dans le système d'informations. D'un point de vue théorique nous ne considèrons jamais cette possibilité, tandis que d'un point de vue pratique c'est une opération à manipuler avec la plus grande précaution.

Exemple 3.3.

P	
A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_2

Q	
A	C
a_1	c_1
a_1	c_2
a_2	c_2
a_4	c_1

$P * Q$		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2

$P \hat{*} Q$		
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2
a_3	b_2	'b'
a_4	'b'	c_1

Remarques: (a) *Dans le cas où les espaces de constituants des collections P et Q sont disjoints, c'est-à-dire X.Y = ∅ alors on définira :*

$$P * Q \triangleq P,Q$$

(b) *Nous venons de donner plusieurs définitions de la composition normale de deux collections. Ceci afin d'offrir à l'utilisateur un plus grand choix pour exprimer ses traitements. Toutefois, d'un point de vue formel et dans la suite de notre travail, nous utiliserons plus particulièrement la définition donnée au paragraphe 3.4.3.*

3.4.4. Propriétés de la composition normale

(a) commutativité P * Q = Q * P. Cela résulte de la définition de la composition normale.

(b) associativité Etant donné trois collections de données P,Q,R dont les espaces de constituants sont respectivement (A,B), (B,C) et (C,D) il est possible de montrer que les trois expressions suivantes sont égales :

P * (Q * R)
(P * Q)* R
(P * R)* Q

Montrons, par exemple l'identité entre P *(Q * R) et (P * R)* Q, cette dernière expression s'écrivant en fait (P,R)* Q.

Si l'entité <T:t> = (<A:a>,<B:b>,<C:c>,<D:d>) ε (P,R)* Q

cela implique que les entitées (<A:a>,<B:b>) ε P
(<B:b>,<C:c>) ε Q
(<C:c>,<D:d>) ε R

donc <T:t> ε P *(Q * R) et par conséquent :

(P,R)* Q ⊂ P *(Q * R)

On montrerait de la même façon que P *(Q * R) ⊂ (P,R)* Q d'où l'égalité annoncée.

(c) distributivité Etant donné deux collections P et Q ayant le même espace de constituants alors R *(P ∪ Q) = (R * P) ∪ (R * Q)
R *(P ∩ Q) = (R * P) ∩ (R * Q)

(d) <u>absorption</u> Etant donné deux collections P et Q dont les espaces sont respectivement X et Y avec $Y \subset X$ alors

$$P * Q \subseteq P$$

en particulier $P * P = P$

(e) Etant donné trois collections de données P,Q et R dont les espaces de définition sont respectivement (A,B), (B,C) et (B,C) vérifiant :

$$P * Q = P * R$$

alors $Q = R$

Pour le montrer il suffit d'exprimer la composition normale, soit :

$$\bigcup_{<B:b> \in [B]P = [B]Q} <B:b>,[<B:b>,A]P,[<B:b>,C]Q = \bigcup_{<B:b> \in [B]P=[B]R} <B:b>,[<B:b>,A]P,[<B:b>,C]R$$

qui implique que pour chaque valeur b :

$$[<B:b>,A]P,[<B:b>,C]Q = [<B:b>,A]P,[<B:b>,C]R$$

ou bien encore :

$$[<B:b>,C]Q = [<B:b>,C]R$$

d'où en regroupant pour toutes les valeurs b:

$$Q = R$$

<u>Remarque</u> : Si l'espace des constituants du support de P et Q et de P et R est différent alors Q peut être différent de R comme le montre l'exemple :

P		Q	R	
A	C	A	A	B
b_1	c_1	a_1	a_1	b_1
b_1	c_2	a_2	a_2	b_1

Les collections P,Q,R vérifient $P * Q = P * R$ avec $Q \neq R$

D'autres propriétés concernant la composition normale sont exposées dans l'<u>annexe A</u>. En particulier, on établira un lien entre les collections de données et les relations binaires et n-aires. On donnera aussi des conditions pour que la famille des composés $\mathscr{C}(P,Q)$ soit réduite à un seul élément.

3.5. R-Composition de collections de données

3.5.1. Définition

Dans la pratique du traitement de l'information il arrive fréquemment que l'opérateur de composition normale, tel que nous l'avons défini, ne puisse représenter certaines opérations voisines de la notion de composition. Ce phénomène se présente dans plusieurs cas :

- si P est une collection de données, nous avons vu que l'opération P * P ne crée pas une collection de données nouvelles. Néanmoins il serait souhaitable dans le cas où une collection P est définie sur des constituants A et B de même champ U de pouvoir composer P sur P afin d'obtenir les successeurs d'une donnée a. C'est par exemple le cas lorsqu'on cherche les pièces élémentaires constituant un objet (voir exemple 2.3).

- si P et Q sont deux collections de données où les listes de champ X et Y comportent des champs non distincts. Dans ce cas il y aurait ambiguïté lors d'une composition normale.

Pour ces raisons nous sommes amenés à définir une nouvelle opération appelée R-composition.

Définition de la R-composition

Etant donné deux collections P et Q définies respectivement par :

- les ensembles E et F
- les ensembles de champs $X = \{X_1, X_2, \ldots, X_p\}$ $Y = \{Y_1, Y_2, \ldots, Y_r\}$
- les ensembles d'applications :

$$\{A_i : E \to X_i : i = 1,2,\ldots,n\}$$
$$\{B_j : E \to Y_j : j = 1,2,\ldots,m\}$$

Soit A et B deux parties de l'ensemble des applications par exemple :

$$A = \{A_i : i = 1,2,\ldots, k \leqslant n\}$$
$$B = \{B_j : j = 1,2,\ldots, \ell \leqslant m\}$$

et une relation R, comprenant $k + \ell$ composantes, construite sur les champs associés aux applications appartenant à A et B.

Sous ces conditions on définira une nouvelle collection de données appelées le R-composé de P et Q en prenant :

- pour ensemble de base $G' \subset G$ où G est une partie de $E \times F$ définie par :
  (a) $G \triangleq \{(e,f) : e \in F, f \in F, (A_1(e),\ldots,A_k(e), B_1(f),\ldots,B_\ell(f)) \in R\}$
  (b) la projection de G' sur E et F vérifie $G'_E = E$ et $G'_F = F$
- les champs X U Y
- les applications $C_1, C_2, \ldots, C_n, C_{n+1}, \ldots, C_{n+m}$ définies par :

$$1 \leqslant i \leqslant n \quad C_i : G' \to X_i \text{ avec } C_i(e,f) = A_i(e)$$
$$n+1 \leqslant i \leqslant n+m \quad C_i : G' \to Y_{i-n} \quad C_i(e,f) = B_{i-n}(f)$$

Le R-composé normal de P et de Q sera obtenu en prenant G' = G. D'autre part, il est facile de remarquer que la composition normale est un cas particulier de la R-composition, il correspond au cas où la relation R est la relation d'identité.

Exemple 3.4.

Etant donné deux collections P et Q définies par leurs entités :

P	
A	*B*
a_1	2
a_1	3
a_2	3

Q	
C	*D*
4	d_1
5	d_2
4	d_2

et une relation R s'exprimant par :

$$R = \{(b,c) : b \in B(e), c \in D(f), b \leqslant c\}$$

soit :

R

B	*C*
2	*4*
2	*5*
3	*4*
3	*5*

Les collections T et T' répondent au problème :

T

A	*B*	*C*	*D*
a_1	*2*	*4*	d_1
a_1	*2*	*5*	d_2
a_1	*3*	*4*	d_2
a_2	*3*	*5*	d_2

T'

A	*B*	*C*	*D*
a_1	*2*	*4*	d_1
a_1	*2*	*5*	d_2
a_1	*2*	*4*	d_2
a_1	*3*	*4*	d_1
a_1	*3*	*5*	d_2
a_1	*3*	*4*	d_2
a_2	*3*	*4*	d_1
a_2	*3*	*5*	d_2
a_2	*3*	*4*	d_2

3.5.2. Applications

Dans de nombreux cas la relation R peut s'exprimer à l'aide des symboles mathématiques usuels, si R est une relation d'égalité ou d'ordre on utilisera :

=, >, >, <, <

Dans l'exemple 3.4 nous conviendrons d'écrire :

T' = P ✱(B < C) Q

et P ✱ (B=C)Q exprimera une R-composition de P et Q où la composition s'effectue sur les constituants B et C à l'aide de la relation identité.

3.6. Problème de la décomposition d'une collection de données

3.6.1. Définition du problème

Etant donné une collection P et les opérateurs (de composition normale et projection) est-il possible de trouver des collections R et S telle que P puisse s'exprimer à partir de R et S en utilisant les opérateurs de composition normale et de projection.

L'application pratique de ce problème correspond à la représentation d'un système d'informations par une famille de collections de données. En effet, si par exemple la collection P peut s'exprimer à partir de R et S, on pourra définir le même système d'information à partir de deux familles de collections de données, donc par conséquent de choisir la famille de collections la mieux adaptée aux traitements.

Nous définirons deux formes de décomposition :
- la décomposition standard,
- la décomposition par élimination d'un constituant introduite dans [17].

La première est du point de vue pratique la plus intéressante et sera illustrée au cours du chapitre 11 mais les conditions de décomposition sont assez strictes. Par contre la deuxième forme de décomposition réussit quand la première tombe en échec, nous donnerons même un moyen de trouver des relations R et S dans l'annexe B. Son intérêt réside beaucoup plus dans la compréhension du phénomène de décomposition que dans son aspect pratique.

3.6.2. Décomposition standard

Etant donné une collection P d'espace de constituants (A,B,C), nous dirons que R et S constituent une décomposition standard de P si :

$$P = R * S$$

avec

$$R = [A,B]\ P$$

$$S = [A,C]\ P$$

3.6.3. Décomposition par élimination d'un constituant

Etant donné une collection P d'espace de constituants (A,B,C), on dira que les collections R et S dont les espaces de constituants sont respectivement (A,B,X) et (A,C,X) constituent une décomposition par élimination du constituant X si :

$$P = R \overset{X}{\circ} S \triangleq [A,B,C]\ (R * S)$$

On pourra remarquer que dans le cas où la liste X est vide alors la décomposition par élimination se réduit à la décomposition simple.

L'annexe B donne les conditions dans lesquelles cette décomposition intervient.

3.6.4. Conditions de décomposition standard

Proposition 3.1.

Une condition nécessaire et suffisante pour que :

$$P = R * S$$

avec $R = [A,B]P$, $S = [A,C]P$ et $B \,.\, C = \emptyset$

est que pour toute entité $\langle A:a\rangle \in [A]P$

$$[\langle A:a\rangle,B,C]P = [\langle A:a\rangle,B]P,[\langle A:a\rangle,C]P$$

Si cette équation est vérifiée on peut écrire :

$$\langle A:a\rangle,[\langle A:a\rangle,B,C]\ P = \langle A:a\rangle,[\langle A:a\rangle,B]P,[\langle A:a\rangle,C]\ P$$

puis en prenant l'union :

$$\bigcup_{\langle A:a\rangle \in [A]P} \langle A:a\rangle,[\langle A:a\rangle,B,C]P = \bigcup_{\langle A:a\rangle \in [A]P} \langle A:a\rangle,[\langle A:a\rangle,B]\ P,[\langle A:a\rangle,C]\ P$$

Le premier membre de l'équation ci-dessus est la collection P, tandis que le second exprime la composition de $[A,B]P$ avec $[A,C]P$ (voir paragraphe 3.4.3 pour l'expression de la formule de composition), c'est-à-dire :

$$P = [A,B]P * [A,C]P$$

Inversement si P = [A,B]P * [A,C]P, en exprimant la formule de composition, on obtient la condition énoncée ci-dessus.

La figure 3.2 exprime tous les liens reliant les données entre elles pour qu'une collection P puisse être décomposée. Cette figure est relative à une valeur a.

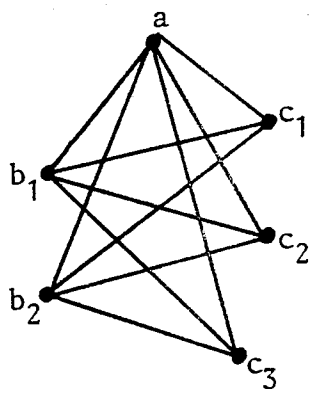


FIGURE 3.2.

Proposition 3.2.

Cette proposition est un corollaire de la proposition 3.1. :

(Eq. 3.1.) $P = [A,B]P * [A,C]P$

si et seulement si pour toute valeur $\langle A:a \rangle \in [A]P$ et $\langle B:b \rangle \in [B]P$ on a l'égalité :

$$[\langle A:a \rangle, \langle B:b \rangle, C]P = [\langle A:a \rangle, C]P$$

Si P = [A,B]P * [A,C]P d'après la proposition 3.1. on a pour toute valeur $\langle A:a \rangle$

(Eq. 3.2.) $[\langle A:a \rangle, B, C]P = [\langle A:a \rangle, B]P, [\langle A:a \rangle, C]P$

et si on affecte à B la valeur b il vient :

(Eq. 3.3.) $[\langle A:a \rangle, \langle B:b \rangle, C]P = [\langle A:a \rangle, C]P$

Réciproquement si l'équation 3.3. est vraie pour toute valeur <A:a> et <B:b> l'équation 3.2 est aussi vraie et par conséquent l'équation 3.1 par application de la proposition 3.1.

Il est possible de donner une interprétation concrète de la proposition 3.2. En effet, si une telle collection de données est un élément du système d'informations, toute requête qui consisterait à rechercher les valeurs du constituant C attachées aux données a,b pourraient se transformer en une requête équivalente : rechercher les valeurs du constituant C attachées seulement à la donnée a. Normalement cette dernière requête devrait être plus rapide puisque le nombre de critères de sélection a diminué.

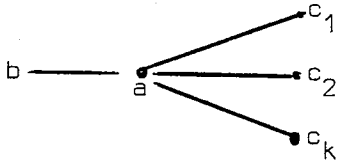
Ceci nous montre que la connaissance du système d'informations ne doit pas être composée uniquement des collections de données, mais aussi de propriétés intrinsèques pouvant faciliter les traitements d'une requête, comme par exemple l'existence d'une décomposition.

Proposition 3.3.

Les propositions ci-dessous sont des corollaires des propositions 3.1 et 3.2.

(a) Etant donné une collection P d'espaces de constituants (A,B,C) où A,B,C sont des listes disjointes, si pour les collections [A,B]P et [A,C]P il n'existe pas de point multiple (voir annexe A) dans le support A alors P = [A,B]P * [A,C]P.

Si <A:a> n'est pas un point multiple il est clair que :



$$[<A:a>,B,C]P = \{(<B:b>,<C:c_1>),\ldots,(<B:b>,<C:c_k>)\}$$
$$= <B:b>,\{<C:c_1>,\ldots,<C:c_k>\}$$
$$= [<A:a>,B]P,[<A:a>,C]P$$

Si cette propriété est vraie pour toute valeur <A:a> d'après la position 3.1 on peut alors écrire P = [A,B]P * [A,C]P

(b) Si les collections [A,B]P et [A,C]P possèdent la propriété d'unicité (voir Annexe A) alors P = [A,B]P * [A,C]P.

C'est évident puisque d'après la proposition A.1 la propriété d'unicité n'implique pas de point multiple et par application de la proposition 3.3.(a) on obtient le résultat annoncé.

(c) Etant donné une suite P_i de collections définies sur (ABC) dont l'invariant est une relation fonctionnelle ℓ : A → B,pour ∀i on peut écrire $P_i = [A,B]P_i * [A,C]P_i$

L'existence d'une relation fonctionnelle implique pour tout i l'existence d'une application ℓ_i: $[A]P_i \rightarrow [B]P_i$ ou en d'autres termes à une valeur a ne correspond qu'une valeur b donc les collections $[A,B]P_i$ et $[A,C]P_i$ ne possèdent pas de point multiple par rapport au support A, donc on peut écrire :

$$P_i = [A,B]P_i * [A,C]P_i$$

Ce corollaire est important car il relie la notion de relation fonctionnelle et celle de décomposition. Or l'examen d'un système d'informations montre l'existence de nombreuses relations fonctionnelles d'où la possibilité de décomposer une collection et par là-même d'obtenir des représentations différentes du système d'informations. Les chapitres 7 et 11 seront consacrés à l'étude des relations fonctionnelles et à l'application de la décomposition dans la conception du système d'informations.

Une collection de données P peut possèder la propriété de décomposition standard soit par construction, soit par interprétation d'une propriété intrinsèque de la collection de données. Les exemples donnés ci-dessous illustreront ce dernier cas.

Exemple 3.5.

Etant donné une collection P d'espace de constituants :

A : clients

B : modèle ordinateur

C : inspecteur de maintenance

et une entité (<A:a>,<B:b>,<C:c>) ∈ P si le client a peut faire réparer l'ordinateur b par l'inspecteur c.

De plus P possède la propriété que tout modèle d'ordinateur possèdé par un client peut être réparé par n'importe quel inspecteur affecté à ce client.

Cette propriété peut s'exprimer par :

$$P = [A,B]P * [A,C]P$$

Exemple 3.6.

Etant donné une collection de données P sur les constituants

A : cellule

B : prisonnier

C : garde

Une entité (<A:a>, <B:b>, <C:c>) ε P si le prisonnier b est dans la cellule a surveillé par le garde c.

De plus chaque garde affecté à une cellule surveille tous les prisonniers de la cellule ce qui s'exprime par :

(Eq. 3.4) $P = [A,B]P * [A,C]P$

Si on fait l'hypothèse qu'un prisionnier ne peut être que dans une cellule on a la relation fonctionnelle ℓ : B → A qui fournira la décomposition :

(Eq.3.5.) $P = [B,A]P * [B,C]P$

Nous dirons que l'équation 3.4. exprime une décomposition directe tandis que l'équation 3.5. est une décomposition fonctionnelle.

Ce dernier exemple nous donne trois modèles différents pour représenter un système d'informations sur les constituants A,B,C.

- le premier sera d'utiliser une seule collection définie sur A,B,C.
- le deuxième comprendra deux collections de données l'une sur (A,B) et l'autre sur (A,C).
- le troisième comprendra aussi deux collections de données l'une sur (A,B) et l'autre sur (B,C).

La décision de choisir un modèle plus qu'un autre ne sera pas discuté dans le cadre de ce chapitre. Disons qu'elle résulte des traitements que l'on veut effectuer sur le système d'informations. Nous discuterons ce problème dans les chapitres 11 et 12.

3.6.5. Représentation matricielle d'une décomposition standard

A une collection de données P définie sur l'espace (A,B,C) on peut faire correspondre un tableau $\mathcal{C}^{P}_{AB,AC}$ dont chacune des lignes correspond à une valeur appartenant à la projection P sur (A,B), dont chacune des colonnes correspond à une valeur appartenant à la projection de P sur (A,C) et dont les éléments peuvent prendre l'une des trois valeurs 1,0,'@'.

Si (<A:a>,<B:b>,<C:c>) est une entité appartenant à P alors à l'intersection de la ligne (a,b) et de la colonne (a,c) la valeur de l'élément sera 1 sinon 0. A l'intersection de la ligne (a,b) et de la colonne (a',c) avec a ≠ a' on placera la valeur '@'.

Exemple 3.8.

A la collection P on associe le tableau $\mathcal{C}^{P}_{AB,AC}$

	P	
A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_2
a_2	b_2	c_1

		a_1		a_2	
		c_1	c_2	c_1	c_2
a_1	b_1	1	1	@	@
	b_2	1	1	@	@
a_2	b_1	@	@	0	1
	b_2	@	@	1	0

. On notera $\mathcal{C}^{P}_{aB,aC}$ *le sous-tableau de* $\mathcal{C}^{P}_{AB,AC}$ *correspondant à une valeur a.*

Proposition 3.4.

Une condition nécessaire et suffisante pour que la collection P admette une décomposition standard est que tous les sous-tableaux $\mathcal{C}^{P}_{aB,aC}$ du tableau $\mathcal{C}^{P}_{AB,AC}$ ne comprennent que des 1.

Cette proposition n'est que la transposition de la proposition 3.1., puisque si le tableau : $\mathcal{C}^{P}_{aB,aC}$ ne comprend que des 1 cela est équivalent à :

$$[<A:a>,B,C]\ P = [<A:a>,B]P,\ [<A:a>,C]\ P$$

Cette représentation de la collection P sous forme d'un tableau présente deux avantages :

- de pouvoir utiliser la théorie des décompositions d'une fonction (Pichat [20]) pour étudier la décomposition d'une collection,
- de construire une solution pour la décomposition par élimination si la condition exprimée dans la proposition 3.4 n'est pas vérifiée (voir Annexe B)

3.7. Dérivation de collection de données

Toutes les opérations que nous avons vues jusqu'à maintenant ne modifiaient pas la valeur des collections de départ, même nous avons étudié une opération de composition dont la principale propriété était de conserver l'information. Dans ce paragraphe, nous allons étudier des opérations qui créent de nouveaux constituants et de nouvelles données.

3.7.1. Dérivation d'entités

Les opérations de dérivation sont les opérations inverses de celles que nous avons vues dans le problème de la décomposition. Les deux opérations de dérivation que l'on va considérer seront définies à partir d'une collection P d'espace de constituant (A,B) et d'une collection K d'espace de constituants (A,A').

Comme il arrive fréquemment en traitement de l'information la collection K sera définie par un tableau de valeurs. Toutefois, il arrive des cas où le prédicat de définition de la collection exprime une opération de calcul entre les valeurs des constituants A et A'. Dans ce dernier cas la collection K est définie de façon implicite.

La première opération de dérivation consistera simplement en une opération de composition, c'est-à-dire que l'on fabriquera une collection Q définie sur (A,B,A') par :

$$Q = P * K$$

La deuxième opération permettra d'obtenir une collection notée P^K :

$$P^K = P \overset{A}{\circ} K = [A',B]\ (P * K)$$

Les exemples ci-dessous illustreront ces deux opérations.

Exemple 3.9.

Etant donné une collection P exprimant la population d'un département et une collection K reliant le nom du département au numéro du département.

P	
DPT	*POP*
Ain	*1000295*
Paris	*3275495*
Yvelines	*2496780*
Indre	*936842*

K	
DPT	*NO-DPT*
Ain	*01*
Paris	*75*
Yvelines	*78*
Indre	*36*

P^K	
NO-DPT	*POP*
01	*1000295*
75	*3275495*
78	*2496780*
36	*936842*

Si on définit une collection K' sur les constituants POP et POP-A qui exprime la population arrondie par défaut à 10^3 près, on obtiendra :

$$(P^K)^{K'}$$

NO-DEPT	*POP-A*
01	*1000000*
75	*3275000*
78	*2496000*
36	*936000*

Exemple 3.10.

Etant donné une collection P d'espace de constituants (NO-FACTURE, NO-PRODUIT, QUANTITE, PRIX-UNITAIRE) qui exprime les objets à facturer, on veut construire une collection Q comprenant les mêmes constituants que P avec en plus le constituant PRIX-PRODUIT.

On définira une collection K sur les constituants (QUANTITE, PRIX-UNITAIRE, PRIX-PRODUIT) et une entité :

$(\langle QUANTITE{:}q\rangle, \langle PRIX\text{-}UNITAIRE{:}p\rangle, \langle PRIX\text{-}PRODUIT{:}pp\rangle) \in K$

si

$$pp = q \times p$$

Sous ces conditions la collection Q recherchée s'exprime par :

$$Q = P * K$$

L'opération de dérivation P^K peut être perçue comme le désir de remplacer la collection P par P^K. Dans une telle hypothèse il peut être intéressant d'examiner dans quelles conditions l'invariant de la collection P se transmet à la collection P^K. La proposition 3.5. apporte une réponse dans le cas où l'invariant est exprimé par une relation fonctionnelle.

Définition : Recouvrement et sous-recouvrement.

Si U est un ensemble quelconque et $\{U_i\}_{i \in I}$ une famille de partie de U, la famille $\{U_i\}_{i \in I}$ constitue un recouvrement si :

$$U = \bigcup_{i \in I} U_i$$

Une famille $\{V_j\}_{j \in J}$ de parties de U est un sous-recouvrement de la famille $\{U_i\}_{i \in I}$ si pour $\forall j \in J$ il existe un i tel que $V_j \subset U_i$

Proposition 3.5.

Etant donné :

- une collection P dont l'espace contient au moins les constituants A et B et caractérisée par la relation fonctionnelle A → B,
- une collection K d'espace (A,A')

alors, une condition nécessaire et suffisante pour que la collection P^k possède la relation fonctionnelle A' → B est que la famille :

$$\mathcal{F}_1 = \{[<A':a'>, A]\ K\}_{a'} \in [A']K$$

soit un sous-recouvrement de la famille :

$$\mathcal{F}_2 = \{[<B:b>, A]P\}_b \in [B]P$$

Si la famille $\mathcal{F}_1$ est sous-recouvrement de $\mathcal{F}_2$ alors il existe un b tel que

$$[<A':a'>, A]\ K \subset [<B:b>, A]\ P$$

et tous les éléments $<A:a> \in [<A':a'>, A]K$ donnent naissance au cours de l'opération de dérivation au même élément $(<A':a'>, <B:b>)$ et comme de plus à un a ne correspond qu'un b, à un a' ne correspond qu'un b, c'est-à-dire qu'il existe bien une relation fonctionnelle ℓ' : A' → B.

Inversement si la collection P^K possède la relation fonctionnelle ℓ' : A' → B, on va d'abord montrer que pour toute paire $<A:a_1>$ et $<A:a_2> \in [<A':a'>, A]K$ avec :

$$(<A:a_1>, <B:b_1>) \in P$$

et

$$(<A:a_2>, <B:b_2>) \in P$$

implique $b_1 = b_2$. En effet si $b_1 \neq b_2$ on aurait :

$$(<A':a'>, <B:b_1>) \text{ et } <A':a'>, <B:b_2> \in P^K$$

et la relation fonctionnelle ℓ': A' → B ne serait pas vérifiée.

La propriété, ci-dessus signifie que pour $\forall <A:a> \in [<A':a'>, A]K$ la valeur a est attachée au même b dans la collection P, c'est-à-dire :

$$[<A':a'>, A]K \subset [<B:b'>, A]P$$

qui est bien la propriété de recouvrement.

3.7.2. Dérivation par partition

Soit une collection de données P définies sur un espace de constituants X = (E,F) où E est une liste de constituants déterminés et nous allons considérer une partition ω_E (voir paragraphe 2.3.3.) de la collection P. Nous noterons P/ω_E l'ensemble des classes d'équivalence, et [<X:x>] une classe d'équivalence.

De plus nous considérons une relation binaire R entre les classes d'équivalence et une valeur y d'un constituant Y.

Nous définirons une collection de données Q sur les constituants (E,Y) par le prédicat.

Une entité (<E:e>,<Y:y>) ε Q s'il existe une classe d'équivalence [<X:x>] telle que :

- Π_E(<X:x>) = <E:e>
- [<X:x>] R y

Dans la pratique la relation R est de la classe des fonctions c'est-à-dire qu'à une classe d'équivalence ne correspond qu'une seule valeur y et les fonctions usuelles sont :

- le comptage des entités de la classe d'équivalence,
- la recherche d'un élément maximal ou minimal,
- la totalisation des valeurs associées à un même constituant,
- la totalisation pondérée,
- le calcul d'une moyenne.

Exemple 3. 11.

A partir de l'exemple 3.10 où la collection Q est définie sur : (NO-FACTURE, NO-PRODUIT, QUANTITE, PRIX-UNITAIRE, PRIX-PRODUIT) on veut définir une collection S sur (NO-FACTURE, TOTAL-FACTURE) permettant de calculer le total d'une facture.

Pour cela on considérera la partition $\omega_{NO\text{-}FACTURE}$ de Q et on fera une totalisation des valeurs du constituant PRIX-PRODUIT à l'intérieur d'une classe d'équivalence.

Dans le chapitre 5 consacré à un langage de manipulation de collections de données, nous verrons comment le langage permettra d'exprimer soit les collections K soit les relations associées aux classes d'équivalence.

Conclusion

Le chapitre avait pour but de présenter les opérations fondamentales, sur les collections de données. La plupart de ces opérations, composition normale, projection, R-composition, seront considérées comme des primitives pour le langage que nous définirons dans le chapitre 5.

Nous avons aussi étudié certaines propriétés mathématiques de ces opérations. Si ces propriétés restent de nature théorique nous pensons qu'elles permettent de mieux comprendre ces opérations.

CHAPITRE 4

LES OPERATIONS DE MISES A JOUR ET LES OPERATIONS COMPATIBLES

Un système de collections de données est un ensemble de collections indexées par le temps et possèdant une ou plusieurs propriétés invariantes au cours du temps. Après avoir examiné les opérations fondamentales on étudiera les opérations qui permettent d'effectuer des mises-à-jour.

Bien que ces opérations peuvent s'exprimer à l'aide des opérateurs déjà introduits nous les considérons comme des opérations primitives car elles sont spécifiques des traitements de mise à jour.

Il est important que ces opérations de mise à jour préservent l'invariant du système de collection. Nous rechercherons dans quelles conditions certaines opérations maintiennent les relations fonctionnelles définies comme invariant.

4. LES OPERATIONS DE MISES A JOUR ET LES OPERATIONS COMPATIBLES

4.1. Les opérations de mises-à-jour

Dans le chapitre 3 les opérations étudiées sur les collections étaient des opérations algébriques. Dans ce paragraphe la présentation des opérations de mise-à-jour se fera comme dans un langage de programmation à l'aide d'instructions élémentaires.

4.1.1. Addition et suppression d'entité

Les opérations d'addition et de suppression peuvent être réalisées à l'aide de l'opérateur d'union (U) et de différence (-) à condition que ces opérations soient possibles (voir paragraphe 3.1).

4.1.2. Modification d'une collection à partir d'une autre collection

L'opération sera notée :

MODIFIER nom-de-collection 1 CONSTITUANT liste de constituant 1 PAR
nom-de-collection 2 CONSTITUANT liste de constituant 2

Par exemple :

MODIFIER P CONSTITUANT B,C PAR Q CONSTITUANT B',C'

où P et Q sont des collections dont l'espace de définition est respectivement :

- P : X = (A,B,C,D) E = (B,C)
- Q : Y = (A,B',C') F = (B',C')

Cette opération suppose que les champs élémentaires associés aux constituants subissant la modification sont identiques et elle est définie à partir de l'équation :

$$P := (P - [X](P * Q)) \cup ([\overline{E}] (P*Q))$$

, où $\overline{E}$ désigne la liste de constituants complément par rapport à la liste (X,Y), qui signifie que l'on remplace les entités de la collection P par un ensemble d'entités exprimés par l'expression figurant dans le membre de droite de l'équation.

Exemple 4.1.

P	
A	B
a_1	b_1
a_2	b_2
a_3	b_3
a_4	b_4

Q	
A	B'
a_2	b'_2
a_4	b'_4
a_5	b'_5

après l'opération :

MODIFIER P CONSTITUANT B PAR Q CONSTITUANT B'

on obtiendra la collection P dont le tableau de valeurs est :

P	
A	B
a_1	b_1
a_2	b'_2
a_3	b_3
a_4	b'_4

4.1.3. Modification d'une collection à partir d'une liste de valeurs

L'opération sera notée :

MODIFIER nom de collection 1 PAR entité DANS nom de collection 2

Dans cette opération il est nécessaire que la collection 2 soit une sous collection de la collection 1.

Par exemple : si P est une collection définie sur X = (A,B,C,D,E) alors l'opération :

MODIFIER P PAR (<A:a>,<B:b>) DANS Q

remplacera dans toutes les entités appartenant à la collection Q les valeurs correspondantes aux constituants A et B par les données a et b.

De façon générale l'opération :

MODIFIER P PAR <Y:y> DANS Q

est équivalente à l'équation :

$$P := (P-Q) \cup ([\overline{\vee}]P, \langle Y:y \rangle)$$

Exemple 4.2.

P			Q		
A	B	C	A	B	C
a_1	b_1	c_1	a_1	b_1	c_1
a_1	b_2	c_2	a_1	b_2	c_1
a_2	b_3	c_2			
a_3	b_3	c_3			

l'opération MODIFIER P PAR <B:b_o> *DANS Q génèrera la collection P*

P		
A	B	C
a_1	b_0	c_1
a_2	b_3	c_2
a_3	b_3	c_3

Dans cette opération il faut remarquer que la collection Q doit avoir été définie au préalable, c'est-à-dire par son prédicat associé. Nous verrons dans le chapitre suivant comment il est possible de définir la collection Q

Dans le cas où on utilise non pas une collection P dans l'instruction de modification, mais une entité X, l'instruction se simplifie puisque le "DANS" n'a plus de raison d'exister. L'instruction s'exprime alors :

MODIFIER nom d'entité PAR entité

Par exemple : MODIFIER X PAR <A:a>, <B:b> où X est une entité de la collection P, aura pour effet de substituer dans X les valeurs correspondantes aux constituants A et B par les données a et b.

4.2. Opérations compatibles et opérations incompatibles

4.2.1. Présentation de la notion de compatibilité

L'ensemble des opérations qui a été défini dans les paragraphes précédents permet à partir d'un ensemble donné de collections de générer de nouvelles collections. Si les collections de départ étaient caractérisées par un invariant il n'est pas sûr qu'après un certain nombre d'opérations, elles soient encore de même type.

En conséquence nous dirons qu'une opération est incompatible avec un système de collections si elle détruit l'invariant du système. Dans le cas contraire l'opération sera dite compatible.

Il faut tout de suite remarquer que parmi l'ensemble des opérations que nous avons définies certaines ne sont pas incompatibles :

(1) - L'opération de composition conserve intégralement les propriétés des collections composantes. Ceci est dû au fait que si les collections de données P et Q possèdent respectivement les invariants $\mathcal{R}_1$ et $\mathcal{R}_2$, alors toute composition de P et Q et, par exemple, la composition normale P * Q est caractérisée par l'invariant $\mathcal{R}_1 \wedge \mathcal{R}_2$.

(2) - L'opération de sélection qui consiste à extraire d'une collection une sous-collection.

(3) - La projection d'une collection est caractérisée par une restriction de l'invariant de la collection servant de point de départ à la projection.

D'une façon générale, les opérations de mise à jour sont incompatibles parce que l'introduction ou la modification de valeurs peuvent changer les propriétés initiales de la collection.

Exemple 4.3.

Etant donné une collection de données P dont l'espace de définition est (NOM, SECTION) et que l'invariant de P soit la relation fonctionnelle ℓ : NOM → SECTION, qui signifie qu'une personne ne travaille que dans une seule section. L'introduction d'une nouvelle entité pour une personne qui appartient déjà à la collection entraînerait que cette personne travaille pour deux sections ce qui est incompatible avec la propriété ℓ : NOM → SECTION.

Dans la perspective de concevoir et d'implémenter un système d'informations fondé sur la notion de collection de données, ces opérations incompatibles sont importantes.

Elles ne pourront être décelées parfois qu'après un examen complet de la collection de données, ce qui en temps machine peut être fort long, inversement la détection d'une telle incompatibilité préserve l'utilisateur de l'introduction de données parasites.

Nous allons examiner des situations où il est possible de prévoir si l'opération est incompatible ce qui évitera un examen complet de la collection de données.

4.2.2. Opérations incompatibles

Si P est une collection de données d'espace de constituants (X,A,B) avec comme invariant la relation fonctionnelle ℓ : A → B, les opérations :

MODIFIER P PAR <A:a> DANS Q

MODIFIER P PAR <B:b> DANS Q

sont incompatibles si A n'est pas un ensemble discriminatoire de la collection P.

Du fait que A n'est pas un ensemble discriminatoire il peut exister dans P deux entités (<X:x>,<A:a>,<B:b'>) et (<X:x'>, <A:a>, <B:b'>), la première appartenant à Q comme l'indique la figure 4.1.(a)

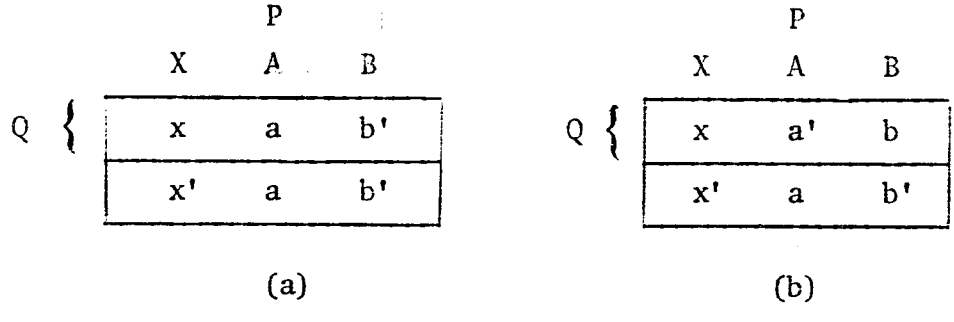


FIGURE 4.1.

alors il est immédiat que l'opération MODIFIER P PAR <B:b> DANS Q détruira la relation fonctionnelle ℓ : A → B.

Une autre situation possible est celle indiquée par la figure 4.1.(b) et dans ces conditions l'opération :

MODIFIER P PAR <A:a> DANS Q détruit la relation fonctionnelle :

ℓ : A → B

Par contre si A est un ensemble discriminatoire de la collection P alors pour tout constituant de P on a, si B est un tel constituant

ℓ : A → B.

et l'opération MODIFIER P PAR <B:b> DANS Q n'est pas incompatible.

En effet, comme A est un ensemble discriminatoire, il ne peut exister deux entités <X:x> et <X:x'> , la première appartenant à Q, avec

$$\pi_A(<X:x>) = \pi_A(<X:x'>).$$

Ces propriétés sont générales et pour définir des conditions plus spécifiques d'incompatibilité il sera nécessaire d'étudier avec soin les propriétés des relations fonctionnelles. Nous reviendrons sur ce problème au cours du chapitre 13.

CHAPITRE 5

LANGAGE DE TRAITEMENT DE COLLECTIONS DE DONNEES (L.T.C.D.)

Ce chapitre a pour but de présenter un langage de traitement de collections de données qui servira de base pour la représentation d'un réseau d'informations.

Ce langage permet de décrire d'une part les relations qui existent entre les données (langage de définition) et d'autre par les traitement qui peuvent être effectués sur ces données (langage de traitement). Ces traitements utilisent les opérations définies au cours des chapitres précédents : produit, composition normale, R-composition normale et modification. A ces opérations on ajoutera des fonctions de sélection et d'itération.

De plus, bien que ce langage ne se veuille pas un langage de programmation nous montrerons à l'aide d'exemples comment il permet d'exprimer les requêtes d'un utilisateur dans un environnement de base de données. Dans ce contexte nous indiquerons de façon sommaire la structure d'un système qui accepterait ce langage.

5. LANGAGE DE TRAITEMENT DE COLLECTIONS DE DONNEES (L.T.C.D.)

5.1. Langage de programmation ou langage d'analyse

Nous pensons que les modèles fondés sur la notion de collection de données, et par extension un langage s'appuyant sur ces modèles, permettent de décrire les éléments composant un système d'informations.

Dans ce chapitre, nous présenterons un langage décrivant les traitements,tandis que le chapitre 6 sera consacré à l'utilisation de ce langage pour décrire le réseau de l'information.

Les langages, développés à partir d'un modèle représentant sous forme de relations les données, peuvent être classés en deux grandes catégories :

- approche algébrique : cette approche a été proposée par CHILD [7], GOLDSTEIN et STRNAD [13], E.F. CODD [21].

Bien que l'algèbre de l'information proposée par le comité CODASYL [1] soit plus un modèle qu'un langage, elle contenait néanmoins les principales caractéristiques des langages actuels.

- calcul de prédicat : le calcul des prédicats est un outil puissant et de nombreux auteurs ont fait appel à lui, on peut citer, E.F. CODD [23],[24], LEVIEN et MARON [11], le système TRAMP [25].

Parallèlement, nous nous sommes inspirés des concepts utilisés dans le langage LEAP [29] de FELDMAN et ROVNER, du système PLANNER [30 par HEWITT, ainsi que des développements du projet SOCRATE [15] par J.R. ABRIAL.

Le langage que nous proposons ne se veut pas un langage de programmation mais plutôt un langage descriptif des traitements sur les collections de données. Pour cette raison, nous souhaitons rester proche des conceptions de l'utilisateur et s'exprimer dans un langage relativement synthétique.

Notre approche est de nature algébrique et les propositions que nous faisons se distinguent des propositions déjà existantes sur les points suivants :

- les opérations d'accès aux données s'expriment par une équation algébrique d'où une unification d'écriture.
- introduction de fonctions primitives grâce aux opérateurs de composition, R-composition et produit permettant d'exprimer les requêtes de l'utilisateur.

L'approche algébrique a pour but de former de nouvelles collections de données à partir d'un ensemble de base de collections en utilisant des opérateurs.

Il est important de souligner que ces opérateurs agissent sur la collection tout entière, exactement comme les opérations matricielles considèrent la matrice comme un opérande.

Nous sommes conscients qu'un tel langage ne peut exprimer sans lourdeur les finesses permises par un langage de programmation, mais là n'est pas notre objectif. Notre but est de fournir à l'utilisateur un langage de communication qui lui permette d'exprimer de façon simple les traitements qu'il désire effectuer.

Un tel langage pourrait être dans une certaine mesure un langage permettant une simulation des désirs de l'utilisateur. En effet, la construction d'un système d'information est un processus long et complexe, il peut se mesurer en hommes années et toute erreur est coûteuse. On constate la plupart du temps que la communication entre l'utilisateur et l'analyste-concepteur n'est pas parfaite, par erreur d'interprétation, et que l'utilisateur constate ces erreurs dès le début de fonctionnement du système.

Dans la perspective d'une implémentation, on pourrait effectuer une simulation sur des collections de données qui seraient des échantillons des collections réelles. Les résultats obtenus pourraient être confrontés avec les désirs de l'utilisateur et ainsi on aurait une base de départ pour construire le système d'informations; que le système définitif

soit un système "batch processing" ou de télétraitement avec une banque de données.

Cette simulation présenterait l'avantage d'insérer le système d'informations dans un système de décision ce qui permettrait de connaître les réactions globales du système. Dans l'état actuel, cette étape n'est pas atteinte et nous nous sommes essentiellement placés dans une perspective descriptive qui présente néanmoins l'avantage pour l'utilisateur de définir clairement ses besoins.

5.2. Langage de définition

L'utilisateur doit déclarer les collections de données utilisées par le système d'informations en vue de leur manipulation ultérieure. Les noms des collections de données sont choisis par un utilisateur ou par un ensemble d'utilisateurs, si plusieurs utilisateurs travaillent sur le même système d'informations.

Conformément aux définitions données au chapitre 2, une collection de données est vue comme composée d'entités, ces entités se composant de constituants, chacun des constituants possède un type caractérisant les données.

5.2.1. Les types de constituants

Le type d'un constituant a pour but de définir la nature des valeurs que peut prendre les données. On distinguera :

(a) Type élémentaire :

- Numérique qui se subdivisera en :

  entier

  réel

- alphanumérique qui se subdiviserea en :

  mot chaîne de caractères permettant d'effectuer une recherche sur le constituant ayant ce type dans une requête ou un traitement

  texte zone servant uniquement à l'édition, aucune recherche ne pourra être faite sur un constituant ayant ce type.

- logique booléen donnée ne pouvant prendre que deux valeurs codées par zéro ou un.

Il est fréquent de rencontrer des notions telles que DATE et ADRESSE qui définissent des ensembles de valeurs obtenues par concaténation de valeurs élémentaires. On parle alors d'un constituant composé et on indique les constituants qui le compose. Il va de soi que le type d'un constituant composé doit être compatible avec les types des constituants qui le compose.

Dans le cas d'un constituant composé on devra effectuer une déclaration du type suivant :

```
constituant ADRESSE mot
        début
            RUE mot
            VILLE mot
            CODE-POSTAL entier
        fin
```

Le fait d'utiliser un constituant composé dans la définition d'une collection de données aura des répercussions dans la suite du langage. Par exemple si le constituant ADRESSE apparaît dans une collection de nom P, il sera possible d'effectuer l'opération [ADRESSE]P alors que l'opération [CODE-POSTAL]P sera impossible. Cette impossibilité correspond au souci de distinguer le fait que ADRESSE est un ensemble de valeurs et qui si on désire accèder au code postal il suffit d'effectuer une opération de décodification sur les données représentant l'adresse. D'autre part, cette présentation permet, dans la déclaration d'une collection que nous verrons au chapitre suivant, de n'avoir aucune hiérarchisation des constituants.

Après le type du constituant on précisera éventuellement la liste des valeurs possibles. Pour les données numériques on aura le choix entre deux formes :

- liste de valeurs, par exemple MOIS (1 2 3 4 5 ... 12)
- par bornes, par exemple MOIS (1 à 12)

Pour les données alphanumériques on précisera la liste des valeurs possibles seulement dans le cas du type mot et seule la forme de liste de valeurs pourra être acceptée.

5.2.2. Déclaration de collection

La déclaration de collection a pour but d'indiquer la liste des constituants élémentaires sur laquelle cette collection est définie, c'est-à-dire qu'elle se présente sous la forme :

collection <nom de collection>
début
<liste constituant>
fin

Exemple 5.1.

```
collection EMPLOYE
        début
            NO-EMPLOYE entier
            NOM        mot
            ADRESSE    texte
            SEXE       mot (M F)
            AN-NAISSANCE entier (1970 à 2000)
            MOIS-NAISSANCE entier (1 à 12)
            JOUR-NAISSANCE entier (1 à 31)
            ETAT-CIVIL     mot (M C V D)
            ANNEE-MARIAGE  entier
            NOM-JEUNE-FILLE mot
            PRENOM-ENFANT   texte
            AN-NAISSANCE-ENFANT entier
            SALAIRE        réel
            NO-DPT         entier
        fin

collection DEPARTEMENT
        début
            NO-DPT  entier
            CHEF-DPT entier
            NOM-CHEF-DPT mot
        fin
```

Parmi l'ensemble des collections déclarées par l'utilisateur nous distinguerons deux classes :

- les collections permanentes qui correspondent à l'information enregistrée, à partir desquelles on va créer de nouvelles collections,
- les collections temporaires qui sont créées pour répondre aux besoins de traitements et qui peuvent être détruites une fois la séquence de travail terminée.

Pour allèger l'écriture des déclarations de collections on pourra :

(1) soit utiliser la clause idem, par exemple

collection EMPLOYE-NOUVEAU idem EMPLOYE

(2) soit ne pas préciser la liste des constituants, si cette collection est obtenue par une suite d'opérations qui permet de déterminer sans ambiguïté l'ensemble des constituants élémentaires sur lequel elle est définie. Nous verrons ultérieurement dans quel cas une collection est définie sans ambiguïté par une suite d'opérations.

Compte-tenu des remarques ci-dessus nous ajouterons les clauses "permanente" et "temporaire" ce qui donnera :

collection permanente EMPLOYE

collection temporaire EMPLOYE-NOUVEAU idem EMPLOYE

5.2.3. Déclaration d'entités

A une collection on désire souvent associer un ou plusieurs nom(s) d'entités qui désigneront des éléments distincts de la collection. Ce besoin correspond à la nécessité au cours d'un traitement sur la collection EMPLOYE d'avoir deux réalisations d'entités différentes, l'une sera identifiée par l'identificateur X et l'autre par l'identificateur Y.

Nous écrirons alors :

entité (EMPLOYE) X Y fin

où sous une forme plus formelle :

entité (nom de collection) <liste d'entités> fin

avec cette forme de déclaration les constituants des entités X et Y sont ceux de la collection EMPLOYE. S'il n'est pas fait référence à une collection on utilisera alors la forme :

entité <nom d'entité> début <liste-constituant> fin

Les entités X et Y ayant été déclarées il sera possible d'effectuer sur elles des opérations de calcul et des opérations d'affectation. Pour cela il faut pouvoir accèder aux valeurs contenues dans l'entité, ceci sera réalisé par l'opérateur ".". Par exemple :

NOM.X

SEXE.X

désigneront respectivement la valeur du constituant NOM dans l'entité X et le sexe de X. Dans le cas où l'on désire accèder aux valeurs élémentaires d'un constituant composé, on écrira :

CODE-POSTAL.ADRESSE.X

(voir exemple de constituant composé donné au paragraphe 5.2.1.)

La valeur d'un constituant d'une entité pourra être dans l'un des quatre états suivants :

- définie	NOM.X égal à dupont
- indéfinie	NOM.X égal à 'b' c'est-à-dire que X a un nom que l'on ne connaît pas
- nulle	PRENOM-ENFANT.X égal à nul car l'entité X représente une personne qui n'a pas d'enfant.
- impossible	ANNEE-MARIAGE.X égal ' @ ' car l'état-civil de l'entité X est C (célibataire).

Ceci nous montre que les valeurs des constituants d'une entité ne sont pas toutes compatibles entre elles. Il sera donc important de définir les restrictions associées à une entité, ce qui sera fait au stade de la création des entités d'une collection.

Le but de ces restrictions est de préserver l'intégrité des données en permettant éventuellement un contrôle de validité chaque fois que l'utilisateur introduit ou met à jour des données, (voir paragraphe 2.2.3.).

5.3. Opérations de création de collections de données sans modification de valeur

Ces opérations ont pour but de créer de nouvelles collections de données à partir de l'ensemble des collections de données décrites à l'aide du langage de définition. Pour désigner ces nouvelles collections de données on utilisera les opérateurs définis et étudiés au chapitre 3, à savoir les opérateurs de produit, de projection, de composition normale et de R-composition. A ces opérations on adjoindra des fonctions de sélection et des fonctions itératives.

5.3.0. Opérations d'union, d'intersection et de différence de collections de données

Si P et Q sont des noms de collections nous noterons par :

P inter Q

P union Q

P diff Q

respectivement l'intersection, l'union et la différence de collections. Ces opérations sont uniquement justifiées dans le cas où P et Q ont les mêmes ensembles de constituants.

5.3.1. Opération de projection

L'opération de projection d'une collection a été définie au paragraphe 2.3.2 et dans le langage nous adopterons les conventions suivantes pour distinguer la projection d'une entité d'une part et la projection d'une collection d'autre part.

Projection d'entité

NOM.X désignera la valeur du constituant NOM de l'entité X

Projection de collection

La projection d'une collection définit une nouvelle collection de données dont l'espace de définition est celui de la liste des noms de constituants figurant dans l'expression; ainsi :

[NOM AN-NAISSANCE SALAIRE] EMPLOYE

désigne une collection de données d'espace de définition :

(NOM, AN-DATE-NAISSANCE, SALAIRE).

De même :

[<NOM : dupont> PRENOM-ENFANT AN-NAISSANCE-ENFANT] EMPLOYE

désignera une collection de données contenant les enfants des diverses personnes qui s'appellent dupont.

On peut appliquer successivement l'opérateur de projection, ainsi :
[AN-NAISSANCE-ENFANT] [<NOM:dupont> PRENOM-ENFANT AN-NAISSANCE-ENFANT]EMPLOYE
sera les années de naissance des enfants de dupont. Ceci sera possible car il faut se rappeler qu'il y a plusieurs entités de nom dupont, plus précisément autant que dupont a d'enfants.

Si dupont n'avait pas eu d'enfant, le résultat est une collection de données définies sur le constituant élémentaire AN-NAISSANCE-ENFANT et qui possède une seule entité dont la valeur du constituant est nulle.
Si une collection de données comporte une seule entité dont tous les constituants ont une valeur nulle, on considèrera la collection comme vide. Cette remarque est nécessaire pour la définition de l'instruction conditionnelle, car on pourra tester si une collection est vide.

Remarque : *L'opérateur de projection définit sans ambiguïté de nouvelles collections temporaires, on n'est donc pas obligé de les déclarer.*
D'autre part les constituants de la liste entre crochets ne peuvent appartenir qu'à l'espace de définition de la collection.

5.3.2. Opération de produit

L'opération de produit a été définie au paragraphe 3.2. Si P et Q sont deux collections de données d'espaces de définition respectifs (A,B) et (C,D) alors P,Q est une collection d'espace de définition (A,B,C,D).

Toutefois, dans le cas où les constituants élémentaires de P et Q ne sont pas tous distincts il est nécessaire de déclarer la collection résultante. Dans ce cas le transfert des valeurs se fera dans l'ordre des constitutants déclarés et les valeurs seront affectées à ces nouveaux noms.

5.3.3. Opération de composition normale

L'opération de composition normale a été définie au paragraphe 3.4.2. L'opération de composition normale se fait toujours sur l'intersection des espaces de définition et par conséquent est définie sans ambiguïté.

5.3.4. Opération de R-composition normale

Nous avons vu au paragraphe 3.5.1 que la R-composition normale peut être considérée comme une extension de la composition normale si on a défini une relation R entre certains constituants des deux collections composantes.

Supposons par exemple que P et Q soient de collections de données d'espace de définition (A,B,C,D) et (E,F,G) alors :

$$P * (C = E \wedge D = F)\ Q$$

désigne une collection de données d'espace (A,B,C,D,E,F,G) telle qu'une entité U vérifie C . U = E . U et D . U = F . U, le symbole "$\wedge$" étant mis à la place du "et".

A la place de l'opérateur =, on peut utiliser les opérateurs : $>$, $\geqslant$, $<$, $\leqslant$, $\neq$.

Remarque : Dans le cas où P et Q comportent des constituants élémentaires communs et que l'on désire effectuer une R-composition, il est nécessaire d'utiliser une règle de transmission des noms. Nous utiliserons la même règle que celle donnée pour le produit (cf. paragraphe 5.3.2).

5.3.5. Affectation et expression d'affectation de collections

Les opérateurs de projection, composition normale et R-composition permettent de créer de nouvelles collections de données.

L'instruction d'affectation se notera comme en ALGOL par le symbole :=

Par exemple, l'instruction :

$$R := P * Q$$

créera une collection de données de nom R comme étant le résultat de l'opération P * Q. L'espace de définition de R est l'espace de définition de P * Q.

Bien que le but du langage L.T.C.D. soit de décrire les opérations de traitement dans un système d'informations, il est intéressant de signaler comment de telles expressions pourraient être implémentées dans un langage de programmation.

En effet, on peut considérer que dans un système acceptant le langage L.T.C.D. l'espace de stockage de l'information est divisé en deux parties :
- une zone de travail,
- une zone pour les collections de données permanentes.

C'est dans cette zone de travail que sera rangé la collection de données R.

Si aucune notion d'ordre n'est indiquée pour la relation R alors, l'ordre de R sera déterminé par le système compte tenu de l'organisation physique des collections de données P et Q.

Si on désire indiquer que la collection de données doit être ordonnée suivant un certain ordre, cela pourra se faire par l'opération suivante inspirée du langage APL, avec en plus l'indication des constituants intervenant dans le critère de tri :
- ordre croissant ↑ (<liste des constituants>)
- ordre décroissant ↓(<liste des constituants>)

Exemple :

R := ↑(A B) P

La collection R créée en zone de travail est ordonnée suivant les constituants A et B à partir de la collection P.

Remarque :

Si l'espace de définition de la collection est réduite à un seul constituant, alors il n'est pas nécessaire de faire figurer après les symboles ↑, ↓ la liste des constituants intervenant dans le critère de tri.

Dans le cas où une expression d'affectation fait apparaître plusieurs opérateurs, il est nécessaire de donner la règle indiquant l'ordre des opérations.

L'ordre d'exécution de l'instruction dépendra :
- des parenthèses,
- de l'ordre d'apparition des opérateurs dans l'instruction.

Règle sur les opérateurs

L'argument droit d'un opérateur (par rapport à l'argument gauche est tout ce qui est à sa droite. Un opérateur agit sur l'argument droit.

Si on utilise l'opérateur de produit ou de R-composition normale l'expression d'affectation ne doit contenir que l'opération correspondante. Ceci afin d'éviter toute ambiguité dans le mécanisme d'affectation.

Règle sur les parenthèses

L'utilisation des parenthèses est celle habituellement utilisée. C'est-à-dire, que les opérations à l'intérieur des parenthèses sont exécutées avant les opérations à l'extérieur des parenthèses.

Toutefois, dans le cas d'expression manipulant des collections de données, la règle classique qui veut qu'une expression à l'intérieur de parenthèses doit être évaluée en premier conduirait dans certains cas au moment de l'exécution à des baisses de performances sensibles. Ce problème d'optimisation des séquences de traitement ne sera pas abordé dans le cadre de ce travail.

Exemple :

$R := \uparrow(A\ B)\ \ [A\ B\ C]\ P * [C\ D]\ Q$

est équivalent à la suite d'opérations :

$U := [C\ D]\ Q$

$V := P * U$

$T := [A\ B\ C]\ V$

$R := \uparrow(A\ B)\ T$

Par contre l'opération :

$$R := \uparrow(A\ B)\ ([A\ B\ C]\ P) * [C\ D]\ Q$$

est équivalente à :

$$U := [C\ D]\ Q$$

$$V := [A\ B\ C]\ P$$

$$T := U * V$$

$$R := \uparrow(A\ B)\ T$$

5.3.6. Opération d'affectation d'entité

On pourra désigner une réalisation d'entités par un identificateur d'entité, si cette entité appartient à une collection de données. On peut aussi construire une entité à partir de l'opérateur de produit. Par exemple, l'opération :

U := <NOM : dupont>, <AGE : 42>

créera de façon implicite une entité U d'espace de définition (NOM, AGE).

Dans le cas où l'entité aurait été préalablement déclarée, il sera possible d'écrire l'instruction d'affectation sans faire figurer le nom des constituants à condition que l'ordre des valeurs soit identique à l'ordre de leur affectation aux constituants.

Par exemple, si X est une entité de la collection DEPARTEMENT (voir exemple 1), alors :

X := 16, 1395, durand

est équivalent à :

X := <NO-DPT : 16>, <CHEF-DPT : 1395>, <NOM-CHEF-DPT : durand>

Il est important de connaître si une entité est présente dans une collection de données faisant partie du système d'informations.

L'instruction qui permettra de le vérifier sera un test d'existence qui retournera une valeur 1 (vrai) si l'entité appartient à la collection, sinon 0 faux) dans le cas contraire.

Ce test d'existence s'exprimera sous la forme :

[<entité>] <nom de collection>

Par exemple, si P est une collection de données d'espace de définition (NOM, AGE, SEXE) le test qui aura pour expression :

[<NOM : durand>, <AGE : 15>, <SEXE : f>] P

délivrera un 1 si l'entité appartient à la collection P.

Il sera intéressant de définir la réponse du système dans le cas où l'entité dans la collection P serait : <NOM : durant>, <AGE : 'b'> <SEXE : f> qui signifierait que l'on ne connaîtrait pas l'âge de la personne.

On peut adopter une clause qui indiquerait au système que dans le cas d'un blanc 'b' on accepte la réponse comme valable ou bien qu' au contraire on rejette ces réponses incomplètes.

D'une façon plus générale, si l'utilisateur désire savoir si il existe une entité ayant pour âge : 15 et pour sexe : f, il devra écrire le test :

[<SEXE : f>, <AGE : 15>]P.

5.3.7. Opération d'introduction d'entité

Inversement, une instruction voisine du test d'appartenance permet d'ajouter ou de retirer une entité à une collection de données.

L'instruction, [<entité>] <nom de collection> := 1

aura pour effet d'ajouter une entité à la collection si les valeurs ajoutées sont compatibles avec le type des constituants de la collection. Dans le cas où il y aurait incompatibilité, le système retournerait un message indiquant les causes de cette incompatibilité.

L'instruction :

[<entité>] <nom de collection> := 0

provoquera la soustraction de l'entité désignée par la liste des données dans la collection. Cette opération n'a aucun effet si l'entité n'est pas présente dans la collection.

Nous avons vu, qu'en général les opérations d'introduction ou de suppression d'entités peuvent être des opérations qui ne respectent pas des restrictions définies sur une collection de données. C'est-à-dire qu'il est souhaitable de pouvoir contrôler ces opérations, en écrivant l'instruction suivante :

[<entité>] <nom de collection> := $\left\{\begin{matrix}1\\0\end{matrix}\right\}$ contrôle <nom de procédure>

Cette instruction provoquera le déroutement du programme vers une procédure dont le paramètre d'entrée sera l'entité et vérifiera s'il est possible d'introduire l'entité ou de la supprimer. En fin de procédure le programme pourra être soit dans l'état ECHEC ou SUCCES. Cette notion d'échec ou succès a été introduite par J.R. ABRIAL dans [15] et permet de savoir si l'introduction ou la suppression peut être effectuée. Cette procédure pourra même avoir des effets de bord, c'est-à-dire qu'elle pourrait elle-même, provoquer des modifications dans d'autres collections de données. Nous reviendrons sur cette opération de contrôle avec le paragraphe sur les procédures. Si l'on souhaite introduire, non pas une entité, mais plusieurs entités formant une collection on pourra écrire :

[EMPLOYE-NOUVEAU] EMPLOYE := 1 contrôle INTROEMPLOYE

Cette opération n'est valide que si les espaces de définition des collections sont identiques. D'autre part, le contrôle se fera pour chaque entité de la collection EMPLOYE-NOUVEAU.

5.3.8. Les opérations d'accès à une collection

Dans la plupart des langages de programmation les instructions d'accès aux fichiers sont distinctes des autres instructions.

Il nous a semblé intéressant de pouvoir exprimer l'opération d'accès à une collection de données comme une opération d'affectation, ceci est possible grâce à une propriété de l'opérateur de composition normale.

Supposons que P soit une collection de données d'espace de définition (NOM, AGE, SEXE) et que le constituant NOM soit un index (cf. paragraphe 2.3.4.) pour la collection P, alors l'expression :

E := <NOM : durand>* P

affectera dans une zone identifiée par le symbole E une collection de données dont les entités correspondantent aux personnes de nom durant. Toutefois comme le constituant NOM est un index et si on a déclaré E comme une entité de la collection P alors l'instruction sera valide. Dans le cas où E aurait été déclaré comme une entité mais que NOM ne soit pas un index alors l'instruction serait considérée comme valide mais il ne serait pas possible de connaître laquelle des entités est amenée par le système dans la zone de nom E.

Si on avait désiré l'âge de durant, il aurait suffi d'écrire :

E := [AGE] <NOM : durand> * P.

Cette remarque nous permet de définir la liste des valeurs d'une collection liée à une valeur donnée d'un constituant ou à plusieurs couples constituant-valeur.

Par exemple l'expression :

E := <AGE : 15> , <SEXE : f> * P.

définit la collection des entités de P qui ont pour âge 15 et pour sexe f, tandis que l'instruction,

E := [NOM] <AGE : 15>, <SEXE : f> * P

donne les noms de personnes.

Dans tout ce paragraphe nous avons considéré jusqu'à maintenant que le composant élémentaire d'une entité est le couple : constituant-valeur. La valeur étant exprimée sous forme d'une donnée que l'on attache au constituant. Une conception plus générale consiste à considérer qu'à un constituant on peut attacher toute valeur qui résulte d'un calcul effectué au cours du programme. Ceci signifie qu'une valeur pourra être éventuellement une variable.

Par exemple si E désigne une entité appartenant à la collection département (cf. paragraphe 4.2.5.) alors :

<NO-EMPLOYE : CHEF-DPT . E>

est une expression valide car elle consiste à considérer que la valeur de NO-EMPLOYE est celle obtenue lors de l'évaluation de l'expression CHEF-DPT.E

D'autre part l'expression :

<NO-EMPLOYE : CHEF-DPT.E> * EMPLOYE

sélectionnera une entité d'employé correspondant au chef de département de l'entité E.

REMARQUE :

Si X désigne un nom de constituant et E le nom d'une entité des expressions de la forme :

<X:X.E>

sont courantes, nous conviendrons par souci de simplification d'écrire seulement X.E .

5.3.9. Expressions conditionnelles

Une expression conditionnelle a pour but de sélectionner parmi les entités d'une collection celles qui possèdent un critère donné.

La structure de l'instruction aura une des deux formes suivantes :

si <expression-conditionnelle> alors
:
sinon
:
fin

si <expression conditionnelle> alors ... fin

Dans le bloc alors et sinon il pourra être effectué soit des opérations exprimant des traitements sur des collections, soit des calculs arithmétiques, soit des expressions conditionnelles.

Une expression conditionnelle sera définie à partir de la notion d'élément atomique.

Un élément atomique est de la forme :

<Ea> → <partie gauche> <opc> <partie droite>

<opc> → > | ≤ | ≥ | = | ≠

<partie gauche> → <atome>

<partie droite> → <atome> | <identificateur> | <donnée>

<atome> → <nom de constituant> . <nom d'entité>

Exemple :

AGE.X ≥ 20

NOM.X = NOM.Y

PRIX.X = PRIXTOTAL

Une expression conditionnelle (EC) sera définie récursivement de la façon suivante :

(1) tout élément atomique est une expression conditionnelle.

(2) si EC est une expression conditionnelle alors $\neg$EC est ausii une expression conditionnelle. Le symbole "$\neg$" est celui de la négation

(3) si EC_1 et EC_2 sont deux expressions conditionnelles, alors ($EC_1 \vee EC_2$) et ($EC_1 \wedge EC_2$) sont des expressions conditionnelles. La signification des symboles "$\vee$" et "$\wedge$" est respectivement celui de l'union et de l'intersection.

(4) si EC est une expression dans laquelle figure une entité X d'une collection P qui apparaît comme une variable libre alors :

$\forall X$ dans P(EC)

$\exists X$ dans P(EC)

sont des expressions conditionnelles.

Exemples 5.2.

(1) Si PERSONNE désigne une entité de la collection EMPLOYE (cf. exemple 5.1) l'expression ci-dessous est une expression conditionnelle.
si (AGE.PERSONNE = 20 $\vee$ SEXE.PERSONNE = m) $\wedge \neg$(SALAIRE.PERSONNE $\geqslant$ 1000 $\wedge$ SALAIRE.PERSONNE $\leqslant$ 2000) alors [PERSONNE]T := 1 fin

(2) Si FOUR, PROJET et FR désignent respectivement trois collections de données qui représentent le nom et l'adresse d'un fournisseur, le projet et son nom, enfin les pièces fournies par un fournisseur pour un projet donné, c'est-à-dire que les déclarations de collections seraient:

```
collection FOUR
           début
           NUMERO-FOUR
           NOM
           ADRESSE
           fin
collection PROJET
           début
           NUMERO-PROJET
           NOM-PROJET
           fin
```

collection FR
début
NUMERO-FOUR
PIECE
NUMERO-PROJET
fin
entité (FOUR) X fin
entité (PROJET) Y fin
entité (FR) Z fin

Si on veut vérifier qu'un fournisseur désigné par l'entité X fournit tous les projets, on pourra écrire :

si $\forall$*Y dans PROJET* ($\exists$*Z dans FR (NUMERO-FOUR.Z = NUMERO-FOUR.X*
$\wedge$ *NUMERO-PROJET.Y = NUMERO-PROJET.Z))*
alors ...
fin

5.3.10. Opérations itératives

Dans la plupart des langages de base de données la fonction itérative est implicite eu égard au caractère généralement fonctionnel de ces langages. Il nous a semblé préférable de démonter ce mécanisme d'itérations implicites en donnant la possibilité à l'utilisateur d'exprimer une boucle d'itération.

La forme de l'instruction sera :

pour <nom-d'entité> dans <collection> faire
début
⋮ instructions
fin

Cette fonction itérative exprime que pour chaque entité de la collection les instructions comprises entre début et fin seront exécutées.

Remarque :

Il pourrait être intéressant d'avoir deux mécanismes d'exécution de la boucle "pou", mais cela nécessiterait d'avoir deux types de déclarations d'entité.

Le premier type serait entité repère [<nom de collection>] <liste nom d'entité> fin

Le deuxième type serait entité <nom d'entité> début <liste constituant> fin

Dans une telle hypothèse dans le premier type une entité repèrerait une entité de la collection de données. Dans le deuxième type il y aurait création d'une entité. Sous ces conditions le mécanisme de la boucle "pour" pourrait être différent.

(a) mécanisme associé avec une entité-repère.

soit la séquence d'instructions :

```
entité repère (P) X fin
collection P

   pour X dans P faire
   {
   }
   fin
```

X repère une entité de la collection P et X repèrera successivement toutes les entités de la collection P.

(b) mécanisme associé avec une entité

```
collection P
entité X
pour X dans P faire
   :
fin
```

Chaque entité de la collection P est recopiée successivement dans la zone définie par l'entité X.

Dans notre cas nous admettrons que n'ayant pas fait de distinction sur les entités, la boucle "pour" correspond au mécanisme (b).

Les différents exemples ci-dessous illustreront les possibilités de la fonction itérative associée à des expressions conditionnelles.

Exemple 5.3.

Calculer le salaire de toutes les PERSONNES de la collection EMPLOYE ?

```
Réel TOTAL-SALAIRE
     pour PERSONNE dans EMPLOYE faire
          début
               TOTAL-SALAIRE := TOTAL-SALAIRE + SALAIRE . PERSONNE
          fin
```

Exemple 5.4.

Calculer le salaire de tous les chefs de département ?

```
Réel TOTAL-CHEF
     pour E dans DEPARTEMENT faire
          début
               X := <NO-EMPLOYE : CHEF-DPT.E> * EMPLOYE
               TOTAL-CHEF := TOTAL + SALAIRE.X
          fin
```

Exemple 5.5.

Rechercher les noms de personnes ayant tous leurs enfants âgés de moins de 10 ans ?

```
booléen D
pour X dans [NO-EMPLOYE] PERSONNE
     début
          D := 1
          pour Y dans [<NO-EMPLOYE : NO-EMPLOYE.X>NOM AGE] EMPLOYE
               début
                    si AN-NAISSANCE.Y ⩾ 1963 alors D := D ∧ 1
                                             sinon D := D ∧ 0 fin
               fin
          si D = 1 alors [NOM.X] T := 1
     fin
```

La collection T contient la réponse à la question posée.

Une autre expression de ce même programme peut être obtenue en introduisant la notion d'expression conditionnelle quantifiée qui a été déjà définie :

```
pour X dans [NO-EMPLOYE] EMPLOYE
début
    si ∀Y dans [<NO-EMPLOYE : NO-EMPLOYE.X> NOM AGE] EMPLOYE
      (AN-NAISSANCE.Y ⩾ 1963)
    alors [NOM.X]T := 1 fin
fin
```

Cet exemple avait pour but de monter les relations qui existent entre l'expression conditionnelle avec quantificateur et la boucle "pour".

5.4. Opérations de dérivations et de modifications de collections de données

5.4.1. Dérivation de collection

Nous voulons montrer comment à partir du langage L.T.C.D., il est possible de créer de nouvelles collections de données, en effectuant des calculs sur les constituants initiaux.

Exemple 5.6.

Cet exemple sera inspiré de la facturation d'articles commandés pour lequel est défini les collections ci-dessous.

```
collection COM
        début
            NO-COM entier
            NO-ART entier
            QUANTITE entier
        fin
```

```
collection ARTICLE
        début
            NO-ART entier
            DESCRIPTION texte
            PRU réel
        fin
réel PA, PT

pour S dans  [NO-COM] COM faire
      début
          PT := 0
          pour T dans [<NO-COM : NO-COM.S> NO-ART QUANTITE] COM faire
          début
              U := <NO-ART : NO-ART.T> * ARTICLE
              PAR := PRU.U   QUANTITE.T
              [S U QUANTITE.T <PRIXA : PA>] R := 1
              PT := PT + PA
          fin
          [S<PRT : PT>]  V := 1
       fin
fin
```

Après exécution de ces instructions on a généré deux nouvelles collections de données R et V définies respectivement sur les constituants :

(NO-COM, NO-ART, PRU, QUANTITE, PRIXA) et (NO-COM,PT)

Ces deux collections contiennent bien toutes les informations nécessaires à l'édition de la facture.

Comme on peut le constater à partir de cet exemple, si le langage permet de définir de façon logique les opérations à effectuer, il

n'indique pas pour autant, comme le permet un langage de programmation de façon précise, l'ordre dans lequel les opérations élémentaires sont effectuées. Ceci résulte du fait que les langages tels que COBOL, PL1, etc... tiennent compte très fortement de l'organisation physique des données et de la présentation successive en mémoire centrale des enregistrements.

Cette remarque justifie notre objectif défini dans l'introduction de ce chapitre, à savoir que ce langage est un modèle, indépendant de l'organisation des données, pour traduire les requêtes de l'utilisateur ou bien de servir comme un langage permettant d'analyser les traitements que l'utilisateur désire effectuer.

5.4.2. Modifications de collections

Aux paragraphes 4.1.1., 4.1.2 et 4.1.3 nous avons défini les opérations d'addition, suppression et modifications de collections dont les paramètres sont des collections.

Exemple 5.7.

Augmenter de 8 % le salaire des personnes ayant un salaire inférieur à 1500 francs et de 6 % les autres.

```
pour PERSONNE dans EMPLOYE faire
   début
        si SALAIRE.PERSONNE < 1500
            alors SA := SALAIRE.PERSONNE × 1.08
            sinon SA := SALAIRE.PERSONNE × 1.06
        fin
            modifier PERSONNE par <SALAIRE : SA>
   fin
```

De la même façon que nous avons vu qu'une opération d'addition ou de suppression pouvait être contrôlée par une procédure, l'opération de modification pourra respecter la même règle, avec la seule différence que l'on devra transmettre à la procédure, l'entité sur laquelle on effectue la modification, ainsi que les valeurs modifiées.

5.4.3. Procédure

Les opérations d'addition, de suppression et de modification d'entités peuvent être contrôlées par des procédures.

Nous considérerons qu'une procédure peut produire une valeur ou un ensemble de valeurs, ces valeurs pouvant être du type entité et/ou collection. D'autre part, on devra indiquer les paramètres de la procédure.

La représentation formelle d'une déclaration de procédure sera :

procédure <nom de procédure> $\alpha(\mu_1\ \mu_2\ \ldots\ \mu_n)$

où α est le type du résultat produit par la procédure et μ_1, μ_2,..., μ_n des déclarations de paramètres formels.

Si la procédure ne produit pas de résultat l'écriture sera alors :

procédure <nom de procédure> $(\mu_1\ \mu_2\ \ldots\ \mu_n)$

Si elle n'a pas de paramètres :

procédure <nom de procédure> α,

et enfin si elle n'a pas ni paramètres, ni résultat

procédure <nom de procédure>

Lors de l'appel de la procédure on donnera aux paramètres formels leurs valeurs actuelles et il sera considérer comme un appel par nom.

Illustrons ceci par un exemple tiré de l'introduction d'une entité dans une collection d'emploi du temps.

Exemple 5.8.

```
collection EMPT
        début
            PROFESSEUR
            HEURE
            SALLE
            CLASSE
        fin
```

Dans la collection EMPT nous ferons l'hypothèse que toutes les entités vérifient le fait qu'un professeur à une heure donnée ne peut être que dans une seule salle avec une seule classe.

```
PROGRAMME
    .
    .
    .
  [X]EMPT := 1 contrôle INTRO-EMPT
  si ECHEC alors allera FIN fin
    .
    .
    .
FIN : fin
procédure INTRO-EMPT (entité (EMPT)X)
  si (<PROFESSEUR : PROFESSEUR.X>, <HEURE : HEURE.X>)* EMPT ≠ nul
  alors ECHEC allera FIN
  sinon [X]EMPT := 1 fin
FIN : fin
```

Lorsque l'exécution du programme rencontre l'instruction [X]EMPT := 1 contrôle INTRO-EMPT, il est dérouté vers la procédure INTRO-EMPT sans exécuter l'opération d'introduction.Si la procédure se termine sans échec, alors l'opération d'introduction est effectivement exécutée sinon une autre séquence peut être exécutée. Dans l'exemple ci-dessus l'échec correspond au fait qu'il existe déjà une entité où le professeur enseigne à cette même heure.

Le lecteur pourrait nous faire remarquer qu'étant donné que la première partie de l'instruction :

[X]EMPT := 1 contrôle INTRO-EMPT

est sans effet, il n'est pas nécessaire de la mentionner, et que l'instructio se comporte comme un appel de procédure. Nous avons retenu cette forme afin que l'utilisateur indique le nom de l'entité et la collection concernée, tout en précisant la nature de l'opération de mise-à-jour qui subit le contrôle.

Il est bien important de remarquer que l'opération effective d'introduction de l'entité se fait à l'intérieur du corps de la procédure.

Cette façon de procéder laisse à l'utilisateur, toute possibilité d'effectuer, à bon escient, des opérations de mise à jour lorsqu'il y a plusieurs contrôles en cascade.

On peut aussi à l'aide de la notion de procédure, construire de nouvelles relations. Pour cela nous allons prendre un exemple que l'on rencontre fréquemment.

Exemple 5.9.

Etant donné une arborescence qui peut être décrite par une collection de données :

```
collection SUIVANT
        début
            PERE
            FILS
        fin
```

A l'entité <PERE:A> on veut faire correspondre une collection de données :

```
collection DESCENDAND
        début
            FILS
        fin
```

qui donne toutes les entités successeurs de <PERE:A>.

```
procédure DESCENDANT collection début FILS fin
        (entité X début PERE fin)
    entité U début PERE fin
    entité T Z début FILS fin
    pour Z dans [<PERE:PERE.X> FILS] SUIVANT faire
                [Z]DESCENDANT := 1
                U := <PERE : FILS.Z>
            pour T dans DESCENDANT(U) faire
                [T]DESCENDANT := 1
            fin
    fin
fin
```

Conclusion

Le langage que nous venons de présenter, bien que ne comportant pas toutes les fonctions que l'on peut rencontrer dans des langages beaucoup plus évolués, correspond à nos besoins pour la description d'un modèle de système d'informations que nous décrirons dans le chapitre 6.

5.5. Application à la formulation des questions dans une base de données

Nous avons au cours de la présentation du langage, donner de nombreux exemples permettant de formuler des requêtes. Nous voulons illustrer dans ce paragraphe que des questions qui paraissent complexes, s'expriment très synthétiquement à l'aide du langage.

Exemple 5.10.

Nous supposerons que la base de données est composée de trois collections correspondant aux déclarations suivantes :

```
collection PERSONNE
        début
            NOM mot
            NO-VOITURE entier
            COULEUR-VOIT mot
            NO-PARKING est NO-GARAGE
        fin

collection GARAGE
        début
            NO-GARAGE mot
            LIEU est NO-MAISON
            PROPRIETAIRE est NOM
        fin

collection MAISON
        début
            NO-MAISON mot
            COULEUR-MAISON mot
            LOCATAIRE est NOM
        fin
```

La figure 5.2 donne un ensemble d'occurences de valeur pour chaque collection.

Les questions à formuler sont les suivantes :

Question 1 : Liste des numéros de voiture dont la couleur est identique à la couleur de la maison dont le locataire est le propriétaire de cette voiture.

Question 2 : Liste des personnes ayant une voiture dont ils sont propriétaires du parking et locataires de la maison du lieu de parking et de plus la couleur de la voiture est identique à la couleur de la maison.

Question 3 : Liste des personnes ayant une voiture de couleur différente des couleurs des autres voitures des autres personnes.

PERSONNE

NOM	NO-VOITURE	COULEUR-VOIT	NO-PARKING
durant	1	bleu	α
durant	2	vert	β
dupont	3	bleu	γ
zoé	4	rouge	δ
durant	5	rouge	ε

GARAGE

NO-GARAGE	LIEU	PROPRIETAIRE
α	paris	zoé
β	marseille	dupont
γ	versailles	zoé
δ	sèvres	durant
ε	sèvres	durant

MAISON

NO-MAISON	COULEUR-MAISON	LOCATAIRE
paris	bleu	zoé
marseille	rouge	dupont
versailles	blanc	dupont
sèvres	rouge	durant

FIGURE 5.2.

Question 1 :

T := [NO-VOITURE] PERSONNE ✱(NOM COULEUR-VOIT =
LOCATAIRE COULEUR-MAISON) MAISON

Question 2 :

Pour répondre à cette question il suffit de construire successivement les collections T et U.

T := PERSONNE ✱(NO-PARKING NOM = NO-GARAGE PROPRIETAIRE) GARAGE·
U := [NOM] MAISON ✱(LOCATAIRE COULEUR-MAISON = NOM COULEUR-VOIT) T

La collection de données U contient les éléments de réponse. Dans l'exemple donné,U ne possède qu'une seule valeur 'durand'.

Question 3 :

```
pour X dans PERSONNE faire
     si ∀Y ≠ X dans PERSONNE (COULEUR.Y ≠ COULEUR.X) alors
               [NOM.X] T := 1
     fin
fin
```

CHAPITRE 6

MODELISATION DES SYSTEMES D'INFORMATIONS D'ENTREPRISE

Le but de ce chapitre est de présenter un modèle de systèmes d'informations d'entreprise. Ce modèle comprendra la description des données et les traitements que l'on pourra effectuer en utilisant le langage LTCD (chapitre 5) mais aussi la modélisation des processus d'organisation et de décision de l'entreprise. Cette conception correspond au fait, que dans notre esprit, on ne peut dissocier les traitements et les données.

Les éléments essentiels du modèle sont : les niveaux, les flux, les réseaux, les procédures de traitement et de transfert.

6. MODELISATION DES SYSTEMES D'INFORMATIONS D'ENTREPRISE

6.1. Introduction

6.1.1. Définition du problème de la modélisation des systèmes d'informations

Le problème de l'analyse des systèmes d'informations relève plus d'approche expérimentale que rationnelle. Les raisons de cette situation tiennent essentiellement au fait qu'il est difficile d'avoir simultanément un modèle qui représente le fonctionnement de l'entreprise, et le système de l'information de l'entreprise.

Les premiers outils que l'analyste-concepteur a disposé pour son travail ont été les graphes de circulation de documents utilisés par les organisateurs. Ces outils, d'un point de vue information, présentent plusieurs inconvénients :

- ils décrivent seulement la circulation des documents et non pas celle de l'information,
- ils ne montrent pas quelles sont les opérations de traitements qui sont effectués sur ces documents, mis à part des traitements élémentaires tels que :
  - . transcription
  - . report
  - . duplication
  - . contrôle.
- il est difficile de reconstruire le modèle de l'entreprise à partir du graphe de circulation des documents bien que la circulation des documents devrait reflèter ce modèle.

Parmi les modèles construits sur la notion de graphe de circulation de documents on peut citer AUTOSATE [2]et MODSIN [3].

A l'opposé des graphes de circulation de documents, les travaux de J.W. FORRESTER [6] avec "Industrial Dynamics" portent sur la construction des modèles d'entreprise.

6.1.2. Rappels sur le modèle de J.W. FORRESTER

Dans "Industrial Dynamics" les activités industrielles et économiques de l'entreprise constituent un système continu de contrôle permanent.

La méthode conçue pour simuler le comportement dynamique de certaines grandeurs physiques, présente une grande homogénéité; elle procède de l'analyse quantitative et empirique du système que l'on veut étudier puis après une première étape de symbolisation graphique, elle achève le modèle par une formulation mathématique simple et particulièrement adaptée au traitement sur ordinateur.

Le principe fondamental du modèle est que l'entreprise peut être considérée comme un système de contrôle permanent car les actions ne résultent que de l'écart entre l'état du système observé et l'état désiré défini par les objectifs de l'entreprise. Ce système de contrôle peut être représenté par le schéma ci-dessous (figure 6.1).

Sur cette figure nous voulons affirmer que toute décision est prise en fonction des objectifs fixés et de ce que nous avons appelé l'état apparent du système. Celui-ci, est l'image donnée par différents canaux d'informations branchés sur l'état réel du système. Ces canaux ne sont pas parfaits car la transmission des informations n'est pas instantanée et parce que d'autre part, elles subissent des déformations dues aux erreurs de mesure, de traitement et d'interprétation des résultats.

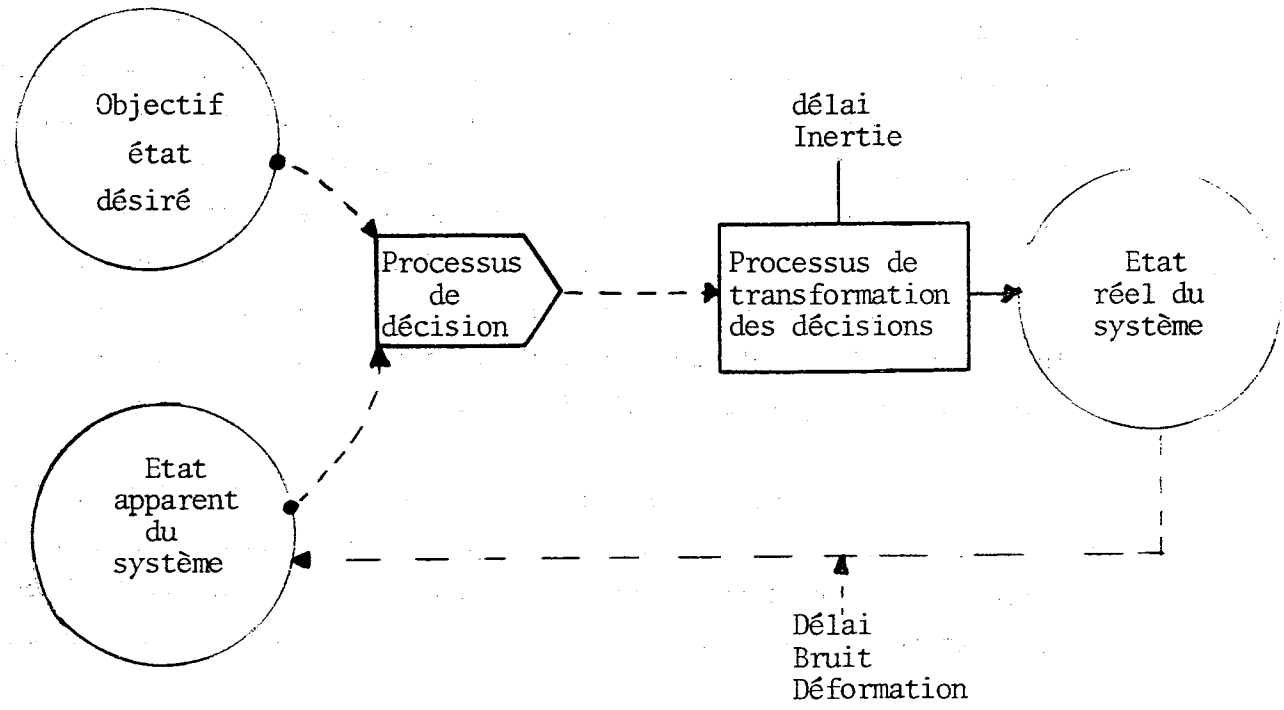


FIGURE 6.1. : Processus de décision

Les décisions ne se transforment pas immédiatement en actions car elles subissent l'inertie des structures mises en place. En effet, il existe toujours un délai entre la prise de décision et la réalisation en actions qu'elle implique, car ces actions mettent en jeu des facteurs multiples (personnel, argent, moyen de production, etc...) que l'on ne peut pas modifier n'importe quand et n'importe comment.

Une hypothèse fondamentale de cette théorie est le principe de continuité. Les informations circulent dans les canaux de manière continue, parce que suscitées par des modifications à variations continues. Ceci peut choquer, car une première analyse fait croire que les décisions sont presque toujours prises de façon discontinue. En fait, il est possible de justifier l'hypothèse fondamentale de FORRESTER par les remarques suivantes

- une décision ne modifie pas brusquement l'évolution du système,
- lorsqu'une décision est prise, elle est le fruit de toutes les influences subies auparavant par celui qui la prend et elle est modifiée ensuite par toutes les autres influences qui peuvent apparaître et contredire cette décision. Par exemple, la décision typiquement discontinue (annuelle) d'établissement d'un budget sera continuellement modifiée par les variations de l'environnement économique de l'entreprise. De même la signature d'un contrat n'est qu'apparemment discontinue; elle est précédée d'une période de négociations et suivie de toute une série d'aménagements.

Ceci nous montre les intentions de J.W. FORRESTER : créer un modèle qui, après fonctionnement, donnerait des résultats analogues à ceux constatés dans la réalité; il montrerait que la symbolisation de la politique de gestion et des canaux d'informations qu'elle utilise est adéquate.

La structure de base du modèle peut être comparée à un ensemble de réservoirs reliés entre eux par des canalisations possédant des robinets. Cette vue est peut être un peu simpliste, mais elle correspond très bien à la réalité.

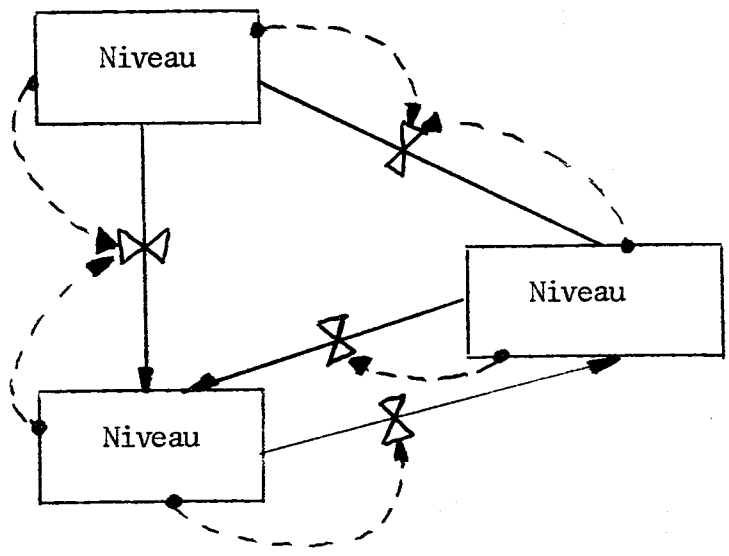


FIGURE 6.2. : Structure de base du modèle

La figure 6.2 contient tous les éléments que nous allons exposer séparément:

- les niveaux ou réservoirs,
- les flux qui transportent le contenu d'un niveau dans un autre
- les fonctions de décisions correspondent aux robinets et elles ont pour but de contrôler le débit du flux,
- le circuit des informations qui agissent sur les fonctions de décisions.

LES NIVEAUX : Les niveaux sont les points d'accumulation du système, du point de vue mathématique ils sont équivalents à une intégration. Cela peut être, un stock, la caisse, le nombre d'employés. Les niveaux existent dans les six réseaux que nous exposerons plus tard : information, matière, ordre argent, personnel, capital et équipement.

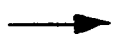
Il est très important de remarquer qu'un niveau peut être mesuré par la même grandeur qu'un flux, ce qui crée souvent des confusions. Pour différencier un niveau d'un flux, il suffit de remarquer que si le flux cesse, le niveau continue à exister.

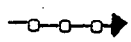
Dans un réseau d'information, on trouve souvent des niveaux qui peuvent être confondus avec des flux. Par exemple, la moyenne des ventes qui se mesurent en articles par mois ou par an parce que la moyenne des ventes est obtenue en intégrant sur une période les flux "vente". On peut arrêter les ventes sans pour cela détruire le concept et la valeur numérique de la moyenne des ventes de l'année écoulée.

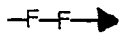
LES FLUX : L'idée de flux se dégage d'après ce que nous venons de dire du niveau. Le débit d'un flux caractérise l'activité du système tandis que le niveau en mesure le résultat.

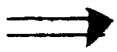
LES FONCTIONS DE DECISIONS : Les fonctions de décisions appelées parfois équations de débit, déterminent à partir de la valeur des niveaux la quantité circulant entre deux niveaux.

Pour représenter l'activité industrielle d'une firme, il est nécessaire d'utiliser plusieurs réseaux. La distinction en six réseaux est arbitraire mais permet une classification commode.

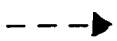
 - réseaux de matières : Ils sont constitués par les stocks, les produits finis, les matières premières.

 - réseaux d'ordres : Les ordres sont le résultat de décisions, cela peut être des demandes d'achats, des demandes d'embauche.

 - réseaux d'argent : Le mot argent est utilisé ici dans le sens paiement. Le réseau d'argent correspond aux différentes sommes d'argent qui peuvent circuler dans le système. Les acomptes recevables représentent une information et ne font pas partie du réseau. Le prix est aussi une information.

 - réseau du personnel : Il comprend tous les employés de la société, mais le réseau est constitué par la circulation de ce personnel.

 - réseau capital et investissement : ce réseau comprend les moyens nécessaires à la production : espace, machines, outillage, etc... Il décrit le cycle du matériel au cours de son existence.

 - réseau d'informations : c'est certainement le réseau le plus important car il permet d'interconnecter tous les autres réseaux. Il transmet les informations prises au stade des niveaux vers les fonctions de décisions. Le réseau d'information comporte lui-même ses propres niveaux et flux. Par exemple le niveau moyen des ventes.

Les hypothèses fondamentales faites par FORRESTER sont que premièrement, le système évolue de façon continue et que deuxièmement les flux et niveaux représentent des unités physiques mesurables. Pour cela, il est nécessaire que par exemple le flux matière, soit défini comme l'aggrégation de plusieurs matières élémentaires.

Si cette aggrégation de variables est acceptable pour les buts poursuivis par FORRESTER, à savoir, la simulation d'un modèle d'entreprise, elle l'est difficilement pour l'analyse des systèmes d'informations.

Nous proposons donc d'utiliser les principes de FORRESTER pour représenter l'entreprise et d'apporter de nombreuses modifications pour décrire le système d'informations et les traitements effectués. Pour la description des traitements nous utiliserons le langage L.T.C.D.

6.1.3. Un modèle : pourquoi faire ?

Avant d'indiquer les raisons qui nous ont conduit à construire un modèle de système d'informations,nous donnerons quelques indications générales sur l'utilité d'un modèle.

La construction d'un modèle permet d'une part, de représenter un phénomène en le séparant de son milieu naturel et d'autre part de simuler la réalité en incluant tous les éléments qui semblent essentiels à l'explication du mécanisme du phénomène.

L'utilisation d'un modèle présente de nombreux avantages. En particulier, il permet des essais contrôlés, à moindre coût, à l'inverse des systèmes réels où il est impossible de changer la valeur d'un paramètre, sans changer les autres et par conséquent de voir les répercussions de ce changement. D'autre part, les confrontations successives entre les essais et l'image que l'on se fait de la réalité apportent des améliorations et des explications sur le mécanisme que l'on analyse.

Dans l'utilisation d'un modèle, il est possible de donner aux paramètres une valeur correspondant à une éventualité future : il permet de mettre en oeuvre des prévisions. Ces prévisions ne sont pas des prédictions, elles sont établies sur les bases du modèle, par conséquent à la limite elles résultent de cette boutade "si rien ne change, tout se passera comme avant". Bien que ceci soit vrai, il est déjà fort intéressant de pouvoir situer des prévisions si aucun changement ne vient bouleverser les bases sur lesquelles a été établi le modèle.

La construction d'un modèle du système d'informations d'une entreprise peut permettre de répondre à deux besoins distincts d'égale importance :

- examiner si le réseau d'informations, qui comprend les données et les traitements, satisfait les besoins de décision des responsables de l'entreprise. Cette possibilité suppose qu'une modélisation de certaines décisions a été faite afin de mesurer la qualité du réseau d'informations par rapport à ces décisions. On peut aussi envisager de modifier le réseau d'informations sans changer les processus de décision ce qui permettra d'étudier différentes configurations du réseau d'informations.

- la construction d'un modèle d'un système d'informations comme on vient de le voir ci-dessus, implique d'avoir une représentation globale des processus d'organisation et de gestion de l'entreprise, ce qui permettra d'envisager une simulation.

Par processus de gestion nous entendons :

- la gestion des opérations,
- la gestion budgétaire,
- la gestion du plan de développement.

La gestion des opérations correspond aux problèmes à court terme, c'est-à-dire à des problèmes d'adaptation aux budgets définis dans le cadre de la gestion budgétaire. Parmi les problèmes de la gestion des opérations on peut citer :

- gestion des commandes, gestion de la production (ordonnancement, approvisionnement),
- gestion financière (compte client, compte fournisseur, trésorerie),
- gestion du personnel,
- gestion commerciale (analyse des ventes, prévisions et quotas des ventes).

La gestion budgétaire a pour but de déterminer pour une période en général annuelle l'activité de l'entreprise compte tenu des objectifs. C'est à ce stade qu'est remis en cause les standards qui s'expriment en unité physique et en valeurs. La gestion du plan de développement a pour but, à partir d'une situation initiale, de projeter l'entreprise sur les années à venir en fonction des objectifs à long terme.

Le modèle que nous allons présenter utilisera d'une part la structure de base proposée par J.W. FORRESTER et d'autre part le langage L.T.C.D. Les propositions que nous faisons nous ont semblé intéressantes à donner bien qu'elles devront être confirmées par des expérimentations car elles sont des applications de la notion de collection de données et elles apportent une amélioration dans la description du réseau d'informations.

6.2. Les éléments du modèle

Les symboles que nous utiliserons dans ce modèle seront ceux proposés par J.W. FORRESTER mais leur signification sera précisée dans chaque cas.

Notre hypothèse fondamentale est qu'à un niveau ou un flux on peut toujours associer une collection de données qui représentera :

- pour un niveau, l'ensemble des valeurs concernant les unités physiques contenues dans le niveau.
- pour un flux, les valeurs circulant dans le flux.

6.2.1. Les niveaux

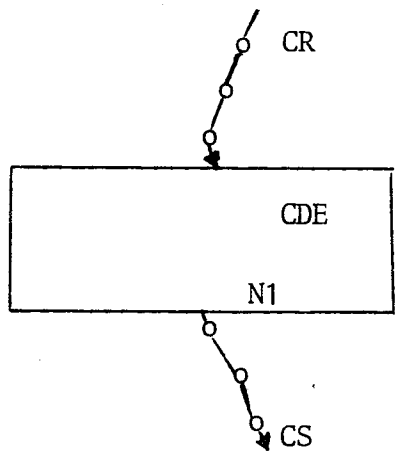
Chaque niveau, représenté symboliquement par un rectangle, est un point d'accumulation de l'entreprise. Cette notion de niveau a déjà été définie au paragraphe 6.1.2.

Chaque niveau représentera une collection de données qui évoluera avec le temps, ce qui signifie que les entités composant le niveau à l'instant t pourront être différentes des entités composant le niveau à l'instant t+1.

Un niveau (figure 6.3.) sera caractérisé par trois paramètres :

- le nom de la collection de données associé au niveau,
- la liste des constituants de la collection de données,
- la référence à un numéro de procédure qui définit l'équation de niveau écrite en langage L.T.C.D. (cette équation sera définie ultérieurement).

Exemple 6.1.



collection CDE
début
NO-COMMANDE
NO-CLIENT
DATE-COMMANDE
NO-PRODUIT
QUANTITE
fin

FIGURE 6.3. : Symbolisation d'un niveau

6.2.2. Les flux

Les flux sont les entrées et les sorties des niveaux. Le symbole utilisé pour les représenter est celui du réseau auquel ils appartiennent. La symbolisation des flux a été donnée au paragraphe 6.1.2.

Chaque flux représente une collection de données. La figure 6.3 a deux flux CR et CS représentant respectivement les commandes reçues et les commandes satisfaites.

Dans l'exemple 6.1 l'espace des constituants de la collection CR est identique à celui de la collection CDE tandis que l'espace de la collection CS est :

collection CS
début
NO-LIVRAISON
NO-COMMANDE
NO-CLIENT
DATE-COMMANDE
NO-PRODUIT
QUANTITE-LIVREE
fin

Il est important de souligner que chaque flux est contrôlé par une vanne qui a pour but de mesurer la quantité qui circule dans le flux. Nous verrons ultérieurement comment est définie cette quantité.

6.2.3. Les collections de données

Nous avons vu que les niveaux et les flux définissent des collections de données, mais il peut exister d'autres collections de données Ces dernières décrivent, soit des éléments extérieurs à l'entreprise, tels que clients et les fournisseurs, soit des faits propres à l'entreprise tels que la nomenclature des produits, les gammes de fabrication, qui sont des données techniques n'ayant aucune existence physique. De telles collections de données seront symbolisées par un rond portant le nom de la collection de données.

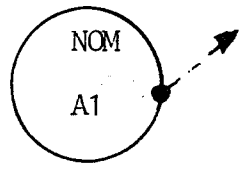


FIGURE 6.4. : Collection de données

Cette symbolisation exprime le fait que le modèle considérera la collection de données comme invariable, c'est-à-dire dans le cas de la collection "nomenclature" le modèle ne pourra intégrer les modifications de nomenclature.

Une collection de données représentant les clients pourra être considérée soit sous la forme d'un niveau, c'est-à-dire que l'on étudiera les entrées et les sorties de clients, soit sous une forme invariante au cours du temps. Ce choix dépend des buts poursuivis dans la construction du modèle.

6.2.4. Les fonctions de traitement

Les fonctions de traitement ont pour but de définir de nouvelles collections de données à partir d'autres collections de données. Dans les fonctions de traitement, on rencontre deux catégories :

- fonction de traitement définissant la valeur d'un flux en agissant sur la vanne de contrôle de ce flux, ceci correspond au schéma de la figure 6.5.

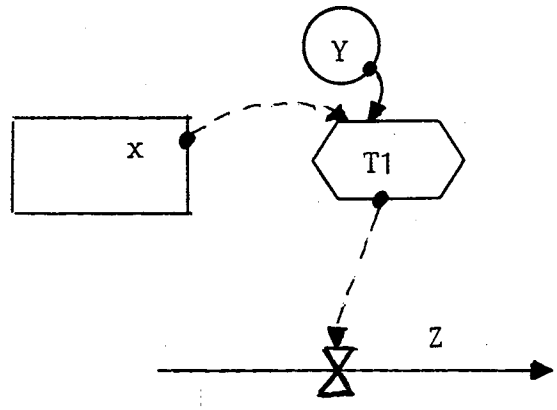


FIGURE 6.5.

Sur ce schéma, le flux Z (ou la collection de données Z) est définie par le traitement T1 qui utilise le niveau X (ou la collection X) et la collection de données Y. Le traitement T1 sera décrit à l'aide du langage L.T.C.D.

Les flèches en pointillé représentent le réseau d'informations.

- fonction de traitement définissant seulement une nouvelle collection de données (figure 6.6.)

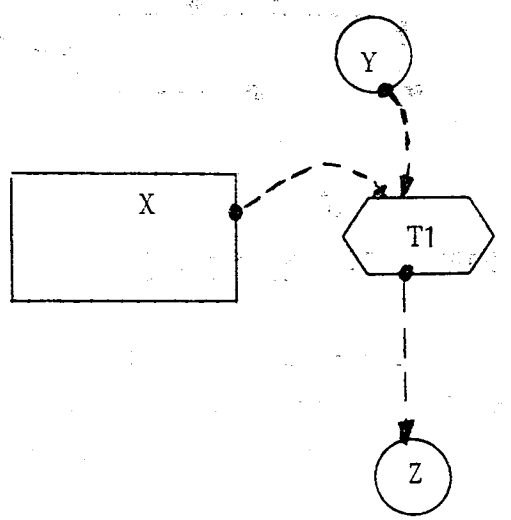


FIGURE 6.6.

Dans chaque cas, on fera figurer dans le symbole représentant la fonction de traitement le numéro de la procédure correspondante.

De plus une fonction de traitement peut contrôler plusieurs flux ou définir plusieurs collections de données.

6.2.5. Sous-système

Il arrive fréquemment que dans la construction du modèle on rencontre des processus de transformation. Par exemple, la matière première se transforme en pièces détachées, puis les pièces détachées en produits finis. La représentation de ces processus est souvent complexe et parfois n'est pas indispensable dans le fonctionnement du modèle. Dans ce cas, on adoptera une symbolisation pour représenter le sous-système (figure 6.7.).

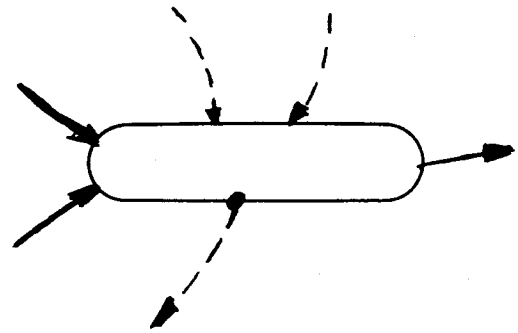


FIGURE 6.7. : Sous-système

Il est important de faire figurer les flux entrants et sortants du sous-système et surtout les informations qu'il reçoit pour l'exécution de sa tâche et celles qu'il émet pour rendre compte de son activité.

6.2.6. Sources et puits

Les flux ont des points d'entrée et de sortie qui sont parfois en-dehors des limites du modèle. Par exemple un flux de commandes satisfaites devient sans intérêt pour le modèle, il tombe dans un puits; à l'inverse on considère le fournisseur approvisionnant l'entreprise en matière première comme une source, si dans le modèle on ne s'intéresse pas aux caractéristiques du fournisseur.

Dans de telles circonstances le flux a pour origine une source et se perd dans un puits, comme le montre la figure 6.8.

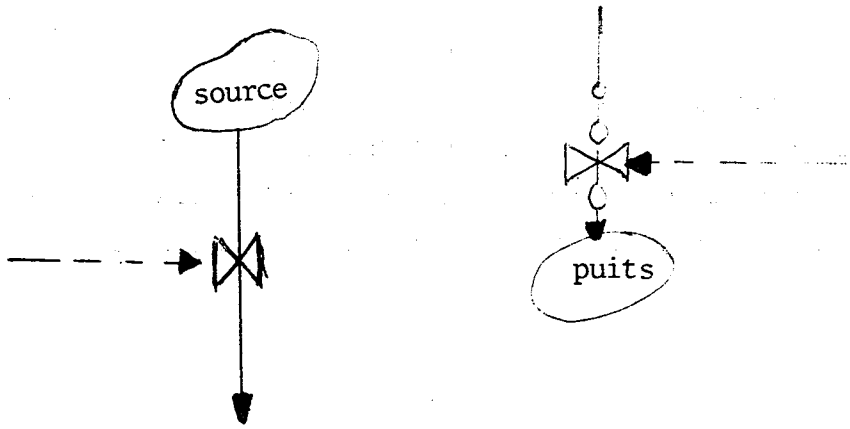


FIGURE 6.8. : Source et puits

6.2.7. Les équations du modèle

Le modèle de FORRESTER est régi par deux grandes catégories d'équations :

- les équations de niveaux qui expriment que si N_t désigne la collection à l'instant t alors on a :

$$N_t = N_{t-1} + E_{t,t-1} - S_{t,t-1}$$

Cette écriture est symbolique, elle signifie en fait que la collection N_{t-1} est mise à jour à partir des flux d'entrées $E_{t,t-1}$ durant la période (t,t-1) et des flux de sorties $S_{t,t-1}$ durant la même période.

Cette mise-à-jour sera exprimée de façon correcte à l'aide du langage L.T.C.D. d'où l'intérêt de référencer le niveau, par son équation de niveau.

- les équations de flux sont les équations qui définissent la valeur d'un flux. De la même façon que les équations de niveau, elles s'exprimeront à l'aide du langage.

Dans toutes équations apparaît la notion de temps, ce qui est normal puisque nous sommes intéressés dans l'évolution du système d'informations au cours du temps.

L'intervalle de temps entre deux évaluations du système doit être assez court pour ne pas affecter les calculs, et aussi long que possible pour éviter des calculs inutiles. La détermination de ce paramètre résulte à la fois d'expériences et de considérations sur la nature des phénomènes que l'on désire prendre en compte dans le modèle.

6.3. Application

L'exemple que nous allons présenter correspond à une partie d'un système d'informations d'une entreprise. La partie choisie est la réception des commandes, la livraison et la fabrication de ces produits.

Notre entreprise fabrique des produits dont la composition en pièces élémentaires est donnée par la collection de données nomenclature : NOM. Le délai de montage de ces produits est court et inférieur à 1 semaine, ce montage est réalisé dans un atelier de montage qui sera considéré comme un sous-système : MONTAGE. La fabrication de ces produits se fait à partir d'un stock de pièces détachées, qui dans le schéma de la figure 6.9 sera le niveau SPD. Une fois les produits fabriqués ils sont mis dans un stock de produits finis, niveau SPF.

La gestion de la production de cette entreprise est caractérisée par le fait que l'entreprise recevant une commande, elle essaye de la satisfaire dans un délai de l'ordre de la semaine. Ceci est possible car deux cas peuvent se présenter :

- soit le produit est en stock et auquel cas la commande peut être satisfaite immédiatement,
- soit le produit n'est pas en stock et auquel cas on lance un ordre de produit à fabriquer, ce qui est désigné par le flux PAF. Ce flux est défini par une fonction de traitement qui tiendra compte du niveau du stock, du niveau des commandes mais aussi de caractéristique de nature technique, telle que la quantité économique de fabrication PRS.

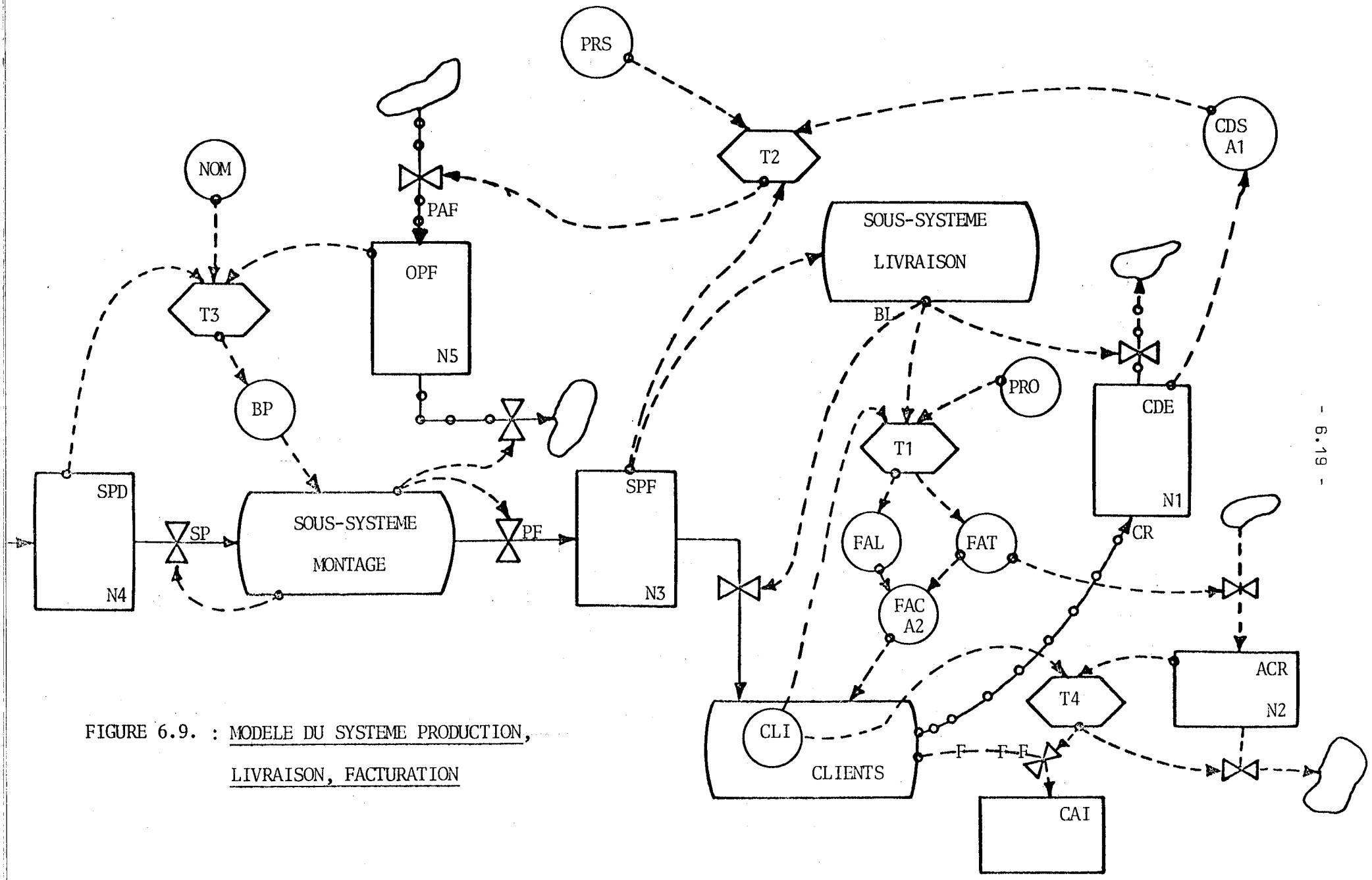


FIGURE 6.9. : MODELE DU SYSTEME PRODUCTION,
LIVRAISON, FACTURATION

Le flux des produits à fabriquer donne naissance à un niveau des ordres de produit à fabriquer OPF. Ce niveau sera mis à jour en fonction de la production du sous-système montage, ce flux de production PF alimente le stock de produit fini.

A partir du niveau des ordres de produits à fabriquer on peut déterminer en fonction du niveau stock pièces détachées SPD, les besoins de pièces BP nécessaires au montage, comme le montre le schéma de la figure 6.9., nous indiquons que l'information est transmise au sous-système montage.

Sur la figure,nous n'avons pas fait figurer la façon dont était approvisionné le stock de pièces détachées, ceci, puisque nous nous sommes limités à une partie du système. Dans une étude plus générale cette analyse devrait être faite.

Parallèlement au processus de production, nous avons indiqué le processus de facturation. Le fait générateur de la facturation est la livraison d'un produit, ce qui est exprimé par le bordereau de livraison BL. La procédure T1 élabore les collections FAL et FAT qui a leur tour définissent la facture FAC. La collection FAT représente le montant de la facture tandis que la collection FAL le détail de la facture. A partir de la collection FAT il est possible de déterminer le montant des accomptes à recevoir (niveau ACR). Puis la procédure T4 exprime la rentrée du flux monétaire dans la caisse CAI.

Ce schéma est incomplet pour la caisse puisqu'il ne fait apparaître que des entrées d'argent et aucune sortie.

Le tableau de la figure 6.10 donne la liste des symboles utilisés ainsi que leur signification.

A chaque niveau et chaque procédure de traitement on associe un ensemble d'équations qui exprime l'évolution du système. Nous illustrerons ces équations en prenant seulement une partie des équations qui régissent tout le système. Nous avons retenu :

- la procédure T1 qui est la plus complexe (figure 6.11),
- l'équation de niveau N1 (figure 6.12),
- une équation auxiliaire A1 (figure 6.13).

SPD : Stock de pièces détachées
SP : sortie de pièces
BP : besoins nets de pièces
NOM : Nomenclature
OPF : niveau des ordres de produits à fabriquer
PAF : produit à fabriquer
PRS : série économique de fabrication par produit
CDS : commande à fabriquer une semaine donnée
CDE : commande
CR : commande reçue
PRO : produit
SPF : stock de produits finis
BL : bordereau de livraison
PF : produits fabriqués
FAC : facture
FAL : facture ligne
FAT : facture totale
CLI : clients
ACR : acomptes à recevoir
EA : entrée argent
CAI : caisse

T1 : procédure de fabrication
T2 : calcul des ordres de fabrication
T3 : calcul des besoins nets de pièces détachées
T4 : calcul de l'entrée d'argent.

FIGURE 6.10.: Tableau des symboles utilisés dans la figure 6.8

```
collection BL
        début
              NBL (numéro de bordereau)
              NCD (numéro de commande)
              NCL (numéro de client)
              DCD (date commande)
              DBL (date bordereau de livraison)
              NPR (numéro de produit)
              QL  (quantité livrée)
        fin

collection CLI
        début
              NCL (numéro de client)
              NOM (nom client)
              ACL (adresse du client)
        fin

collection PRO
        début
              NPR (numéro de produit)
              DPR description du produit)
              PPR (prix unitaire du produit)
        fin
pour S dans [NBL] BL faire
        début
              PT := 0
              pour T dans [NBL.S NPR QL] BL faire
                    début
                          U := NPR.T * PRO
                          PAR := PPR × QL.T
                          [NBL.S, U, ⟨PLA:PA⟩]  FAL := 1
                          PT := PT + PA
                    fin
              [( [NBL.S CNB] BL * CLI)   [NBL.S NCD DCD DBL] BL  NBL.S
                                         ⟨PTF : PT⟩] FAT := 1
        fin
FAC := FAL * FAT
```

FIGURE 6. 11. : Procédure T1

```
collection BL (voir description page précédente)

collection CDE
        début
            NCD (numéro de CDE)
            NCL (numéro de client)
            DCD (date de commande)
            NPR (numéro de produit)
            QCD (quantité commandée)
            DLC (date livraison commande)
        fin

collection CR idem CDE

T := BL * CDE
pour S dans T faire
    début
        Si QL . S ≤ QCD . S alors
            début
                QCD . S := QCD .S - QL . S
                [[NCD NCL DCD NPR QCD DLC] S] U := 1
            fin
        sinon [[NCD NCL DCD NPR QCD DLC] S] V := 1
    fin
BL := BL UNION (U DIF V) UNION CR
```

FIGURE 6.12. : N1 :Equations de mise à jour du niveau CDE

```
CDS := [<DLC : n>  NCD  NCL  DCD  NPR  QCD]  CDE
   n désigne le numéro de la semaine en cours
```

FIGURE 6.13. : A1 : Equation auxiliaire définissant la collection CDS

6.4. CONCLUSIONS

Ce chapitre avait pour but de montrer comment il était possible de construire un modèle de système d'informations tout en respectant le schéma d'organisation de l'entreprise.

A partir de l'état dans lequel se trouve notre travail, on peut explorer différentes voies de recherche. La première serait de construire un outil permettant d'effectuer une simulation du modèle sur une période de temps. Une deuxième aurait rapport avec les processus de décision. Nous avons fait l'hypothèse que les décisions prises en compte dans le modèle était de nature opérationnelle, c'est-à-dire qu'il existait un algorithme de décision. Je pense que l'on pourrait dépasser ce stade en étudiant des processus décisionnels faisant appel à des algorithmes heuristiques ou d'intelligence artificielle. De nombreuses décisions dans l'entreprise relèvent de cette approche.

D E U X I E M E P A R T I E

La première partie de notre travail avait pour but de montrer qu'un système d'informations pouvait être représenté par un ensemble de collections de données définies chacune sur un ensemble de constituants. La caractéristique fondamentale d'une telle représentation est qu'il existe de nombreux recouvrements entre les constituants des collections de données, ce qui permet d'effectuer à l'aide d'opérateurs particuliers des calculs permettant d'obtenir de nouvelles collections de données.

Durant cette deuxième partie, nous allons nous préocupper essentiellement d'examiner la possibilité de construire un ensemble de collections de données répondant à certains critères. Pour cela nous serons amenés à étudier les propriétés et l'utilisation des notions de relations fonctionnelles et de décomposition.

CHAPITRE 7

LES RELATIONS FONCTIONNELLES

Le concept de relation fonctionnelle entre constituants est une notion fondamentale qui peut être utilisée dans la construction d'un système d'informations. Ces relations fonctionnelles sont significatives à la fois pour l'utilisateur dans la compréhension des propriétés intrinsèques des données et,pour le concepteur dans l'implémentation d'un système. Ce chapitre a pour but de dégager les propriétés fondamentales des relations fonctionnelles qui seront utilisées dans les prochains chapitres.

7. LES RELATIONS FONCTIONNELLES

7.1. Définition

Pour l'étude de ce chapitre nous nous plaçons dans l'hypothèse où l'on considère un système de collection de données $(\mathcal{T}, \mathcal{R})$ où les éléments de $\mathcal{T}$ constituent une suite de collections de données T_i définies sur un espace de constituants $\mathcal{A}$ et l'invariant du système s'exprimera à l'aide de relations fonctionnelles.

La notion de relation fonctionnelle a été définie au paragraphe 2.3.2., elle correspond au fait que si le constituant B est relié fonctionnellement à A (A et B ε $\mathcal{A}$), ce que nous noterons :

$$\ell : A \rightarrow B$$

nous avons la propriété suivante pour tout i :

pour toute entité (<A:a>, <B:b>) ε $[A,B]T_i$ il n'existe pas d'entité (<A:a>, <B,b'>) ε $[A,B]T_i$ avec $b \neq b'$

Nous avons vu que cette définition peut être étendue à des constituants composés E et F avec $E = (E_1, E_2, \ldots, E_n)$ $F = (F_1, F_2, \ldots, F_p)$ nous écrirons :

$$\ell : E \rightarrow F$$

ou bien

$$\ell : E_1, E_2, \ldots, E_n \rightarrow F_1, F_2, \ldots, F_p$$

Le concept de relation fonctionnelle est par nature identique à celui d'ensemble de constituants discriminatoire (2.3.4). En effet, si une collection de données a pour espace de définition (E,F) avec $E \rightarrow F$ alors une valeur donnée du constituant E détermine de façon unique une entité de la collection de donnée.

Remarque : Il est rappelé que la notation $\ell : A \rightarrow B$ représente un ensemble d'applications (voir paragraphe 2.3.2). Le symbole ℓ est une étiquette pour désigner la relation fonctionnelle. Dans certains cas, nous noterons plus simplement $A \rightarrow B$.

7.2. Relation fonctionnelle élémentaire

Une relation fonctionnelle $\ell : E \rightarrow F$ est dite élémentaire si pour tout $E' \subset E$ et $E' \neq E$ il n'existe pas de relation fonctionnelle entre E' et F.

L'ensemble des relations fonctionnelles élémentaires peut être considéré comme une propriété intrinsèque de l'information dans une collection de données, cette propriété existe soit par conception, soit comme le résultat de la façon dont apparaissent les données dans une collection. De toute façon le concepteur n'a aucun contrôle sur cette structure, il ne peut faire que des constatations et à partir de ces constatations déduire de nouvelles relations fonctionnelles comme nous allons l'étudier.

Par exemple, dans la collection de données exprimant un emploi du temps figure 7.1 construit sur les constituants : P (professeur), H (heure), N (salle), Y (classe), T (matière enseignée), nous pouvons avoir les relations fonctionnelles élémentaires suivantes :

ℓ_1 : P → T
ℓ_2 : P, H → Y
ℓ_3 : P, H → N
ℓ_4 : H, N → P
ℓ_5 : H, N → Y
ℓ_6 : H, Y → P
ℓ_7 : H, Y → N

Ces relations fonctionnelles élémentaires seront fournies par quelqu'un de familier avec l'élaboration d'un emploi du temps. Elles expriment sa façon de percevoir les propriétés intrinsèques des données. La première relation précise qu'un professeur n'enseigne qu'une seule matière, la troisième qu'un professeur à une heure donnée ne fait cours que dans une seule salle.

Un autre observateur, a qui on demanderait de fournir une telle liste de relations fonctionnelles, pourrait, comme nous le verrons, exprimer les mêmes propriétés mais avec une liste différente.

P	T	Y	N	H
DURAND	MATH	4	2B	6
DUPONT	FRANCAIS	2	1A	3
DUPONT	FRANCAIS	1	3C	5
NIMBUS	FRANCAIS	2	3C	10
DUPONT	FRANCAIS	2	1A	7
NIMBUS	FRANCAIS	3	2A	3
DURAND	MATH	5	1A	5

FIGURE 7.1. : Ensemble d'entités de la collection emploi du temps

Il faut aussi remarquer que dans la figure 7.1 nous avons une application de P,Y dans N, mais cette application ne revêt pas le caractère de permanence en fonction du temps, car un professeur avec une classe peut faire cours dans des salles différentes. Cette remarque est liée à la propriété (b) de la définition d'une relation fonctionnelle.

Ceci nous montre qu'un ensemble de relation fonctionnelle n'existe que par rapport à un référentiel,et que ce que nous allons étudier est vrai à l'intérieur de ce référenciel. Un problème très intéressant, et que nous étudierons uniquement en vue de le poser, sera : que se passe-t-il si on change de référentiel ?

7.3. Représentation d'un ensemble de relations fonctionnelles élémentaires

Il sera utile pour la suite de notre étude de représenter un ensemble de relations fonctionnelles élémentaires par un multigraphe orienté et valué.

$$G = (\mathcal{A}, \Gamma, \Phi)$$

où :

- $\mathcal{A}$ est l'ensemble des noms de constituants
- Φ l'ensemble des relations fonctionnelles élémentaires

  $\Phi = \{\ell_1, \ell_2, \ldots, \ell_k\}$
- $\Gamma \subset \mathcal{A} \times \mathcal{A} \times \Phi$ est une relation définit par $(A, B, \ell_i) \in \Gamma$ si :

  A et B sont des constituants figurant dans la relation ℓ_i avec A dans la partie droite et B dans la partie gauche.

La figure 7.2 ci-dessous correspond à l'exemple donné au paragraphe 7.2

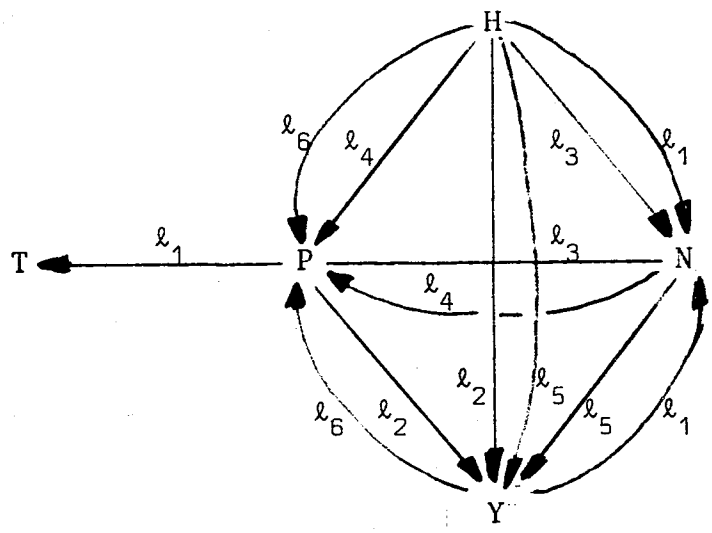


FIGURE 7.2 : Représentation graphique d'un ensemble de relations fonctionnelles

7.4. Propriétés des relations fonctionnelles

7.4.1. Transitivité

Si E → F et F → G alors E → G

$E \to F \Longleftrightarrow \forall\, i$ il existe une application $f_i : [E]\, T_i \to [F]\, T_i$

$F \to G \Longleftrightarrow \forall\, i$ il existe une application $g_i : [F]\, T_i \to [G]\, T_i$

l'application composée $g_i \circ f_i$ est aussi une application surjective puisque, comme composition d'applications surjectives. Elle définit donc une relation fonctionnelle E → G

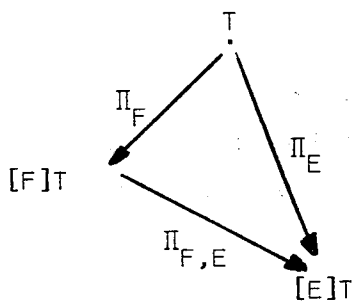
7.4.2. Reflexivité E → E

En effet pour tout i il existe l'application identique

$1_i : [E]\, T_i \to [E]\, T_i$

7.4.3. Projectivité

Si E et F sont des parties de $\mathcal{a}$ avec $E \subset F$ alors F → E.
Cette propriété résulte du fait que la projection vérifie toujours la propriété $\Pi_E = \Pi_{F,E} \circ \Pi_F$



On pourra remarquer que dans ce cas la relation fonctionnelle F → E ne peut être une relation fonctionnelle élémentaire.

7.4.4. Augmentation

Si E → G alors pour tout $F \subset \mathcal{a}$ E,F → G

En effet, si l'entité (<E:e>, <F:f>, <G:g>) ε [E,F,G] T il ne peut exister d'élément (<E:e>, <F:f>, <G:g'>) ε [E,F,G]T avec g ≠ g' puisque E → G.

7.4.5. Additivité

Si E,F,G sont des constituants tels que : E → F et E → G alors E → F,G.

En effet, si l'entité (<E:e>, <F:f>, <G:g>) ε [E,F,G]T il ne peut exister d'entités (<E:e>, <F:f'>, <G:g'> ε [E,F,G]T avec

$$f \neq f' \text{ et } g \neq g'$$

puisqu'il existe deux relations fonctionnelles :

$$E \rightarrow F$$

$$E \rightarrow G$$

D'une façon plus générale si E → F et G → H alors E,G → F,H

E → F E,G → F par augmentation de la partie G

G → H E,G → H par augmentation de la partie E

puis par additivité

$$E,G \rightarrow F,H.$$

Remarque : Si il existe la relation fonctionnelle E,F → E,G il n'est pas possible de simplifier par E les deux membres de l'expression.

7.4.6. Pseudo-transitivité

Si E → F et F,G → H alors E,G → H

E → F E,G → F par augmentation de la partie G

E, G ⊃ G E,G → G selon la propriété de projectivité d'où par addivité :

$$E,G \rightarrow F,G$$

puis par transitivité

$$E,G \rightarrow H$$

Remarque : Dans le cas où G est la partie vide on retrouve la propriété de transitivité ce qui est tout à fait normal.

7.5. Applications des propriétés

Une application de la pseudo-transitivité et de la transitivité est de générer des relations fonctionnelles supplémentaires à partir de l'ensemble de départ. Par exemple, par transitivité on déduit de ℓ_1 et ℓ_6 (voir paragraphe 7.2) :

$$H,Y \rightarrow T$$

tandis que de ℓ_2 et ℓ_7 par pseudo-transitivité, la relation fonctionnelle

$$P,H \rightarrow N$$

qui est l'élément ℓ_3 de l'ensemble de départ. Ceci signifie que notre ensemble de départ comporte certaines redondances.

Le chapitre 9 aura pour but d'étudier les dérivations qui peuvent être faites à partir d'un ensemble de départ et d'éliminer éventuellement les redondances qu'il peut contenir.

Toutefois, avant cela il sera nécessaire d'étudier la structure de l'espace généré par un ensemble de relations fonctionnelles.

Dans la conception de nouveaux principes régissant le fonctionnement d'une base de données, le concept de relations fonctionnelles élémentaires pourra être utilisé :

- en les considérant comme des propriétés préservant l'intégrité de la base de données lors de l'introduction ou de la mise à jour des données. Par exemple, si l'utilisateur a déclaré qu'un employé ne peut travailler que dans un département soit EMPLOYE → DPT alors toute introduction de donnée qui détruirait cette relation devrait provoquer une réaction du système en informant l'utilisateur. A partir de ce moment l'utilisateur peut soit déceler une erreur, soit informer le système que la relation fonctionnelle élémentaire précédemment déclarée n'est plus valable. Ce dernier cas peut avoir des répercussions graves sur l'organisation même de la base de données que nous étudierons au chapitre 13.

- en donnant au système un moyen automatique de rechercher les index associés à une entité.

- en décelant dans des questions formulées par l'utilisateur certaines incompatibilités ou en accélérant le processus de recherche de la réponse (chapitre 10).

Les relations fonctionnelles forment une classe particulière parmi toutes les relations qui peuvent exister entre constituants. Dans la pratique on rencontre les trois cas suivants :

- les constituants dont la valeur est calculée à partir d'autres valeurs de constituants, par exemple le constituant SALAIRE-ANNUEL est dérivé du constituant SALAIRE-MENSUEL.

- les relations non-fonctionnelles, car une valeur d'un constituant correspond à plusieurs valeurs d'un autre constituant et inversement. Dans ce cas, par l'introduction d'un constituant fictif δ on pourrait exprimer une relation non-fonctionnelle par l'écriture :

$$E, \delta \rightarrow F$$
$$F, \delta \rightarrow E$$

le constitutant fictif δ jouant le rôle d'un compteur d'occurence des réalisations d'entités de la relation.

Ces différentes remarques justifient l'étude détaillée au chapitre 8, si l'on désire pouvoir intégrer la notion de relation fonctionnelle élémentaire dans un système de base de données.

CHAPITRE 8

ETUDE D'UN ENSEMBLE DE RELATIONS FONCTIONNELLES

Le but de ce chapitre est d'étudier la structure d'un ensemble de constituants organisé par un ensemble de relations fonctionnelles. Cette étude a permis de dégager une définition opérationnelle de la notion de forme d'une collection de données introduite par E.F. CODD. De nombreux exemples sont donnés pour illustrer ces concepts.

8. ETUDE D'UN ENSEMBLE DE RELATIONS FONCTIONNELLES

8.1. Le préordre induit par les relations fonctionnelles définies sur l'ensemble $\mathcal{A}$ des constituants d'une collection de donnée T.

Nous avons vu au chapitre précédent que la relation $\rightarrow$ définie entre les parties de $\mathcal{A}$ est une relation transitive et réflexive, c'est donc une relation de préordre.

Définissons sur les parties de $\mathcal{A}$ les relations suivantes :

(a) $E \equiv F \Leftrightarrow E \rightarrow F$ et $F \rightarrow E$

(b) $E < F \Leftrightarrow E \rightarrow F$ et $E \subset F$

La relation binaire $\equiv$ définit une relation d'équivalence sur les parties de $\mathcal{A}$. Nous noterons [E] une classe d'équivalence et $\mathcal{A}/\equiv$ l'ensemble des classes d'équivalence.

La relation $<$ est une relation d'ordre partiel à l'intérieur d'une classe d'équivalence, qui est en fait, l'ordre de l'inclusion.

Proposition 8.1.

Si $E \equiv F$ alors $E \equiv E,F$

D'après les hypothèses nous avons :

$$E \rightarrow F \text{ et } F \rightarrow E$$

d'après la propriété d'addivité (paragraphe 7.4.5) nous déduisons :

$$E \equiv E,F$$

et comme $EF \rightarrow E$ ceci implique :

$$E \equiv E,F$$

Proposition 8.2.

Si $E \equiv F$ alors $C/\omega_E = C/\omega_F$ où C/ω_E et C/ω_F désignent respectivement l'ensemble des classes d'équivalence de la partition ω_E et ω_F.

Par hypothèse puisque E et F appartiennent à la même classe d'équivalence : $E \rightarrow F$ et $F \rightarrow E$ et par application de la proposition 2.1. (paragraphe 2.3.3.), nous pouvons écrire :

$$C/\omega_E = C/\omega_{E,F}$$

$$C/\omega_F = C/\omega_{F,E}$$

soit :

$$C/\omega_E = C/\omega_F$$

8.2. Définition du supremum de deux constituants composés

Si E et F sont deux parties de $\mathcal{A}$ nous appellerons supremum de E et de F, noté E V F, le plus petit majorant des majorants communs à E et à F, c'est-à-dire :

$$Maj(E) \triangleq \{G : G \subset \mathcal{A},\ E < G\}$$

$$Maj(F) \triangleq \{G : G \subset \mathcal{A},\ F < G\}$$

si il existe G_o tel que :

$$G_o \in Maj(E) \cap Maj(F)$$

et pour $\forall G \in Maj(E) \cap Maj(F)$ $G > G_o$

alors G_o sera le supremum de E et de F : $G_o = E \vee F$

Proposition 8.3.

Le supremum de E et F est E,F si $E \equiv F$ ou bien encore

$$E \vee F = E,F$$

Selon la proposition 8.1 si $E \equiv F$ alors $E \equiv E,F$ donc on peut écrire $E \rightarrow E,F$ et $F \rightarrow E,F$ et selon la définition de la relation d'ordre partiel $E < E,F$ et $F < E,F$, il en résulte que $E,F \in Maj(E) \cap Maj(F)$

Nous allons montrer que $G_o = E,F$. Pour cela soit H une partie quelconque de $\mathcal{a}$ avec $H \in Maj(E) \cap Maj(F)$, nous allons montrer que $E,F < H$.

Par hypothèse, $E < H$ et $F < H$ ce équivalent à :

$E \subset H$ et $E \rightarrow H$

$F \subset H$ et $F \rightarrow H$

ce qui permet de déduire :

$E, F \subset H$ et $E,F \rightarrow H \Longleftrightarrow E,F < H$

Il résulte des propositions 8.1 et 8.3 que le supremum de toutes les parties d'une classe d'équivalence appartient à cette classe d'équivalence et qu'une classe d'équivalence définie par la relation $\equiv$ sur les parties de $\mathcal{a}$ est sup-demi-treillis.

Le majorant universel d'une classe d'équivalence sera appelé la <u>borne supérieure</u> et il est unique, tandis que les **minorants** universels seront appelés les <u>bornes inférieures</u>. Nous désignerons respectivement par $\overline{E}$ et $\underline{E}$ la borne supérieure et une borne inférieure de la classe d'équivalence $[E]$.

<u>**Proposition 8.4.**</u>

Si $\underline{E}$ est une borne inférieure alors $\underline{E}$ est un index pour la collection de donnée $[\overline{E}]$ T, où $[\overline{E}]$ T désigne la projection de la collection de donnée T sur la borne supérieure $\overline{E}$.

Etant donné que $\overline{E}$ et $\underline{E} \in [E]$ on a $\underline{E} \rightarrow \overline{E}$, ce qui exprime $\underline{E}$ est un ensemble discriminatoire de la collection de données $[\overline{E}]T$, et c'est un index puisque c'est une borne inférieure.

Proposition 8.5.

L'application qui fait correspondre à une partie E de $\mathcal{a}$ la borne supérieure $\overline{E}$ de la classe d'équivalence de E: [E] est une fermeture.

Cette application possède, en effet, les propriétés suivantes

(1) monotone : montrons que $E \subset F \Rightarrow \overline{E} \subset \overline{F}$
puisque $E \subset F \Rightarrow F \rightarrow E$, mais aussi :
$\overline{F} <-> F \rightarrow E \leftrightarrow \overline{E}$
soit : $\overline{F} \rightarrow \overline{E}$, en conséquence puisque $\overline{F} \rightarrow \overline{E}$ on peut dire que $\overline{F}$ et $\overline{F}$, $\overline{E}$ appartiennent à la même classe d'équivalence et puisque $\overline{F}$ est une borne supérieure cela implique que :
$\overline{F} \supset \overline{F}, \overline{E}$ et par conséquent $\overline{F} \supset \overline{E}$

(2) extensive : $\overline{E} \subset E$, ceci est vrai par définition de la borne supérieure.

(3) indempotente : $\overline{\overline{E}} = \overline{E}$ puisque la borne supérieure de toute classe d'équivalence appartient à cette classe d'équivalence.

On peut déduire de cette proposition que l'ensemble des classes d'équivalence représenté par leur borne supérieure et ordonné par l'ordre d'inclusion est un treillis.

8.3. Représentation des classes d'équivalence et du préordre

Pour représenter le préordre, on utilisera comme support le treillis des parties de $\mathcal{a}$. Une arrête de ce treillis reliera les éléments

E et E,A appartenant à la même classe d'équivalence si il existe une relation fonctionnelle de la forme E → E,A où E⊂$\mathcal{A}$ et A ε$\mathcal{A}$. Les éléments appartenant à une même classe d'équivalence sont connectés entre eux par une double ligne.

La figure 8.1 donne **la** représentation du préordre pour l'ensemble des relations fonctionnelles données au paragraphe 7.2. Tandis que la figure 8.2 donne le treillis des classes d'équivalence représentées par leur borne supérieure.

On peut faire les remarques suivantes :

- certaines classes d'équivalence sont réduites à un seul élément, cela veut dire que la borne supérieure est confondue avec la borne inférieure. Ces classes d'équivalence ont une signification particulière qui sera étudiée au chapitre 10.
- une classe d'équivalence peut se déduire **par translation** d'une partie donnée de$\mathcal{A}$. Sur la figure 8.1 c'est le cas de la classe d'équivalence :

$$\{\{P,Y,N\}\ ,\ \{T,P,Y,N\}\}$$

qui peut s'obtenir à partir de la classe d'équivalence

$$\{\{P\}\ ,\ \{P,T\}\}$$

par addition de la partie {Y,N}.

Pour cette raison nous définissons un ordre partiel sur les classes d'équivalence qui sera notée [E] ≺ [F]si :

(a) $\overline{E} \subset \overline{F}$

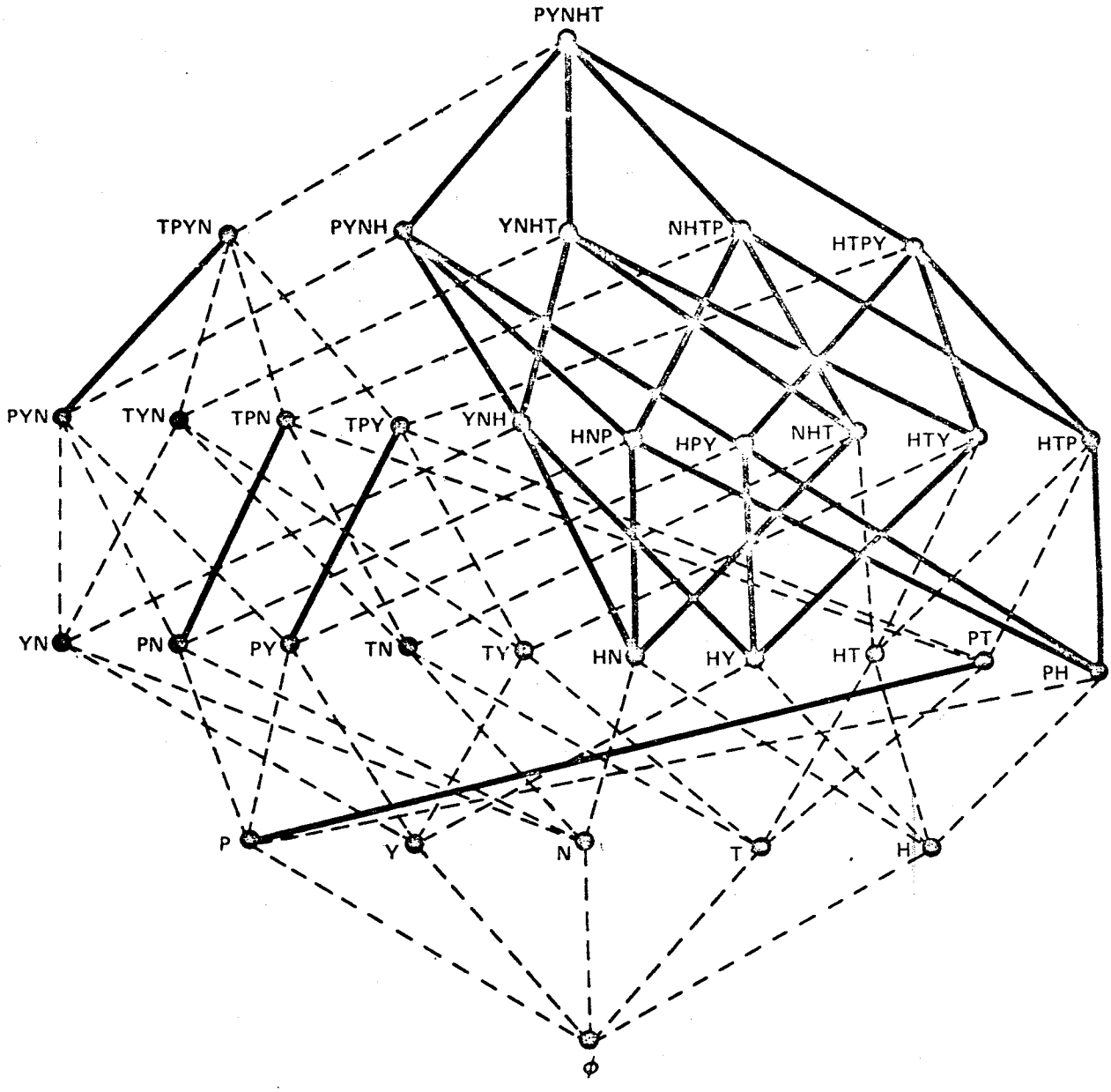
(b) tout élément de [F] est obtenu à partir d'un élément de [E] par addition de l'ensemble $\overline{F} - \overline{E}$.

Les éléments minimaux de cette relation d'ordre partiel sur les classes d'équivalence seront appelés des <u>classes d'équivalence primaires</u>.

La partie vide est une classe d'équivalence primaire sans intérêt, par contre {{P}, {P,T}} est une classe d'équivalence primaire. Ces classes d'équivalence primaires jouent un rôle particulier et on peut faire à leur sujet la remarque ci-dessous.

<u>*Remarque*</u> :

Seules les arêtes partant d'une borne inférieure d'une classe d'équivalence primaire peuvent éventuellement représenter une relation fonctionnelle élémentaire.



P : Professeur
Y : Année
N : N° de salle
H : Heure
T : Matière enseignée

Figure 8.1 : Représentation du préordre de la collection de données représentant un emploi du temps.

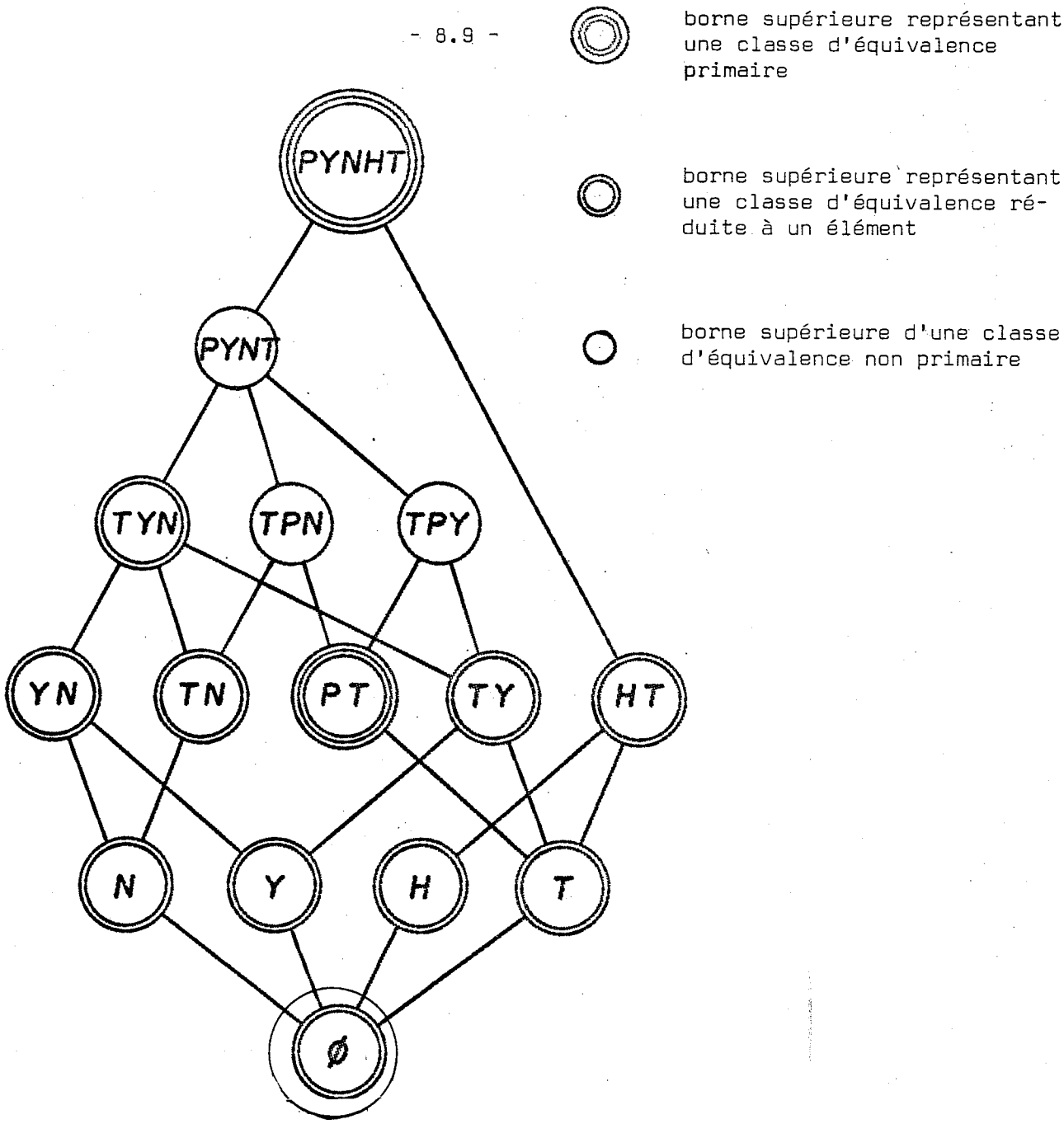


Figure 8.2 : Treillis des bornes supérieures associés à la figure 8.1.

Cette distinction entre classe d'équivalence permet seulement de retenir l'essentiel du préordre.

E.F. CODD dans [26] a été amené à classer les collections de données en trois catégories en se fondant sur la nature des liens existant entre constituants. Bien que cette classification soit parfaitement définie, l'auteur ne donne pas de méthode opérationnelle pour indiquer la classe d'une collection de données. La représentation du préordre permet d'apporter une réponse à ce problème.

8.4. Relation entre la structure du préordre et les formes d'une collection

8.4.1. Définitions

Nous rappelons que nous considérons l'ensemble $\mathcal{a}$ des constituants d'une collection de données T.

Une relation fonctionnelle élémentaire E → F sera dite directe s'il n'existe pas une partie G $\subset \mathcal{a}$, avec G différent d'un index, tel que :

E → G → F où les relations E → G et G → F sont élémentaires

Dans le cas où G serait un index, on se trouverait dans le cas suivant : E ↔ G → F. Par extension, on considèrera que la relation E est une relation fonctionnelle élémentaire directe.

Troisième forme : Une collection de données T sera en troisième forme si pour tout constituant A relié fonctionnellement à un index I de la collection de T, la relation fonctionnelle I A est élémentaire directe.(Il est sous-entendu que l'on n'envisage pas les relations fonctionnelles triviales du type XY → X).

Dans le cas, aussi, où il n'existe aucune relation fonctionnelle élémentaire entre constituant, la collection sera en troisième forme.

Seconde forme : Une collection de données T sera en seconde forme si pour tout constituant A relié fonctionnellement à un index I, la relation fonctionnelle I → A est élémentaire.

Première forme : Toute collection de données qui n'est ni en troisième forme, ni en seconde forme est en première forme.

La classification, ainsi que les définitions données ci-dessus correspondent, avec toutefois une légère modification, à celles proposées par E.F. Codd. La modification correspond au fait que E.F. Codd a introduit la notion de "constituant primaire" et que cette notion n'est pas nécessaire et permet ainsi une simplification dans la présentation.

Lorsque l'on passe de la première forme à la troisième forme les conditions sont de plus en plus restrictives. En effet, si une collection de données satisfait les conditions pour être en troisième forme elle

satisfait aussi les conditions pour être en seconde forme. A la fin du chapitre, nous illustrerons à l'aide d'exemples les avantages et les inconvénients pour une collection de données d'être dans une certaine "forme". Donnons toutefois les principales raisons et les motivations qui conduisent à cette caractérisation :

- une collection de données en troisième forme est moins redondante (du point de vue de l'information) qu'une collection en deuxième forme, voire en première forme.
- les opérations de mise-à-jour sur une collection en troisième forme sont plus simples que sur une collection en deuxième ou première forme.
- l'opération de décomposition d'une collection définie au paragraphe 3.6 tend, partant d'une collection en première forme, à la décomposer en des collections de deuxième et troisième forme, puis en répétant ce processus on peut aboutir uniquement à des collections en troisième forme. Ceci sera montrer dans la proposition 11.5.

La difficulté majeure des définitions données ci-dessous est qu'elle suppose que l'on a pu obtenir de façon opérationnelle les index de la collection T. Dans le cas de nombreuses relations fonctionnelles où les constituants sont fortement imbriqués cette tâche n'est pas facile. Nous donnerons toutefois un moyen simple d'y parvenir, grâce à l'utilisation de l'algèbre de Boole (paragraphe 11.3, proposition 11.2).

Un autre moyen pour caractériser une collection est d'utiliser la représentation graphique du préordre en définissant les deux propriétés P1 et P2.

Nous allons caractériser maintenant le préordre par les deux propriétés P1 et P2.

P1 : La classe d'équivalence dont la borne supérieure est a, est primaire, et aucune borne inférieure d'une classe d'équivalence primaire n'est contenue dans une borne inférieure d'une autre classe d'équivalence.

P2 : La classe d'équivalence, dont la borne supérieure est a, est primaire, et c'est la seule classe d'équivalence primaire.

8.4.2. Equivalence entre propriétés du préordre et formes d'une collection de données

Proposition 8.6.

Troisième forme $\Leftrightarrow$ P2

Seconde forme $\Leftrightarrow$ P1

$P_2 \Rightarrow$ Troisième forme

1er cas : Toutes les classes d'équivalence sont réduites à un élément, alors il n'existe aucune relation fonctionnelle entre les constituants, ce qui fait que l'espace de définition a de la collection T est lui-même un index et alors la collection satisfait à la définition de 3ème forme.

2ème cas : Il existe une seule classe d'équivalence primaire dont les bornes inférieures sont I_1, I_2, ..., I_K. Si A est un constituant relié fonctionnellement à un index nous devons montrer que la relation fonctionnelle $I_i \rightarrow A$ est élémentaire et directe.

Raisonnons par l'absurde, si la relation $I_i \rightarrow A$ n'était pas élémentaire il existerait $E \subset I_i$ tel $E \rightarrow A$, du fait qu'il y a une seule classe d'équivalence, E appartient à cette classe d'équivalence et par conséquent on peut déduire que $E \leftrightarrow I_i$, donc E serait une borne inférieure contenue dans la borne inférieure I_i, ce qui n'est pas possible.

Supposons maintenant que $I_i \rightarrow A$ ne soit pas directe, il existe donc G différent d'un index tel que :

$$I_i \rightarrow G \rightarrow A$$

avec $G \rightarrow A$ est une relation fonctionnelle élémentaire.
G appartient à la classe d'équivalence (hypothèse d'unicité) et par conséquent $G \leftrightarrow I_i$, de plus l'arrête $\{\{G\}, \{G,A\}\}$ a comme extrémité G et comme elle représente une relation fonctionnelle élémentaire G est une borne inférieure et par conséquent un index. Il y a donc contradiction et G ne peut exister.

P1 ⇒ Seconde forme.

Soient $I_1, I_2, \ldots, I_k$ les bornes inférieures de la classe d'équivalence primaire dont la forme supérieure est a, et une autre classe d'équivalence primaire dont les bornes inférieures sont $J_1, J_2, \ldots, J_p$, comme le montre la figure 8.3, et telle qu'aucune borne inférieure ne soit contenue dans une autre borne inférieure.

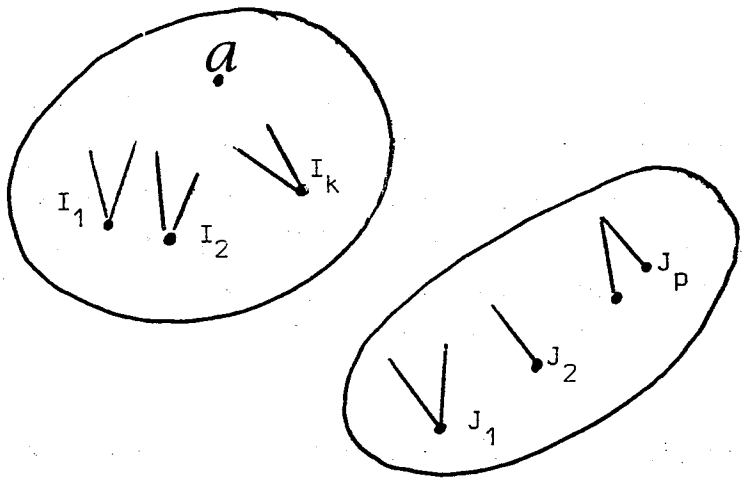


Figure 8.3.

Si A est un constituant relié fonctionnellement à un index par la relation fonctionnelle $I_i \rightarrow A$, nous devons montrer que cette relation fonctionnelle est élémentaire. Si elle ne l'était pas il existerait E tel que $E \rightarrow A$ soit élémentaire et par conséquent cette relation fonctionnelle élémentaire serait associée à une **arête** de la forme $\{\{E\}, \{E,A\}\}$ qui appartiendrait à une classe d'équivalence primaire et de plus E serait nécessairement une borne inférieure, par exemple J_1. En conséquence on obtiendrait $I_i \rightarrow A$ et $J_1 \rightarrow A$ avec $J_1 \subset I_i$ ce qui est contraire à l'hypothèse.

Toute représentation graphique du préordre qui n'a pas ni la propriété P2, ni la propriété P1 est en **première** forme.

La démonstration en sens inverse présente peu d'intérêt puisque ce que nous venons de montrer c'est que l'on peut caractériser une collection de données en examinant la représentation graphique du préordre.

Pour illustrer ces propriétés nous donnerons quelques exemples.

Exemple 8.1

La collection de données relative à un emploi du temps (représentée par la figure 8.1) dont l'ensemble des relations fonctionnelles élémentaires est donné au paragraphe 7.2, est en première forme, car la classe d'équivalence primaire $\{\{P\}, \{P,T\}\}$ *a pour borne inférieure P qui est contenue dans la borne inférieure* $\{P,H\}$ *de la classe d'équivalence primaire qui contient la borne supérieure* $\{P,Y,N,H,T\}$.

Si de l'ensemble des relations fonctionnelles on retire la relation fonctionnelle $\ell_1 : P \rightarrow T$ *, on pourra constater qu'il existe une classe d'équivalence primaire ne contenant pas* a*, la collection reste en 1ère forme. Par contre si on élimine la relation fonctionnelle et le constituant T alors la représentation graphique du préordre ne comprend plus qu'une classe d'équivalence primaire, et la collection de données associée sera en troisième forme.*

Exemple 8.2.

La figure 8.4 donne différents exemples très simples représentant les différents cas qui peuvent être rencontrés.

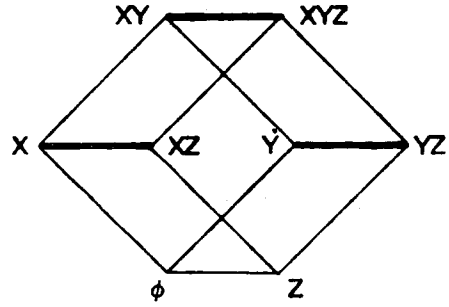
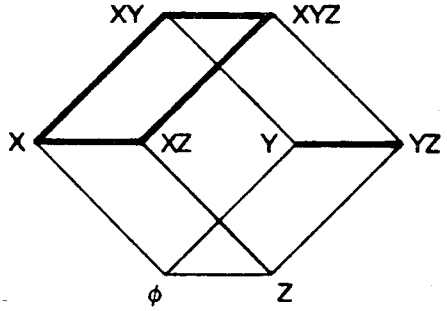
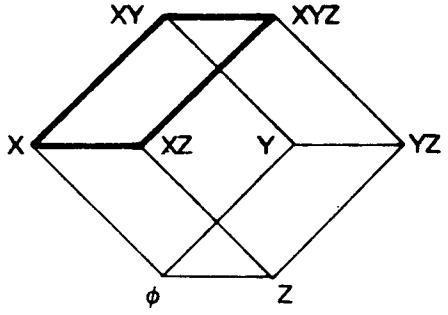
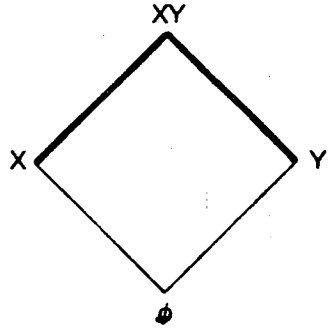
	Relations fonction-nelles	Représentation	Classes d'équivalence	forme
I	X→Z Y→Z		/XY, XYZ/ /X, XZ/ /Y, YZ/ /Z/ /ϕ/	1
II	X→Y Y→Z		/X, XY, XZ, XYZ/ /Y,YZ/ /Z/ /ϕ/	2
III	X→Y X→Z		/X, XY, XZ, XYZ/ /YZ/ /Y/ /Z/ /ϕ/	3
IV	X→Y Y→X		/X, Y, XY/ /ϕ/	3

FIGURE 8.4 : Représentations de 1°,2°,3° forme

8.4.3. Remarques sur la dimension de l'ensemble .

Si l'application de gestion que l'on décrit recouvre plusieur fonctions de l'entreprise, le nombre total de constituants peut être de l'ordre de 500 à 600. Dans de telles conditions il existerait de sérieuses difficultés pour construire pratiquement le préordre même à l'aide de moyens automatiques. Il serait souhaitable de pouvoir réduire la taille du problème sans détruire les caractéristiques de l'application. Ceci peut être fait en partant du multigraphe qui représente un ensemble de relations fonctionnelles. Dans le multigraphes nous dirons qu'un constituant est terminal si tous les arcs arrivant en ce noeud sont "entrants".

Si $E_1,E_2,\ldots E_n \rightarrow F_1,F_2,\ldots,F_p$ est une relation fonctionnelle où $F_1,F_2,\ldots, F_k$ (à une numérotation près) avec $k \leqslant p$ sont des constituants terminaux, on définira un constituant composé : $X = F_1,F_2,\ldots,F_k$ et on utilisera alors la relation fonctionnelle : $E_1,E_2,\ldots,E_n \rightarrow X,F_{k+1},\ldots,F_p$.

Illustrons ceci à l'aide d'un exemple ayant trait à la gestion du personnel :

A1 : Numéro d'employé
A2 : Nom
A3 : Sexe
A4 : Date de Naissance
A5 : Adresse
A6 : Date
A7 : Qualification
A8 : Fonction
A9 : Salaire
A10 : Salaire actuel
A11 : Département auquel l'employé appartient
A12 : Division auquel l'employé appartient
A13 : Numéro Projet sur lequel l'employé travaille
A14 : Nom du responsable du projet.

Si on suppose que nous avons les relations fonctionnelles suivantes :

$A1 \rightarrow A_2,A_3,A_4,A_5,A_{10},A_{11},A_{13}$

$A1,A6 \rightarrow A7,A8,A9$

$A_{11} \rightarrow A_{12}$

$A_{13} \rightarrow A_{14}$

Nous créerons un constituant composé X si aucun élément composant est membre de gauche d'une autre relation fonctionnelle. Ici, nous créerons :

$X = (A_2, A_3, A_4, A_5, A_{10})$

$Y = (A_7, A_8, A_9)$

La liste des relations fonctionnelles sera alors :

$A_1 \rightarrow X, A_{11}, A_{13}$

$A_1, A_6 \rightarrow Y$

$A_{11} \rightarrow A_{12}$

$A_{13} \rightarrow A_{14}$

La plupart du temps ce processus peut être appliqué pour donner une très grande réduction de l'ensemble de départ.

8.5. Opérations de mise-à-jour et formes d'une collection de données

Considérons une collection de données définie sur les constituants :

A : numéro de projet

B : numéro de produit

C : numéro de fournisseur

D : prix du produit

et les relations fonctionnelles :

ℓ_1 : $A, B \rightarrow C$

ℓ_2 : $B \rightarrow D$

qu'un produit d'un certain prix entre dans la composition d'un projet et est approvisionné par un fournisseur.

Les figures 8.5 et 8.6 représentent respectivement un ensemble d'entités pour cette collection et la représentation du préordre des relations fonctionnelles.

A	B	C	D
1	a	α	5
1	b	α	10
1	c	β	7
2	b	γ	10
2	d	α	6
2	a	α	5
2	c	δ	5

Figure 8.5 :

Ensemble d'entités pour la collection T

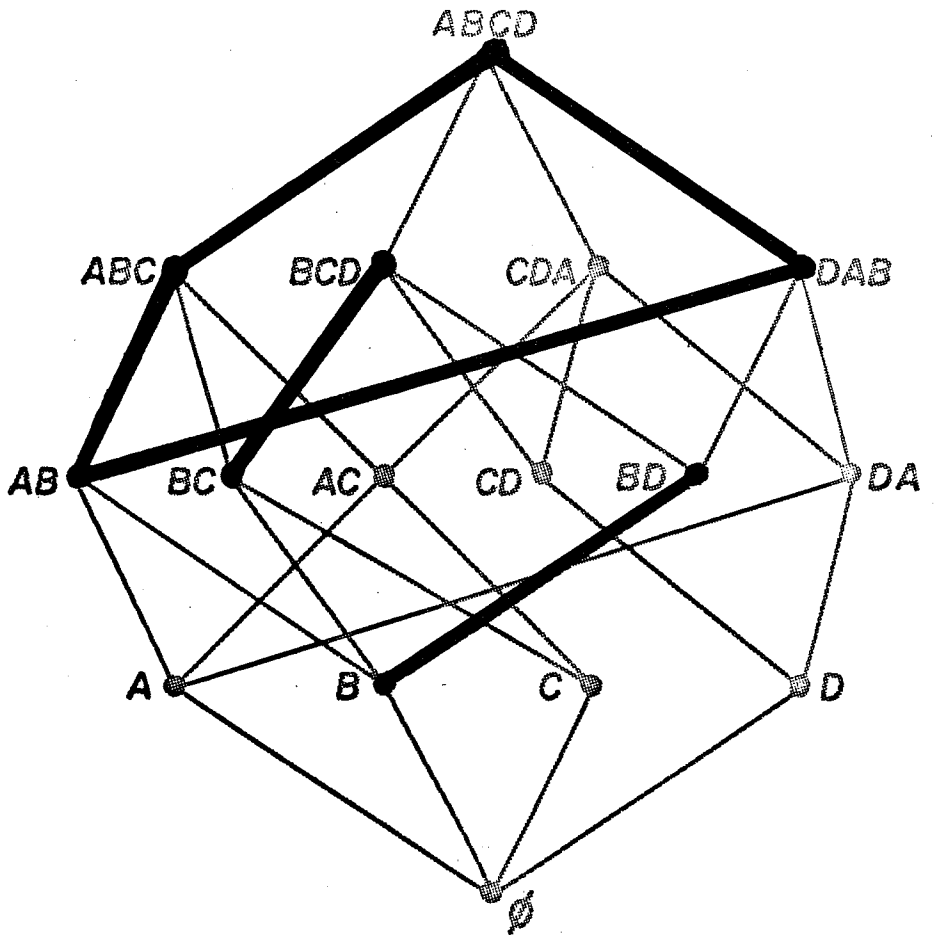


Figure 8.6 : Préordre des relations fonctionnelles de la collection T

A partir de la figure 8.6 il est facile de conclure que la collection de données T est en 1° forme.

Si on regarde l'ensemble des entités de la collection de données T (figure 8.5) nous observons par exemple que la modification du prix d'un produit nécessitera la mise à jour de plusieurs entités. De plus le nombre actuel d'entités dans la collection T peut être modifié par addition ou suppression d'entités. Par exemple, la suppression du projet 1 entraîne la suppression de trois entités et entraîne la disparition seulement du produit c et du fournisseur β, ce qui est génant si l'on avait voulu retenir le prix du produit c et que le fournisseur β approvisionnait un projet.

De même l'insertion d'un produit nouveau et d'un prix ne peut être fait tant que ce produit n'apparaît pas dans un projet.

Pour remédier à ces inconvénients on peut remplacer la collection T par deux de ces projections (figure 8.7) conformément à la proposition 3 (paragraphe 3.7.4) puisqu'il existe la relation fonctionnelle ℓ_2 : B → D.

[A,B,C] T

A	B	C
1	a	α
1	b	α
1	c	β
2	β	γ
2	d	α
2	a	α
2	e	δ

[B,D] T

B	D
a	5
b	10
c	7
d	6
e	5

Figure 8.7 : Projections de la collection T

On pourra remarquer que maintenant les collections [A,B,C]T et [B,D]T sont maintenant en 3° forme et que les phénomènes signalés ci-dessus ont disparu.

Le problème de remplacer une collection par plusieurs de ces projections est ce que nous appelons le problème de la décomposition d'une collection et sera étudié en détail dans le chapitre 11. E.F. CODD a conjecturé que des collections de données en 3° forme seraient une représentation efficace pour une base de donnée.

CHAPITRE 9

FERMETURE ET COUVERTURE MINIMALE

Le présent chapitre a pour but de montrer que les propriétés des relations fonctionnelles peuvent être formulées en terme d'algèbre booléenne. L'avantage de cette formulation est qu'il est possible d'explorer avec efficacité les propriétés d'un modèle sur la structure de l'information dans la mesure où les calculs booléens sont effectués très simplement par programme. Cette propriété est très importante pour le concepteur d'un système d'informations car elle se trouve dans les perspectives d'une conception assistée par ordinateur qui aurait pour but de générer et d'évaluer plusieurs organisations de données.

9. FERMETURE ET COUVERTURE MINIMALE

Dans le chapitre 7 nous avons vu que les propriétés des relations fonctionnelles permettaient de déduire de nouvelles relations fonctionnelles. Nous avons même mis en évidence le fait que certaines relations fonctionnelles sont redondantes. Dans ce chapitre nous allons étudier des procédures automatiques qui permettront de générer de nouvelles relations fonctionnelles et d'éliminer la redondance.

9.1. Définitions

Commençons par définir des expressions standardisées de l'information contenue dans un ensemble de relations fonctionnelles. Nous nous limiterons aux relations fonctionnelles élémentaires puisqu'elles sont des expressions les moins redondantes des relations fonctionnelles.

Soit $\Phi = \{\ell_1, \ell_2, \ldots, \ell_r\}$ un ensemble de relations fonctionnelles élémentaires défini par l'utilisateur. Si par application de la propriété de pseudo-transitivité de la relation "$\rightarrow$" la paire non-ordonnée (ℓ_i, ℓ_j) produit la relation fonctionnelle élémentaire ℓ_m, nous dirons que ℓ_m est dérivé de (ℓ_i, ℓ_j) et nous noterons cette dérivation par l'opération :

$$T[(\ell_i, \ell_j)] = \ell_m$$

Par analogie nous noterons :

$T[\Phi]$ l'ensemble des relations fonctionnelles dérivées de Φ.

Comme ℓ_i est dérivable de (ℓ_i, ℓ_i) nous avons $\Phi \subset T[\Phi]$. La fermeture de Φ notée Φ^+ sera obtenue par application successive de l'opérateur de dérivation, c'est-à-dire : il existe un rang n à partir duquel :

$$T^{n+k}[\Phi] = T^n[\Phi]$$

nous poserons alors :

$$\Phi^+ = T^n[\Phi]$$

$\Phi_0 \subset \Phi$ sera une <u>couverture minimale si</u> :

- $\Phi \subseteq \Phi_0^+$

- $\nexists\ \Phi_0' \subseteq \Phi_0$ tel que

$$\Phi \subseteq \Phi_0'^+$$

<u>Exemple 9.1</u> : *Pour illustrer la notion de fermeture et couverture minimale reprenons l'exemple donné au paragraphe 7.2 où*

$$\Phi = \{\ell_1, \ell_2, \ldots, \ell_7\}$$

Φ^+ *est composé de* $\ell_1, \ell_2, \ldots, \ell_7$ *et de*

ℓ_8 : H,Y → T

ℓ_9 : H,N → T

On peut ainsi trouver deux couvertures minimales :

ℓ_1 : P → T	ℓ_1 : P → T
ℓ_2 : P,H → Y	ℓ_3 : P, H → N
ℓ_4 : H,N → P	ℓ_5 : H, N → Y
ℓ_7 : H,Y → N	ℓ_6 : H,Y → P

Cette notion de couverture minimale nous semble importante pour les raisons suivantes :

- *elle permet de comprendre pourquoi deux utilisateurs différents peuvent décrire la même application en des termes différents,*
- *elle sera utiliséedans le problème de la décomposition d'une collection de donnée (voir chapitre 11).*
- *nous avons dit que l'on pourrait éventuellement utiliser la notion de relation fonctionnelle élémentaire comme une déclaration préservant l'intégrité de la base de donnée lors de la mise à jour. Nous montrerons dans le chapitre 13 sur la restructuration de la base de donnée que les couvertures minimales jouent un rôle fondamental.*

9.2. Représentation d'un ensemble de relations fonctionnelles par une fonction booléenne

Cette possibilité là a été soulignée la première fois par J. BOITTIEAUX et a été généralisée à des structures de treillis par BOUCHER. Nous en donnerons une démonstration différente qui servira de base à l'écriture d'un programme.

Les opérations "ET", "OU", et "complément" d'une fonction booléenne seront notées +, ., ',

Forme relationnelle

Pour représenter un ensemble $\Phi = \{\ell_1, \ell_2, \ldots, \ell_n\}$ de relations fonctionnelles, avec

$$\ell_j : X_j \to Y_j \quad j = 1,2,\ldots,n$$

on construira une fonction booléenne d'un type particulier que nous appellerons une forme relationnelle.

Cette fonction booléenne est une somme de monôme et chaque monôme correspond à une relation fonctionnelle. D'une façon générale à la relation fonctionnelle $X_j \to Y_j$ nous associerons le terme :

$$x_j\, y'_j$$

dans le cas où X_j et Y_j seraient des constituants composés :

$$X_j = X_{j_1}, X_{j_2}, \ldots, X_{j_k}$$

$$Y_j = Y_{j_1}, Y_{j_2}, \ldots, Y_{j_m}$$

on obtiendra :

$$x_j\, y'_j = (x_{j_1} \cdot x_{j_2} \cdot \ldots \cdot x_{j_k})(y_{j_1} \cdot y_{j_2} \cdot \ldots \cdot y_{j_m})'$$

$$= x_{j_1} \cdot x_{j_2} \cdot \ldots \cdot x_{j_k}\, y'_{j_1} + x_{j_1} \cdot x_{j_2} \cdot \ldots \cdot x_{j_k}\, y'_{j_2} + \ldots + x_{j_1} \cdot x_{j_k}\, y'_{j_m}$$

A un ensemble de n relations fonctionnelles on associera la fonction booléenne :

$$f = \sum_{i}^{n} x_j\, y'_j$$

En conséquence la fonction booléenne associée avec un ensemble de relations fonctionnelles peut toujours être exprimée, comme une somme de monômes ne contenant qu'une seule variable primée. D'une façon générale une expression booléenne ayant cette propriété sera appelée aussi une forme relationnelle.

Il est simple de constater que toute forme relationnelle détermine de façon unique un ensemble de relations fonctionnelles. En effet, à chaque monôme on associe une relation fonctionnelle où les termes de gauche correspondent aux variables non primées et la partie droite à la variable primée.

9.3. Equivalence entre relation fonctionnelle et forme relationnelle

Si g est une fonction booléenne équivalente à la forme relationnelle f au sens de l'algèbre de boole, alors le résultat fondamental que nous allons montrer est que g est une forme relationnelle et peut être interprété comme un ensemble de relations fonctionnelles. Ce qui signifie que si il existe une relation fonctionnelle déduite de g, mais absente de f, alors cette relation fonctionnelle dérive de relations fonctionnelles de f. On voit alors immédiatement l'utilisation qui peut être faite de cette propriété pour trouver la fermeture d'un ensemble de relations fonctionnelles.

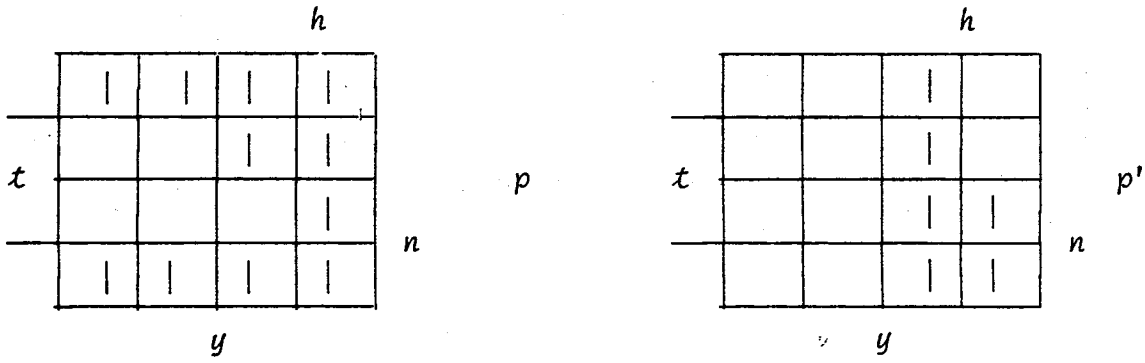
Illustrons cette propriété par un exemple avant de discuter le cas général.

Exemple 9.2 :

La fonction f associée à l'exemple de la collection de données représentant un emploi du temps (paragraphe 7.2) est

$$f = pt' + phy' + phn' + hnp' + hny' + hyp' + hyn'$$

Cette fonction de 5 variables peut être représentée par le diagramme de Karnaugh :



Une fonction équivalente ayant le même diagramme est

$$g = pt' + hyt' + hytn' + pthn' + nhty' + nythp' + nht'$$

Cette fonction représente les relations fonctionnelles.

(a) P → T (d) PTH → N (g) NH → T

(b) HY → T (e) NHT → Y

(c) HYT → N (f) NYTH → P

Chacune des relations fonctionnelles (a) à (g) est une conséquence des relations ℓ_1 à ℓ_7. C'est ainsi que ℓ_1 et (a) sont identiques; (b) est dérivé par transitivité de ℓ_1 et ℓ_6; (c) provient de ℓ_7; (d) est impliqué par ℓ_3; (e) est une conséquence de ℓ_5; (f) est une représentation plus faible de ℓ_4 ou ℓ_6; et (g) est obtenue à partir de ℓ_1 et ℓ_4.

Réciproquement les relations ℓ_1 à ℓ_7 peuvent être déduites de (a) - (f). Nous allons montrer l'équivalence entre les formes booléennes et les relations fonctionnelles. Cette affirmation aura pour corollaire :

(1) La fermeture d'un ensemble de relations fonctionnelles est l'image des monômes premiers de la forme booléenne

(2) Une couverture minimale dans le domaine relationnelle est l'image de la couverture minimale de la forme booléenne

Soit Φ un ensemble de relations fonctionnelles sur un espace $\mathcal{a}$.

$$\Phi = \{U_j \rightarrow V_j : j = 1,2,\ldots,n\}$$

Il peut être supposé sans nuire à la généralité de la démonstration que les U_j sont des constituants composés, tandis que les V_j sont des constituants élémentaires.

Supposons qu'à partir de relations fonctionnelles appartenant à Φ il soit possible d'obtenir une nouvelle relation $X \rightarrow Y$ par l'application de deux propriétés fondamentales : pseudo-transitivité et augmentation.

Soit $\Psi = \Phi \cup \{X \rightarrow Y\}$ alors nous écrirons : $\Phi \simeq \Psi$

où le symbole d'équivalence $\simeq$ doit être compris comme il est possible d'obtenir Ψ à partir de Φ et inversement Φ à partir de Ψ en éliminant les relations redondantes.

Associées à Φ et Ψ correspond les deux formes relationnelles :

$$f = \sum_j u_j v_j' \qquad g = \sum_j u_j v_j' + xy'$$

Nous allons montrer que :

Proposition 9.1.

Deux ensembles de relations fonctionnelles Φ et Ψ sont équivalents si et seulement si les formes relationnelles associées sont équivalentes.

La proposition ci-dessus est clairement extensible par induction à Φ et Ψ ou f et g si ces ensembles diffèrent de plus d'un élément.

(1) $\Phi \simeq \Psi \Rightarrow f \simeq g$

le terme additionnelle $X \rightarrow Y$ est dérivé de Φ par application des règles de pseudo transitivité et augmentation. Nous avons seulement besoin de montrer que la contrepartie booléenne de ces opérations produit des termes correspondants pour prouver (1).

pseudo-transitivité : $\{A \rightarrow B, BD \rightarrow C\} \simeq \{A \rightarrow B, BD \rightarrow C, AD \rightarrow C\}$

la forme relationnelle associée à la partie gauche est :

$$f = ab' + bdc'$$

qui peut s'écrire par utilisation de la loi d'absorption

$$f = (ab' + ab'dc') + (abdc' + bdc')$$
$$= ab' + bdc' + ade'$$

qui est bien la forme associée à la partie droite.

augmentation : $\{A \rightarrow B\} \simeq \{A \rightarrow B, AC \rightarrow B\}$

dans le domaine booléen on peut écrire :

$$ab' \simeq ab' + acb'$$

qui est une conséquence de la loi d'absorption. Ainsi, la propriété d'augmentation pour les relations fonctionnelles est l'équivalent de la loi d'absorption du calcul booléen.

(2) $f \simeq g \Rightarrow \Phi \simeq \Psi$

La démonstration dans ce sens est beaucoup plus difficile. En effet, des formes booléennes équivalentes peuvent être générées en utilisant d'autres opérations que la contrepartie de la pseudo-transitivité et l'augmentation.

Nous allons démontrer que toutes formes canoniques équivalentes peuvent être produites en utilisant uniquement ces deux opérations Il suivra alors que Ψ est dérivé de Φ en utilisant les opérations correspondantes à celles qui permettent d'obtenir g à partir de f.

La démonstration repose sur le lemme suivant.

Lemme : Tous les monômes premiers de f peuvent être obtenus par utilisation de la contrepartie booléenne de la pseudo-transitivité et de l'augmentation.

Un monôme premier est un terme irréductible dérivable de f par des opérations booléennes. Il s'en suit que chaque terme de g est soit un monôme premier soit un terme obtenu à partir d'un monôme premier de f par le moyen de la loi d'absorption. Ainsi, si le lemme est vrai, g est dérivé de f par utilisation seulement des contreparties des opérations relationnelles, ce qui prouvera (2).

Démonstration du lemme

La démonstration du lemme repose sur l'algorithme ✱ de génération des monômes premiers d'une expression booléenne sous forme disjonctive [voir Réf. 27 pp. 156-163]. L'algorithme construit l'ensemble des monômes premiers par itérations successives à partir de deux opérations. Une de ces opérations est la loi d'absorption et l'autre est appelée l'opération ✱.

Alors la preuve du lemme reposera sur la nécessité de montrer que l'opération ✱ sur les formes canoniques est expressible en termes d'absorption et de pseudo-transitivité booléenne.

L'opération ✱ définit un nouveau terme à partir de deux termes donnés. Ce nouveau terme a la propriété fondamentale que si u et v sont deux termes :

$$u + v \simeq u + v + u \ast v$$

où u et v sont des termes définis sur les variables de l'espace a soit $a_1, a_2, \ldots, a_n$. Nous représenterons un terme par un vecteur de dimension n

$$u = [u_1, u_2, \ldots, u_n]$$

où

$$u_j = \begin{cases} a_j \\ a'_j \\ x \end{cases}$$

Le symbole x signifiant que la variable correspondante est absente du terme.

Si u et v sont deux vecteurs, l'opération u * v définit un vecteur w de la façon suivante :

(a) w est indéfini (c'est-à-dire il n'y a pas de terme généré) si il existe deux indices distincts i et j tel que :

$$(u_k, v_k) \in \{(a_k, a'_k), (a'_k, a_k)\}$$

pour à la fois k = i et k = j.

Exemple :

$u = (a_1, a'_2, x, a_4)$

$v = (x, a_2, a_3, a'_4)$

*u * v est indéfini.*

(b) si la propriété (a) n'est pas vraie alors :

$w_j = a_j$ si $(u_j, v_j) \in \{(a_j, a_j), (a_j, x), (x, a_j)\}$

$w_j = a'_j$ si $(u_j, v_j) \in \{(a'_j, a'_j), (a'_j, x), (x, a'_j)\}$

$w_j = x$ si $(u'_j, v_j) \in \{(a_j, a'_j), (a'_j, a_j), (x, x)\}$

Ainsi w est déterminé par neuf combinaisons différentes de valeurs de u et v. Il est facile de remarquer que par un changement de variable et réarrangement de l'ordre des variables, il est toujours possible de ramener les vecteurs u et v de longueur n à un vecteur de longueur 9.

En effet, désignons par u et v deux vecteurs de longueur n.

$u = [u_1, u_2, \ldots, u_n]$

$v = [v_1, v_2, \ldots, v_n]$

où u_j et v_j peuvent prendre comme valeur a_j, a'_j, x pour j = 1, 2, ...,n. Si nous définissons le changement de variables par :

$$b_1 = \{a_i : u_i = a_i \quad v_i = a_i\}$$

$$b_2 = \{a_i : u_i = a_i \quad v_i = x\}$$

$$b_3 = \{a_i : u_i = x \quad v_i = a_i\}$$

$$b_4 = \{a_i : u_i = a'_i \quad v_i = a'_i\}$$

$$b_5 = \{a_i : u_i = a'_i \quad v_i = x\}$$

$$b_6 = \{a_i : u_i = x \quad v_i = a'_i\}$$

$$b_7 = \{a_i : u_i = a_i \quad v_i = a'_i\}$$

$$b_8 = \{a_i : u_i = a'_i \quad v_i = a_i\}$$

$$b_9 = \{a_i : u_i = x \quad v_i = x\}$$

on pourra toujours écrire :

$$u = [b_1, b_2, x, b'_4, b'_5, x, b_7, b'_8, x]$$

$$v = [b_1, x, b_3, b'_4, x, b'_6, b'_7, b_8, x]$$

Dans le cas u $\divideontimes$ v est défini nous obtiendrons :

$$u \divideontimes v = [b_1, b_2, b_3, b'_4, b'_5, b'_6, x, x, x]$$

Par exemple si les vecteurs de u et v sont :

$$u = [a_1, a_2, a'_3, x, a_5, a'_6, a_7, x, a_9, a'_{10}, a_{11}, a'_{12}]$$

$$v = [a'_1, a_2, x, x, a'_5, a'_6, a'_7, a_8, x, x, a'_{11}, a'_{12}]$$

par le changement de variable

$$b_1 = a_2 \quad b_2 = a_9 \quad b_3 = a_8 \quad b_4 = (a_6, a_7, a_{12})$$

$$b_5 = (a_3, a_{10}) \quad b_6 = x \quad b_7 = (a_1, a_5, a_{11}) \quad b_8 = x \quad b_9 = a_4$$

nous obtenons :

$$u \divideontimes v = [b_1, b_2, b_3, b'_4, b'_5, x, x, x, x]$$

$$= [a_2, a'_3, a'_6, a'_7, a_8, a_9, a'_{10}, a'_{12}]$$

Il ne faut pas oublier que u et v sont des images des relations fonctionnelles, à savoir une seule variable primée par terme. Ce qui fait que u ne peut avoir qu'un des éléments de la forme b'_4, b'_5, b'_8. De même v ne peut avoir qu'un des éléments de la forme b'_4, b'_6, b'_7.

Ainsi les formes possibles pour u et v sont :

$u(1) = b_1\ b_2\ b'_4\ b_7$ $\qquad$ $u(2) = b_1\ b_2\ b'_5\ b_7$ $\qquad$ $u(3) = b_1\ b_2\ b_7\ b'_8$

$v(1) = b_1\ b_3\ b'_4\ b_8$ $\qquad$ $v(2) = b_1\ b_3\ b'_6\ b_8$ $\qquad$ $v(3) = b_1\ b_3\ b'_7\ b'_8$

Maintenant si on considère les 9 combinaisons possibles on constate que certaines sont incompatibles par suite du changement de variable et que u et v ne contiennent qu'une seule variable primée.

Le tableau ci-dessous exprime ces 9 cas et donne la forme réduite qui sera considérée :

	Combinaison	forme réduite u	forme réduite v	u * v	
1	u(1) * v(1)	$b_1\ b_2\ b'_4$	$b_1\ b_3\ b'_4$	$b_1\ b_2\ b_3\ b'_4$	$\subset u(1)$
2	u(1) * v(2)	pas possible	pas possible		
3	u(1) * v(3)	pas possible	pas possible		
4	u(2) * v(1)	pas possible	pas possible		
5	u(2) * v(2)	$b_1\ b_2\ b'_5$	$b_1\ b_3\ b'_6$	$b_1\ b_2\ b_3\ b'_5\ b'_6$	$\subset u(2)$
6	u(2) * v(3)	$b_1\ b_2\ b'_5\ b_7$	$b_1\ b_3\ b'_7$	$b_1\ b_2\ b_3\ b'_5$	
7	u(3) * v(1)	pas possible	pas possible		
8	u(3) * v(2)	$b_1\ b_2\ b'_8$	$b_1\ b_3\ b'_6\ b_8$	$b_1\ b_2\ b_3\ b'_6$	
9	u(3) * v(3)	$b_1\ b_2\ b_7\ b'_8$	$b_1\ b_3\ b'_7\ b_8$	indéterminé	

Montrons pourquoi dans le cas de la combinaison u(1) ✱ v(1) on ne peut envisager que les formes $b_1\ b_2\ b_4'$ ✱ $b_1\ b_3\ b_4'$.

Si b_7 était présent dans l'expression de u(1), par construction du b_7 on aurait deux variables primées dans v(1) ce qui n'est pas possible. De même si b_8 était présent dans v(1), on aurait deux variables primées dans u(1). Les deux formes compatibles avec nos hypothèses sont celles indiquées dans le tableau.

Montrons maintenant que la combinaison u(1) ✱ v(2) ne peut être considérée. Premièrement le même raisonnement que celui indiqué ci-dessous nous indique b_7 et b_8 ne peuvent figurer. Une combinaison possible serait donc à la rigueur $b_1\ b_2\ b_4'$ ✱ $b_1\ b_3\ b_6'$. Mais b_4' et b_6' sont incompatibles car la présence de b_4' entraîne deux variables primées dans v. Le lecteur pourra vérifier qu'en utilisant ces raisonnements on aboutit au tableau ci-dessus.

Dans le cas (1) et (5) nous avons indiqué que les termes générés peuvent être absorbés puisqu'ils sont contenus dans les termes primitifs et ceci est fait chaque fois que possible dans l'algorithme ✱. Dans le cas (9) il n'y a pas de terme généré car nous avons l'hypothèse (a) de la définition de l'opération ✱.Nous allons montrer que dans les cas restants l'opération ✱ peut être obtenue en utilisant la pseudo-transitivité.

Par exemple dans le cas (6) on peut écrire :

$$\begin{aligned} u(2) + v(3) &= b_1\ b_2\ b_5'\ b_7 + b_1\ b_3\ b_7' \\ &= b_1\ b_2\ b_5'\ b_7 + b_1\ b_2\ b_3\ b_5'\ b_7 + b_1\ b_3\ b_7' + b_1\ b_2\ b_3\ b_5'\ b_7' \\ &= b_1\ b_2\ b_5'\ b_7 + b_1\ b_3\ b_7' + b_1\ b_2\ b_3\ b_5' \\ &= u(2) + v(3) + u(2) \text{ ✱ } v(3) \end{aligned}$$

On pourrait procéder de la même façon dans le cas (8). A chaque fois on montrerait que le nouveau terme u ✱ v est exactement le même que celui obtenu en appliquant la règle de pseudo-transitivité.

Ainsi l'utilisation de l'algorithme * pour déterminer les monômes premiers d'une forme relationnelle booléenne produit les mêmes monômes qui auraient pu être trouvés en utilisant les opérations sur les relations fonctionnelles.

Il en résulte que chaque monôme premier d'une forme relationnelle a une contrepartie relationnelle (c'est-à-dire chaque monôme premier à une seule variable primée) et que l'on peut interpréter chaque forme relationnelle comme un ensemble de relations fonctionnelles.

Remarque : *Le cas où l'opération * n'est pas définie correspond à une relation fonctionnelle triviale. En effet, soit par exemple, l'opération :*

$$a_1\ a_2\ a_3' * a_4\ a_2'\ a_3$$

qui correspond aux relations fonctionnelles :

$$A_1\ A_2 \rightarrow A_3$$

$$A_4\ A_3 \rightarrow A_2$$

par application de la pseudo-transitivité on obtiendrait :

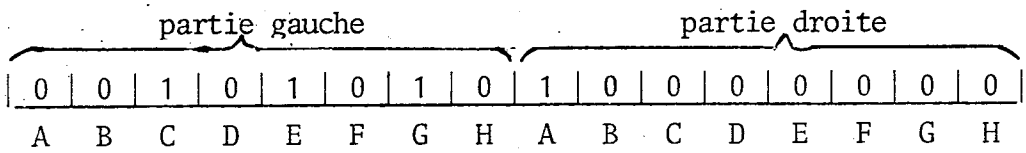
$$A_1\ A_2\ A_4 \rightarrow A_2$$

ce qui est toujours vérifié.

9.4. Applications

Le programme donné en annexe A permet de générer la couverture d'un ensemble de relations fonctionnelles élémentaires en se fondant sur les propriétés booléennes qui viennent d'être étudiées.

Dans ce programme chaque relation fonctionnelle est représentée par une chaîne binaire de caractère de longueur 2n si n est la dimension de l'ensemble des constituants. A chaque position binaire correspond un constituant. Par exemple :



représente la relation fonctionnelle C, E, G → A

Remarque : *Si l'ensemble de départ des relations fonctionnellles n'est pas un ensemble élémentaire, on peut à l'aide des propriétés booléennes rechercher une couverture de monôme premier qui sera nécessairement composé uniquement de relations fonctionnelles élémentaires. Ce travail est aussi réalisé par le programme.*

Exemple 9.3 :

Les résultats ci-joints sont l'illustration du programme et d'un ensemble de relations fonctionnelles élémentaires, où la signification des constituants est donné par le tableau suivant :

Constituants	Symbole Relationnel	Symbole Booléen
Nom	N	n
Qualification	S	s
Niveau de qualification	SL	l
Nb d'années d'expérience dans la qualification	YS	x
Nom des enfants	C	c
Nom de l'animal favori des enfants	P	p
Espèce de l'animal	PK	k
Diplôme (de Nom)	D	d
Date du diplôme	YD	z

Relations fonctionnelles élémentaires

ℓ_1 : P → C ℓ_6 : D → S ℓ_2 : P → PK

ℓ_7 : N,D → YD ℓ_3 : C → N

ℓ_4 : N,S → YS ℓ_5 : N,S → SL

Monômes premiers

pc' + ds' + pk' + ndz' + cn' + nsx' + nsl'
+ pn' + ndx' + ndl' + cdr' + scx' + scl' + pdz'
+ spx' + spl' + csx' + cdl' + pdx' + pdl'

CHAPITRE 10

MESURE DE LA PUISSANCE D'UN SYSTEME D'INFORMATIONS

Nous avons dit dans le chapitre 8 consacré à l'espace généré par un ensemble de relations fonctionnelles, que les bornes inférieures des classes d'équivalence du préordre jouaient un rôle important dans différents problèmes. Nous avons, tout d'abord vu, que les bornes inférieures représentaient des index pour des collections de données (proposition 8.4.) puis qu'elles permettaient de caractériser les formes d'une collection de données.

Dans ce chapitre, nous allons montrer que les bornes inférieures d'une part peuvent être reliées à la notion de question et d'autre part que la cardinalité de cet ensemble représente une image de la faculté du système d'informations à produire de nouvelles informations. Nous donnerons les algorithmes qui permettent de calculer la cardinalité de l'ensemble des bornes inférieures.

Le problème que nous abordons dans ce chapitre est une étude préliminaire à celui présenté par V. LUM dans [31] et par KNHUTH dans [32] (tome III page 557).

Ce dernier problème consiste à déterminer un ensemble de fichiers d'index secondaires en vue de minimiser les accès à un fichier principal pour un ensemble donné de questions. Pour résoudre ce problème, V. LUM fait l'hypothèse que les constituants apparaissant dans les questions sont indépendants les uns des autes. Cette hypothèse n'est plus vraie dans le cas où il existe une relation fonctionnelle entre des constituants et par conséquent modifie les données du problème. Les résultats de ce chapitre peuvent être alors utilisés pour reprendre le problème de V. LUM dans le cas de la dépendance entre constituants.

10.1. Questions et bornes inférieures

10.1.1. La notion de question

Un système d'informations a pour fonction d'enregistrer des données concernant des faits en vue de certaines applications. Que ces applications soient ponctuelles ou répétitives elles ont pour but d'extraire, de traiter, et d'éditer les données résultantes du traitement. Dans tous les cas l'application définit des ensembles de données sur lesquels des opérations sont effectuées. On appellera question toute formulation qui permet d'isoler un ensemble de données du système d'information. Ces questions sont parfois complexes et un langage de programmation est nécessaire pour exprimer les besoins de l'utilisateur. Ceci a été exposé dans le chapitre 5.
Ici nous voulons nous limiter à une classe de questions : les questions booléennes qui dans les applications de nature statistiques sont la plupart du temps valables..

10.1.2. Les questions booléennes

Il est bien connu que toute expression booléenne peut être transformée en une formule équivalente : la forme disjonctive normale. Dans tout ce chapitre nous considèrerons seulement une collection de données T définie sur l'ensemble $\mathcal{A}$ des constituants et structurée par un ensemble de relations fonctionnelles.

Définition : Nous dirons qu'une question est exprimée sous la forme disjonctive normale si elle est de la forme :

$$Q = Q_1 \vee Q_2 \vee \ldots \vee Q_k$$

où chaque Q_i peut s'écrire :

$$Q_i = K_{i_1} \wedge K_{i_2} \wedge \cdots \wedge K_{i_{pi}}$$

avec :

$$K_{i_j} = \begin{cases} A_{ij} = a_{ij} \\ \text{ou} \\ A_{ij} \neq a_{ij} \end{cases}$$

A_{ij} désigne un constituant de l'ensemble $\mathcal{A}$, et les opérateurs Λ, V représentent les opérations d'intersection et d'union booléenne.

Q_i sera appelé une question élémentaire de la question Q et associé avec Q_i nous définissons l'ensemble E_i :

$$E_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{p_i}}\}$$

Définition : Q est une question bien formulée si tous les ensembles E_i $i = 1,2,\ldots,k$ associés à la question $Q = Q_1 \vee Q_2 \vee \ldots \vee Q_k$ sont les bornes inférieures des classes d'équivalence du préordre engendré par les relations fonctionnelles.

Dans le cas où une classe d'équivalence du préordre est réduite à un seul élément, l'ensemble E_i est identique à cet élément.

Proposition 10.1. : **Si une question élémentaire Q_i est bien formulée il peut exister des entités de la collection de données répondant à la question, sinon il se produit un phénomène d'incompatibilité que l'étude ci-dessous décrira.**

1er cas :

$$Q_1 = K_{i_1} \wedge K_{i_2} \wedge \ldots \wedge K_{i_{p_i}}$$

avec $K_{i_j} = (A_{i_j} = a_{i_j})$ $\quad j = 1,2,\ldots, p_i$

- si Q_i est bien formulée les entités répondant à la question Q_i appartiennent à une classe d'équivalence ω_{E_i} de la partition de la collection de données (voir paragraphe 2.3.3.)
- si Q_i n'est pas bien formulée l'ensemble E_i n'est pas une borne inférieure, supposons par exemple que $F_i \subset E_i$ soit une borne inférieure. Dans ce cas, il existe une application m associée à la relation fonctionnelle $F_i \rightarrow E_i$

$$m : [F_i]T \rightarrow [E_i]T$$ où T désigne la collection de données.

Désignons par :

$$<E_i:a_i> = <A_{i_1}:a_{i_1}>, <A_{i_2}:a_{i_2}>, \ldots, <A_{i_{p_i}}:a_{i_{p_i}}>$$

l'entité associée à la question Q_i et par

$$\Pi_{F_i}(<E_i:a_i>)$$

la projection de l'entité sur le constituant composé F_i.

Si $m[\Pi_{F_i}(<E_i:a_i>)] = <E_i:a_i>$ alors il existe des entités répondant à la question. On peut même préciser que ces entités qui répondent à la question constituent une classe d'équivalence de la partition ω_{F_i}.

Si, par contre $m[\Pi_{F_i}(<E_i:a_i>)]$ n'est pas défini, il n'existe pas 'entités répondant à la question.

Le constituant F_i étant contenu dans E_i la recherche se trouve simplifiée car le nombre de constituants à examiner est plus petit.

Si la condition ci-dessus n'est pas vérifiée aucune entité ne répond à la question par suite de redondance et d'incompatibilité entre valeurs de constituants.

Exemple 10.1.

Cet exemple se rapporte à la collection de données du paragraphe 7.2.

Question : Trouver les professeurs qui enseignent le français en 4ème à la 6ème heure. L'expression de cette question sera :

$$Q = (T=français) \wedge (Y=4) \wedge (H=6)$$

L'ensemble $E = \{T,Y,H\}$ n'est pas une borne inférieure, par contre : $F = H,Y$ en est une. Il existe donc une application m telle que :

$$m[(<Y:4>,<H:6>)] = (<T:math>, <Y:4>, <H:6>)$$

On constate qu'il y a incompatibilité, par conséquent aucune entité ne répond à la question. Si la question avait été :

$$Q = (T=math) \wedge (Y=4) \wedge (H=6)$$

il y aurait eu une entité répondant à la question bien que celle-ci soit mal formulée.

2ème cas

$$Q_i = K_{i_1} \wedge K_{i_2} \wedge \ldots K_{i_r} \wedge K_{i_{r+1}} \wedge \ldots \wedge K_{i_{p_i}}$$

avec :

$$K_{ij} = (A_{ij} = a_{ij}) \qquad j = 1,2,\ldots,r$$

$$K_{ij} = (A_{ij} \neq A_{ij}) \qquad j = r+1,\ldots,p_i$$

on notera $\overline{K}_{ij}$ le complément de K_{ij}

Si Q_i est bien formulée nous considérons les questions :

$$Q'_i = K_{i_1} \wedge K_{i_2} \wedge \ldots \wedge K_{i_r}$$

$$Q''_i = K_{i_1} \wedge \ldots \wedge K_{i_r} \wedge \overline{K}_{i_{r+1}} \wedge \ldots \wedge \overline{K}_{i_{p_i}}$$

et on désigne par $C(Q_i)$, $C(Q'_i)$, $C(Q''_i)$ les entités répondant à ces questions, il est aisé de voir que :

$$C(Q_i) = C(Q'_i) - C(Q''_i) \quad \text{car} \quad C(Q''_i) \subset C(Q'_i).$$

Le processus de recherche fait appel aux deux partitions $\omega_{E'_i}$ et $\omega_{E''_i}$ avec :

$$E''_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{p_i}}\}$$

$$E'_i = \{A_{i_1}, A_{i_2}, \ldots, A_{i_r}\}$$

Si Q_i n'est pas bien formulée il est nécessaire d'avoir le phénomène de compatibilité, décrit dans le premier cas, pour que la question ait une réponse.

La formalisation d'une question booléenne nous montre l'importance des bornes inférieures des classes d'équivalence du préordre.

Actuellement dans les systèmes d'interrogation d'une banque de données on vérifie si la syntaxe de question est correcte sans que pour autant la question ait un sens. L'introduction des relations fonctionnelles permet dans le cas de questions booléennes d'informer l'utilisateur des questions mal formulées par suite des relations existantes entre constituants. Le problème n'est pas résolu pour un ensemble de questions plus lar-

ges que celui des questions booléennes mais il est permis de penser que cette première approche pourrait être développée.

10.1.3. Puissance d'un système d'informations

Le but fondamental de tout système d'information est d'extraire les données répondant à une question. Dans certains cas des calculs sont appliqués sur cet ensemble de valeurs : calcul d'une moyenne, valeur maximale ou minimale, totalisation, nombre d'élément de l'ensemble. Dans tous les cas les partitions ω_{E_i} où E_i est une borne inférieure, interviennent pour déterminer ces ensembles. Pour cette raison nous pensons que le nombre de bornes inférieures des classes d'équivalence du préordre est un indicateur des possibilités de la banque d'informations pour produire de nouvelles informations.

Définition

Nous noterons $\mathcal{J}$ l'ensemble des bornes inférieures des classes d'équivalence du préordre d'un ensemble de relations fonctionnelles, et par $\text{card}(\mathcal{J})$ la cardinalité de $\mathcal{J}$ que nous appellerons la puissance du système d'informations.

Si le préordre est défini sur un ensemble de n constituants et qu'il n'existe aucune relation fonctionnelle entre ces constituants alors $\text{card}\,\mathcal{J}$ est maximum et égal à 2^n, dans tous les cas :

$$\text{card}\,\mathcal{J} \leq 2^n$$

Le paragraphe suivant a pour but d'étudier les propriétés de l'ensemble $\mathcal{J}$, ce qui permettra de déduire un algorithme pour calculer $\text{card}\,\mathcal{J}$

10.2. Propriétés des bornes inférieures

Une voie intéressante pour étudier si un constituant composé $E \subset \mathcal{A}$ est une borne inférieure est de considérer les chaînes ou mots* qui peuvent être construits à partir de l'ensemble $\mathcal{A}$ des constituants en munissant cet ensemble de règles de la forme :

$$EF \vdash E$$

où E et F sont des éléments du monoïde $\mathcal{A}^*$ obtenu à partir de $\mathcal{A}$ en définissant une opération binaire notée EF :

- indempotente EE = E
- commutative EF = FE
- associative (EF)G = E(FG) = EFG

Définition :

Nous dirons que X est un mot réductible et Y sa dérivation s'il existe deux mots G et H tels que :

$$X = G\ E\ F\ H$$
$$Y = G\ E\ H$$

et une règle de dérivation $EF \vdash E$.

Si X n'a pas de dérivation on dira que c'est un mot irréductible.

Dans notre cas la règle de dérivation $EF \vdash E$ sera réduite de la relation fonctionnelle $E \rightarrow F$.

Il serait facile de transposer l'étude sur l'ensemble des relations fonctionnelles en une étude sur les mots du monoïde $\mathcal{A}^*$, en particulier on peut montrer que :

Proposition 10.2.

L'ensemble des mots irréductibles et l'ensemble des bornes inférieures $\mathcal{J}$ sont identiques.

---

* le lecteur ne devra pas confondre la notion de chaîne et de mot avec celle que l'on peut rencontrer dans d'autres ouvrages où ces notions sont utilisées lorsque les opérations sont non commutatives.

Nous avons vu (proposition8.4) que les bornes inférieures d'une classe d'équivalence sont des index pour une collection de données dont l'espace de définition est la borne supérieure de cette même classe d'équivalence. On peut donner une analogie avec le calcul booléen pour trouver les dérivations irréductibles (les index) d'un mot (constituant composé).

Proposition 10.3.

Si $\Phi = \{U_j \rightarrow V_j : j = 1,2,\ldots,p\}$ désigne un ensemble de relations fonctionnelles et E un constituant composé de $E_1, E_2, \ldots, E_k$ alors les monômes premiers de la fonction booléenne :

$$f = \prod_{i=1}^{k} e_i + \prod_{j=1}^{p} u_j \, v_j'$$

qui ne contiennent pas de variables complémentées sont des dérivations irréductibles du mot E (constituant composé E).

La démonstration de cette proposition sera donnée au chapitre 11 car cette propriété est aussi utilisée dans le problème de la décomposition d'une collection de données.

10.3. Connexion de deux graphes d'ensemble de relations fonctionnelles

10.3.1. Principe général

Notre but est de rechercher un moyen de déterminer de façon algorithmique l'ensemble des bornes inférieures.

Etant donné deux préordres des classes d'équivalences construitsrespectivement sur les ensembles de constituants $\mathcal{a}$ et $\mathcal{a}'$ et par les ensembles de relations fonctionnelles Φ et Φ' dont on connait l'ensemble des bornes inférieures $\mathcal{J}$ et $\mathcal{J}'$. Le problème est d'étudier l'ensemble des bornes inférieures du préordre défini par :

- l'ensemble des constituants $\mathcal{A} \cup \mathcal{A}'$
- les relations fonctionnelles $\Phi \cup \Phi' \cup \{X \rightarrow Y'\}$ où Y' est un constituant élémentaire de $\mathcal{A}'$ et X un constituant composé dont tous les constituants élémentaires appartiennent à $\mathcal{A}$.

Le schéma de la figure 10.1 représente le graphe résultant de la connexion si G et G' sont les graphes représentant les relations fonctionnelles Φ et Φ'.

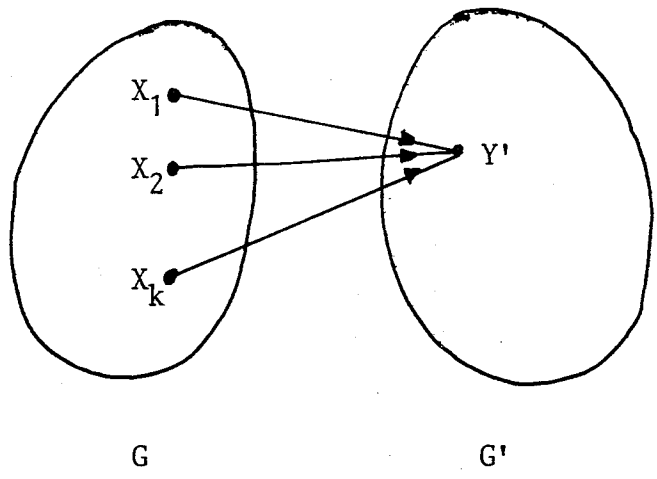


FIGURE 10.1. : CONNEXION DE DEUX GRAPHES G ET G' PAR LA RELATION FONCTIONNELLE $X_1, X_2, \ldots, X_k \rightarrow Y'$

Cette façon de poser le problème correspond dans une certaine mesure, à la situation où un système se trouverait pour intégrer deux bases de données indépendantes à un moment donné, en une base unique en précisant les règles d'intégration.

Divisons l'ensemble $\mathcal{J}$ en deux classes $\mathcal{J}_1$ et $\mathcal{J}_2$ et $\mathcal{J}'$ en trois classes $\mathcal{J}'_1, \mathcal{J}'_2, \mathcal{J}'_3$ définies par :

$\mathcal{J}_1 = \{E : E \in \mathcal{J}$ et $\exists G \subset E$ tel que $GY' \vdash G\}$

$\mathcal{J}_2 = \{H : H \in \mathcal{J}$ et $H \notin \mathcal{J}_1\} = \mathcal{J} - \mathcal{J}_1$

$\mathcal{J}'_1 = \{Y'H'\}$

$\mathcal{J}'_2 = \{F'G' : \forall F'_i \in F', F'_i\ Y' \vdash Y'$, et $F'G' \in \mathcal{J}'\}$

$\mathcal{J}'_3 = \{L' : L' \in \mathcal{J}', L' \notin \mathcal{J}'_1, L' \notin \mathcal{J}'_2\}$

Il est important de remarquer pour la suite du raisonnement que les mots H' et G' qui entrent dans la définition de $\mathcal{J}'_1$ et $\mathcal{J}'_2$ sont dans la classe $\mathcal{J}'_3$.

Les mots irréductibles du préordre :

$\Phi \cup \Phi \cup \{X \to Y'\}$

sont les mots irréductibles de Φ et Φ', plus tous les mots irréductibles obtenus par combinaison d'un mot irréductible de Φ avec un mot irréductible de Φ'. La classification faite sur $\mathcal{J}$ et $\mathcal{J}'$ nous conduit à étudier les six cas qui sont résumés dans le tableau de la figure 10.2.

Combinaison	Forme	Réduction
$\mathcal{J}_1$ $\mathcal{J}'_1$	E Y' H'	E H'
$\mathcal{J}_1$ $\mathcal{J}'_2$	E F' G'	E G'
$\mathcal{J}_1$ $\mathcal{J}'_3$	E L'	Irréductible
$\mathcal{J}_2$ $\mathcal{J}'_1$	H Y'H'	Irréductible
$\mathcal{J}_2$ $\mathcal{J}'_2$	H F' G'	Irréductible
$\mathcal{J}_2$ $\mathcal{J}'_3$	H L'	Irréductible

FIGURE 10.2. : ETUDE DES COMBINAISONS POSSIBLES

Montrons que E Y'H' $\vdash$ E H', comme E $\in \mathcal{J}_1$ il existe G $\subset$ E tel que $GY' \vdash G$ on peut donc écrire E = GK et

G K Y'H' $\vdash$ G K H'

soit E Y'H' $\vdash$ E H'.

De même, montrons que $EF'G' \vdash EG'$, on peut écrire

$EF'G' = GK\ F'_1\ F'_2 \ldots F'_K\ G'$ où $F'_1, F'_2, \ldots, F'_k$ sont des constituants élémentaires, par construction de G et F' on a les propriétés suivantes :

$$G \rightarrow Y' \text{ et pour } i = 1,2,\ldots,k \quad Y' \rightarrow F'_i$$

soit : $G \rightarrow F'_i \quad i = 1,2,\ldots,k$ on en déduit donc :

$$GF'_i \vdash G \quad i = 1,2,\ldots,k$$

il est alors immédiat par application successive de la règle de dérivation que : $GKF'_1F'_2\ldots F'_kG' \vdash GKG'$ soit $EF'G' \vdash EG'$

Comme les mots de la forme H' et G' sont contenus dans la classe $\mathfrak{J}'_3$ il en résulte que les mots de la forme EH' et EG' sont inclus dans la combinaison $\mathfrak{J}_1, \mathfrak{J}'_3$ on peut donc établir la proposition suivante :

Proposition 10.4.

Les bornes inférieures générées par l'ensemble des relations fonctionnelles $\Phi \cup \Phi' \cup \{X \rightarrow Y'\}$ est défini par :

$$\mathfrak{J} \cup \mathfrak{J}' \cup (\mathfrak{J}_1, \mathfrak{J}'_3) \cup (\mathfrak{J}_2, \mathfrak{J}')$$

où $(\mathfrak{J}_1, \mathfrak{J}'_3)$ et $(\mathfrak{J}_2, \mathfrak{J}')$ désignent tous les mots qui peuvent être obtenus par combinaison des classes composantes. On obtient aussi pour formule de dénombrement :

$$\text{card}(\mathfrak{J}) + \text{card}(\mathfrak{J}') + \text{card}(\mathfrak{J}_1)\ \text{card}(\mathfrak{J}'_3) + \text{card}(\mathfrak{J}_2)\ \text{card}(\mathfrak{J}')$$

10.3.2. Application dans le cas où le graphe des relations fonctionnelles est une arborescence

Considérons deux arborescences T et T' représentatives de deux ensembles de relations fonctionnelles définies sur des espaces $\mathcal{a}$ et $\mathcal{a}'$ disjoints. Soient U et V les sommets de ces deux arborescences, nous voulons étudier la puissance d'un système d'informations de racine W construit à partir de T et T' comme l'indique la figure 10.3. Pour cela nous allons procéder en deux étapes, dans la première calculer la puissance du système composé de W et T avec l'addition de relation fonctionnelle $W \rightarrow U$, puis dans la deuxième étape déterminer la puissance du système formé par (W,T) et T' avec addition de la relation fonctionnelle $W \rightarrow V$.

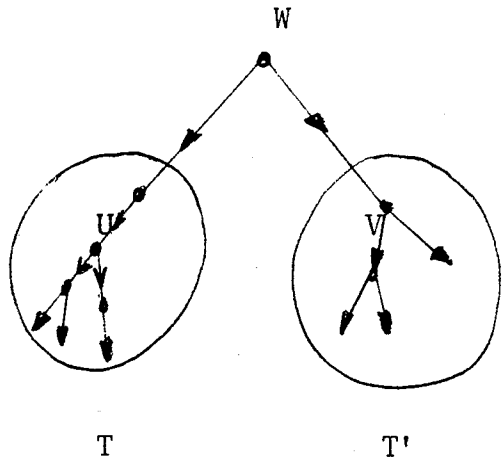


FIGURE 10.3. : CONNEXION DE DEUX ARBORESCENCES

1ère étape :

Connexion de W avec T par l'intermédiaire de la relation fonctionnelle W → U. Désignons respectivement par $\mathfrak{J}^W$ et $\mathfrak{J}^T$ les bornes inférieures des deux systèmes W et T. Si on divise $\mathfrak{J}^W$ et $\mathfrak{J}^T$ en classes on constate :

$$\mathfrak{J}^W_1 = \{W\},\ \mathfrak{J}^W_2 = 0$$

$$\mathfrak{J}^T_1 = \{U\}\quad \mathfrak{J}^T_2 = \mathfrak{J}^T - \mathfrak{J}^T_1 \qquad \mathfrak{J}^T_3 = \emptyset \text{ parce que T est}$$

une arborescence.

On en déduit par application de la proposition 10.4.

$$\text{card}_{W,T} = 1 + \text{card}_T$$

2ème étape

Connexion de (W,T) avec T' par l'intermédiaire de la relation fonctionnelle W → V. Désignons respectivement par $\mathfrak{J}^{W,T}$ et $\mathfrak{J}^T$ les bornes inférieures. Il vient :

$$\mathfrak{J}_1^{W,T} = \mathfrak{J}^{W,T} = \{W\}$$

$$\mathfrak{J}_2^{W,T} = \mathfrak{J}^{W,T} - \{W\}$$

$$\mathfrak{J}_1^{T'} = \{V\}$$

$$\mathfrak{J}_2^{T'} = \mathfrak{J}^{T'} - \mathfrak{J}_1^{T'}$$

$$\mathfrak{J}_3^{T'} = 0$$

$$\text{card}_{(W,T),T'} = \text{card}_{W,T} + \text{card}_{T'} + (\text{card}_{W,T}-1)\,\text{card}_{T'}$$

$$= 1 + \text{card}_T + \text{card}_{T'} + \text{card}_T\,\text{card}_{T'}$$

$$= (1+\text{card}_T)(1+\text{card}_{T'})$$

Dans le cas d'arborescences le résultat aurait pu être obtenu directement sans application de la formule générale. En effet, les mots irréductibles du système est l'ensemble :

- mots irréductibles de T
- mots irréductibles de T'
- des mots obtenus par combinaison des mots de T et des mots de T' qui sont irréductibles car T et T' sont des arborescences
- du mot W

ce qui donne bien :

$$1 + \text{card}_T + \text{card}_{T'} + \text{card}_T \times \text{card}_{T'} = (1+\text{card}_T)(1+\text{card}_{T'})$$

Cette formule peut être généralisée au cas de n arborescences.

Proposition 10.5.

Si $T_1, T_2, \ldots, T_n$ désignent n arborescences de racine $U_1, U_2, \ldots, U_n$ et que l'on établisse plusieurs connexions comme l'indique la figure 10.4., alors $\text{card}_{T_1 T_2 \ldots T_n} = (1+\text{card}_{T_1})(1+\text{card}_{T_2}) \ldots (1+\text{card}_{T_n})$

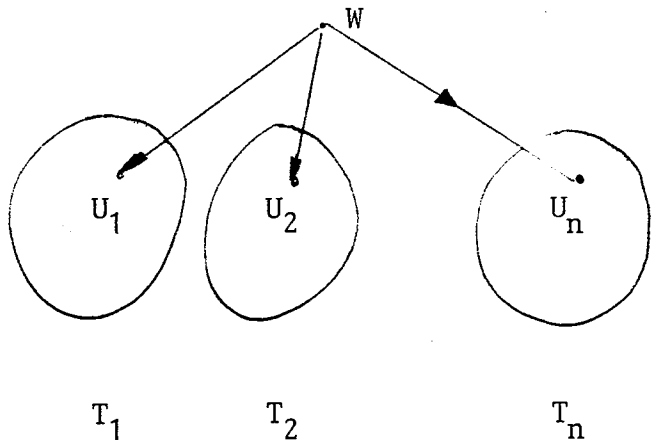


FIGURE 10.4. : Connexion de n arborescences

Cette proposition se démontre par récurrence, elle est vraie, pour n=1, n=2 supposons la vraie pour n-1. D'après la proposition 10.4., on peut écrire :

$$\text{card}_{T_1 T_2 \ldots T_n} = \text{card}_{T_1 T_2 \ldots T_{n-1}} + \text{card}_{T_n} + (\text{card}_{T_1 T_2 \ldots T_{n-1}} - 1)\,\text{card}_{T_n}$$

$$= \text{card}_{T_1 T_2 \ldots T_{n-1}} (1+\text{card}_{T_n})$$

$$= (1+\text{card}_{T_1})(1+\text{card}_{T_2}) \ldots (1+\text{card}_{T_n})$$

ce qui assure le résultat pour n.

Exemple 10.2. : *Application numérique*

Pour l'arborescence de la figure 10.5. comprenant 12 constituants, il vient, si on désigne par $T_1, T_2, \ldots, T_6$ des sous-arborescences de T.

$$card_T = 1 + card_{T_1}$$

$$= (1+card_{T_2})(1+card_{T_3})$$

$$= (1+card_{T_4})(1+card_{T_5})(1+card_{T_6})$$

avec

$$card_{T_3} = 3, card_{T_4} = 2, card_{T_5} = 8, card_{T_6} = 2$$

$$card_T = 329$$

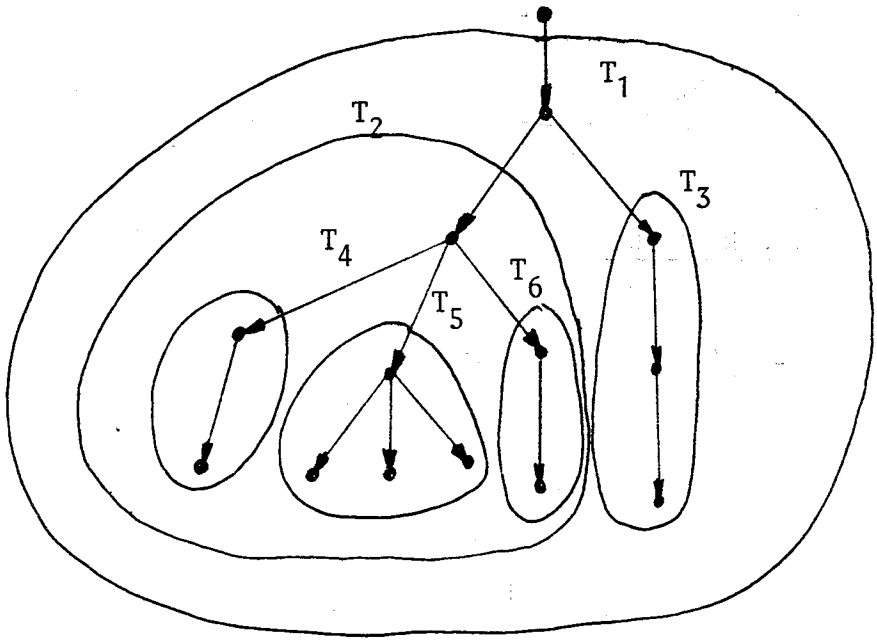


FIGURE 10.5. : ARBORESCENCE T

10.3.3. Cas d'une connexion sur le graphe lui-même des relations fonctionnelles

Etant donné un ensemble de relations fonctionnelles Φ définie sur un espace a, auquel on adjoint une relation fonctionnelle supplémentaire $X \rightarrow Y$ où les constituants X et Y appartiennent à a. Dans ces conditions que devient l'ensemble des bornes inférieures ?

Si $\mathcal{J}$ désigne l'ensemble des mots irréductibles, les seuls mots réductibles ce sont ceux qui contiennent XY. Soit $E \supset XY$ ou $E = XYH$ alors $XYH \vdash XH$.

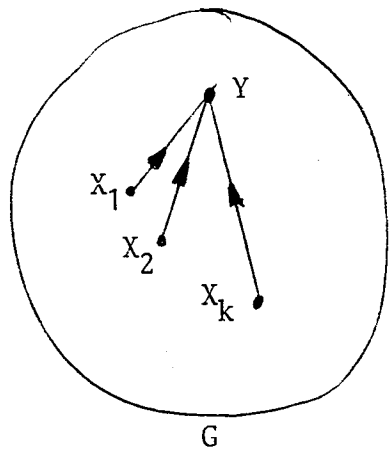


FIGURE 10.6.

Cas d'une connexion correspondant au schéma de la figure 10.7.

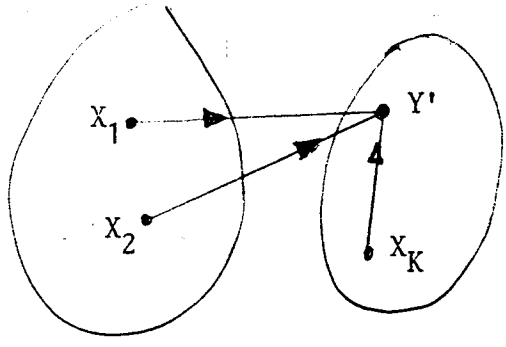


FIGURE 10.7.

Dans ce cas on peut se ramener au cas précédent en procédant en deux étapes :

1ère étape

Etude de la puissance du système formé par G et les points isolés Y' et X_K, ceci correspondant à la figure 10.8.

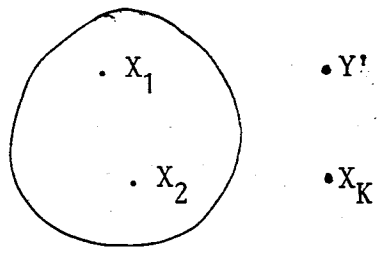


FIGURE 10.8.

2ème étape cas d'une connexion sur le graphe lui-même

10.3.4. Algorithmes

Les propriétés que nous venons d'étudier sur la connexion de deux graphes de relations fonctionnelles permettent de trouver de proche en proche le nombre de mots irréductibles d'un préordre généré par un ensemble de relations élémentaires Φ. Pour cela, il suffit de prendre la première relation élémentaire et le préordre associé, puis la deuxième relation élémentaire et le préordre associé, on connectera les deux préordres et ainsi de suite. A chaque étape on décomposera les préordres en classes comme elles ont été définies.

Nous avons aussi mis en oeuvre un autre algorithme qui part du principe que si E est un mot irréductible tous les sous-mots de E sont irréductibles. On procède alors de la façon suivante :

- Etant donné $\Phi = \{\ell_i\}$ ensemble des relations élémentaires sur $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ on commence par générer la fermeture de Φ : Φ^+ selon le processus décrit dans le chapitre 9.

- D'autre part, on sait que les mots $A_1, A_2, \ldots, A_n$ sont des mots irrédutibles, partant de ce fait à A_1 on ajoute A_2 ce qui donne le mot A_1A_2 et on regarde si dans l'ensemble Φ^+ il n'existe pas une relation élémentaire qui permette de réduire le mot, si oui, A_1A_2 n'est pas irréductible, si non, A_1A_2 est irréductible et en procédant ainsi de proche en proche on obtient l'ensemble des mots irréductibles.

Nous donnons dans le tableau de la figure 10.9 les résultats fournis par un programme ordinateur pour l'exemple du paragraphe 7.2. La fermeture a été donnée dans l'exemple 9.1. du pagraphe 9.1.

P	Mots irréductibles
H	
Y	
N	
T	
PH	
PY	
PN	
HY	
HN	
HT	
YN	
YT	
NT	
PYN	
YNT	

FIGURE 10.9.

CHAPITRE 11

DECOMPOSITION D'UNE COLLECTION DE DONNEES

La décomposition d'une collection de données en sous-collections est un problème fondamental dans la conception d'un système d'informations. On remarquera qu'en fait il existe des décompositions multiples mais de différentes natures, les unes appelées "décomposition primaire" respectent la conception de l'utilisateur, les autres appelées "décomposition secondaire" sont uniquement valides pour la représentation mémoire des données. Des algorithmes de décomposition seront étudiés et en particulier celui de la décomposition fonctionnelle. De plus une algèbre des décompositions sera examinée.

11. DECOMPOSITION D'UNE COLLECTION DE DONNEES

11.1. Définition du problème de la décomposition

La première phase dans la construction d'un système est de choisir la structure logique la mieux adaptée pour représenter l'information. Il nous semble que la représentation de cette structure logique à l'aide d'un ensemble de collections de données répond à cet objectif. Dans cette conception on peut remarquer qu'aucune notion d'ordre n'est impliquée (ni ordre sur la succession des constituants, ni ordre sur la succession des entités). Cette façon de voir est fondamentale lorsqu'on désire séparer la structure logique des nombreuses représentations physiques de cette même structure logique.

Le niveau de la représentation logique correspond à celui de l'utilisateur qui donne la façon dont il se représente un système d'informations.Le problème que nous voulons étudier est de rechercher si un même système d'informations peut être représenté par différents découpages, si certains découpages présentent des avantages, donc répondraient mieux aux besoins de l'utilisateur.

N'importe quel ensemble de données peut être représenté par une collection de données (ex. du paragraphe 7.2.). Une telle représentation serait, dans la plupart des cas, redondante et difficile à manipuler; néanmoins c'est un concept utile pour les raisons suivantes. Si une banque de données peut être décomposée en collections de données, chaque collection de données exprimant une relation sur un sous-ensemble de l'espace de définition de la banque de données, alors il est fondamental que la fonction de regénération conduise au même ensemble de départ.

C'est ainsi, que dans une décomposition hiérarchique, dans laquelle la description globale est décomposée progressivement en ensemble de plus en plus petit, à chaque étape du processus il est possible de regénérer les données associées à l'étape précédente.

L'opération de composition normale, définie au paragraphe 3.4.2., est une procédure fondamentale de la décomposition d'une collection de données en deux sous-collections.

La proposition 3.1 du paragraphe 3.6.4 donne une condition nécessaire et suffisante de décomposition. La décomposition d'une collection de données T signifie que la collection T peut être recréée à partir des projections de T à l'aide de l'opérateur de composition normale. Par exemple, l'expression :

$$[A,B,C,D,E]T = [A,B]T * [B,C]T * [A,D,C]T * [D,E]T$$

implique que la collection de données T peut être représentée par un ensemble de collections de données constitué par ses projections sur (A,B), (B,C), (A,D,C) et (D,E).

Il est clair que lorsqu'il existe une relation fonctionnelle $A \rightarrow B$ la proposition 3.1 s'applique et permet d'écrire : $[ABC]T = [AB]T * [AC]T$. Mais inversement l'existence d'une décomposition n'implique pas l'existence d'une relation fonctionnelle (voir exemple 3.5. paragraphe 3.6.4.).

Notre objectif dans ce chapitre sera d'étudier les propriétés d'une décomposition et de rechercher les décompositions possibles pour une collection de données.

Nos hypothèses pour aborder ce problème seront :

- la donnée d'un ensemble de relations fonctionnelles,
- la donnée d'expressions de décomposition du type $[ABC]T = [AB]T * [AC]T$ à condition que de telles expressions ne soient pas dérivées des relations fonctionnelles (l'espace sur lequel la décomposition est définie peut être un sous-espace de l'espace total). De telles expressions expriment des propriétés intrinsèques de la collection T, indépendamment des relations fonctionnelles qui peuvent exister entre les constituants.

Dans l'état actuel, ce problème n'a pas de solution générale. Si notre objectif est de générer le plus grand nombre possible de décompositions il n'est pas possible d'affirmer que toutes les décompositions sont obtenues. Bien que cela soit un problème de nature théorique la plupart des solutions pratiques ne nécessitent pas un tel outillage.

11.2. Formalisation du problème de la décomposition

Nous considérons tout d'abord le cas où c'est un système de collections qui est donné (voir paragraphe 2.2.3.) et non une seule collection de données.

Etant donné un système $(\mathcal{F}', \mathcal{R}')$ dont un élément de $\mathcal{F}'$ est une famille de collections de données $(T'_1, T'_2, \ldots, T'_k,)$ définies sur un ensemble $\mathcal{a}$ de constituants.

Premièrement, nous devons transformer ce système en remplaçant toutes les paires (T'_i, T'_j) par leur composition normale chaque fois que cela est possible. Seule l'utilisateur peut décider si une paire (T'_i, T'_j) est composable, grâce à la signification des collections de données T'_i et T'_j. Par répétition de ce processus nous arrivons à un nouveau système $(\mathcal{F}, \mathcal{R})$ avec pour élément de $\mathcal{F}$ une famille $(T_1, T_2, \ldots, T_k)$ où pour tout $i \neq j$ la composition $T_i * T_j$ n'est plus possible. Dans ce nouveau système $\mathcal{R}$ est défini par :

$\mathcal{R}(T_1, T_2, \ldots, T_k)$ est vrai si et seulement si

$\mathcal{R}'(T'_1, T'_2, \ldots, T'_k,)$ est vrai.

Désignons par $\mathcal{a}_i$ l'ensemble des constituants qui définissent la collection de données T_i.

Supposons que nous ayons pu déterminer toutes les décompositions possibles de T_i, c'est-à-dire toutes les paires de sous-ensembles de $\mathcal{a}_i$

$$\{(E_{i1}, F_{i1}), (E_{i2}, F_{i2}), \ldots, (E_{ik(i)}, F_{ik(i)})\}$$

telles que :

- $E_{ij}, F_{ij} = \mathcal{a}_i \qquad E_{ij} \cdot F_{ij} \neq \emptyset$

- $T_i = [E_{ij}]T_i * [F_{ij}]T_i$

pour $j = 1, 2, \ldots, k(i)$.

Si maintenant on répète le même processus sur $[E_{ij}]T_i$ et $[F_{ij}]T_i$, on arrivera à exprimer toutes les décompositions de T_i par des expressions de la forme :

(Eq.11.1) $$T_i = T_{i1} * T_{i2} * \ldots * T_{im(i)}$$

où les éléments T_{ij} sont des projections de T_i.

Finalement, on obtient un autre système : $(\mathcal{F}'', \mathcal{R}'')$ avec pour éléments de $\mathcal{F}''$ une famille :

$$(T_{11}, T_{12}, \ldots, T_{1m(1)}, \ldots, T_{k1}, T_{k2}, \ldots, T_{km(k)})$$

qui est une représentation différente du système de départ et qui lui est équivalente, dans le sens, que l'on peut toujours retrouver, sans perte, l'information contenue dans le système de départ.

Une décomposition sera dite <u>irréductible</u> si aucun des éléments ne peut être décomposé dans ce cas un élément irréductible est de la forme :

$$T_{ij} = M * N$$

ce qui implique soit $M = T_{ij}$, soit $N = T_{ij}$

Il est évident que dans ce problème de décomposition on va essentiellement s'intéresser aux décompositions irréductibles, et par là même aux éléments irréductibles. Il est possible de donner une caractérisation d'une collection de données irréductible en terme de relation fonctionnelle.

<u>Proposition 11.1</u>

Si une collection de données T définie sur l'ensemble de constituants $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ est irréductible alors s'il existe une relation fonctionnelle élémentaire elle est de la forme $\mathcal{A} - A_i \rightarrow F$, où F est constituant composé.

Si nous supposons qu'il existe une relation fonctionnelle élémentaire de la forme :

$$E \rightarrow F$$

où nous définissons $\overline{E}$ par $\overline{E} = \mathcal{A} - E = \{A_{i1}, A_{i2}, \ldots, A_{ik}\}$ $K > 1$ et telle que $F \not\subset E$, alors par application de la proposition 3.1, on obtiendrait :

$$[E,F]\, T * [E,\overline{F}]T = T$$

En choisissant une partie F si nécessaire, on peut supposer que :

$$F = A_j \subset \overline{E}$$

mais alors $(E,\overline{F}) = \overline{F}$ et (E,F) sont strictement contenus dans $\mathcal{A}$ et $(E,\overline{F}), (E,F) = \mathcal{A}$. Ce qui signifierait que T ne serait pas irréductible. Par conséquent $K \leqslant 1$ et $\overline{E} = \{A_i\}$ ou vide.

Remarque :

Dans la suite de ce chapitre, nous nous intéresserons à la décomposition d'une collection de données T. Il existe de nombreux systèmes où partant d'un système $(\mathcal{F}', \mathcal{R}')$, dont un élément de $\mathcal{F}'$ est une famille $(T'_1, T'_2, \ldots, T'_k)$, il est possible de le transformer en un système $(\mathcal{F}, \mathcal{R})$ où un élément de $\mathcal{F}$ est une famille réduite (T). Si cela n'était pas le cas, les raisonnements que nous donnerons et qui s'appliquent pour une collection T, doivent être utilisés pour chacune des collections de données composant le système. Dans une telle hypothèse si la collection T est définie sur l'ensemble $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ nous remplacerons la notation :

$[A_1, A_2, A_3]\, T$ *par* (A_1, A_2, A_3) *sans aucune ambiguïté. De même l'écriture :*

$$(A_1, A_2, A_3) = (A_1, A_2) * (A_1, A_3)$$

sera équivalente à :

$$[A_1, A_2, A_3]T = [A_1, A_2]T * [A_1, A_3]\, T.$$

Nous aborderons ce problème, tout d'abord en nous limitant à ce que nous appelons les décompositions fonctionnelles. Sur ce point là il sera possible d'établir une théorie assez générale, dont on établira les limites.

11.3. Décomposition fonctionnelle

11.3.1. Décomposition fonctionnelle et forme booléenne

11.3.1.1. Définition du problème

Nous dirons qu'une décomposition est fonctionnelle si nous utilisons uniquement les relations fonctionnelles pour rechercher les décompositions.

Soit $\mathcal{A} = \{A_i : i = 1,2,\ldots,n\}$ un ensemble de constituants et $\{E_i \rightarrow A_i : i = 1,2,\ldots,k\}$ un ensemble de relations fonctionnelles.

La relation fonctionnelle $E_i \rightarrow A_i$ fournit la décomposition :

$$(A_1\ A_2\ \ldots\ A_n) = (E_i\ A_i) * (A_1\ A_2 \ldots\ A_{i-1}\ A_{i+1}\ \ldots\ A_n)$$

On peut maintenant chercher à décomposer les parties (E_i, A_i) et $(A_1, A_2, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n)$ en utilisant les autres relations fonctionnelles.

Si le processus décrit ci-dessus est simple du point de vue logique, il peut paraître, à première vue, très long car il faudra explorer différentes possibilités qui ne conduisent pas nécessairement à des décompositions différentes. Les performances d'un tel procédé sont grandement améliorées par le fait qu'il existe une correspondance entre ce processus et les propriétés d'une classe de fonctions booléennes. De plus, cette analogie permet d'obtenir un résultat indépendant de l'ordre dans lequel les relations fonctionnelles sont utilisées dans la décomposition. L'analogie est similaire à celle vue au paragraphe 9.2., pour l'étude des relations fonctionnelles. Dans ce cas nous allons construire la fonction booléenne :

$$h = \prod_i^n a_i + \sum_i e_i\ a_i'$$

Le terme $\prod_i^n a_i$ est introduit pour servir de cible aux relations fonctionnelles.

Illustrons l'analogie en supposant qu'il n'existe qu'une seule relation fonctionnelle $E_i \rightarrow A_i$ alors nous avons :

$$h = a_1\ a_2 \cdots a_n + e_i\ a_i'$$

$$h = a_1\ a_2 \cdots a_n + a_i'\ a_1\ a_2 \cdots a_{i-1}\ a_{i+1} \cdots a_n + e_i\ a_i'$$

$$h = a_1\ a_2 \cdots a_{i-1}\ a_{i+1} \cdots a_n + e_i\ a_i'$$

qui est bien l'image de la décomposition donnée ci-dessus.

Pour arriver à montrer cette propriété nous allons procéder par étapes. Tout d'abord, nous allons établir la proposition 11.2 qui est à la base de la propriété.

11.3.1.2. Forme relationnelle et index

Proposition 11.2.

Soit $f(a_1,a_2,\ldots,a_n)$ une forme relationnelle, dont les monômes premiers sont $\{f_i : i = 1,2,\ldots,m\}$. Si un terme $p = a_{k+1}a_{k+2}\cdots a_n$ dont chaque variable est non complémentée est un monôme premier de la fonction booléenne :

$$h = \prod_i^n a_i + f$$

alors $A_{k+1}A_{K=2}\cdots A_n$ est index, c'est-à-dire :

$$A_n \rightarrow A_r \quad r = 1,2,\ldots,n$$

Cette proposition est la conséquence l'algorithme ✱
L'algorithme ✱ dans ce cas commence avec l'ensemble :

$$C_o = \{\text{termes de } f,\ \prod_i^n a_j\}$$

et forme de façon itérative des ensembles de termes :

$$C_{j+1} = C_j \cup C_j \text{✱} C_j \qquad j = 0,1,2,\ldots$$

A chaque étape toutes les absorptions possibles sont effectuées dans C_{j+1} et tous les éléments ayant plus de n-j variables sont éliminés. Le terme p doit apparaître au $k^{\text{ième}}$ pas, et p ne peut dérivé uniquement de f puisque les monômes premiers de f ont une variable complémentée.

En conséquence le terme $\prod_1^n a_i$ doit apparaître dans le processus conduisant à p.

Remarque :

L'opération $*$ *possède la propriété que si deux termes* v_1 *et* v_2 *ont* m_1 *et* m_2 *variables absentes alors* $v_1 * v_2$ *a au moins* $min(m_1,m_2)$ *variables absentes.*

En conséquence à chaque pas de la séquence des opérations sur $\prod_1^n a_i$ une variable est éliminée. Pour arriver en k pas à un élément ayant k variables absentes nous devons avoir une séquence de la forme :

$$p = (((\prod_1^n a_j * w_1) * w_2) \ldots * w_k)$$

où chaque w_j peut être le produit d'autre opération $*$. Cette séquence peut être écrite sous une autre forme :

$$u_j = u_{j-1} * w_j \qquad j = 1,2,\ldots,k$$

$$u_o = \prod_1^n a_j \qquad u_k = p$$

et une représentation serait la suivante (à un arrangement près des variables $a_1\ a_2 \ldots a_k$)

u_o	a_1	a_2	a_3	...	a_{k-1}	a_k	a_{k+1}	... a_n
w_1	a'_1	y	y	...	y	y	y	... y
u_1	x	a_2	a_3	...	a_{k-1}	a_k	a_{k+1}	... a_n
w_2	x	a'_2	y	...	y	y	y	... y
u_2	x	x	a_3	...	a_{k-1}	a_k	a_{k+1}	... a_n
w_3	x	x	a'_3	...	y	y	y	... y
w_{k-1}	x	x	x	...	a'_{k-1}	y	y	... y
w_{k-1}	x	x	x	...	x	a_k	a_{k+1}	... a_n
w_k	x	x	x	...	x	a'_k	y	... y
u_k	x	x	x	...	x	x	a_{k+1}	... a_n

un x précise que la variable est absente tandis que le y est soit a_j ou x.

Cette séquence d'opérations montre deux propriétés fondamentales de w_j :

(1) - une variable apparaissant sous forme complémentée au $l^{ième}$ pas n'apparaît plus pour les pas $l+1, l+2, \ldots, k$

(2) - une seule variable complémentée peut être présente dans un élément w_j.

En effet si w_j ne contenait pas de variables complémentées cela signifirait que :

$$w_j + \prod_{i=1}^{n} a_i = w_j ,$$

mais alors l'opération $w_j * u_{j-1}$ n'éliminerait aucune variable et cette séquence ne conduirait pas à p.

Si w_j contient une et une seule variable primée, il est contenu dans un monôme premier de f et on peut interpréter w_j comme une relation fonctionnelle :

$$w_j \Longleftrightarrow Y_{j+1}\ Y_{j+2} \ldots Y_n \rightarrow A_j \qquad j = 1,2,\ldots,k$$

où Y_{j+r} désigne $\begin{cases} A_{j+r} \\ \text{ou l'élément absent.} \end{cases}$

En utilisant la propriété d'augmentation, il est possible d'écrire en notant ℓ_i la relation fonctionnelle :

$$\ell_i : A_{j+1}\ A_{j+2} \ldots A_n \rightarrow A_j \qquad j = 1,2,\ldots,k$$

Si nous définissons :

$$\mu_j : A_{j+1}\ A_{j+2} \ldots A_n \rightarrow A_1\ A_2 \ldots A_j$$

alors on peut montrer par récurrence que $(\ell_1, \ell_2, \ldots, \ell_k)$ implique μ_k.

(ℓ_1, ℓ_2) implique μ_2 par pseudo-transitivité.

Supposons que :

$(\ell_1, \ell_2, \ldots, \ell_{j-1})$ implique u_{j-1}

alors $(\ell_1, \ell_2, \ldots, \ell_{j-1}, \ell_j)$ implique (μ_{j-1}, ℓ_j) qui par pseudo-transitivité implique μ_j.

En conséquence si p est un monôme premier de **h** on a :

$A_{k+1}\ A_{k+2} \cdots A_n \rightarrow A_1\ A_2 \cdots A_k$

Il reste à montrer que c'est une relation élémentaire, donc qu'il n'est pas vrai que :

$$A_{k+2},\ A_{k+3} \cdots A_n \rightarrow A_1 A_2 \cdots A_k\ A_{k+1}$$

Si cette hypothèse était vraie, alors la fonction f contiendrait tous les termes :

$$t_j = a'_j\ a_{k+2} \cdots a_n \qquad j = 1,2,\ldots,k+1$$

et l'on aurait :

$$(\ldots((\prod_1^n a_i * t_1) * t_2)\ldots * t_{k+1}) = a_{k+2} \cdots a_n$$

ce qui serait contraire à l'hypothèse que p est un monôme premier de h.

Remarque :

La séquence des opérations u_o, w_1, u_1, w_2,..., u_{k-1}, w_k, u_k peut être interprétée, à l'aide de la représentation graphique du préodre (chapitre 8, paragraphe 8.3.) comme un chemin de la borne supérieure $A_1\ A_2 \ldots A_n$ à la borne inférieure $A_{k+1}\ A_{k+2} \cdots A_n$. Les u_i correspondent aux noeuds du graphe, tandis que chaque arrête (u_i, u_{i+1}) peut être libellée par w_{i+1} représentant la relation fonctionnelle qui permet de passer de u_i à u_{i+1}.

Proposition 11.3.

Si $A_{k+1}\ A_{k+2} \cdots A_n$ est un index et $p = a_{k+1}\ a_{k+2} \cdots A_n$ sa représentation booléenne, alors il existe un monôme premier a a'_i de la forme relationnelle $f = (a_1, a_2, \ldots, a_n)$, dont l'interprétation est la relation fonctionnelle $A \rightarrow A_i$, tel que :

$$A_{k+1}\ A_{k+2} \cdots A_n \supseteq A$$

Tout d'abord introduisons la notion qu'une fonction booléenne f est <u>inclue</u> dans g, ce qui sera noté $f \subset g$ si :

$$f = 1 \quad \text{implique} \quad g = 1$$

Conformément à la proposition 11.2., si $p = a_{k+1}\, a_{k+2} \ldots a_n$ est un monôme premier $A_{k+1}\, A_{k+2} \ldots A_n$ est un index, c'est-à-dire que

$$A_{k+1}\, A_{k+2} \ldots A_n \rightarrow A_r \qquad r = 1,2,\ldots,k$$

Si nous faisons l'hypothèse que tous les termes de la forme

$$e = a_{k+1}\, a_{k+2} \ldots a_n\, a'_r$$

vérifient la propriété suivante :

il n'existe pas $\ell \supseteq e$ tel que ℓ soit un monôme premier.

On peut alors affirmer qu'il existe deux relations fonctionnelles, dont les termes sont des monômes premiers, telles que :

$$\left.\begin{array}{l} L_1\, L_2 \ldots L_{q_r} \rightarrow F_r \\ F_r,\, G_r \rightarrow A_r \end{array}\right\} \qquad r = 1,2,\ldots,k$$

avec $L_1\, L_2 \ldots L_{q_r} \subset A_{k+1}\, A_{k+2} \ldots A_n$

$$G_r \subset A_{k+1}\, A_{k+2} \ldots A_n$$

qui engendrent par pseudo-transitivité et augmentation la relation fonctionnelle :

$$A_{k+1}\, A_{k+2} \ldots A_n \rightarrow A_r$$

Nous allons en déduire qu'il y a une contradiction. En effet, deux cas sont à considérer.

(a) si F_r contient un élément $A_j, j \in \{1,2,\ldots,k\}$ la relation fonctionnelle $L_1\, L_2 \ldots L_{q_r} \rightarrow A_j$ est telle que le monôme :

$$\ell_1\, \ell_2 \ldots \ell_{q_r}\, a'_j \quad \text{est premier et contient}$$

$$a_{k+1}\, a_{k+2} \ldots a_n\, a'_j$$

ce qui est contraire à l'hypothèse.

(b) si F_r contient un élément A_j $j \in \{k+1,\ldots,n\}$
la relation fonctionnelle :

$$L_1 L_2 \ldots L_{q_r} \rightarrow A_j$$

implique que $A_{k+1} \ldots A_n$ n'est pas un index, donc contradiction.

En conséquence l'hypothèse formulée au départ est fausse et on peut affirmer qu'il existe au moins un monôme premier :

$$\ell = \ell_1 \ell_2 \ldots \ell_q a_i'$$

associé à la relation fonctionnelle

$$L_1 L_2 \ldots L_q \rightarrow A_i$$

tel que :

$$A_{k+1} A_{k+2} \ldots A_n \supset L_1 L_2 \ldots L_q$$

11.3.1.3. Définition de la notion de chaîne

Etant donné un terme de la forme $b = \prod_{i=1}^{k} b_i$ et une fonction booléenne f, on dira que b est <u>chaîné</u> à un monôme x de f, si :

- chaque b_i apparaît dans x
- ou s'il existe un produit de variable c_i tel que :
  - $c_i b_i'$ soit un monôme de f
  - c_i soit chaîné à x.

L'interprétation relationnelle de cette définition est qu'il existe une suite de relation fonctionnelle :

$$B \rightarrow F_1$$
$$E_1 \rightarrow F_2$$
$$\vdots$$
$$E_i \rightarrow F_{i+1}$$
$$F_{i+1} \rightarrow X$$

telle que par pseudo-transitivité on obtienne $B \rightarrow X$.

11.3.1.4. Conditions pour qu'une expression booléenne soit une décomposition

Nous considérerons qu'une expression booléenne de la forme :

$$g = \sum_{i=1}^{p} g_i$$

peut représenter la décomposition d'une collection de données dont la fonction booléenne est $h = \Pi a_i + f(a_1, a_2, \ldots, a_m)$. C'est-à-dire que l'on pourra écrire :

$$(A_1\ A_2\ \ldots\ A_n) = G_1 * G_2 * \ldots * G_p$$

si les conditions suivantes sont vérifiées :

(1) chaque g_i est contenu dans h,

(2) il existe au moins un terme g_i tel que g_i soit contenu dans l'élément représentant l'index,

(3) toutes les variables $a_1, a_2, \ldots, a_n$ apparaissent dans g,

(4) chaque variable a_i (i = 1,2,...,n) est chaînée à l'élément index dans h.

Exemple 11.1.

Etant donné quatre constituants A,B,C,D dont les relations fonctionnelles sont :

$$AB \rightarrow C$$

$$C \rightarrow D$$

Nous avons alors h = abcd + abc' + cd' avec comme seul élément index ab.

Les quatre fonctions g ci-dessous :

$$g = \begin{cases} ab + abc' + abd' \\ abc' + cd' \\ abc' + abd' \\ abc' + bcd' \end{cases}$$

satisfont les quatre conditions pour qu'une fonction booléenne puisse donner naissance à une décomposition. Les décompositions sont :

$$ABCD = \begin{cases} AB * ABC * ABD \\ ABC * CD \\ ABC * ABD \\ ABC * BCD \end{cases}$$

Parmi l'ensemble des décompositions, une, offre un intérêt particulier et correspond à la procédure de construction suivante :

- <u>étape 1</u> : déterminer un élément index de la fonction :

$$h = \Pi a_i + f$$

- <u>étape 2</u> : déterminer une couverture minimale de f, soit :

$$\sum_{1}^{p} g_i$$

- <u>étape 3</u> : si il existe un monôme g_i contenu dans l'élément index, on prendra :

$$g = \sum_{1}^{p} g_i$$

sinon :

$$g = (\text{élément index}) + \sum_{1}^{p} g_i$$

La fonction g ainsi construite satisfait les conditions (1) et (2) par construction. Elle satisfait aussi (3) puisque g_i est une couverture minimale. Quant à la condition (4) elle est une conséquence de la propriété 11.3., et que $\sum_{i} g_i$ est une couverture minimale.

On peut aussi montrer les propriétés suivantes qui ont l'intérêt de relier la décomposition fonctionnelle avec les formes d'une collection de données (chapitre 8).

<u>Proposition 11.4.</u>

Si la forme relationnelle f représente un ensemble de relations fonctionnelles associé à une collectionn de données en seconde ou troisième forme selon les définitions de CODD alors la fonction g de la forme :

$$g = \sum_{1}^{p} g_i$$

est suffisante pour établir une décomposition.

Par application de la proposition 11.3 si $A_{k+1}\ A_{k+2} \cdots A_n$ est un index, il existe un monôme premier de f, par exemple a a_i' tel que :

$$A_{k+1}\ A_{k+2} \cdots A_n \supset A$$

$$A_{k+1}\ A_{k+2} \cdots A_n \rightarrow A_i$$

$$A \rightarrow A_i$$

Comme la collection de données est en seconde ou troisième forme toutes les A_i sont en relation fonctionnelle élémentaire avec l'index, ceci n'est possible que si :

$$A = A_{k+1}\ A_{k+2} \cdots A_n$$

Dans ces conditions les termes de la décomposition seront :

$$A_1\ A_2 \cdots A_n = A * A\ A_i * \text{(autres termes)}$$

$$= A\ A_i * \text{(autres termes)}$$

Proposition 11.5.

Etant donné $g = \sum_{i=1}^{p} a_i\ c_i'$ une couverture minimale de la forme relationnelle f et les relations fonctionnelles élémentaires associées à g :

$$A_i \rightarrow C_i \qquad i = 1,2,\ldots, p$$

où A_i peut désigner un ensemble de constituants, tandis que C_i un constituant élémentaire.

Alors, il ne peut exister un indice $k \in \{1,2,\ldots,p\}$ et un ensemble de constituants $X = \{X_1, X_2,\ldots, X_q\}$ tels que les propriétés :

(1) $X_j \notin A_k$ pour chaque $j = 1,2,\ldots,q$

(2) $C_k \neq X_j$ pour chaque j

(3) $C_k \not\mapsto X_j$ pour chaque j

(4) $X \not\mapsto A$ pour tout $A \subset A_k$

(5) $A_k \rightarrow X \rightarrow C_k$

soient vraies.

Le fait qu'il ne peut exister un indice k et un ensemble X permet d'assurer que tous les éléments $(A_i\ C_i)$ de la décomposition donnent naissance à des collections de données en troisième forme. En effet, dans une collection de données en 3ème forme toutes les relations fonctionnelles élémentaires sont directes, c'est-à-dire que la propriété (5) ne peut exister.

Pour démontrer cette proposition nous allons faire l'hypothèse qu'il existe un tel indice k et un ensemble X. Sans nuire à la généralité de la proposition on peut considérer k = 1. Définissons alors :

$$\overline{g} = \sum_{i=2}^{p} g_i$$

Si nous faisons les hypothèses qui seront justifiées juste après que :

(a) $a_1\ x' \subset \overline{g}$

(b) $x\ c_1' \subset \overline{g}$

il vient :

$$\overline{g} = a_1x' + xc_1' + (\text{autres termes}) = a_1x' + xc_1' + a_1c_1' + (\text{autres termes})$$

$$\overline{g} = \overline{g} + a_1c_1' = g.$$

Ce qui n'est pas possible puisque g est une couverture minimale, et par conséquent il n'existe pas un indice k et un ensemble X.

Il reste bien entendu à justifier les hypothèses :

(a) $a_1x' \subset \overline{g}$

(b) $xc_1' \subset \overline{g}$

Démonstration de (a)

Puisque $a_1x' \subset g$ nous avons $(a_1x') \cdot g_1' \subset \overline{g}$

$(a_1x') \cdot (a_1' + c_1) \subset \overline{g}$

$a_1c_1x' \subset \overline{g}$

soit $\sum_{t=1}^{q} a_1c_1x_t' \subset \overline{g}$

Cette expression montre que pour tout t il existe un indice s = s(t), avec $s \neq 1$ tel que :

$$a_1 c_1 x'_t \subset g_s$$

ce qui fait que g_s ne peut avoir que les formes suivantes :

$$\left.\begin{array}{l} - \; c_1 x'_t \\ - \; z c_1 x'_t \\ - \; z x'_t \end{array}\right\} \quad \text{avec } a_1 \subset z, \quad c_1 \not\subset z$$

La première forme n'est pas possible conformément à la propriété (3).
Si g_s à la forme 2, il vient :

$$g_s + g_1 = a_1 c_1 x'_t + a_1 c'_1 = a_1 x'_t + a_1 c'_1$$

ce qui montre que $a_1 c_1 x'_t$ ne serait pas un monôme premier.

En conséquence g_s à la troisième forme, soit $g_s = a_1 x'_t$, c'est-à-dire que pour chaque t, nous avons :

$$a_1 x'_t \subset \bar{g}$$

ce qui donne bien :

$$a_1 x' \subset \bar{g}$$

Démonstration de (b)

Puisque $xc' \subset g$ nous avons :

$$(x c'_1) \cdot g'_1 \subset \bar{g}$$

$$(x c'_1) \cdot (a_1 c'_1) \subset \bar{g}$$

$$x a'_1 c'_1 \subset \bar{g}$$

Le constituant A_1 est en fait un ensemble de constituants, c'est-à-dire que a_1 est un produit de variables :

$$a_1 = \prod_r a_{1r}$$

$$x\, a'_1\, c'_1 = \sum_r x\, a'_{1r}\, c'_1$$

La condition $\sum_r x a'_{1r} c'_1 \subset \bar{g}$ implique que pour chaque r

$$x a'_{1r} c'_1 \subset \bar{g}$$

ce qui revient à dire qu'il existe un indice u = u(r), u ≠ 1, tel que

$$xa'_{1r}c'_1 \subset g_u$$

g_u ne peut avoir que deux formes :

$$\left.\begin{array}{l} - \; za'_{1r} \\ - \; zc'_1 \end{array}\right\} \text{ avec } x \subset z$$

La première forme n'est pas possible puisque $x \subset z$ est équivalente à $Z \subset X$ et comme $Z \rightarrow A_{1r}$ on aurait $X \rightarrow A_{1r}$, ce qui n'est pas possible selon la propriété (3).

Seule la deuxième forme est possible et nous avons :

$$xc'_1 \subset zc'_1 \subset g_u \subset \overline{g}$$

11.3.2. Application de la décomposition fonctionnelle

Pour illustrer le problème de la décomposition fonctionnelle, on peut reprendre l'exemple 9.3. du paragraphe 9.4. La figure 9.1., représente le processus de décomposition sous forme d'une arborescence. Au sommet figure l'espace de départ et chaque feuille représente un élément de la décomposition.

Dans cet exemple, le résidu de la décomposition est l'élément (P,D) qui correspond au monôme premier non-complémenté pd de la fonction booléenne.

Peut-on considérer l'élément (P,D) comme un élément significatif pour l'utilisateur ?

Les diplômes et l'animal sont reliés entre eux par l'intermédiaire de la variable "nom de personne" qui n'est pas une relation élémentaire puisqu'elle correspond à la projection sur l'espace P,D de la chaîne de composition :

$$\Pi_{P,D}(N,C * CP * ND).$$

Bien que d'un point de vue stockage de l'information cette décomposition soit parfaitement valable elle contient un élément non nécessairement souhaité par l'utilisateur.

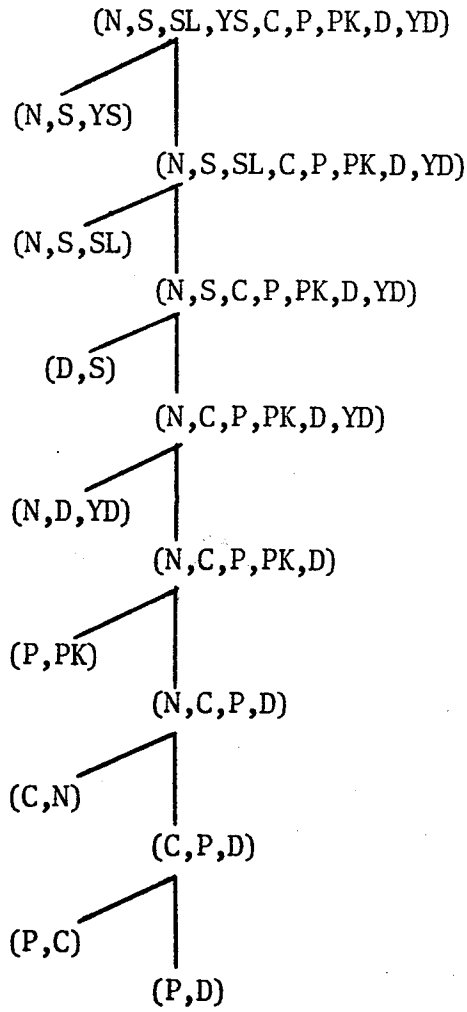


FIGURE 11.1. : DECOMPOSITION FONCTIONNELLE ARBORESCENTE

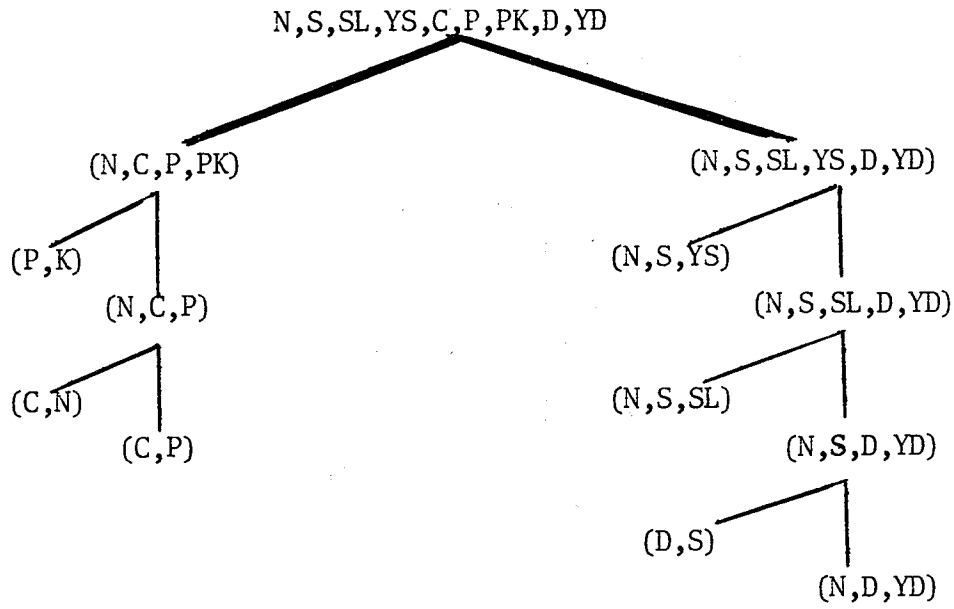


FIGURE 11.2. : DECOMPOSITION ARBORESCENTE PRIMAIRE

11.3.3. Décomposition primaire, décomposition secondaire

Une collection de données d'une décomposition sera primaire si l'utilisateur peut sans aucun calcul de composition ou de projection donner les entités associées à cette collection de données. Ceci revient à dire qu'une collection est primaire si elle enregistre des faits élémentaires du système d'informations.

Dans l'exemple précédent la collection de données définie sur l'ensemble des constituants (P,D) ne répond pas une telle définition. Par contre la collection de données construite sur (N,D) est primaire car elle correspond au fait élémentaire d'associer un diplôme à une personne.

A la notion de collection primaire nous associerons la notion de décomposition primaire si toutes les collections de la décomposition sont primaires. Dans la pratique, établir une décomposition primaire c'est se donner un système de départ ($\mathcal{F}'$, $\mathcal{R}'$) comme nous l'avons considéré au début de ce chapitre (paragraphe 11.2).

On pourra remarquer que si dans l'exemple du paragraphe 11.3.2., on fait l'hypothèse que :

(N,S,SL,YS,C,P,PK,Y,YD) = (N,C,P,PK) * (N,S,SL,YS,D,YD)

qui exprime que deux constituants apparaissant dans chaque opérande sont reliés indirectement par le constituant N et si aux parties (N,C,P,PK) et (N,S,SL,YS,D,YD) on applique le principe de la décomposition fonctionnelle on obtient une décomposition primaire. La figure 9.2. illustre cette propriété.

11.4. Méthodologie pour la détermination d'une décomposition primaire

Comme nous l'avons dit au paragraphe précédent se donner un système de départ $(\mathcal{F}', \mathcal{R}')$ c'est se donner une décomposition primaire. Il peut être intéressant d'aider l'utilisateur dans la recherche d'un tel système de départ. Nous proposons de développer une méthode qui peut le guider dans la recherche du système de départ. Nous utilisons volontairement le mot méthode et non celui d'algorithme, car cette méthode expérimentale repose sur des propriétés des relations fonctionnelles et de la décomposition ; il n'est pas possible de l'étayer sur un processus rigoureux.

Cette méthode utilise le graphe des relations fonctionnelles comme point de départ (voir paragraphe 7.3.). Plus précisément, comme nous avons vu qu'un ensemble de relations fonctionnelles n'est pas caractéristique de l'application nous utiliserons le graphe des relations fonctionnelles élémentaires de la couverture minimale.

Nous considérons toujours que $\mathcal{a}$ désigne l'ensemble des constituants. Donnons les définitions de constituant terminal et de graphe connexe.

Un constituant élémentaire A ε $\mathcal{a}$ sera <u>terminal</u> simple s'il n'existe qu'une seule relation fonctionnelle élémentaire telle que E → A et que A n'apparaît dans aucun membre de gauche des relations fonctionnelles élémentaires.

Nous dirons que (A,B) est une arête du graphe si les constituants du couple (A,B) apparaissent dans une relation fonctionnelle élémentaire. Une arête est non orientée.

Un graphe G sera connexe si pour tout couple de constituants élémentaires (A,B) il existe une chaîne (suite d'arêtes) de A à B. Si cette propriété n'est pas vérifiée nous dirons que le graphe n'est pas connexe.

Le principe de la méthode consiste à rechercher les constituants terminaux simples du graphe des relations fonctionnelles, chaque fois qu'il existe un constituant terminal on élimine du graphe le sommet terminal et les arêtes correspondant à la relation fonctionnelle E → A si A est le constituant terminal. Parallèlement on génère un élément de la décomposition qui est la collection (E,A).

Après cette étape on examine si le graphe est connexe, alors deux cas peuvent se présenter, le graphe est encore connexe et on répète la première étape. Si le graphe est déconnecté on divise le graphe en autant de graphes connexes que nécessaire et l'on recherche une collection de données qui établit le lien entre ces différentes composantes connexes. Ceci est justifié par le fait qu'il existe des liens d'une autre nature que fonctionnelle entre constituants et que seul l'utilisateur peut les déterminer. Par exemple, la figure 9.3. exprime que le graphe initial sur les constituants a est divisé en deux parties a_1 et a_2 déconnectés.

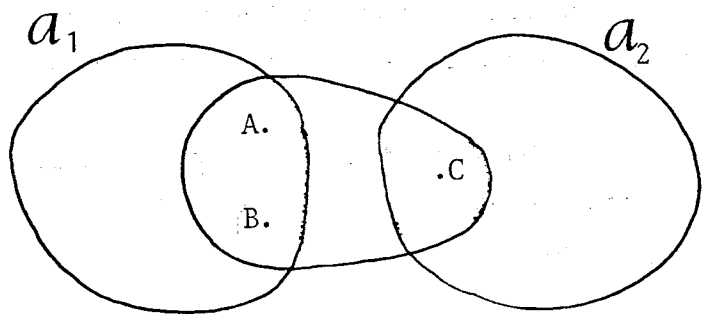


FIGURE 11.3. : ETABLISSEMENT D'UN LIEN ENTRE LES PARTIES a_1 ET a_2 DECONNECTES

Arrivé à ce stade l'utilisateur indique, par exemple, qu'il existe une collection de données définie sur (A,B,C) qui établit le lien entre a_1 et a_2 c'est-à-dire que l'on peut écrire :

$$a_1 . a_2 = a_1 * (A,B,C) * a_2$$

Après on recommence le processus successivement sur a_1 et a_2. Ce processus s'arrête dans deux cas, soit que la partie a_1 est réduite à un seul élément, soit a_1 est connexe et il n'existe plus de constituants terminaux.

Cette dernière situation correspond au cas où les constituants sont fortement reliés par des relations fonctionnelles et on peut alors appliquer la théorie de la décomposition fonctionnelle pour obtenir le résultat.

La figure 11.4. donne l'organigramme de la méthode. Dans l'organigramme nous utilisons deux piles l'une dénommée PILE, l'autre PILE-EN-COURS telles que chaque élément de la pile contient un ensemble de constituants. Au départ PILE(1) est composé de l'ensemble de tous les constituants $\{A_1, A_2, \ldots, A_n\}$. A chaque élément de la pile on associe le graphe correspondant.

La figure 11.5. illustre la méthode sur l'exemple 9.3. du paragraphe 9.4. Dans cette figure nous avons fait figurer d'une part le graphe des relations fonctionnelles à chaque étape et d'autre part les collections de données générées à chaque étape.

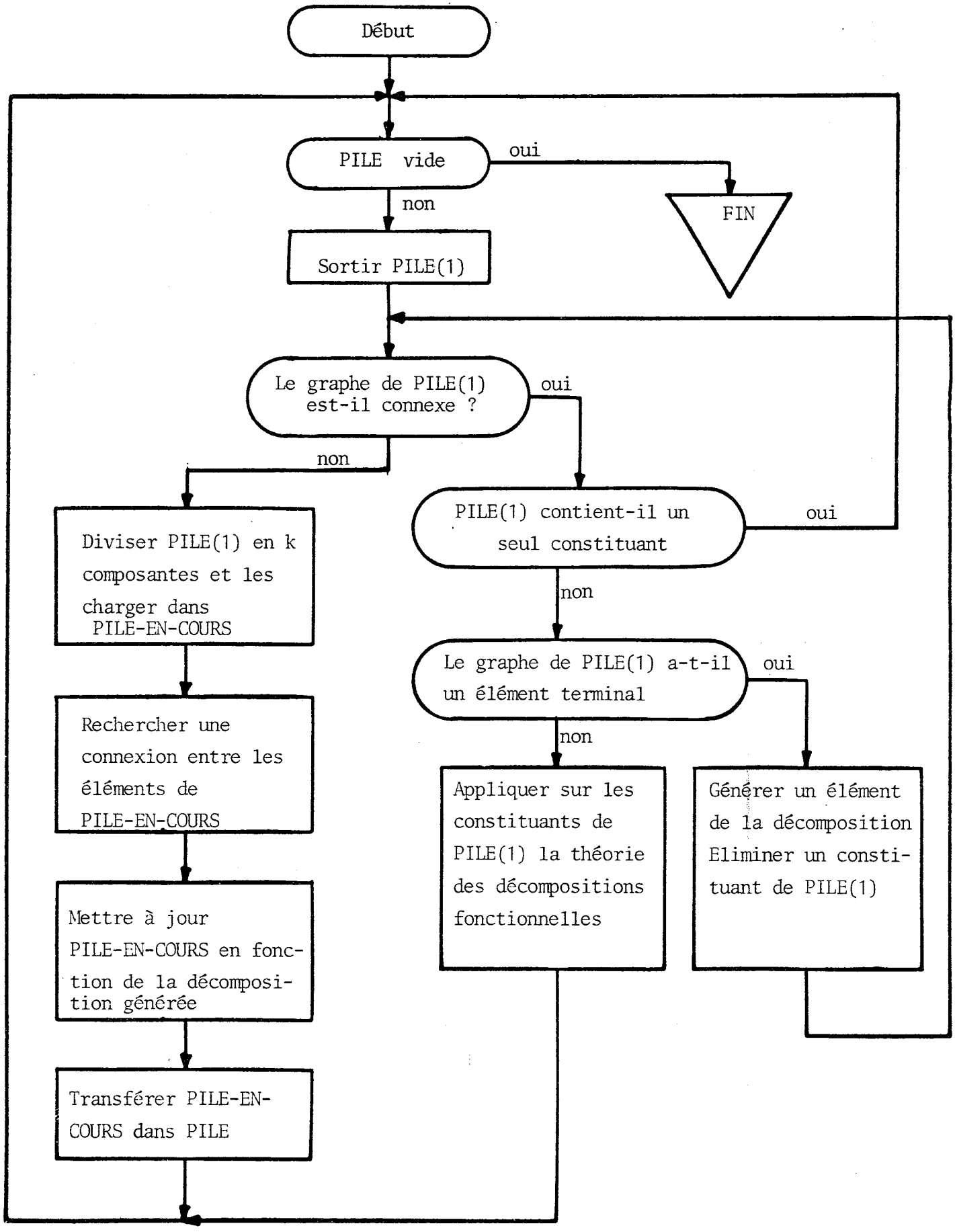


FIGURE 11.4 : Organigramme de la méthode

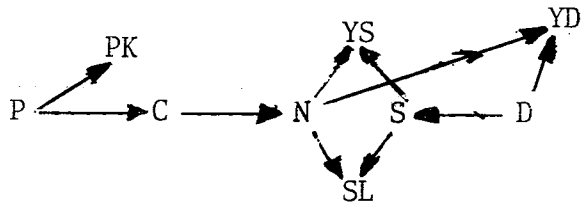
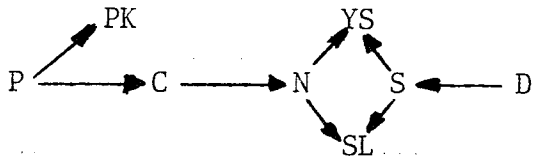
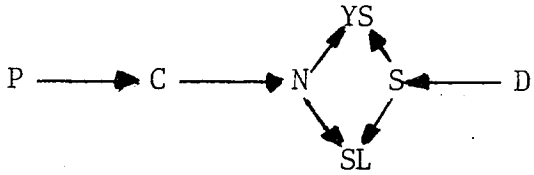
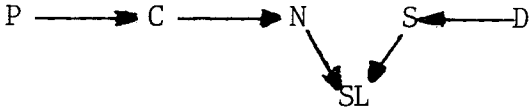
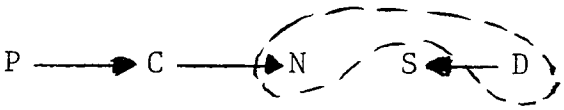
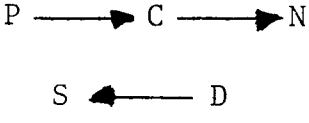
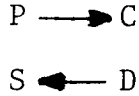
	GRAPHE	Collection générée à chaque étape
0		
1		(N,D,YD)
2		(P,PK)
3		(N,S,YS)
4		(N,S,SL)
5		A cette étape le graphe est déconnecté et c'est la collection (N,D) qui est généré mais elle est absorbée par la collection (N,D,YD)
6		(C,N)
7		(P,C) (S,D)

FIGURE 11.5.: ILLUSTRATION DE LA METHODE DE RECHERCHE

11.5. Algèbre des décompositions

Le paragraphe 11.2. a été consacré à l'étude de la décomposition fonctionnelle. Dans ce paragraphe nous allons montrer que si nous connaissons une décomposition il est possible d'en générer de nouvelles. Les premières recherches mathématiques concernant une algèbre desdécompositions ont été faites par J. BOITTIEAUX, malheureusement les résultats sont peu connus et imprécis. Nous rappellerons les propriétés élémentaires et nous donnerons les conditions d'utilisation de chaque proposition. De plus, nous énoncerons de nouvelles règles de génération, ce qui devrait permettre de donner un algorithme général de décomposition.

11.5.1. Notation et définition

Nous désignerons par P une collection de données définie sur l'ensemble des constituants $\mathcal{A}$.

Si il est possible d'écrire : P = [X]P ✱ [Y]P
nous noterons symboliquement :
(X,Y) = X ✱ Y
où X et Y sont des constituants composés.

Définition : Une décomposition sera arborescente si il existe une partie X de $\mathcal{A}$ tel que :
(X,Y) ✱ (X,Z) ✱ (X,U) = (X,Y,Z,U)
où Y,Z,U sont disjoints deux à deux et disjointes de X. Dans un tel cas on notera : X : Y | Z | U

Nous avons utilisé le mot de décomposition arborescente parce que la collection de données associée à cette décomposition peut être représentée par une organisation hiérarchisée comme l'indique la **figure** 11.6.

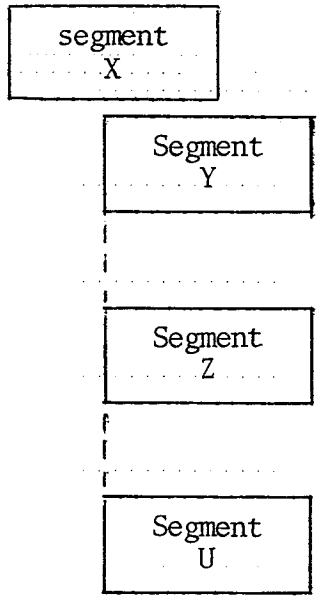


FIGURE 11.6. : ORGANISATION HIERACHISEE DE DONNEES

Ce type d'organisation est utilisée dans de nombreux systèmes de base de données. A ce sujet le lecteur pourra se reporter à la référence [9]. Ce qui est important,c'est l'analogie qui existe entre un type d'organisation possible et une expression de nature algébrique.Une définitio plus formelle de la notion d'organisation hiérarchisée sera donnée (§12.2.1

D'autre part nous dirons que X constitue la racine et Y,Z,U des branches.

11.5.2. Propriétés élémentaires

11.5.2.1. Groupement d'éléments

Proposition 11.6.

Si (X,Y,Z) = X * Y * Z alors si par exemple X et Y sont deux éléments choisis parmi les éléments X,Y,Z alors (X,Y,Z) = (X,Y) * Z

Une des propriétés fondamentales de la composition normale est que :

$X * Y \supset (X,Y)$ ce qui implique

$X * Y * Z \supset (X,Y) * Z$ soit $(X,Y,Z) \supset (X,Y) * Z$

mais on a aussi :

$(X,Y) * Z \quad X,Y,Z$

ce qui donne bien :

$(X,Y,Z) = (X,Y) * Z$

11.5.2.2. Addition d'éléments

Proposition 11.7.

Si $(X,Y,Z) = X * Y * Z$ et $T \subset (X,Y,Z)$ alors :

$X * Y * Z * T = (X,Y,Z)$

Puisque $T \subset (X,Y,Z)$ d'après la propriété d'absorption de l'opérateur de composition normale on peut écrire :

$T * (X,Y,Z) = (X,Y,Z)$

et en remplaçant X,Y,Z par son expression, on obtient :

$T * X * Y * Z = (X,Y,Z)$

11.5.2.3. Suppression d'éléments

Proposition 11.8

Si $(X,Y,Z) = X * Y * Z * T$ avec $T \subset (X,Y)$ alors

$(X,Y,Z) = (X,Y) * Z$

D'après la propriété d'absorption on peut écrire :

$T * (X,Y) = (X,Y)$

puis par composition :

$T * (X,Y) * Z = (X,Y) * Z$

Si on applique la proposition 11.6 à (X,Y,Z) = X * Y * Z * T on obtient

(X,Y,Z) = (X,Y) * Z * T

d'où par rapprochement :

(X,Y) * Z = (X,Y,Z)

11.5.2.4. Projection d'un élément

Proposition 11.9.

Si X : Y | Z est une décomposition arborescente et T tel que T ⊂ Y avec T ≠ ∅ alors X : T | Z est aussi une décomposition arborescente.

La décomposition arborescente X : Y | Z se traduit par l'écriture :

(Eq. 11.1.) (X,Y,Z) = (X,Y) * (X,Z)

Conformément à la proposition 3.2 paragraphe 3.6.4., l'équation 11.1., donne :

(<X:x>,<Y:y>,Z) = (<X:x>,Z)

mais puisque T ⊂ Y

(<X:x>,<Y:y>,Z) = (<X:x>,<T:t>,Z)

soit par rapprochement :

(<X:x>,Z) = (<X:x>,<T:t>,Z)

ce qui est équivalent à :

(X,Z,T) = (X,Z) * (X,T) ou X : T | Z

La proposition 11.9. permet de projeter une branche de la décomposition arborescente. La proposition que nous allons établir maintenant permet de projeter la racine elle-même sous certaines conditions.

Proposition 11.10.

Si X,Y,Z : A | B est une décomposition arborescente telle que l'équation 11.2 soit vérifiée :

(Eq. 11.2.) (<X:x>,<Y:y>,<Z:z>,A,B) = (<X:x>,<Y:y>, A,B)

alors X,Y : Z | A | B est une décomposition arborescente.

L'équation 11.2. se traduit par :

$$(X,Y,Z) * (X,Y,A,B) = (X,Y,Z,A,B)$$

ou :

(Eq.11.3.) X,Y : Z | A,B

Par application de la proposition 11.9. on peut projeter la branche A,B successivement sur A, et sur B, ce qui permet de générer les équations :

(Eq. 11.4.) $(X,Y,Z,A) = (X,Y,Z) * (X,Y,A)$

(Eq. 11.5.) $(X,Y,Z,B) = (X,Y,Z) * (X,Y,B)$.

Comme X,Y,Z : A | B se traduit par $(X,Y,Z,A,B) = (X,Y,Z,A) * (X,Y,Z,B)$ il vient en utilisant les équations 11.4 et 11.5. :

(Eq.11.6.) $(X,Y,Z,A,B) = (X,Y,Z) * (X,Y,A) * (X,Y,Z) * (X,Y,B)$

$= (X,Y,Z) * (X,Y,A) * (X,Y,B)$

ce qui est bien la décomposition arborescente :

X,Y : Z | A | B

Proposition 11.11.

Cette proposition est un corollaire de la proposition 11.10.

Si X,Y : A | B est une décomposition arborescente telle que $X \rightarrow Y$ alors X : A | B est aussi une décomposition arborescente.

En effet, si $X \rightarrow Y$ alors (<X:x>,<Y:y>, A,B) = (<X:x>,A,B) ce qui permet d'appliquer la proposition 11.10.

11.5.3. Génération de décompositions

11.5.3.1. Alourdissement de la racine

Proposition 11.12.

Si l'expression 11.7. est une décomposition arborescente :

(Eq.11.7.) $X : (A,B) \mid (C,D)$

alors si B et D sont deux constituants choisis, parmi A,B,C,D l'expression :

(Eq.11.8.) $(X,B,D) : A \mid C$

est une décomposition arborescente.

Le constituant composé (X,B,D) est contenu dans (X,A,B,C,D) ce qui permet d'écrire :

(Eq.11.9.) $(X,B,D) * (X,A,B,C,D) = (X,A,B,C,D)$

et en utilisant l'équation 11.7. on déduit :

(Eq.11.10.) $(X,B,D) * (X,A,B) * (X,C,D) = (X,A,B,C,D)$

ou bien encore :

$$(X,B,D) * (X,A,B) * (X,B,D) * (X,C,D) = (X,A,B,C,D)$$

et par application de la proposition 11.6. :

(Eq.11.11.) $(X,A,B,D) * (X,B,C,D) = (X,A,B,C,D)$

L'équation 11.11. est bien équivalente à l'équation 11.8.

11.5.3.2. Décomposition d'une branche

Proposition 11.13.

Etant donné deux décompositions arborescentes :

(Eq.11.12.) $X : A \mid B$

(Eq.11.13.) $X : Y \mid Z \mid T$

telles que $(Y,Z,T) \subset A$ alors :

(Eq.11.14.) $X : Y \mid Z \mid T \mid B$ est une décomposition arborescente.

Comme (Y,Z,T) ⊂ A il est possible de projeter les branches A sur Y,Z,T ce qui donne :

(Eq.11.15.) X : (Y,Z,T) | B ou (X,Y,Z,T,B) = (X,Y,Z,T) ✱ (X,B)

et par substitution de (X,Y,Z,T)

(X,Y,Z,T,B) = (X,Y) ✱ (X,Z) ✱ (X,T) ✱ (X,B)

qui représente bien l'équation 11.13.

11.5.3.3. Principe général de génération

Si on considère une décomposition arborescente de la forme :

(Eq.11.16.) $X : X_1 \mid X_2 \mid \ldots \mid X_k$

où X, X_1, X_2,..., X_k sont des constituants composés disjoints, alors toute collection de données construite sur les constituants : $(X,A_1,A_2,\ldots,A_k)$ où :

$$A_i \subset X_i \qquad i = 1,2,\ldots,k$$

est un élément de plusieurs décompositions possibles.

En effet, comme $A_1 \subset X_1$ on peut par application de la proposition 11.9. écrire :

(Eq.11.17.) $X : A_1 \mid X_2 \mid \ldots \mid X_k$

ensuite on peut alourdir la racine X en écrivant (proposition 11.2.)

(Eq.11.18) $X, A_2,\ldots,A_k : A_1 \mid X_2 - A_2 \cdot X_2 \mid \ldots \mid X_k - A_k \cdot X_k$

ce qui donne :

$$(X,A_1,X_2,\ldots,X_k) = (X,A_1,A_2,\ldots,A_k) \text{ ✱ } (X_1,A_2,\ldots,A_k,X_2-A_2.X_2) \text{ ✱ } \ldots \text{ ✱ } (X,A_2,\ldots,A_k,\ X_k-A_k.X_k)$$

qui montre que $(X,A_1,A_2,\ldots,A_k)$ est bien un élément d'une décomposition. En fait on a choisi au départ de projeter X_1 sur A_1 on aurait pu aussi bien le faire sur $A_2,\ldots,A_k$. Ceci signifie qu'en fait l'élément $(X,A_1,A_2,\ldots,A_k)$ est l'élément d'au moins K-décompositions distinctes.

On aurait pu aussi procéder de la façon suivante :

- on projette les branches X_1, X_2 sur A_1, A_2 soit :

(Eq.11.19.) $$X : A_1 \mid A_2 \mid X_3 \mid \ldots \mid X_k$$

- on alourdit la racine :

(Eq.11.20.) $$X, A_3, \ldots, A_k : A_1 \mid A_2 \mid X_3 - A_3 . X_3 \mid \ldots \mid X_k - A_k . X_k$$

ce qui donne après regroupement :

$$(X, A_1, A_2, X_3, \ldots, X_k) = (X, A_1, A_2, A_3, \ldots, A_k) * \ldots * (X, A_3, \ldots, A_k, X_k - A_k . Y_k)$$

En poursuivant ce raisonnement il est aisé de remarquer qu'au total, l'élément $(X, A_1, A_2, \ldots, A_k)$ est un élément pouvant intervenir dans $2^k - 1$ décompositions.

On peut alors remarquer que si l'élément $(X, A_1, A_2, \ldots, A_k)$ est lui-même décomposable, on génère alors $2^k - 1$ nouvelles décompositions.

Ainsi à partir d'une décomposition exprimée par l'équation 11.16 et d'une autre décomposition portant sur l'élément $(X, A_1, A_2, \ldots, A_k)$ on peut générer de nouvelles décompositions : ceci constitue le principe général de décomposition.

Si une décomposition est mise sous la forme de l'équation 11.16., il y a $2^{n_1+n_2+\ldots n_k} - 1 = 2^n - 1$ expressions possibles de la forme $(X_1, A_1, \ldots, A_k)$ ou $n_1, n_2, \ldots, n_k$ sont les nombres de constituants des branches $X_1, X_2, \ldots, X_k$. Plus ce nombre sera grand, plus il existera de chance pour obtenir des décompositions nouvelles. Pour augmenter ces chances il est indispensable d'utiliser la proposition 11.10. d'allègement de la racine puisque si on peut écrire :

(Eq.11.21.) $$Y : Z \mid X_1 \mid X_2 \mid \ldots \mid X_k \quad \text{avec } (Y, Z) = X$$

le nombre possible devient $2^{n+n_z} - 1$ si n_z désigne le nombre de constituants de Z.

La proposition 11.15 que nous verrons ultérieurement répond aussi à ce souci.

Si ce que nous venons d'exposer est le principe général de la décomposition, il est possible aussi d'utiliser d'autres règles lorsque certaines conditions sont vérifiées, sans utiliser nécessairement le principe général.

11.5.3.4. Décomposition particulière

Proposition 11.14.

$$X_1 : Y_1 \mid Z_1 \mid T_1 \mid U_1 \qquad \text{(Eq.11.22.)}$$

$$X_2 : Y_2 \mid Z_2 \mid T_2 \mid U_2 \qquad \text{(Eq.11.23.)}$$

sont deux décompositions arborescentes telles que :

si $X_1.X_2 = \emptyset$ alors $Y_1,Z_1,T_1,U_1 \supset X_2$ (Eq.11.24.)

$Y_2,Z_2,T_2,U_2 \supset X_1$ (Eq.11.25.)

ou

si $X_1.X_2 \neq \emptyset$ alors $Y_1,Z_1,T_1,U_1 \supset X_2.X_1'$ (Eq.11.26.)

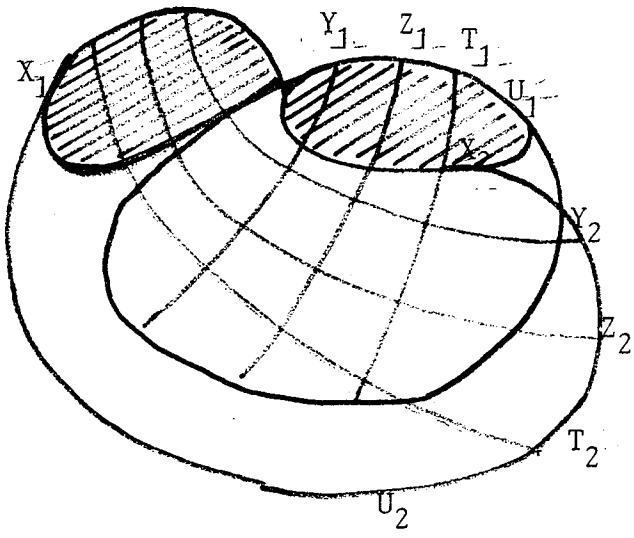
$Y_2,Z_2,T_2,U_2 \supset X_1.X_2'$ (Eq.11.27.)

où X_1' et X_2' désignent le complément de X_1 et de X_2 par rapport à a, alors si Y_1 et Z_1 sont deux branches choisies parmi toutes les branches de la première décompositions alors :

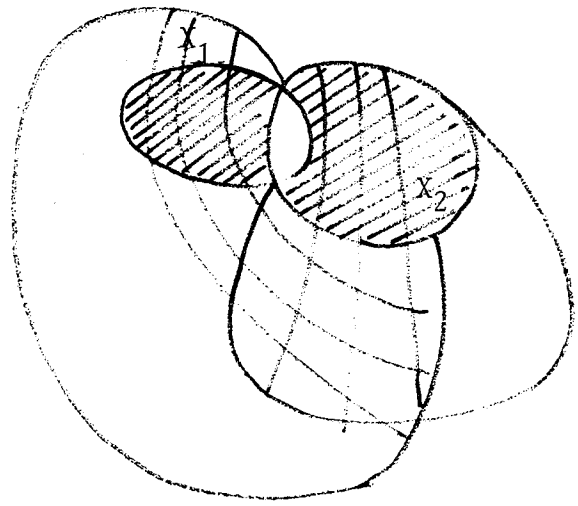
$$\text{(Eq.11.28.)}\quad X_1,(Y_1,Z_1).X_2 : Y_1.Y_2 \mid Y_1.Z_2 \mid Y_1.T_2 \mid Y_1.U_2 \mid Z_1.Y_2 \mid$$
$$\mid Z_1.Y_2 \mid Z_1.Z_2 \mid Z_1.T_2 \mid Z_1.U_2 \mid T_1 \mid U_1$$

est une décomposition arborescente.

Les conditions exprimées par les équations 11.24. et 11.25 et 11.26. et 11.27. correspondent aux deux cas représentés par les figures ci-dessous.



conditions 11.24. et 11.25.



conditions 11.26. et 11.27.

FIGURE 9.7. : REPRESENTATION DES CONDITIONS DE DECOMPOSITION

Par alourdissement de la racine de l'équation 11.22., on peut écrire :

(Eq.11.29.) $X_1,(Y_1,Z_1).X_2 = Y_1\text{-}Y_1.X_2 \mid Z_1\text{-}Z_1.X_2 \mid T_1 \mid U_1$

par allègement de toutes les branches de l'équation 11.29.

(Eq.11.30.) $X_1,(Y_1,Z_1).X_2 = Y_1.Y_2,Y_1.Z_2,Y_1.T_2,Y_1.U_2 \mid$

$\mid Z_1.Y_2,Z_1.Z_2,Z_1.T_2,Z_1.U_2 \mid$

$\mid T_1.X_2 \mid U_1.X_2$

par alourdissement de 11.23., on obtient :

(Eq.11.31.) $X_1,X_2 : Y_2\text{-}Y_2.X_1 \mid Z_2\text{-}Z_2.X_1 \mid T_2\text{-}T_2.X_1 \mid U_2\text{-}U_2.X_1$

par allègement de 11.31. on peut obtenir les deux équations 11.32. et 11.33.

(Eq.11.32.) $X_1,X_2 : Y_1.Y_2 \mid Y_1.Z_2 \mid Y_1.T_2 \mid Y_1.U_2$

(Eq.11.33.) $X_1,X_2 : Z_1.Y_2 \mid Z_1.Z_2 \mid Z_1.T_2 \mid Z_1.U_2$

et de 11.30 les deux équations 11.34. et 11.35.

(Eq.11.34.) $X_1,(Y_1,Z_1).X_2 = Y_1.Y_2,Y_1.Z_2,Y_1.T_2,Y_1.U_2 \mid T_1.X_2 \mid U_1.X_2$

(Eq.11.35.) $X_1,(Y_1,Z_1).X_2 = Z_1.Y_2,Z_1.Z_2,Z_1.T_2,Z_1.U_2 \mid T_1.X_2 \mid U_1.X_2$

ce qui est équivalent à :

$$(x_1,(y_1,z_1)x_2,Y_1Y_2,Y_1Z_2,Y_1T_2,Y_1U_2) = (x_1,(y_1,z_1,t_1,u_1)x_2,Y_1Y_2,\ldots,Y_1U_2)$$
$$= (x_1,x_2,Y_1Y_2,\ldots,Y_1U_2)$$

et par application de la propriété de la projection de la racine (proposition 11.10.) on peut écrire :

(Eq.11.36.) $X_1,(Y_1,Z_1)X_2 = Y_1.Y_2 \mid Y_1.Z_2 \mid Y_1.T_2 \mid Y_1.U_2$

de même :

(Eq.11.37.) $X_1,(Y_1,Z_1)X_2 = Z_1.Y_2 \mid Z_1.Z_2 \mid Z_1.T_2 \mid Z_1.U_2$

on peut alors dans l'équation 11.29. décomposer les deux premières branches ce qui donne bien l'expression 11.28. annoncée au début.

Exemples d'application

$$\left.\begin{array}{l} AB : GC \mid EHID \\ BCD : AIG \mid FH \end{array}\right\} \Rightarrow ABCD : F \mid G \mid H$$

11.5.4. Décomposition et relations fonctionnelles

Proposition 11.15.

Si X : Y | Z est une décomposition arborescente alors :

$\overline{X} : \overline{X+Y}-\overline{X} \mid \overline{X+Z}-\overline{Z}$ est aussi une décomposition, la notation $\overline{X}$ exprime la borne supérieure de la classe d'équivalence de X.

$X : Y \mid Z \Leftrightarrow (<X:x>,Y,Z) = (<X:x>,Y) \times (<X:x>,Z)$

mais comme $X \leftrightarrow \overline{X}$ est équivalent à $x \leftrightarrow \overline{x}$ et on peut écrire :

$$(<X:x>,Y,Z) = (<\overline{X}:\overline{x}>,Y,Z) = (<\overline{X}:\overline{x}>,Y) \times (<\overline{X}:\overline{x}>,Z) \Leftrightarrow \overline{X} : Y \mid Z$$

de même :

$$\overline{X} : Y \mid Z \Leftrightarrow [<X:\overline{x}>,Z] = [<X:\overline{x}>,<Y:y>,Z]$$

mais $\overline{X},Y \rightarrow \overline{X+Y}$ ou bien encore $\overline{x},y \rightarrow \overline{x,y}$ soit

$$[<\overline{X}:\overline{x}>,Z] = [<\overline{X,Y}>:<\overline{x,y}>,Z] \iff \overline{X} : \overline{X+Y} - \overline{X}\ Z$$

finalement on obtiendrait :

$$\overline{X} : \overline{X+Y} - \overline{X} \mid \overline{X+Z}-\overline{X}$$

Dans l'état actuel il est difficle de pouvoir affirmer que l'ensemble des règles données dans ce chapitre permettent de générer toutes les décompositions d'une collection de données. D'un point de vue théorique il manque des outils pour déterminer si un ensemble de constituants est un élément d'une décomposition, sans avoir à se référer à la liste des décompositions.

CHAPITRE 12

APPLICATION DE LA NOTION DE DECOMPOSITION

La notion de décomposition est un concept utile dans l'étude de la structure des systèmes d'informations. Ce chapitre a pour but d'illustrer l'utilisation des notions de décomposition primaire et secondaire à la conception d'une base de données où la représentation de l'utilisateur est séparée de la représentation physique des données. Le passage du domaine utilisateur au domaine physique des données se faisant à l'aide de règles de conversion.

Nous montrons comment les opérations classiques d'insertion, suppression, mise-à-jour, requête peuvent fonctionner dans un tel environnement.

Une autre application de la notion de décomposition qui est aussi étroitement liée à la représentation physique des données est celle de l'étude des organisations hiérarchisées.

12. APPLICATION DE LA NOTION DE DECOMPOSITION

12.1. Application de la notion de décomposition au problème de l'indépendance dans une base de données

12.1.1. Définition du problème

Le concept "d'indépendance" dans une base de données est des plus difficiles à mettre en oeuvre dans la réalisation d'un système de base de données. En quoi consiste-t-il ?

Une base de données est un système en perpétuelle évolution, l'utilisateur met à jour les données, mais peut aussi vouloir introduire de nouvelles relations, de nouveaux constituants. D'un autre côté les ingénieurs systèmes chargés de la maintenance peuvent désirer pour des raisons de performances, modifier des séquences de tri, insérer de nouveaux points d'entrée, établir des nouveaux liens physiques entre fichiers. Il en résulte que, par la volonté de l'utilisateur ou des techniciens, on fasse subir des modifications au système de la base de données. Sous ces conditions, est-il possible qu'un programme (ou une requête) écrit par l'utilisateur avant toutes modifications puisse continuer à s'exécuter normalement après la modification? Si une telle condition est vérifiée on dira qu'il y a indépendance entre la représentation logique de l'utilisateur et la représentation physique.

Nous voulons donner quelques indications sur ce problèmes à partir de la notion de décomposition. En effet un ensemble de collection de données apparaît comme capable de représenter la conception "informative de l'utilisateur de n'importe quelle organisation actuelle des systèmes d'informations.

Nous avons défini le concept de décomposition primaire qui correspond à la perception de l'utilisateur. En effet, il peut pour chacune des collections associées à la décomposition primaire saisir les valeurs correspondantes à chaque entité. D'un autre côté nous avons montré que pour un même ensemble de constituants il y avait plusieurs décompositions dites secondaires. Nous ferons l'hypothèse qu'une telle décomposition secondaire sera le point de départ de toute organisation physique des données. Cette hypothèse est fondée sur le fait que l'organisation physique ne doit pas être construite à partir de la perception de l'utilisateur mais pour des raisons d'efficacité à partir des requêtes que doit exécuter le système de la base de données.

En effet, si une requête de l'utilisateur fait référence à des constituants attachés à différentes collections de données, ceci entrainera une recherche à travers l'ensemble de ces collections qui prendra plus de temps qu'une requête ne travaillant que sur une seule collection de données.

Nous voulons affirmer que parmi l'ensemble des décompositions secondaires il serait souhaitable de choisir celles qui correspond à un critère d'optimisation (temps, espace). Ce problème d'optimisation n'est pas résolu, nous allons simplement montrer comment il serait possible d'assurer l'indépendance et dans une deuxième étape quelles pourraient être les organisations physiques qui seraient construites à partir d'une décomposition primaire. Donc par conséquent de comparer deux structures entre elles.

12.1.2. Illustration du problème à partir d'un exemple

Nous partirons d'un exemple pour illustrer ce concept d'indépendance. L'exemple est tiré du problème d'un emploi du temps pour examens.

Les constituants auront la signification suivante :

S : salle d'examen

P : professeur chargé d'un examen

X : examen

N : note d'un élève pour un examen

H : heure de l'examen

E : étudiant

R : surveillants.

Cet ensemble de constituants est structuré par des relations fonctionnelles élémentaires, et par une décomposition arborescente .

$\ell_1 = X \rightarrow H$

$\ell_2 : H \rightarrow S$

$\ell_3 : H,S \rightarrow X$

$\ell_4 : H,R \rightarrow S$

$\ell_5 : E,X \rightarrow N$

$\ell_6 : H,E \rightarrow X$

X : E/P/R exprime qu'à un examen est associé un ensemble d'étudiants, de professeurs et de surveillants.

De cet ensemble de relations fonctionnelles élémentaires on pourra dériver :

$\ell_7 : H,E \rightarrow S$

$\ell_8 : H,R \rightarrow X$

La décomposition arborescente X : E/P/R va fournir la décomposition primaire de la façon suivante :

- en prenant les bornes supérieures, on obtient :

X,H,S : E,N/P/R

soit : (X,H,S,E,N,P,R) = (X,H,S,E,N) ✱ (X,H,S,P) ✱ (X,H,S,R)

= (X,H,S) ✱ (X,E,N) ✱ (X,H,S) X (X,P) ✱ (X,H,S) ✱ (X,F

= (X,H,S) ✱ (X,E,N) ✱ (X,P) ✱ (X,R)

En partant de la même décomposition on aurait obtenu la décomposition secondaire suivante :

$$(X,H,S,E,N,P,R) = (X,H,S,E,N) * (X,H,S,P) * (X,H,S,R)$$
$$= (X,H,X) * (X,E,N) * (X,H,S) * (X,P) * (H,S,X) * (H,S,R)$$
parce que $H,S \rightarrow X$
$$= (X,H,S) * (X,E,N) * (X,P) * (H,S,R)$$

Cette décomposition est bien secondaire car elle contient la collection (H,S,R) qui n'est pas une collection primaire.

Nous allons représenter ces deux décompositions par le schéma de la figure 12.1 qui fait apparaître d'une part les relations fonctionnelles qui lient les constituants dans une collection et d'autre part la façon dont on peut passer de la première décomposition à la seconde.

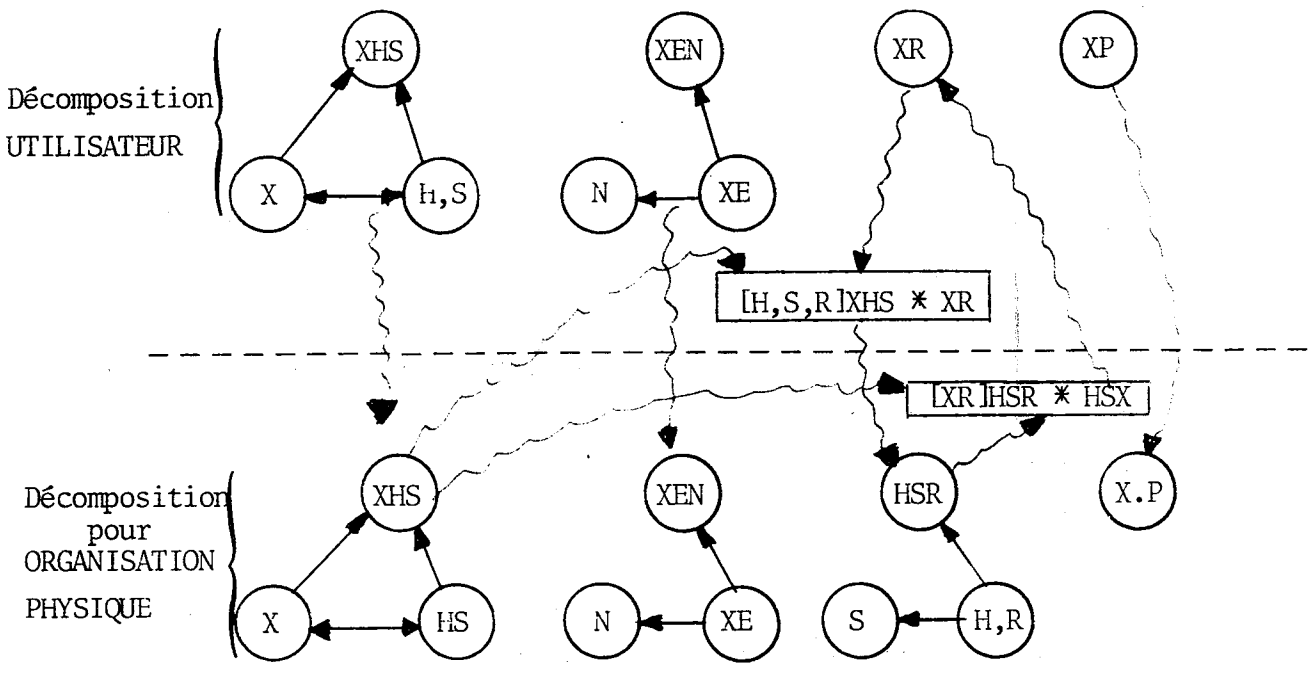


FIGURE 12.1. : Décompositionsprimaire et secondaire

Ce schéma montre que si l'utilisateur transmet une entité selon son découpage elle devra se transmettre soit directement, soit avec traitement vers l'espace de stockage. En particulier, nous avons indiqué que la collection HSR s'obtient par composition puis projection, ce qui peut s'exprimer par :

$$(H,S,R) = [H,S,R]\ (X,H,S) \ast (X,R)$$

Dans une telle conception il y a une séparation totale entre le niveau de l'utilisateur et celui de l'organisation des données. Pour qu'un tel système puisse fonctionner il est indispensable de connaître les règles qui permettent de passer l'un vers l'autre et vice-versa.

Dans le sens :

UTILISATEUR → ORGANISATION PHYSIQUE $(H,S,R) = [H,S,R]\ (X,H,S) \ast (X,R)$

et dans le sens inverse :

ORGANISATION PHYSIQUE → UTILISATEUR $(X,R) = [X,R]\ (H,S,R) \ast (H,S,X)$

Ces règles sont parfaitement définies lorsque les deux décompositions sont connues, ceci veut dire que le système peut être construit de telle sorte qu'il les déduise lui-même sans qu'elles lui soient fournies.

12.1.3. Fonctionnement du système : décomposition primaire décomposition secondaire.

Nous allons montrer comment le système peut fonctionner lorsque l'utilisateur effectue des opérations sur la base de données. Pour l'utilisateur toutes les opérations sont exprimées dans son espace, c'est-à-dire qu'elles font appel aux noms des collections de la décomposition primaire Examinons successivement les opérations qui peuvent être effectuées et mesurons en les conséquences.

12.1.3.1. Opération d'insertion

L'insertion d'une entité appartenant aux collections XHS, XEN, XP ne présente aucune difficulté puisque les mêmes collections sont conservées dans la décomposition secondaire.

Que se passe-t-il si on introduit une entité (<X:x>,<R:r>) dans la collection (X,R) ?

L'opération (<X:x>,<R:r>) * [X,H,S] définit une seule nouvelle entité (<X:x>,<R:r> , <H;h>,<S:s>) qui par projection donne (<R:r>,<H:h>,<S:s>)qui est introduite dans la collection (H,S,R).

12.1.3.2. Opération de suppression

(a) Suppression d'une entité (<X:x>, <R:r>). On applique le même processus que pour l'opération d'insertion en passant par la règle de conversion qui fournit l'entité (<H:h>,<S:s>,<R:r>) à supprimer, comme pour cette entité on connaît l'index (H,R) (d'après les relations fonctionnelles) on accède à l'entité que l'on supprime.

(b) Suppression de toutes les entités (<X:x>,<R:r>) associées au même surveillant r l'opération [H,S,R][<R:r>,X] (X,R) * XHS donne la liste des entités de la forme (<H:h>, <S:s>, <R:r>) à supprimer.

On pourra remarquer en passant que la suppression d'un examen entraîne la nécessité de supprimer l'information dans toutes les collections contenant le constituant X.

12.1.3.3. Mise à jour de valeur

Une telle opération peut toujours se ramener à une opération de suppression puis d'insertion, donc par conséquent on se réfère aux cas déjà étudiés.

12.1.3.4. Requête

Le cas des requêtes est très intéressant. En effet, l'utilisateur exprime sa requête en utilisant les noms de collection de son espace de définition. Comme nous l'avons vu une telle collection n'existe pas nécessairement au niveau de l'organisation physique. En conséquence il est nécessaire d'interpréter son programme à l'aide de règles pour le rendre exécutable.

Donnons quelques exemples :

(1) Supposons que l'utilisateur désire connaître, l'emploi du temps d'un surveillant r. Si cette question est exprimée dans le langage L.T.C.D., nous avons :

U := [H,S] [<R:r>,X](X,R) * (X,H,S)

Comme il se trouve qu'au niveau de l'espace physique la relation (X,R) n'existe pas, cette question devra être transformée en tenant compte des règles de conversion et sera équivalente à :

U := [<R:r>, H,S] (H,S,R)

Cette expression à l'exécution nécessitera que des accès à une seule collection et on peut estimer qu'il y aura gain de temps, par rapport à une requête qui aurait nécessité des accès à plusieurs collections.

(2) La question de l'utilisateur est de donner la liste des surveillants associée à un professeur p. Cette question s'exprimerait par :

T := [R][<P:p>X] (X,P)) * (X,R)

Pour que cette requête puisse s'exécuter, il faut remplacer (X,R) par la règle (X,R) := [X,R] (H,S,R) * (H,S,X) soit :

T := [R][<P:p> X] (X,P) * ([XR](H,S,R) * (H,S,R))

Là à l'inverse de la question précédente, il y aura accroissement du nombre d'accès puisque trois collections de données seront impliquées.

Remarque :

Nous avons vu que les opérations d'insertion dans une telle conception ne posaient aucun problème. Néanmoins, il faut signaler le cas d'insertion incomplète.

Par exemple l'introduction d'une entité (<X:x>,<H:h>,<S:'b'>) qui signifie que la salle de l'examen n'est pas encore connue peut être introduite dans la collection (X,H,S) tout en remarquant qu'elle détruit la relation fonctionnelle H,S → X. Par contre, nous sommes sûrs que nous ne pourrons recouvrer complètement l'information au cours de l'opération :

(X,H,S) → (H,S,R)

car la composition se ferait parfois sur des valeurs blanches.

On peut remarquer que si on prenait comme base de l'organisation physique la décomposition primaire, ce phénomène ne se serait pas produit car toutes les compositions s'effectueraient sur le constituant X, tandis que dans notre schéma au niveau physique les compositions peuvent s'effectuer sur le constituant X et le couple H,S.

Ce problème survient d'une façon générale chaque fois que l'utilisateur perçoit une collection (A,B,C,D) et que le niveau physique est fondé sur la décomposition, par exemple (A,B,C) * (C,D). Faisons de plus l'hypothèse que A est l'index de la collection (A,B,C) et C celui de la collection (C,D). L'introduction d'une entité (<A:a>,<B:b>,<C:c>,<D:d>) au niveau utilisateur entraînera l'introduction de deux entités (<A:a>,<B:b>, <C:c>,<D:d> si la valeur de c est 'b' alors il sera possible d'introduire l'entité <A:a>,<B:b>,<C:'b'> tandis que (<C:'b'>,<D:d>) ne pourra être introduite. Dans une telle situation le système pourrait soit provisoirement créer une zone des entités fictives, soit refuser l'introduction de cette entité tant que la valeur ne soit pas parfaitement définie.

12.2. Décomposition et organisation du niveau physique des données

Dans le paragraphe précédent, nous avons séparé le niveau de l'utilisateur, du niveau physique en indiquant que l'organisation des données au niveau physique pouvait avoir comme point de départ une décomposition secondaire.

Si une décomposition exprime certaines propriétés entre les données, elle ne définit pas pour autant la façon dont seront structurées les données sur les mémoires secondaires. Le problème que nous soulevons est celui de construire une organisation physique des données compatible avec un ensemble de collection de données munies de certaines propriétés de décompositions. Ce problème est fort vaste et complexe et il est en lui-même un travail complet, auquel nous comptons réfléchir. Néanmoins, nous voulons seulement indiquer certains faits caractéristiques entre le problème de la décomposition et les organisations hiérarchisées de données.

12.2.1. Rappels sur les organisations hiérarchisées

De nombreux systèmes de base de données utilisent des organisations physiques de nature hiérarchisée, on peut citer par exemple I.M.S. [38] . Les organisations hiérarchisées sont par nature identiques à la notion de catalogue dont la définition est donnée dans [39], [40]. Rappelons brièvement la définition d'un catalogue à partir des définitions ci-dessous.

Une arborescence* est un ensemble ordonné fini tels que deux éléments quelconques ont un minorant commun mais pas de majorant commun.

On distinguera l'ensemble X des éléments d'une arborescence Y, de l'arborescence elle-même, qui est la donnée de X et d'une relation d'ordre $\mathcal{H}$ qui en fait une arboresence. On notera $Y = (X, \mathcal{H})$.

Une forêt est une union d'arboresences disjointes.

---

* Le terme "d'arborescence" qui va être utilisé dans ce paragraphe ne doit pas être confondu avec celui de "décomposition arborescente". Toutefois les

Une pseudo arborescence est une arborescence valuée, c'est-à-dire qu'à chaque élément de l'arborescence on associe une valeur. Dans les pseudo-arborescences dont il sera question, les éléments distincts comparables porteront des valeurs distinctes. En conséquence, la notion de chaîne d'un ensemble ordonné peut s'appliquer à des pseudo-arborescences.

Un homomorphisme d'ordre strict de première catégorie par rapport à la première variable est une application ψ d'un ensemble ordonné E dans un ensemble ordonné F telle que :

(1) $\quad a < b \implies \psi(a) < \psi(b) \qquad a,b \in E$

(2) $\quad \psi(a) < \psi(b) \implies \forall\, a_1 \in \psi^{-1}\psi(a) \quad \exists b_1 \in \psi^{-1}\psi(b)$ tel que $a_1 < b_1$

pour un homomorphisme par rapport à la deuxième variable (2) est à remplacer par (3).

(3) $\quad \psi(a) < \psi(b) \implies \forall\, b_1 \in \psi^{-1}\psi(b) \quad \exists a_1 \in \psi^{-1}\psi(a)$ tel que $a_1 < b_1$

On appellera carte arborescente du catalogue, la donnée de l'ensemble $\mathcal{A}$ des constituants ordonnés par la relation $\mathcal{H}$, que nous noterons $(\mathcal{A}, \mathcal{H})$.

Une forêt F est dite un précatalogue sur la carte arborescente $(\mathcal{A}, \mathcal{H})$ si il existe une application surjective de F sur $(\mathcal{A}, \mathcal{H})$ qui soit un homomorphisme d'ordre strict de première catégorie successivement par rapport aux deux variables.

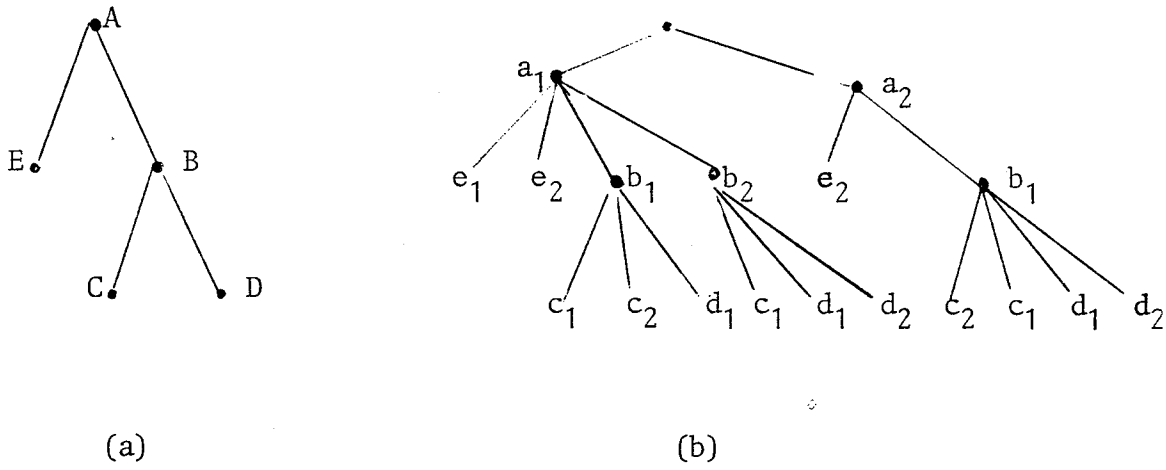
Valuons les éléments du précatalogue de sorte que :

- si $A \in \mathcal{A}$ les éléments de $\psi^{-1}(A)$ sont valués avec des éléments appartenant à l'ensemble A.
- deux éléments n'ont même valuation que si ils appartiennent au même $\psi^{-1}(A)$ et s'ils ne couvrent pas un même élément.

La pseudo arborescence obtenue en ajoutant alors une racine est un catalogue K.

Exemple 12.1.

La figure 12.1.(a) représente la carte arborescente du catalogue 12.1.(b).



(a) (b)

FIGURE 12.1. : Catalogue associé à une carte arborescente

12.2.2. Catalogue et décomposition

A tout catalogue on peut associer une collection de données, en considérant que chaque chaîne du catalogue est représentative d'une entité d'une collection de données.

C'est ainsi qu'à partir du catalogue de l'exemple 12.1 on peut construire les 3 collections de données correspondant aux branches de la carte arborescente :

A	E
a_1	e_1
a_1	e_2
a_2	e_2

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_2	b_1	c_2
a_2	b_1	c_1

A	B	D
a_1	b_1	d_1
a_1	b_2	d_1
a_1	b_2	d_2
a_2	b_1	d_1
a_2	b_1	d_2

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_2	d_1	e_1
a_1	b_2	c_1	d_1	e_1
a_1	b_2	c_1	d_2	e_1
a_2	b_1	c_2	d_1	e_2
a_2	b_1	c_2	d_2	e_2
a_2	b_1	c_1	d_1	e_2
a_2	b_1	c_1	d_2	e_2
a_1	b_1	c_1	d_1	e_2
a_1	b_1	c_2	d_1	e_2
a_1	b_2	c_1	d_1	e_2
a_1	b_2	c_1	d_2	e_2

Inversement si les collections de données qui peuvent être construites sur un ensemble de constituants (A,B,C,D,E) vérifient les décompositions :

$$(A,B,C,D,E) = (A,E) * (A,B,C,D)$$

et

$$(A,B,C,D) = (A,B,C) * (A,B,D)$$

alors toute collection de données d'ensemble (A,B,C,D,E) pourra être représentées par un catalogue dont la carte arborescente est celle de la figure 12.1.(a).

Le premier problème a trait au fait qu'une relation fonctionnelle est une déclaration de type sémantique qui permet de valider l'information au moment de son introduction dans le système d'informations. Ce processus de validation peut être long et le résultat auquel nous sommes parvenus est de le relier au concept de couverture minimale d'un ensemble de relations fonctionnelles.

Nous avons vu au cours du chapitre 11 que les relations fonctionnelles étaient des facteurs déterminants dans la décomposition d'une collection de données et par conséquent dans l'organisation de la base de données. Certaines déclarations faites à la création du système peuvent après un certain temps ne plus être vérifiées. Dans de telles conditions doit-on réorganiser la base de données ? Si oui comment restructurer le système ? Ce problème de restructuration se pose aussi chaque fois que l'utilisateur désire ajouter de nouveaux constituants à des collections de données existantes, créer de nouvelles collections de données, fusionner plusieurs systèmes d'informations en un seul.

CHAPITRE 13

EVOLUTION ET RESTRUCTURATION D'UN SYSTEME D'INFORMATIONS

Une base de données est un système en perpétuelle évolution, par suite d'insertions, de suppressions, de mise à jour. Dans ce chapitre nous voulons étudier des phénomènes liés à ces opérations :

- dans quelles conditions une insertion ou une mise à jour d'entité ne détruit pas les relations fonctionnelles déclarées ?
- est-ce qu'une modification de l'ensemble des relations fonctionnelles entraîne des modifications dans le problème de la décomposition ?
- est-il possible de passer d'une décomposition à une autre décomposition ?

13.1. Insertion et mise-à-jour d'entités

13.1.1. Définition

On considère une collection de données T définie sur un ensemble de constituants $\mathcal{a}$. A cette collection de données on associe l'ensemble $\Phi = \{\ell_1, \ell_2, \ldots, \ell_k\}$ des relations fonctionnelles où

$$\ell_i : E_i \rightarrow F_i \qquad i = 1,2,\ldots,k$$

On dira que l'introduction d'une entité $<\mathcal{a}:a>$ ne détruit pas l'ensemble Φ des relations fonctionnelles si pour toute relation fonctionnelle ℓ_i l'une des deux conditions ci-dessous est vérifiée :

(a) $[E_i] \quad <\mathcal{a}:a> \quad \notin \quad [E_i]T$

(b) si $[E_i] \quad <\mathcal{a}:a> \quad \in \quad [E_i]T$ alors $[F_i] \quad <\mathcal{a}:a> \in [F_i]T$ et l'application m associée à la relation fonctionnelle : $m:[E_i]T \rightarrow [F_i]T$, est telle que : $m([E_i] <\mathcal{a}:a>) = [F_i] <\mathcal{a}:a>$

Exemple 13.1.

L'introduction de l'entité :

(<P:durand>, <T:math>, <Y:4>, <N:3C>, <H:6>)

dans la collection de l'emploi du temps du paragraphe 7.2 détruit la relation fonctionnelle P,H → N. En effet, l'image de la sous-entité :
(<P:durant>, <H:6>) est <N:2B> et non <N:3C>

D'une façon générale nous dirons qu'une insertion ou une mise-à-jour est destructrice si elle détruit au moins une relation fonctionnelle.

A l'aide de cet exemple, on voit qu'à chaque insertion le temps mis pour vérifier chaque relation fonctionnelle peut être fort long.

13.1.2. Destruction de relation fonctionnelle et couverture minimale

Proposition 13.1.

Si une entité <$\mathcal{a}$:a> ne détruit aucune relation fonctionnelle élémentaire de la couverture minimale de Φ alors elle ne détruit aucune relation fonctionnelle élémentaire de la fermeture Φ^+.

Nous avons vu au chapitre 9 que la fermeture Φ^+ peut se déduire uniquement de la couverture minimale par l'opération de pseudo-transitivité. Considérons donc, deux relations fonctionnelles appartenant à la couverture minimale telle que la propriété de pseudo-transitivité puisse s'appliquer, c'est-à-dire :

$$\ell_1 : E \rightarrow F \qquad \ell_2 : F,G \rightarrow H$$

Si l'entité <$\mathcal{a}$:a> ne détruit pas ℓ_1 et ℓ_2 alors elle ne détruit pas ℓ_3 : E,G → H. Pour montrer que cette propriété est vraie nous avons quatre cas à considérer :

(1) [E] <$\mathcal{a}$:a> $\notin$ [E]T et [F,G] <$\mathcal{a}$:a> $\notin$ [F,G]T
(2) [E] <$\mathcal{a}$:a> $\in$ [E]T et [F,G] <$\mathcal{a}$:a> $\notin$ [F,G]T
(3) [E] <$\mathcal{a}$:a> $\notin$ [E]T et [F,G] <$\mathcal{a}$:a> $\in$ [F,G]T
(4) [E] <$\mathcal{a}$:a> $\in$ [E]T et [F,G] <$\mathcal{a}$:a> $\in$ [F,G]T

Il est facile de remarquer que dans les cas (1), (2) et (3)

[E,G] <$\mathcal{a}$:a> $\notin$ [E,G]T,

et par conséquent ℓ_3 n'est pas détruit.

Dans le cas (4) nous avons, si on désigne respectivement par :

m_1 : [E]T → [F]T
m_2 : [F,G]T → [H]T
m_3 : [E,G]T → [H]T
m_1([E] <$\mathcal{a}$:a>) = [F] <$\mathcal{a}$:a>
m_2([F,G] <$\mathcal{a}$:a>) = [H] <$\mathcal{a}$:a>

ou bien encore :

$m_1(\langle E:e\rangle) = \langle F:f\rangle$

$m_2(\langle F:f\rangle, \langle G:g\rangle) = \langle H:h\rangle$

avec :

$\langle E:e\rangle = [E]\ \langle a:a\rangle$

$\langle F:f\rangle = [F]\ \langle a:a\rangle$

$\langle G:g\rangle = [G]\ \langle a:a\rangle$

$\langle H:h\rangle = [H]\ \langle a:a\rangle$

ces conditions impliquent que l'entité $\langle a:a\rangle$ est de la forme :

$\langle a:a\rangle = \langle E:e\rangle, \langle F:f\rangle, \langle G:g\rangle, \langle H:h\rangle, \langle X:x\rangle$

où X est un constituant composé complémentaire de (E,F,G,H) par rapport à a.

On vérifie alors que :

$m_3([F,G]\ \langle a:a\rangle) = m_3(\langle F:f\rangle, \langle G:g\rangle) = \langle H:h\rangle = [H]\ \langle a:a\rangle$

c'est-à-dire que la relation fonctionnelle ℓ_3 : E,G → H n'est pas détruite

13.1.3. Destruction de relation fonctionnelle et décomposition

Dans le cadre du paragraphe précédent nous avons considéré l'insertion ou la mise à jour d'une entité de la collection de données T d'ensemble de constituants a.

Si on considère maintenant le cas où T peut être décomposée en une décomposition primaire et en une décomposition secondaire. Dans une telle situation est-il nécessaire de vérifier toutes les relations fonctionnelles élémentaires de la couverture minimale ? Nous allons montrer, que de la même façon que nous avons réduit le nombre de vérification en passant de la fermeture à la couverture minimale, on va pouvoir passer de la couverture minimale à un nombre restreint de relations fonctionnelles élémentaires.

Le processus que nous allons montrer sera illustré sur l'exemple du paragraphe 11.1.2. sans limitation de généralisation. Ce processus est fondé sur le schéma de la figure 13.1., qui fait apparaître tous les éléments de la décomposition primaire et secondaire et les relations fonctionnelles élémentaires de la couverture minimale.

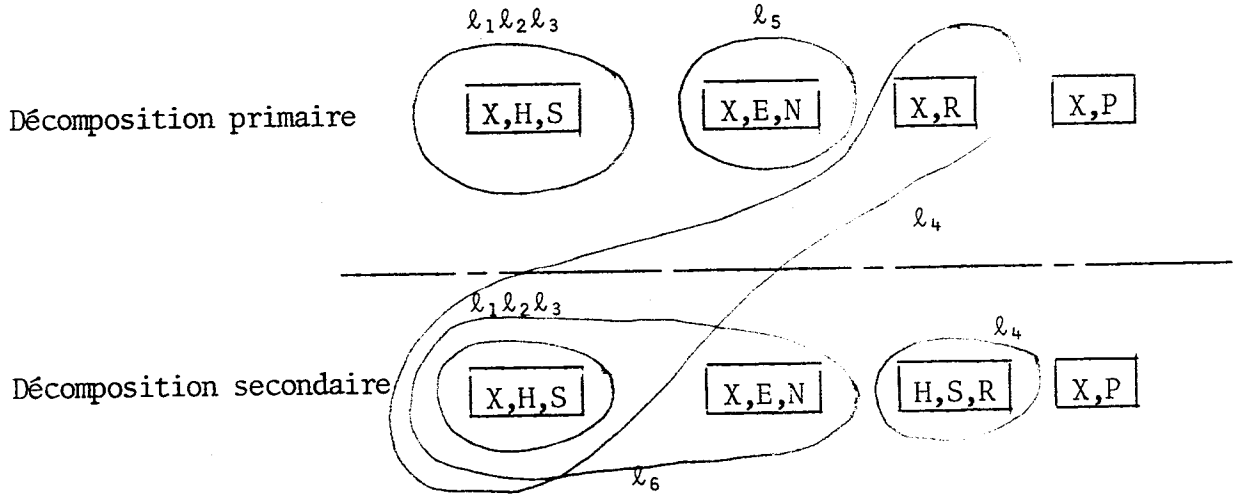


FIGURE 13.1. : DECOMPOSITION ET RELATION FONCTIONNELLE

Une relation fonctionnelle élémentaire sera spécifique d'une collection de données si les constituants de la relation fonctionnelle élémentaire sont contenus dans l'espace de la collection. Si une relation fonctionnelle n'est pas spécifique d'une collection cela signifie que les constituants de la relation fonctionnelles apparaissent dans plusieurs collections de données. En conséquence, pour vérifier la validité de cette relation il est nécessaire d'effectuer la composition de plusieurs collections.

Par exemple ℓ_1, ℓ_2, ℓ_3 sont spécifiques de (X,H,S) tandis que ℓ_4 est spécifique de la collection (H,S,R) et apparaît dans le couple de collection (X,R) et (X,H,S).

Comme nous l'avons vu dans le fonctionnement du système (paragraphe 12.1.3.) l'utilisateur introduit ou modifie une entité à partir des éléments de la décomposition primaire. Si on introduit (ou modifie)

une entité dans une collection de données qui apparait à la fois au niveau primaire et secondaire on commencera par vérifier toutes les relations fonctionnelles élémentaires spécifiques de cette collection. Ensuite, on examinera s'il existe une relation fonctionnelle élémentaire qui recouvre plusieurs collections, dans un tel cas il faudra effectuer la composition de ces collections, puis vérifier la relation fonctionnelle.

Si on introduit une entité dans une collection primaire n' ayant pas son correspondant au niveau physique il faudra utiliser les règles de conversion que nous avons utilisées au paragraphe 12.1.3. Si de telles règles nécessitent la composition de collection il faudra vérifier que pour chaque collection créée les relations fonctionnelles spécifiques ou non sont satisfaites.

Illustrons ceci à l'aide de notre exemple en supposant que les valeurs stockées au niveau secondaire sont données par la figure 13.2.

(X,H,S)		
x_1	h_1	s_1
x_2	h_2	s_2
x_3	h_1	s_2

(X,E,N)		
x_1	e_1	-
x_1	e_2	-
x_1	e_3	-
x_2	e_1	-
x_2	e_3	-
x_3	e_4	-

(H,S,R)		
h_1	s_1	r_1
h_1	s_1	r_2
h_2	s_2	r_1
h_1	s_2	r_3
h_1	s_2	r_4
h_2	s_2	r_2

(X,P)	
x_1	p_1
x_1	p_2
x_2	p_1
x_3	p_2

FIGURE 13.2. : OCCURRENCE DE VALEURS POUR UNE DECOMPOSITION SECONDAIRE

L'introduction de l'entité ($<X:x_3>$, $<R:r_1>$) détruit la relation fonctionnelle ℓ_4. En effet, conformément aux règles de conversion, cette entité donne par la composition :

$(<X:x_3>, <R:r_1>) * (X,H,S) = (<X:x_3>, <R:r_1>, <H:h_1>, <S:s_2>)$
et par projection l'entité $(<H:h_1>, <R:r_1>, <S:s_2>)$, comme il existe l'entité $(<H:h_1>, <R:r_1>, <S:s_1>)$ il y a incompatibilité.

De même l'introduction de $(<X:x_3>, <E:e_1>, <N:'b'>)$ dans la collection (X,E,N) détruit la relation fonctionnelle H,E → X, nous laissons le soin au lecteur de le vérifier.

La principale différence entre ces deux cas d'insertion est que,dans le premier cas une fois la composition faite, la vérification se fait directement par référence à la collection (H,S,R),tandis que dans le deuxième cas il est nécessaire de reconstruire une collection pour faire la vérification.

Dans le cas de modification de valeur, il serait possible de considérer la mise à jour comme l'introduction d'une nouvelle entité, puis de supprimer l'ancienne qui ferait appel au même processus. Néanmoins, il serait possible de procéder de façon plus fine comme le montre l'exemple ci-dessous.

Le remplacement de l'entité $(<X:x_1>, <H:h_1>, <S:s_1>)$ par $(<X:x_1>, <H:h_2>, <S:s_1>)$ ne doit pas être exécuté car cette modification détruirait la relation fonctionnelle ℓ_6 : H,E → X. Par contre, le remplacement de $(<X:x_1>, <H:h_1>, <S:s_1>)$ par $<X:x_1>, <H:h_1>, <S:s_3>$ est possible. De plus, il ne nécessite aucune vérification car d'une part les relations fonctionnelles X → S et H,S → X continuent à être vérifiées, et que d'autre part le constituant S, ici modifié, n'apparaît pas dans la relation fonctionnelle H,E → X.

Un autre problème lié à ces processus d'insertion et de mise à jour est celui de maintenir la base de données dans un état tel que les charnières de composition soient homogènes. Ceci ne peut être fait que si au cours d'une mise à jour il y a des effets de bord.

Par exemple, l'introduction d'un nouvel examen doit avoir des répercussions simultanées sur toutes les collections (X,H,S), (X,E,N), (X,R) et (X,P) sinon l'homogénéité des charnières ne sera pas respectée. Il est donc important qu'un système d'informations n'exécute une opération de mise à jour que si, éventuellement, elle entraîne d'autres actions de mise à jour.

13.2. Modification de la structure

13.2.1. Modifications de l'ensemble des relations fonctionnelles

Nous avons vu dans le paragraphe précédent que l'introduction de nouvelles entités ou la modification de valeurs d'entités peuvent détruire l'ensemble des relations fonctionnelles déclarées . Ceci peut provenir soit d'une insertion erronée auquel cas le système est préservé contre ce risque d'erreur soit que cela résulte d'une modification des relations fonctionnelles du système d'information.

Par exemple, supposons que l'utilisateur ait déclaré qu'un employé ne travaille que pour un seul département, ce qui s'exprimerait par la relation fonctionnelle EMPLOYE → DPT, et qu'il introduise une entité faisant apparaître qu'un employé peut travailler pour plusieurs départements. Dans ces conditions il est évident que la relation fonctionnelle ci-dessus n'est plus vérifiée mais alors est-ce que la structure du système doit être remise en cause ?

Dans l'état actuel de notre travail il n'est pas possible de répondre à cette question sans examiner les nouvelles décompositions de l'ensemble des constituants. Si malgré, la modification des relations fonctionnelles, on peut trouver une décomposition identique à celle existant avant la modification, alors le système n'a pas besoin d'être restructuré, sinon le système nécessite une restructuration.

Nous voulons illustrer ce phénomène à l'aide d'un exemple très simple.

Exemple 13.2.

Soit quatre constituants A,B,C,D sur lesquels est défini l'ensemble de relations fonctionnelles élémentaires.

$$A \rightarrow B$$
$$B \rightarrow C$$
$$C \rightarrow D$$

Une décomposition primaire est :

$$(A,B,C,D) = (A,B) * (B,C) * (C,D)$$

de plus, faisons l'hypothèse que l'organisation physique se superpose à la perception de l'utilisateur, par conséquent les collections stockées sont aussi (A,B), (B,C) et (C,D).

Maintenant, nous voulons étudier quelles seront les conséquences sur la décomposition dans les trois cas ci-dessous :
cas 1 : la relation A → B est détruite
cas 2 : la relation B → C est détruite
cas 3 : la relation C → D est détruite.

Le tableau ci-dessous donne différentes décompositions possibles dans chaque cas.

CAS 1	CAS 2	CAS 3
(B,C) * (B,A,D) (C,D) * (B,C) * (B,A)	(A,B) * (C,D) * (A,C) (A,B,C) * (C,D)	(A,B) * (A,C,D)

Seul le cas 1 a une décomposition identique à la décomposition initiale, pour éviter toute restructuration il faudrait adopter cette décomposition. Dans tous les autres cas il faudra effectuer une restructuration.

13.2.2. Restructuration du système d'informations

La restructuration de la base de données peut être provoquée pour différents motifs :

- la décomposition secondaire n'est pas assez efficace vis-à-vis des requêtes de l'utilisateur, d'où la recherche d'une autre décomposition répondant à ce souci.
- la décomposition adoptée n'est plus valable car il y a eu des modifications dans l'ensemble des relations fonctionnelles (voir paragraphe précédent).
- de nouveaux constituants et de nouvelles relations sont ajoutées à la base de données.

Ce problème n'est pas résolu et nous faisons ici qu'évoquer les perspectives d'évolution de notre travail, à savoir comment effectuer une restructuration en effectuant le minimum d'opération. Il est sûr que ces opérations ne peuvent se concevoir que si l'on a effectué une complète séparation entre le niveau logique et le niveau physique. De plus, pour faire ces opérations on devra définir les opérations de conversion qui permettent de passer d'une structure à une autre structure.

Exemple 13.3.

L'exemple ci-dessous illustre un cas où l'addition d'une relation fonctionnelle et d'un constituant ne détruit pas la structure initiale mais rajoute seulement une collection supplémentaire.

	Structure initiale	*Addition*	*Structure finale*
Relations fonctionnelles	$A \rightarrow B,C$ $B \rightarrow D$	$B,C \rightarrow E$	$A \rightarrow B,C$ $B \rightarrow D$ $B,C \rightarrow D$
Décomposition	$ABC * BD$		$ABC * BD * BCE$

CHAPITRE 14

CONCLUSIONS

Dans ce travail nous avons poursuivi deux objectifs parallèles qui avaient le même point de départ, à savoir, la notion de collection de données. Le premier de ces objectifs était de représenter un système d'informations comme un ensemble de collections de données à partir duquel il était possible d'effectuer des calculs pour donner naissance à de nouvelles collections de données. Ceci nous a conduit à définir plutôt qu'un langage un formalisme pour représenter les manipulations qui pouvaient être effectuées sur le système d'informations. D'autre part, nous avons proposé d'utiliser ce formalisme dans la construction de modèle d'entreprise et plus particulièrement du réseau des informations.

Le deuxième objectif avait essentiellement pour but d'essayer de comprendre pourquoi l'utilisateur lorsqu'il exprime son système d'informations à l'aide de collections de données il le fait d'une certaine façon. C'est ainsi que nous avons mis en lumière le rôle des relations fonctionnelles et de la décomposition et leurs utilisations.

Si ceci représente les intentions que nous avons essayé d'exprimer dans ce travail, il est intéressant nous semble-t-il, de présenter les points qui mériteraient des prolongements.

Le premier serait d'entreprendre la modélisation complète d'un système d'informations d'une entreprise à partir des concepts que nous avons développés. Ce travail nécessiterait d'une part l'adaptation d'un langage de simulation pour inclure les opérations sur les collections de données et d'autre part une analyse assez fine des processus de décision.

Le deuxième, orienté vers la structure des données, serait de donner des critères pour construire de façon algorithmique des représentations physiques des données compatibles avec un ensemble de collections de données. Ce problème a certainement de nombreuses solutions mais il nous permettrait de mieux comprendre la notion de structure de données.

Un troisième prolongement lié aussi à l'organisation physique des données pourrait être d'étudier la meilleure décomposition secondaire correspondant à un ensemble de requête. Dans un tel travail il ne faudrait pas voir uniquement l'aspect statique, car l'ensemble des requêtes d'un utilisateur évolue avec le temps et par conséquent il faudrait voir aussi l'évolution de la décomposition secondaire.

Une dernière idée pourrait être d'utiliser le concept de collections de données pour réaliser sur des petites machines des systèmes appliqués aux tâches de gestion des moyennes entreprises. C'est-à-dire une "mini-base de données" où l'utilisateur pourrait d'une part exprimer les traitements qu'il souhaite effectuer mais aussi accèder aux informations archivées.

ANNEXE A

COLLECTIONS DE DONNEES ET RELATIONS

1. Définitions

Etant donné une collection de données P définie par :

- un ensemble E
- un ensemble $X_1, X_2, \ldots, X_p$ de champs,
- un ensemble de n applications $A_i : E \to X_i \quad i = 1,2,\ldots,n$

on peut considérer une relation n-aire définies sur les champs $X_1, X_2, \ldots, X_p$ comme étant l'ensemble des n-uplets :

$$\{(A_1(e), A_2(e), \ldots, A_n(e)) : \quad e \in E\}.$$

De même si on divise l'ensemble des applications en deux sous-ensembles disjoints A et B, on peut créer une relation binaire notée :

$$P = A \rhd B$$

comme étant l'ensemble des paires de la forme (a,b) avec :

- $a = (A_1(e), A_2(e), \ldots, A_i(e)) \quad A_1, A_2, \ldots, A_i \in A$
- $b = (A_{i+1}(e), A_{i+2}(e), \ldots, A_n(e)) \quad A_{i+1}, \ldots, A_n \in B$
- $e \in E$

La relation transposée de P, notée P^T sera une relation binaire notée :

$P^T : B \rhd A$ et définie par :

$P^T = \{(b,a) : (a,b) \in P\}$

Si P est une relation binaire $P : A \rhd B$ on définira le <u>domaine</u> de P et <u>l'image</u> de P comme les projections de P respectivement sur A et sur B.

Etant donné deux collections P et Q et deux relations binaires associées à P et Q :

$P : A \rhd B$

$Q : B \rhd C$

vérifiant $Im(P) = Dom(Q)$, on définit alors la composition des deux relations binaires par :

$$Q \circ P \triangleq \{(a,c) : \exists\, b \in Im(P), (a,b) \in P \text{ et } (b,c) \in Q\}$$

Il faut bien remarquer que Q ∘ P est une relation binaire orientée :

$$Q \circ P : A \rhd C$$

et non une collection de données.

Si la composition normale des collections P et Q s'effectue sur le support B alors il est possible de relier la composition normale de collections et la composition de relation comme l'indique le diagramme de la figure A.1.

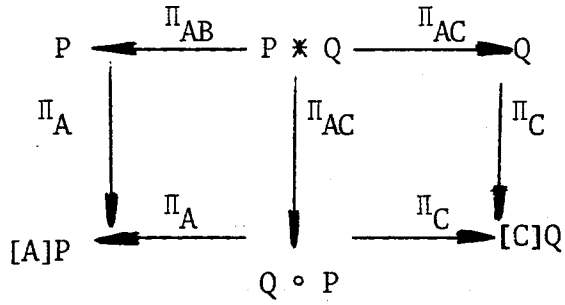


FIGURE A.1.

Définition : **Propriété d'unicité**

On dira que deux relations binaires :

$$P : A \rhd B$$

$$Q : B \rhd C$$

associées aux collections de données P et Q possèdent la propriété d'unicité s'il n'est pas possible de trouver une relation binaire U : A ▷C tel que

(a) $U \subset T$ avec $T = Q \circ P$

(b) Dom(U) = Dom(P)

(c) Im(U) = Im(Q)

2. Conditions d'unicité de la famille des composés de deux collections

Nous avons vu au paragraphe 3.4.1 qu'il existait une famille $\mathscr{C}(P,Q)$ de composés des collections de données P et Q, nous voulons rechercher dans quelles conditions cette famille est réduite au seul élément $P * Q$.

Donc, étant donné deux collections P et Q d'espaces de constituants respectifs (A,B) et (B,C), ainsi que les relations binaires associées :

$$P : A \rhd B$$
$$Q : B \rhd C$$

nous dirons que la composition stricte des collections P et Q est sans point multiple si pour toute entité <B:b> appartenant à [B]P = [B]Q :

$$\text{card } ([<B:b>,C]Q) = 1$$

ou

$$\text{card } ([<B:b>,A]P) = 1$$

Proposition A.1.

(a) card ($\mathscr{C}(P,Q)$) = 1 $\iff$ pas de point multiple

(b) Une condition nécessaire et suffisante pour que les relations binaires P et Q possèdent la propriété d'unicité est que $Q \circ P$ puisse être décomposée en une bipartition de deux relations binaires V et W telles que :

$$V : A \rhd C$$
$$W^T : C \rhd A$$

soit des applications.

(c) Propriété d'unité $\Rightarrow$ pas de point multiple.

Démonstration de la propriété (a) :

Nous avons vu qu'il était possible d'exprimer le nombre d'éléments de la collection P ✱ Q par :

$$\sum_b p_b \cdot q_b$$

où

$$p_b = \text{card } ([<B:b>,A]P)$$
$$q_b = \text{card } ([<B:b>,C]Q)$$

et le nombre d'éléments d'une collection minimale de la famille $\mathscr{C}(P,Q)$ par :

$$\sum_b \text{Max } (p_b, q_b)$$

Un élément minimal est confondu avec P ✱ Q si et seulement si pour toute donnée b :

$$p_b \cdot q_b = \text{Max } (p_b, q_b)$$

équation qui ne peut être vérifiée que dans le cas :

$p_b = 1$ et q_b quelconque

ou

p_b quelconque et $q_b = 1$

ce qui exprime bien que le support de la composition n'a pas de point multiple.

Démonstration de la propriété (b)

Posons T = Q ∘ P, et d'après la propriété d'unicité il n'est pas possible de trouver U ⊂ T tel que Dom(U) = Dom(T) et Im(U) = Im(T).

Définissons les ensembles suivants :

$A_1 = \{a : a \in Dom(T), \forall c,c' \in Im(T)\ (a,c) \in T \text{ et } (a,c') \in T \Rightarrow c = c'\}$

$A_2 = \{a : a \in Dom(T) \text{ et } a \notin A_1\}$

$C_1 = T(A_1) = \{c : c \in Im(T),\ \exists a \in A_1,\ (a,c) \in T\}$

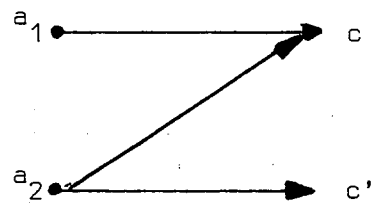
$C_2 = T(A_2) = \{c : c \in Im(T),\ \exists a \in A_2,\ (a,c) \in T\}$

$V = \{(a,c) : \exists a \in A_1,\ (a,c) \in T\}$

$W = \{(a,c) : \exists a \in A_2,\ (a,c) \in T\}$

Montrons tout d'abord que $V \cup W = T$, c'est évident cela résulte des définitions de V et W, montrons aussi que $V \cap W = \emptyset$

Pour montrer que $V \cap W = \emptyset$ cela revient à montrer que $C_1 \cap C_2 = \emptyset$. Faisons l'hypothèse que $C_1 \cap C_2 \neq \emptyset$ et soit $c \in C_1 \cap C_2$ alors il existe $a_1 \in A_1$ et $a_2 \in A_2$ tel que (a_1,c) et $(a_2,c) \in T$, mais il existe aussi c' tel que $(a_2,c') \in T$ puisque $a_2 \in A_2$, sous ces hypothèses si on considère la relation :



$U = T - \{(a_2,c)\}$ alors :

$Dom(U) = Dom(T)$

$Im(U) = Im(T)$

ce qui en contradiction avec le fait qu'il ne puisse exister une relation U. On en conclut donc $C_1 \cap C_2 = \emptyset$ et qui implique $V \cap W = \emptyset$.

Il reste à montrer que W^T est une application puisque la relation binaire V définit une application par construction.

Si (a,c_1) et $(a,c_2) \in W$ nous allons montrer qu'il ne peut exister d'élément $a' \in A_2$ tel que (a',c_1) ou $(a',c_2) \in W$, supposons que cela soit $(a',c_1) \in W$, alors il est possible de constuire une relation

$$U = T - \{(a,c_1)\}$$

avec $Dom(U) = Dom(T)$ et $Im(U) = Im(T)$ ce qui en contradiction avec l'hypothèse et par conséquent (a',c_1) n'existe pas, ce qui montre bien que la relation binaire W^T est une application.

Réciproquement si $T = Q \circ P = V \cup W$ avec $V \cap W = \emptyset$ où les relations V et W^T sont des applications, nous allons montrer qu'il ne peut exister de relation $U \subset T$ avec $Dom(U) = Dom(T)$ et $Im(U) = Im(T)$

Supposons qu'une telle relation U existe, par exemple : $U = T - \{(a,c)\}$, on peut alors avoir deux hypothèses soit $(a,c) \in V$, soit $(a,c) \in W$, examinons les successivement :

- si $(a,c) \in V$, comme V est une application on en déduirait que $Dom(U) \subset Dom(T)$ ce qui contredit l'hypothèse.
- si $(a,c) \in W$, comme W^T est une application on en déduirait que $Im(U) \subset Im(T)$ ce qui contredit aussi l'hypothèse.

Il résulte donc, qu'il ne peut exister de relation $U = T - \{(a,c)\}$ satisfaisant les hypothèses et par conséquent $U = T = Q \circ P$.

Démonstration de la propriété (c)

La propriété d'unicité n'implique pas de point multiple.

Nous avons déjà vu que $T = Q \circ P = V \cup W$ avec $V \cap W = \emptyset$, prenons un élément $(a,c) \in T$ il appartient soit à V, soit à W, supposons que $(a,c) \in V$ alors il existe b tel que $(a,b) \in P$, et $(b,c) \in Q$. Nous allons montrer que b n'est pas un point multiple. En effet, si $card([<B:b>,C]Q) > 1$ il existerait c', tel que $(b,c') \in Q$ et $(a,c') \in T$ ce qui n'est pas possible puisque V est une application. On montrerait la même chose à partir d'un couple $(a,c) \in W$.

Ceci implique bien que la composition n'a pas de point multiple.

La réciproque n'est pas vraie comme le montre l'exemple 3.4. où il existe une relation U contenue dans $T = Q \circ P$ et P et Q n'ont pas de point multiple.

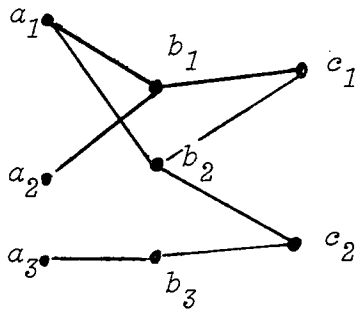
Exemple A.1. :

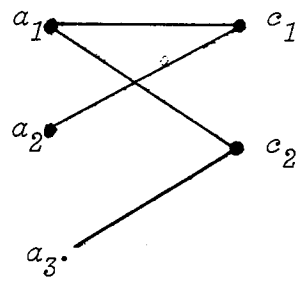
P	
A	B
a_1	b_1
a_1	b_2
a_3	b_3
a_2	b_1

Q	
B	C
b_1	c_1
b_2	c_1
b_3	c_2
b_2	c_2

$T = Q \circ P$	
A	C
a_1	c_1
a_2	c_1
a_3	c_2
a_1	c_2

$U \subset T$	
A	C
a_1	c_1
a_2	c_1
a_3	c_2





A N N E X E B

DECOMPOSITION PAR ELIMINATION

Décomposition par élimination

Etant donné une collection P d'espace de constituants (A,B,C) on dira que les collections R et S d'espace (A,B,X) et (A,C,X) constituent une décomposition par élimination si :

$$P = R \overset{X}{\circ} S \triangleq [A,B,C]\ (R * S)$$

Il a été donné dans [17] une condition nécessaire et suffisante qui s'exprime par :

Si $\langle A:a\rangle \in [A]P$ et si $\langle C:c\rangle \in [C]P$ alors

$$[\langle A:a\rangle,\langle C:c\rangle,B]P = \bigcup_{\langle X:x\rangle \in [\langle A:a\rangle,\langle C:c\rangle,X]S} [\langle A:a\rangle, \langle X:x\rangle,B]\ R$$

Nous allons construire une telle décomposition à partir du tableau $\mathcal{T}^{P}_{AB,AC}$ en considérant les sous-tableaux $\mathcal{T}^{P}_{aB,aC}$ où a est une donnée appartenant à l'ensemble A.

Dans chaque sous-tableau $\mathcal{T}^{P}_{aB,aC}$ nous allons rechercher des blocs rectangulaires ne comprenant que des 1, comme le montre la figure 3.4 et qui seront définis de la façon suivante :

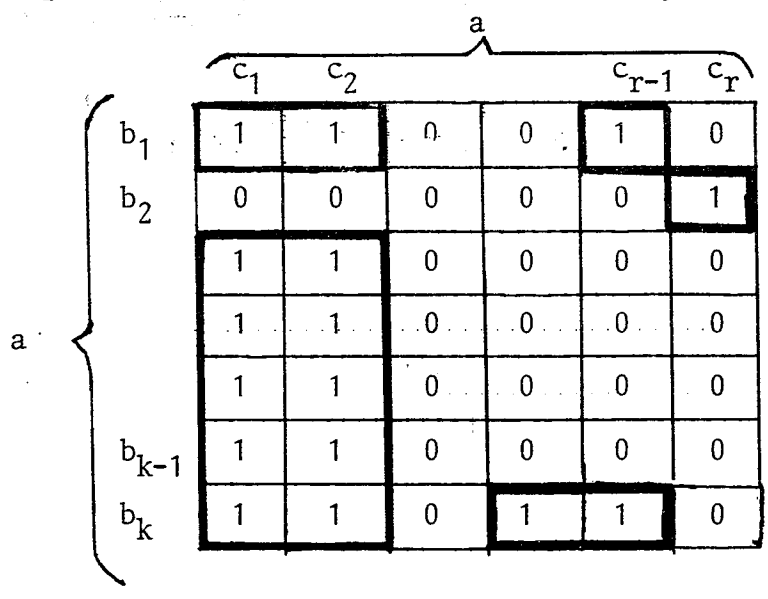


FIGURE 3.4.

- nous désignerons par $\mathcal{C}_a(b,c)$ la valeur de l'élément qui se trouve à l'intersection de ligne b et de la colonne c

- nous définirons l'ensemble :

$$U = \{(b,c) : \mathcal{C}_a(b,c) = 1\}$$

- nous dirons que dans le tableau $\mathcal{C}^P_{aB,aC}$ il existe r blocs $\mathcal{B}^a_i$, i = 1,2,...,r, s'il est possible de trouver des parties B_i et C_i de B et C telles que :

$$\mathcal{B}^a_i = B_i \times C_i$$

et qui vérifient :

$$- \bigcup_i \mathcal{B}^a_i = U$$

- pour tout couple $(b_i, c_i) \in \mathcal{B}^a_i$, $\mathcal{C}_a(b_i,c_i) = 1$

Alors à chacun des r blocs on affectera une valeur x, plus précisément on effectera les valeurs : $x^a_1, x^a_2,\ldots,x^a_r$, aux blocs $\mathcal{B}^a_1, \mathcal{B}^a_2,\ldots, \mathcal{B}^a_r$

Les collections R et S seront alors composées de la façon suivante :

$(\langle A:a\rangle,\langle B:b\rangle,\langle X:x_i^a\rangle) \in R$ et $(\langle A:a\rangle,\langle C:c\rangle,\langle X:x_i^a\rangle) \in S$ si si l'élément situé à l'intersection de la ligne (a,b) et de la colonne (a,c) appartient au $i^{ème}$ bloc valué par x_i^a.

Sous ces conditions, il est aisé de vérifier que la condition nécessaire et suffisante écrite ci-dessus est bien satisfaite.

Remarque :

(a) La décomposition obtenue n'est pas unique puisqu'à l'intérieur d'un sous-tableau $\mathcal{C}^p_{aB,aC}$ *n'importe quelle permutation de ligne et colonne provoquera un ordonnancement différent des blocs rectangulaires.*

(b) Par commodité de présentation, nous avons valué les blocs $\mathcal{B}^a_1, \mathcal{B}^a_2, \ldots, \mathcal{B}^a_r$ *d'un sous-tableau* $\mathcal{C}^p_{aB,aC}$ *par* $x^a_1, x^a_2, \ldots, x^a_r \in X$, *on peut pour chaque sous-tableau* $\mathcal{C}^p_{aB,aC}$ *réutilisé les valeurs x d'un autre sous-tableau* $\mathcal{C}^p_{a'B,a'C}$ *avec* $a \neq a'$ *sans nuire à la généralité. L'avantage de cette procédure est de réduire la cardinalité de X.*

Exemple B.1.

		a_1		a_2		
		c_1	c_2	c_1	c_2	c_3
a_1	b_1	1	1	@	@	@
	b_2	1	1	@	@	@
a_2	b_1	@	@	1	1	1
	b_2	@	@	1	0	1

P

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_1	b_2	c_1
a_1	b_2	c_2
a_2	b_1	c_1
a_2	b_1	c_2
a_2	b_1	c_3
a_2	b_2	c_1
a_2	b_2	c_3

R

A	B	X
a_1	b_1	x_1
a_1	b_2	x_1
a_2	b_1	x_1
a_2	b_2	x_1
a_2	b_1	x_2
a_2	b_1	x_3
a_2	b_2	x_3

S

A	C	X
a_1	c_1	x_1
a_1	c_2	x_1
a_2	c_1	x_1
a_2	c_2	x_2
a_2	c_3	x_3

Une autre solution avec card X = 2 aurait été :

R

A	B	X
a_1	b_1	x_1
a_1	b_2	x_1
a_2	b_1	x_1
a_2	b_2	x_1
a_2	b_1	x_2

S

A	C	X
a_1	c_1	x_1
a_1	c_2	x_1
a_2	c_1	x_1
a_2	c_3	x_1
a_2	c_2	x_2

A N N E X E C

PROGRAMME D'EVALUATION D'UN ENSEMBLE

DE RELATIONS FONCTIONNELLES

```
C PROGRAM TO EVALUATE DATA BASE DECOMPOSITION.
      INTEGER CUM(2,50)
      INTEGER DUM(20),NDUM
      INTEGER IMPLIC(2,50),QUERY(2,50),NVAL(50),ATT(50)
      INTEGER EOR,AND,OR
      INTEGER RELAT(10,2,50),NREL(2,50),MSG(3)
      INTEGER ONE,SGN,BIT(32)
      INTEGER MSG2(3)
      INTEGER EQUIV(2,10)
      DATA ZERO/Z00000000/,ONE/Z00000001/,SGN/Z80000000/
      DATA MSG/'*IMP','LLLL','IES*'/
      DATA MSG2/'*EQU','AAAA','LS* '/
      BIT(1)=SGN
      BIT(32)=ONE
      DO 1122 I=1,30
      J=32-I+1
      BIT(J-1)=BIT(J)*2
 1122 CONTINUE
      DO 1123 I=1,32
 1123 WRITE(6,299)I,BIT(I)
  299 FORMAT(I6,2X,Z8)
      READ(5,200)NATT,NQUER,NIMP
      WRITE(6,200)NATT,NQUER,NIMP
      READ(5,202)(ATT(I),NVAL(I),I=1,NATT)
      WRITE(6,202)(ATT(I),NVAL(I),I=1,NATT)
      READ(5,201)((IMPLIC(I,J),I=1,2),J=1,NIMP)
      WRITE(6,201)((IMPLIC(I,J),I=1,2),J=1,NIMP)
      READ(5,201)((QUERY(I,J),I=1,2),J=1,NQUER)
      WRITE(6,201)((QUERY(I,J),I=1,2),J=1,NQUER)
  200 FORMAT(10I6)
  201 FORMAT(6X,Z8,2X,Z8)
  202 FORMAT(6X,A4,2XI6)
C DECODE IMPLICATIONS
      DO 4 N=1,NIMP
      CALL WORDS(N,BIT,RELAT,NREL,ATT,IMPLIC,MSG)
    4 CONTINUE
C PSEUDO-TRANSITIVITY (DELOBEL):
C IF(A.R.B AND C.R.D) THEN OR(A,EOR(B,C)).R.D
   19 NIMPP=NIMP
      DO 10 I=1,NIMPP
      DO 10 J=1,NIMPP
      IF(I.EQ.J) GO TO 10
      MA=OR(IMPLIC(1,I),EOR(IMPLIC(2,I),IMPLIC(1,J)))
      MB=IMPLIC(2,J)
      IF(NBITS(AND(MB,MA)).NE.0)GO TO 10
C CHECK FOR REDUNDANCY
      DO 17 L=1,NIMP
      IF(NBITS(AND(MB,IMPLIC(2,L))).EQ.0)GO TO 17
      N1=NBITS(AND(MA,IMPLIC(1,L)))
      N2=NBITS(IMPLIC(1,L))
      IF(N1.EQ.N2)GO TO 16
      N3=NBITS(MA)
      IF(N1.EQ.N3)GO TO 16
   17 CONTINUE
```

```
C RELATION IS NOT REDUNDANT.
      NIMP=NIMP+1
      IMPLIC(1,NIMP)=MA
      IMPLIC(2,NIMP)=MB
      WRITE(6,205)NIMP,MA,MB
  205 FORMAT(I5,2(2X,Z8))
   18 CONTINUE
C RELATION SUBASSUMES AN EXISTING RELATION
      IMPLIC(1,L)=MA
   16 CONTINUE
   10 CONTINUE
      IF(NIMP.NE.NIMPP)GO TO 19
      WRITE(6,290)
  290 FORMAT('1CLOSURE OF THE GIVEN FUNCTIONAL RELATIONS'//)
      DO 1 N=1,NIMP
      CALL WORDS(N,BIT,RELAT,NREL,ATT,IMPLIC,MSG)
    1 CONTINUE
C OBTAIN CLOSURE IN SUMMARY FORM
      NCUM=1
      DO 21 J=1,2
   21 CUM(J,NCUM)=IMPLIC(J,1)
      DO 20 I=2,NIMP
      DO 22 K=1,NCUM
      IF(NBITS(EOR(CUM(1,K),IMPLIC(1,I)))*EQ.0)GO TO 23
   22 CONTINUE
      NCUM=NCUM+1
      DO 24 J=1,2
   24 CUM(J,NCUM)=IMPLIC(J,I)
      GO TO 20
   23 CUM(2,K)=OR(CUM(2,K),IMPLIC(2,I))
   20 CONTINUE
C FIND EQUIVALENCES ( A'B'B AND B'B'A ).
      NC1=NCUM-1
      NEQ=0
      DO 30 I=1,NC1
      I1=I+1
      NB=NBITS(CUM(1,I))
      DO 31 K=I1,NCUM
      KTEMP=AND(CUM(1,I),CUM(2,K))
      IF(NBITS(KTEMP).NE.NB)GO TO 31
      LTEMP=AND(CUM(2,I),CUM(1,K))
      IF(NBITS(LTEMP).NE.NBITS(CUM(1,K)))GO TO 31
      NEQ=NEQ+1
      EQUIV(1,NEQ)=CUM(1,I)
      EQUIV(2,NEQ)=CUM(1,K)
      CALL WORDS(1,BIT,DUM,NDUM,ATT,EQUIV(1,NEQ),MSG2)
      GO TO 33
   31 CONTINUE
   32 CONTINUE
   30 CONTINUE
      WRITE(6,210)((CUM(J,K),J=1,2),K=1,NCUM)
  210 FORMAT(2X,Z8,2X,Z8)
C FOR EACH PAIR OF FORM A'B'B CHECK IF THERE IS CUM(K) SUCH THAT C'B'A.
C IF SO, DELETE B FROM CUM(K)
      DO 25 I=1,NIMP
```

```
      LTEMP=OR(IMPLIC(1,I),IMPLIC(2,I))
      NB=NBITS(LTEMP)
      DO 26 K=1,NCUM
      KTEMP=AND(LTEMP,CUM(2,K))
      IF(NB.EQ.NBITS(KTEMP))CUM(2,K)=LOR(CUM(1,K),IMPLIC(2,I))
   26 CONTINUE
   25 CONTINUE
      DO 29 N=1,NCUM
   29 CALL WORDS(1,BIT,CUM,NCUM,ATT,CUM(1,N),MSG)
      RETURN
      END
      SUBROUTINE WORDS(N,BIT,RELAT,NREL,ATT,IMPLIC,MSG)
      INTEGER IMPLIC(2,50),ATT(50),BIT(32)
      INTEGER LOR,AND,OR
      INTEGER RELAT(10,2,50),NREL(2,50),FUNCT(50)
      INTEGER MSG(3),BLANK
      DATA BLANK/'    '/
      DO 5 I=1,2
      NCT=0
      DO 6 J=1,32
      IF(NBITS(AND(BIT(J),IMPLIC(I,N))).EQ.0)GO TO 6
      NCT=NCT+1
      RELAT(NCT,I,N)=J
      NREL(I,N)=NCT
    6 CONTINUE
    5 CONTINUE
      NWD=0
      DO 2 I=1,2
      NCT=NREL(I,N)
      DO 3 J=1,NCT
      NWD=NWD+1
      JJ=RELAT(J,I,N)
      FUNCT(NWD)=ATT(JJ)
      NWD=NWD+1
      FUNCT(NWD)=BLANK
    3 CONTINUE
C     WRITE(6,207)N,I,(RELAT(J,I,N),J=1,NCT)
C 207 FORMAT(2I5,12I5)
      IF(I.EQ.2)GO TO 2
      DO 7 J=1,3
      NWDJ=NWD+J
    7 FUNCT(NWDJ)=MSG(J)
      NWD=NWD+3
      NWD=NWD+1
      FUNCT(NWD)=BLANK
    2 CONTINUE
      WRITE(6,203)(FUNCT(J),J=1,NWD)
  203 FORMAT(/1X20(A4,A1))
      RETURN
      END
```

ERRATA

PAGE	LIGNE	A LA PLACE DE	LIRE
2.15	15	ℓ_2 : P,H,X → N	ℓ_2 : P,H,Y → N
3.6	3	E . B = Ø	Z. B = Ø
5.24	3	boucle "pou"	boucle "pour"
5.33	9	NO PARKING est NO-GARAGE	NO-PARKING entier
5.33	14	LIEU est NO-MAISON	LIEU mot
5.33	15	PROPRIETAIRE est NOM	PROPRIETAIRE mot
8.10	14	la relation fonctionnelle I A	la relation fonctionnelle I → A
9.5	bas de page	$F = \sum_{i}^{n} x_j\ y'_j$	$F = \sum_{1}^{n} x_j\ y'_j$
10.2	2	KNHUTH	KNUTH
10.12	2	$\mathfrak{J}_2$ = {H:H ε et et H ∉ $\mathfrak{J}_1$}=	$\mathfrak{J}_2$ = {H:H ε et H ∉ $\mathfrak{J}_1$}= $\mathfrak{J}$ - $\mathfrak{J}_1$
11.8	23	$h = \prod_{i}^{n} a_i + \sum_{i} e_i\ a'_i$	$h = \prod_{1}^{n} a_i + \sum_{i} e_i\ a'_i$
11.9	15	$h = \prod_{i}^{n} a_i + F$	$h = \prod_{1}^{n} a_i + F$
12.5	28	...(X,H,S) x (X,P) ...	...(X,H,S) * (X,P) ...
12.6	5	H,S → X	HS ↔X
12.6	4	= (X,H,X) * ...	= (X,H,S) * ...
12.10	21	l'introduction de deux entités (<A:a>, <B:b>, <C:c>, <D:d> si ...	l'introduction de deux entités (<A:a>, <B:b>, <C:c>) et (<D:d>, <C:c>) si ...
12.11	bas de page	Toutefois les ...	Toutefois les deux notions sont étroitement liées.
13.7	2	paragraphe 11.1.2	paragraphe 12.1.2.

PREMIERE THESE

CONTRIBUTIONS THEORIQUES A LA CONCEPTION ET L'EVALUATION D'UN SYSTEME D'INFORMATIONS APPLIQUE A LA GESTION

DEUXIEME THESE

PROPOSITIONS DONNEES PAR L'UNIVERSITE

par

C. DELOBEL

Thèse soutenue le 17 Octobre 1973 devant la commission d'examen

J. KUNTZMANN Présiden
J. ARSAC Rapporteur
L. BOLLIET
A. HOCQUENGHEM
C. PAIR