Introduction aux bases de données et aux fondements de l'exemple relationnel

An introduction to databases and to the foundations of the relational case

Claude Del Vigna

Édition électronique

URL : <http://journals.openedition.org/msh/2906>

DOI : 10.4000/msh.2906

ISSN : 1950-6821

Éditeur

Centre d'analyse et de mathématique sociales de l'EHESS

Édition imprimée

Date de publication : 1 mars 2005

ISSN : 0987-6936

Référence électronique

Claude Del Vigna, « Introduction aux bases de données et aux fondements de l'exemple relationnel », *Mathématiques et sciences humaines* [En ligne], 169 | Printemps 2005, mis en ligne le 28 mars 2006, consulté le 23 juillet 2020. URL : <http://journals.openedition.org/msh/2906>

INTRODUCTION AUX BASES DE DONNÉES ET AUX FONDEMENTS DE L'EXEMPLE RELATIONNEL

PREMIÈRE PARTIE

Claude DEL VIGNA¹

RÉSUMÉ – *Cet article est en deux parties réparties sur deux volumes de la revue. Son introduction développe un aperçu des principes généraux des bases de données (BD). Le reste du texte est consacré à l'exposé de certains des aspects mathématiques et informatiques des systèmes relationnels de données. Dans l'introduction, les systèmes BD sont présentés comme des outils pour la programmation et la notion de machine BD est introduite. Les chapitres qui suivent décrivent formellement les étapes de la construction du noyau d'une machine BD relationnelle. Le dernier chapitre examine le pouvoir d'expression de l'outil ainsi élaboré. Par son élégante simplicité, les avancées qu'il a favorisées et aussi sa large diffusion, l'exemple relationnel occupe une place essentielle dans le domaine des BD dont il est, en quelque sorte, représentatif des méthodes. Il couvre un vaste champ de connaissances théoriques et pratiques dont l'article ne présente qu'une partie limitée.*

MOTS-CLÉS – Bases de données, Logique, Modèle relationnel

ABSTRACT – *An introduction to databases and to the foundations of the relational case. This paper is divided between two volumes of the review. The introduction presents some of the general principles of database (DB) systems. The rest of the text addresses certain mathematical and technical aspects of the relational DB systems. In the introduction, DB systems are described as programming tools and the notion of DB machine is introduced. The following chapters formally describe the steps to design a skeleton of a relational DB machine. The last chapter examines the expressive power of the tool previously described. Because of its elegant simplicity, the advances it made possible, and its wide diffusion, the relational case has a special place in the DB discipline. It is, in some ways, representative of the DB methods. This article presents only a limited part of the wide range of theoretical and practical knowledge covered by the relational case.*

KEYWORDS – Databases, Logic, Relational model

Il est assez fréquent qu'on prête aux bases de données des vertus qu'elles n'ont pas et, simultanément, on ignore celles qu'elles ont. *Données*, fichiers de *données*, traitements statistiques des *données*, ensemble de *données*, analyse des *données*, fouille de *données*, classification des *données*, système de bases de *données*, ... Il règne souvent, dans les disciplines autres que l'informatique, une assez grande confusion entre toutes ces expressions qui partagent le mot *données*, confusion qui alimente parfois un syncrétisme bien éloigné de la réalité des choses. Aussi, en préambule, une analogie

¹ Centre d'analyse et de mathématiques sociales, 44 rue de l'Amiral Mouchez 75014 Paris, delvigna@ivry.cnrs.fr

peut-elle être utile pour fixer les frontières, au risque de la caricature. Un menuisier construit des armoires. Il doit les concevoir de façon à ce qu'il ne soit pas nécessaire de totalement les vider pour prendre un simple mouchoir ! Toutefois, au-delà de cette exigence, il n'est pas responsable de comment les vêtements sont utilisés, encore moins de l'élégance vestimentaire de ses clients. Le menuisier construit des meubles de rangement. Ainsi en est-il des bases de données : leur premier objet est le rangement et l'accès. On souligne cependant que certains autres thèmes, qui ne sont pas abordés ici, comme la fouille de données, occupent une place grandissante dans la discipline bien que *stricto sensu* ils l'éloignent du rangement et de l'accès.

0. INTRODUCTION

Un système de gestion de bases de données (SGBD) est un instrument informatique pour la conservation et la gestion d'informations. C'est un instrument générique grâce auquel on construit des bases de données (BD) particulières, de même qu'un langage de programmation est un instrument générique grâce auquel on écrit des programmes particuliers. L'objet principal de cet article est de décrire les fondements des SGBD dits *relationnels*, ceux mis en œuvre dans les logiciels tels ORACLE, MySQL ou ACCESS, pour ne citer qu'eux. Cependant, la présente introduction est plus générale. Elle présente des notions et énonce des principes qui s'appliquent au rangement d'informations en général et à leur accès quelle que soit la nature des SGBD. Elle est en cinq paragraphes. Dans le premier, §0.1, on développe la similitude des SGBD avec les machines programmables usuelles tout en s'interrogeant sur leurs spécificités. À la suite de ce paragraphe, on parlera d'ailleurs plutôt de machine « bases de données » (machine BD) que de SGBD, machine dont le paragraphe suivant, §0.2, dresse un bref cahier des charges. La description d'une machine BD, comme instrument, peut être appréhendée en distinguant, dans sa constitution, trois niveaux : le niveau *interne* ou *physique* (proche des technologies de bas niveau des ordinateurs porteurs), le niveau *logique* ou *conceptuel* (des structures formelles à la base du fonctionnement de la machine), le niveau *externe* (qui individualise le niveau logique selon les catégories d'utilisateurs). Le paragraphe 0.3 présente ces niveaux, explique leur différence et énonce l'exigence de leur autonomie. Le paragraphe 0.4 introduit la notion de *modèle de données*. Un modèle de données caractérise le niveau logique d'une machine BD et constitue l'élément central pour la conception de la machine. Les SGBD se distinguent selon leur modèle de données. Différents modèles ont été proposés et implantés, parmi lesquels les modèles hiérarchique, réseau, relationnel et objet. Le modèle relationnel selon lequel les informations sont rangées dans des tables est introduit au paragraphe 0.5 qui développe, à son propos, la notion d'abstraction calculable et ses exigences en termes d'optimisation. Enfin, le dernier paragraphe, §0.6, est le plan de la suite de l'article. Pour l'ensemble des notions succinctement présentées dans cette introduction, on pourra consulter [Delobel, Adiba, 1982 ; Date, 1998 ; Gardarin, 1983 ; Ullman, 1988] pour des compléments plus détaillés.

0.1 SGBD ET MACHINES BD

Les SGBD sont des systèmes informatiques conçus d'une part pour stocker des données, d'autre part pour les consulter sélectivement, le plus aisément et le plus rapidement possible. Cette définition traduit une idée très statique des SGBD – à la manière de ce que sont les dictionnaires de langue, le Littré, les Larousse ou le Grand Robert – idée qu'illustre parfaitement la définition de Date [1998, p. 3] : « Un système

de base de données n'est essentiellement rien de plus qu'un système de conservation d'enregistrements informatiques... un classeur électronique. ». Or, l'expérience pratique d'applications complexes prouve qu'elles sont utilisées de façon bien plus dynamique. Le contenu des bases de données y est enrichi, appauvri, modifié, à l'instar du contenu des variables d'un programme qui évolue avec l'exécution de celui-ci. Dès lors, dépassant la statique de la conservation et de la consultation, un SGBD apparaît comme une véritable machine programmable et son utilisation comme une activité de programmation à part entière. Ullman [1988, p. 1] écrit d'ailleurs que : « les SGBD sont une importante catégorie de systèmes pour la programmation ... ». De fait, les SGBD doivent disposer d'une mémoire pour conserver les informations et d'un langage pour les décrire, y accéder et les modifier. Ils disposent donc des éléments constitutifs de toute machine programmable, à savoir une mémoire et un langage, celui-ci ayant pour objet, *via* un programme, de modifier l'état courant de celle-là. Visant ici à insister sur l'aspect *programmation* des SGBD, on les nommera plutôt *machine* « bases de données » (machine BD), à l'instar des langages de programmation à propos desquels on parle de machines – machines Lisp, Java ou encore Prolog.

Dès lors, quelles sont les différences entre une machine BD et une machine d'un langage de programmation classique ? On en retiendra trois. La première est que la mémoire d'une machine BD doit être *persistante* – par opposition aux mémoires centrales qui sont volatiles – autrement dit, son contenu doit demeurer après arrêt des programmes ou extinction de l'ordinateur et se retrouver inchangé à l'allumage. La deuxième est que la partie « données » est, en général, incomparablement plus volumineuse en bases de données qu'en programmation de mémoire centrale, des milliards, voire beaucoup plus, d'octets dans le premier cas, comparés à des milliers ou des centaines de milliers dans le second – le sigle d'une des grandes conférences du domaine n'est-il pas VLDB (*Very Large Data Bases*). La troisième enfin est que les langages des machines BD sont en général limités.

La première différence est liée à la persistance. La technicité de la persistance est essentiellement fondée sur les disques durs, ce que n'infirmant pas, sur le fond, ni la place de plus en plus grande prise par les mémoires centrales dans les SGBD suite à l'accroissement considérable de leur capacité, ni l'émergence de mémoires vives persistantes (les clés-mémoire, parfois appelées *mémoires flash*) basées sur la technique EEPROM². Or, les disques durs sont lents : l'accès à une donnée sur un disque dur se mesure en millisecondes alors qu'elle l'est en nanosecondes pour une mémoire centrale [Prieur, 2002]. Les disques durs sont donc de l'ordre de 10⁶ fois plus lents que les mémoires centrales. À l'opposé du « tout électronique » de celles-ci, ils contiennent des parties mécaniques (bras de lecture, disque rotatif) qui expliquent leur lenteur relative. Il n'est donc pas possible d'y transposer sans adaptation les techniques de programmation de mémoire centrale, car les temps d'exécution seraient multipliés par un facteur non négligeable, même si, en pratique, il est bien inférieur aux 10⁶ précédents, facteur « brut » qui ne tient pas compte des techniques logicielles et matérielles pour accélérer les accès, comme les mémoires-cache ou la pagination.

La deuxième différence, liée au volume des données, conduit à une pratique adaptée de la programmation. On n'engranger pas des milliards d'octets pour se rendre compte à la fin qu'il aurait fallu les organiser autrement ! La presse « grand public » fait

² Cf. le site <http://www.webopedia.com>

d'ailleurs état du nouveau métier d'*administrateur de données*³. Les masses importantes de données et les temps de calcul qu'elles induisent réclament beaucoup de prudence, voire une certaine austérité. C'est peut-être la raison pour laquelle on ne trouve pas dans la littérature des bases de données de ces casse-têtes ludiques que les mathématiques et la programmation aiment à proposer [Arsac, 1985], qui mobilisent des heures durant pour aboutir à quelques compendieuses lignes de C++ ou de Prolog. Les exemples de la littérature « bases de données » sont, il faut bien le reconnaître, plus besogneux : les pièces détachées d'une usine, les rouages d'une administration, ... même si certains auteurs font l'effort d'exemples plus festifs, à l'instar de ceux présentés par Gardarin [1983], liés à l'œnologie. On tentera ici un jeu mathématique dans l'intention qu'il éclaire sur la programmation BD (cf. Exemples 2.2 et 3.7).

La dernière différence est liée au langage. L'objectif premier du langage d'une machine BD est de permettre l'extraction sélective des informations stockées et leur composition. Dès lors, doit-il être aussi puissant qu'un langage de mémoire centrale ? Est-il, par exemple, essentiel qu'il permette de programmer des questions comme *quel sont les livres dont le n° d'enregistrement est le produit de trois nombres premiers impairs* ? Ou bien peut-on se limiter ? Et comment ? La réponse à ces questions n'est pas unanime. Le modèle relationnel a limité son langage ; [Aho, Ullman, 1979] ont présenté et étudié la limitation réalisée. À l'inverse, [Atkinson, Buneman, 1987] ont plaidé la nécessité que les langages BD aient la même puissance que les langages de programmation usuels. Ainsi en est-il du langage du SGBD objet O₂ [Adiba, Collet, 1993, p. 79-80]. Du côté de la limitation, on est dans une situation qui n'est pas sans analogie avec la théorie des grammaires [Aho, Ullman, 1973] considérée sous l'angle de la programmation des chaînes de caractères. Les grammaires de type 0 ont la puissance des machines de Turing. Mais ce sont de piètres outils de programmation, de même celles de type 1 (sous contexte). En revanche, celles de type 2 et 3 (hors-contexte et régulières) en sont d'excellents. Certes limités, mais qui font bien, simplement et de façon fiable ce qu'ils sont capables de faire. Il y a là affaire de compromis entre ce qui est utile et souhaitable et la puissance d'expression. Par ailleurs, ainsi que le note Ullman [1999, p. 54-55], à propos du modèle relationnel, limiter le langage permet de résoudre de façon satisfaisante la question des optimisations, dont l'importance essentielle est évoquée plus avant (cf. §0.5). Les bases de données relationnelles ont donc limité leurs langages à l'expression de sélections raisonnables et utiles. Lorsque des traitements plus élaborés s'avèrent nécessaires, comme c'est fréquemment le cas, ils sont dévolus à un langage de mémoire centrale. Celui-ci utilise le langage BD pour acheminer du disque dur vers la mémoire centrale les données que réclament les traitements visés. Le langage de mémoire centrale est alors dit *hôte* du langage BD, lequel est, en retour, qualifié d'*enchâssé*.

0.2 CAHIER DES CHARGES D'UNE MACHINE BD

Le cahier des charges ci-dessous est succinct. Il présente les fonctionnalités nécessaires d'une machine BD. À l'exception de la première, les autres ne sont pas examinées dans cet article. Il est toutefois indispensable de les mentionner sous peine d'une vision tronquée. Une machine BD doit :

³ Magazine *Programmez*, n° 46, octobre 2002, p. 46.

- disposer des moyens pour la description des données, leur accès et leur stockage efficaces ;
- disposer des moyens pour assurer la cohérence des données stockées. Exemple (parmi les plus simples) : exclure la valeur 10000 pour l'âge d'une personne ;
- pouvoir accepter la connexion concurrente, c'est-à-dire simultanée, de plusieurs programmeurs ou usagers sur la même base de données ;
- assurer la sécurité, notamment en disposant des moyens pour limiter les accès selon la catégorie de programmeurs ou d'usagers (cf. §0.3 pour un exemple simple) ;
- être en mesure, en cas de panne, de rétablir les bases dans un état cohérent. Un cas typique d'incohérence causée par une panne est celle que provoquerait l'arrêt inopiné de l'ordinateur d'un établissement bancaire qui surviendrait après le débit d'un compte mais avant le crédit du compte bénéficiaire. La somme débitée n'ayant pas été créditée, la base de données se trouverait dans un état incohérent qu'il faut pouvoir ramener dans un état cohérent.

0.3 NIVEAUX DE DESCRIPTION DES MACHINES BD

Les moyens dont dispose une machine BD sont usuellement classés en trois niveaux : le niveau qualifié d'*interne* ou de *physique* regroupe les moyens pour le stockage physique des données ; le niveau dit *logique* ou *conceptuel* regroupe ceux pour la description des données comme elles sont perçues de l'« extérieur » et pour leur manipulation ; enfin, le niveau *externe* regroupe les moyens pour particulariser le niveau logique selon les catégories d'usagers. Un exemple, rapporté aux informations, fixera les rôles et les différences. On considère le cas d'informations consistant en « *un ensemble de couples (nom, prénom), stockés dans un fichier séquentiel et trié* ». On suppose de plus que « *les agents de l'État Civil sont autorisés à accéder à toutes les informations, mais le grand public uniquement aux prénoms* ». La première partie de cet énoncé, « *un ensemble de couples (nom, prénom)* », relève du niveau logique (elle décrit les informations telles qu'en elles-mêmes), la deuxième partie, « *stockés dans un fichier séquentiel et trié* », relève du niveau interne (elle décrit comment les informations sont stockées). Quant aux précisions relatives à l'accès, « *les agents de l'État Civil sont autorisés à ...* », elles relèvent du niveau externe (elles indiquent à quelle part du niveau logique chaque catégorie d'usagers peut accéder). Une machine BD se décrit par les moyens dont elle dispose pour exprimer et mettre en œuvre chacun de ces niveaux. Le niveau logique d'une machine BD est fondé sur un *modèle de données* (cf. §0.4). Son niveau physique mobilise des technologies de bas niveau, algorithmiques, de structures de données et matérielles (index, mémoire-cache, disques durs performants, etc). Enfin, son niveau externe est fondé sur un mécanisme particulier, celui des *vues*. Seuls les niveaux logique et interne des machines relationnelles sont présentés dans cet article.

Le principe d'autonomie des niveaux logique et physique, connu sous le nom de *principe d'indépendance physique*, est un guide essentiel dans la conception d'une machine BD. Il postule que la machine BD doit permettre d'accéder aux données en ne tenant compte que de leur description logique, donc en faisant abstraction des particularités de leur stockage physique. Autrement dit, l'expression des accès aux informations et de leur traitement ne doit dépendre que de la nature intrinsèque des informations, donc du niveau logique, pas de leur mode de stockage, donc pas du niveau interne. La machine doit être en mesure d'adapter automatiquement l'algorithmique d'un traitement à différents stockages physiques sans qu'il soit nécessaire de modifier la programmation dudit traitement. Poursuivant l'exemple présenté ci-dessus, la machine

respectera le principe d'indépendance physique, si la question *existe-t-il un couple (martin, arthur)* conduit automatiquement à une recherche dichotomique puisque les enregistrements sont triés et conduirait à une recherche séquentielle s'ils ne l'étaient pas. À travers l'autonomie des niveaux logique et physique, l'objectif visé est la réduction, idéalement à aucune, des tâches de programmation lorsque le stockage physique est modifié. Il ne peut être atteint que si le niveau logique est assez abstrait pour ne pas faire intervenir des considérations liées au stockage. Un second principe d'autonomie existe : entre les niveaux logique et externe. Il n'est pas présenté ici.

0.4 MACHINES BD ET MODÈLES DE DONNÉES

Le niveau logique d'une machine BD est construit sur un *modèle de données*. La notion de modèle est ici différente de celle que définit la *théorie des modèles* ou bien encore de celle que sous-tend l'expression *modèle en Sciences Sociales*. En théorie des modèles, un « cas concret » est modèle d'une théorie [Bridge, 1977]. En Sciences Sociales le rapport est inversé puisqu'un modèle est une abstraction qui vise à être au plus près d'un fonctionnement observé, ou encore, comme l'écrit Israel [1995], « une représentation en langage mathématique d'un aspect de la réalité ». En bases de données, la notion de modèle est encore différente. Comme il a déjà été dit, il en existe de plusieurs sortes parmi lesquelles les modèles hiérarchique, réseau, relationnel ou objet. Un modèle de données est constitué de deux volets distincts bien qu'étroitement liés : d'abord une structure, ensuite – et en fonction de la structure – un langage. On peut envisager la structure et le langage d'un modèle de données de deux manières complémentaires selon qu'on les rapporte aux informations ou bien aux machines BD. Rapportée aux informations, la structure explicite la forme sous laquelle les informations doivent être représentées ; cette forme est un arbre dans le modèle hiérarchique, est proche d'un graphe dans le modèle réseau, est une table dans le modèle relationnel, est une structure appelée *objet* dans le modèle du même nom. Rapportée aux machines BD, la structure explicite la nature des données que la mémoire de la machine est conçue pour stocker : des arbres, des graphes, des tables ou des objets. Les deux manières sont bien entendu complémentaires puisqu'en mettant les informations sous la forme prescrite par le modèle, elles deviennent stockables dans toute mémoire fondée sur ledit modèle. Quant au langage, rapporté aux informations, son rôle est de permettre le calcul de nouvelles données à partir de données existantes. Rapporté aux machines BD, son rôle s'augmente des capacités de modifier le contenu de la mémoire. On souligne enfin qu'un modèle de données est de fait un « carcan » puisqu'il impose la forme sous laquelle les informations doivent être transformées pour devenir des données et être ainsi stockables. Ce carcan oblige d'ailleurs parfois à quelques contorsions *ad hoc* lorsque les informations s'y adaptent mal.

0.5 MACHINES BD RELATIONNELLES

Les machines relationnelles sont conçues à partir du modèle relationnel proposé par Codd [1970]. Au-delà de sa spécificité, le modèle est très représentatif de la manière « bases de données ». Son apport fut celui d'une structure simple, les tables, pour la description et le stockage des informations et une limitation naturelle du pouvoir d'expression des langages. Le modèle est, par ailleurs, une des premières expériences de programmation déclarative. Un langage *déclaratif* est un langage de programmation, en général basé sur une abstraction mathématique, qui permet d'exprimer les buts sans se soucier de la manière de les atteindre [Ullman, 1999]. Par opposition, un langage est *impératif* (on dit aussi *procédural*) si toutes les étapes pour atteindre un but, autrement

dit son algorithmique, doivent être précisées. Ce qui ne signifie nullement que les phrases d'un langage déclaratif ne « contiennent » pas d'algorithmique mais que celle-ci y est implicite en ce sens qu'il existe une algorithmique canoniquement associable à chacune de ses phrases. On utilisera le néologisme d'*abstraction calculable*. En général, une perte d'efficacité accompagne une telle abstraction car son algorithmique canonique se révèle lente ou grande consommatrice d'espace-mémoire, ou les deux à la fois. Elle doit donc impérativement être optimisée. Ainsi, abstraction calculable et optimisation apparaissent-elles comme deux moments *distincts* mais *inséparables*. Abiteboul, Hull, Vianu [1995] notent d'ailleurs en substance, p 4, que le modèle relationnel ne se diffusa largement qu'à partir du moment où ses techniques d'optimisation furent suffisamment efficaces. Les bases de données relationnelles sont exemplaires de l'articulation abstraction/optimisation. Elles en couvrent de nombreux aspects. Leur abstraction calculable s'appuie d'une part sur la notion de structure algébrique relationnelle [Bridge, 1977 ; Ebbingaus, Flum, 1999] et, d'autre part, sur le langage du 1^{er} ordre associé. Par ailleurs, son algorithmique canonique s'exprime aussi bien au niveau d'un langage algorithmique qu'au niveau, plus abstrait, d'une collection finie d'opérateurs, appelée *algèbre relationnelle*. L'optimisation relève de la transformation formelle de programmes en liaison avec des structures de stockage et des technologies matérielles adaptées aux disques durs. L'ensemble constitue une machine BD relationnelle, moins puissante qu'une machine de Turing et dont la puissance peut être caractérisée.

0.6 PLAN DU TEXTE

La suite du texte est consacrée à l'abstraction calculable propre au modèle relationnel, à son optimisation et à l'examen de ses limites. Elle expose les chemins mathématique et informatique jusqu'aux marches de la réalisation logicielle d'un noyau de machine BD relationnelle. L'article est en cinq chapitres répartis sur deux volumes de la revue ; le présent volume contient les trois premiers. Le Chapitre 1 présente les préliminaires nécessaires. Il propose, en particulier, une définition simple, mais suffisante pour le propos, des notions de mémoire, de stockage, de supports et de données. Le Chapitre 2 définit formellement la composante mémoire de la machine BD et les primitives pour sa modification. Le Chapitre 3 montre comment le langage de la logique du 1^{er} ordre conduit à un langage pour la machine BD. La notion de *requête indépendante* est la notion centrale de ce chapitre. Elle circonscrit le pouvoir d'expression à atteindre. Il s'agit d'une notion sémantique. Il est montré d'une part qu'elle n'a pas de définition algorithmique mais, d'autre part, que son pouvoir d'expression peut être approché syntaxiquement. L'approche syntaxique présentée est naturelle mais lourde et inefficace. Revenant sur elle, le Chapitre 4 en présente une seconde, plus efficace, décrit son algorithmique canonique et traite de son optimisation. Le Chapitre 5 introduit enfin l'examen du pouvoir d'expression des machines BD relationnelles. Finalement, la conclusion évoque, à partir de l'expérience des Sciences Humaines et Sociales, les modèles de données faiblement (ou semi-) structurés qui visent une souplesse plus grande que celle du modèle relationnel et répondent probablement mieux à leurs besoins. L'approche du modèle relationnel développée ici diffère sur certains éléments des approches usuelles, en particulier de celle d'Abiteboul *et al.* [1995]. Une annexe est adjointe, consacrée à l'examen des différences.

On souligne enfin que l'article, qui se concentre sur les aspects de base du modèle relationnel, laisse inabordés de nombreux thèmes liés aux bases de données. En particulier, il ne présente aucun des langages utilisés en pratique, tout en les mentionnant parfois. Que ce soit SQL, langage de référence en matière relationnelle, ou

Datalog, à visée plus formelle, pour ne citer qu'eux. Par ailleurs, bien que les SGBD existants aient augmenté le noyau relationnel de nombreuses autres capacités, indispensables pour simplifier la programmation des applications, ces extensions sont hors du cadre de ce texte.

Nota bene : les mots *question*, *requête* et *interrogation* seront souvent utilisés dans le texte. Ils n'y sont pas synonymes. Le premier est utilisé dans son acception courante et quotidienne : « demande à l'effet de s'informer de quelque chose » (Littré). Les deux derniers sont au contraire définis formellement, *requête* est une notion syntaxique, *interrogation* une notion sémantique (pour les deux, cf. Chapitre 3). De même, *information* et *donnée* ne sont pas synonymes, le premier est du langage courant – il a d'ailleurs été souvent utilisé comme tel dans cette introduction – tandis que le second est défini formellement (cf. Chapitres 1 et 2).

CHAPITRE 1

1. PRÉLIMINAIRES

Si ces préliminaires commencent par le rappel de quelques unes des notions parmi les plus connues de la théorie des ensembles, c'est dans le but unique d'en préciser les cas « extrêmes », liés à la vacuité des ensembles considérés (cf. Commentaire 1.1). On rappelle qu'une *application* d'un ensemble E dans un ensemble F est un sous-ensemble f du produit $E \times F$ tel que pour tout x de E , il existe un unique élément y de F pour lequel le couple (x, y) appartient à f , cet unique élément étant noté comme toujours $f(x)$. On note F^E l'ensemble des applications de E dans F . Il résulte de la définition, d'une part, que, pour tout ensemble F , y compris s'il est vide, l'ensemble $F^\emptyset$ des applications de l'ensemble vide dans F est un singleton dont le seul élément est le sous-ensemble vide du produit $E \times F$, en d'autres termes l'application vide, qu'on notera $\langle \rangle$ – ainsi, $F^\emptyset = \{\langle \rangle\}$ – et, d'autre part, que si E n'est pas vide, $\emptyset^E = \emptyset$.

Si f est un élément de F^E et si E' est un sous-ensemble de E alors la *restriction* de f à E' est l'application de E' dans F constituée du sous-ensemble $\{(x, y) \mid x \in E' \text{ et } (x, y) \in f\}$. On la note $f \downarrow E'$. On a $f \downarrow \emptyset = \langle \rangle$ y compris si $f = \langle \rangle$. Par ailleurs, soient E' et E'' des sous-ensembles *disjoints* de E . Si f est un élément de $F^{E'}$ et g un élément de $F^{E''}$, on note $f \uplus g$ l'élément de $F^{E' \cup E''}$ égal à l'union de f et de g . Pour tout élément f de F^E et tout sous-ensemble E' de E , on a $f = (f \downarrow E') \uplus (f \downarrow (E - E'))$. Enfin, soient E , F et G des ensembles tels que $F^E \neq \emptyset$ et $G^F \neq \emptyset$. Si f est un élément de F^E et g un élément de G^F , alors il existe un unique élément h de G^E tel que :

$$\forall x \in E, \forall z \in F, \forall y \in G, [(x, z) \in f \ \& \ (z, y) \in g] \Rightarrow (x, y) \in h.$$

L'application h est la *composée* de g et de f , notée $g \circ f$. On souligne que la propriété ci-dessus est encore vraie lorsque E est vide, auquel cas F^E et G^E sont égaux au singleton $\{\langle \rangle\}$. Dès lors, pour tout g dans G^F , $g \circ \langle \rangle = \langle \rangle$, y compris lorsque F est vide, à savoir $\langle \rangle \circ \langle \rangle = \langle \rangle$. Par ailleurs, $\langle \rangle \circ f$ n'est défini que si $f = \langle \rangle$, c'est-à-dire lorsque $E = F = \emptyset$.

Les opérations $\downarrow$, $\uplus$ et $\circ$ s'étendent à des ensembles d'applications. Si $A \subset F^E$ et $E' \subset E$, $A \downarrow E' = \{a \downarrow E' \mid a \in A\}$. Si $A \subset F^{E'}$, $B \subset F^{E''}$ et si E' et E'' sont *disjoints*, alors $A \uplus B = \{a \uplus b \mid a \in A, b \in B\}$. En particulier, si A est vide ou B est vide, alors $A \uplus B = \emptyset$. Enfin, si $A \subset F^E$ et $B \subset G^F$, alors $A \circ B = \{a \circ b \mid a \in A, b \in B\}$. Les règles ci-dessous sont utiles dans la suite :

(1.1a) si $A \subset F^E$, alors $\forall E' \subset E$ (y compris $E' = \emptyset$), $(A = \emptyset) \Leftrightarrow (A \downarrow E' = \emptyset)$;

(1.1b) si E' et E'' sont disjoints, $A_1 \subset F^{E'}$, $A_2 \subset F^{E'}$, $B_1 \subset F^{E''}$ et $B_2 \subset F^{E''}$, alors :

$$(A_1 \uplus B_1) \cap (A_2 \uplus B_2) = (A_1 \cap A_2) \uplus (B_1 \cap B_2).$$

Pour tout ensemble E et pour tout entier $n \geq 1$, on note $E^{[n]}$ l'ensemble des applications de l'intervalle $[1, n]$ des entiers naturels dans E et on étend la notation au cas $n = 0$ en posant $E^{[0]} = E^\emptyset = \{\langle \rangle\}$. On appelle *n-uplet* ou *uplet d'arité $n \geq 0$* sur E tout élément de $E^{[n]}$. On le note $\langle t_1, t_2, \dots, t_n \rangle$ lorsque n est > 0 et, comme déjà mentionné, $\langle \rangle$ lorsque $n = 0$. On désigne enfin par $E^{[*]}$ l'union des ensembles $E^{[n]}$ pour $n \geq 0$. Un élément de $E^{[*]}$ est appelé *uplet sur E* sans que son arité soit précisée. Si $F \subset E$, alors tout uplet sur F peut être canoniquement regardé comme un uplet sur E . Si δ est une application de E dans un ensemble F et $t = \langle t_1, t_2, \dots, t_n \rangle$ un uplet sur E , alors $\delta \circ t = \langle \delta(t_1), \delta(t_2), \dots, \delta(t_n) \rangle$ est un uplet sur F (cf. Figure 1). Si t est un uplet d'arité $n \geq 0$ sur E , l'*image* de t est le sous-ensemble $t([1, n])$ de E si $n > 0$ et $\emptyset$ si $n = 0$. Enfin, on désigne par $\underline{\mathbf{2}}^E$ l'ensemble des ensembles *finis* d'uplets de même arité sur E . Un élément non vide de $\underline{\mathbf{2}}^E$ est de l'arité de ses uplets. L'élément $\emptyset$ de $\underline{\mathbf{2}}^E$ est de l'arité souhaitée.

Figure 1. composition $\delta \circ t$

Par ailleurs, on appellera ici *mémoire* la donnée d'un couple $(\mathbf{S}, \mathbf{D})$ d'ensembles dénombrables, le premier étant appelé l'ensemble des *supports* et le second l'ensemble des *données*. Un *stockage* dans une mémoire $(\mathbf{S}, \mathbf{D})$, ou encore *état* de la mémoire $(\mathbf{S}, \mathbf{D})$, est alors un couple (S, σ) dans lequel S est un sous-ensemble *fini* de $\mathbf{S}$ et σ une application de S dans $\mathbf{D}$. On dira que chaque élément s de S *stocke* la donnée $\sigma(s)$.

Enfin, on supposera connues les notions essentielles de la théorie de la calculabilité. Aussi se limite-t-on à rappeler informellement :

1. qu'une application f d'un ensemble dénombrable E dans un ensemble dénombrable F est *calculable* si, moyennant l'existence d'un codage approprié des éléments de E et de F , un moyen mécanique, i.e. une *machine de Turing* ou encore un *algorithme*, existe qui calcule $f(x)$ pour tout x de E ,
2. qu'on qualifie de *décidable* une application calculable à valeur dans un ensemble à deux éléments *vrai* et *faux* – on parle alors de *problème décidable* – et,
3. qu'un sous-ensemble d'un ensemble dénombrable est *récuratif* si sa fonction caractéristique est calculable. Enfin, une *correspondance de Post* (CP) est la donnée de deux uplets de même arité u et v sur l'ensemble des mots non vides sur l'alphabet $\{0, 1\}$. Une suite finie d'indices (i_j) , $1 \leq j \leq m$, est une solution d'une CP (u, v) si la concaténation $u(i_1) u(i_2) \dots u(i_m)$ est égale à la concaténation $v(i_1) v(i_2) \dots v(i_m)$. Le

problème, dit de correspondance de Post, qui consiste à savoir si une CP admet une solution est indécidable (cf. [Rogers, 1967] pour une approche approfondie de ces notions).

Commentaire 1.1. Une attention particulière a été portée dans ce chapitre des préliminaires à l'examen des cas liés à la vacuité : application vide, uplet d'arité 0, opérandes vides. Ces cas apparaissent fréquemment dans les chapitres qui suivent. Ils réclament une rigueur d'autant plus pointilleuse qu'elle n'est souvent ni naturelle ni intuitive. L'exemple suivant, décrit par Van Gelder, Topor, [1991], Quass *et al.* [1995], Abiteboul *et al.* [1995] en

montre *a contrario* la nécessité. Il s'agit d'un faux ami du langage SQL selon lequel il apparaît, si on n'y prend garde, que la disjonction logique ne correspond pas à l'union ensembliste ! Les requêtes SQL de la Figure 2 montrent le faux ami. Les symboles A , B et C y désignent trois relations à une colonne. Il suffira de considérer ici qu'il s'agit de trois ensembles. La requête (a) utilise une disjonction (**or**) là où la requête (b) utilise une union ensembliste (**union**). Le langage SQL se présentant, et étant souvent présenté, comme l'implantation de la logique du 1^{er} ordre, ces requêtes devraient être équivalentes et correspondre à l'ensemble $A \cap (B \cup C)$. Il suffit de remplacer les notations telle « $A.x$ » par $x \in A$ et le signe « $=$ » par une conjonction. Or, elles ne le sont pas car la sémantique de SQL n'est pas uniquement liée à la logique du 1^{er} ordre, mais aussi au produit cartésien. Ainsi, selon ladite sémantique, l'évaluation de la requête (a) est l'ensemble $T = \{x \mid \exists y \in C, \langle x, x, y \rangle \in A \times B \times C \vee \exists y \in B, \langle x, y, x \rangle \in A \times B \times C\}$ et celle de la requête (b) est l'ensemble $T' = \{x \mid \langle x, x \rangle \in A \times B\} \cup \{x \mid \langle x, x \rangle \in A \times C\}$. Pour des raisons qui relèvent des vacuités de B et de C , les ensembles T et T' ne sont pas toujours égaux. Ils le sont si B et C sont simultanément vides ou simultanément non vides. Mais si B est vide et C non vide, (ou encore B non vide et C vide) l'évaluation de la requête (a) est vide, celle de la requête (b) est $A \cap C$ (ou encore $A \cap B$). Il s'agit bien d'un faux ami lié aux cas extrêmes, d'autant plus dangereux qu'il est peu connu et pas toujours présenté.

select	$A.x$	select	$A.x$
from	A, B, C	from	A, B
where	$A.x = B.x$	where	$A.x = B.x$
or	$A.x = C.x$	union	
		select	$A.x$
		from	A, C
		where	$A.x = C.x$
(a)		(b)	

Figure 2. Un faux ami SQL

CHAPITRE 2

2. MÉMOIRE RELATIONNELLE ET BASES DE DONNÉES

2.1 DÉFINITIONS

On considère le quadruplet $M = (\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$ dans lequel :

- $\mathbf{R}$ est un ensemble dénombrable, appelé *ensemble des symboles relationnels* ;
- Ω est un élément distingué de $\mathbf{R}$;

- *arité* est une application qui associe à tout symbole relationnel un entier ≥ 0 , de sorte que $\text{arité}(\Omega) = 1$ et que, pour tout $n \geq 0$, il existe une infinité de symboles relationnels d'arité n ;
- $\mathbf{D}$ est un ensemble dénombrable, appelé *domaine des valeurs*, disjoint de $\mathbf{R}$.

L'ensemble $\mathbf{D}$ étant dénombrable, l'ensemble $2^{\mathbf{D}}$ des ensembles finis d'uplets de même arité sur $\mathbf{D}$ l'est aussi. Le couple $(\mathbf{R}, 2^{\mathbf{D}})$ est donc une mémoire. On appelle *stockage relationnel* sur le quadruplet M tout stockage (S, Δ) sur la mémoire $(\mathbf{R}, 2^{\mathbf{D}})$ – S est donc un sous-ensemble *fini* de $\mathbf{R}$ et Δ est une application de S dans $2^{\mathbf{D}}$ (voir Chapitre 1) – tel que :

- $\Omega \in S$;
- pour tout élément R dans S , $\Delta(R)$ est un élément de $2^{\mathbf{D}}$ dont l'arité est celle de R , i.e. $\Delta(R)$ est un sous-ensemble *fini* de $\mathbf{D}^{[\text{arité}(R)]}$;
- $\Delta(\Omega) \neq \emptyset$.

On donne au quadruplet $M = (\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$ le nom de *mémoire relationnelle* ; l'ensemble $\mathbf{R}$ est celui de ses *supports* et l'ensemble $2^{\mathbf{D}}$ celui de ses *données*⁴. Un stockage relationnel (S, Δ) sur une mémoire relationnelle M est plus communément appelé *base de données* sur M . L'ensemble S est appelé *schéma* de la base et l'application Δ *réalisation*⁵ du schéma S ou encore *base de données sur S* . On dira de chaque ensemble $\Delta(R)$, $R \in S$, qu'il est une *réalisation* du symbole R , ou bien que R *stocke* $\Delta(R)$. Le rôle du symbole Ω est uniquement technique. Selon la définition ci-dessus, sa réalisation n'est jamais vide. On appellera cette contrainte la *contrainte- Ω* . Elle est inhabituelle dans la littérature des bases de données. Les raisons de sa présence ici, sont présentées dans la glose 2.6 et surtout dans l'annexe. On pourra par ailleurs supposer que la réalisation de Ω est un singleton $\{\omega\}$, ω étant un élément distingué de $\mathbf{D}$. Dans toute la suite, lorsqu'on parlera d'un schéma à k symboles, k désignera le nombre de ceux qui sont différents de Ω . Ainsi, un schéma à deux symboles en contient-il en fait trois si on compte Ω . Finalement, on note $\mathcal{B}(M)$ l'ensemble des bases de données sur une mémoire M et $\mathcal{B}(S)$ l'ensemble de celles de schéma S . Des deux exemples qui suivent, le premier est « de terrain », le second est le divertissement annoncé dans l'introduction. Il s'agit d'un jeu mathématique classique [Arsac, 1985], dont on présente la solution « par les bases de données » – comme on dit « par l'algèbre ».

Exemple 2.1. On considère la mémoire relationnelle $(\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$ dans laquelle l'ensemble des valeurs $\mathbf{D}$ est égal à l'ensemble des chaînes de caractères sur l'alphabet latin, les chiffres et un élément distingué ω . On suppose que $\mathbf{R}$ contient les symboles relationnels *possède* et *marque* d'arité respectivement 3 et 2. On considère le schéma S constitué de ces deux symboles et la base de données (S, Δ) représentée, Figure 3, sous forme tabulaire. À chaque symbole du schéma correspond une table qui représente la donnée que le symbole stocke. Une table a autant de colonnes que l'arité du symbole

⁴ « Donnée », « uplet » et « valeur » ne sont pas ici synonymes comme ils le sont parfois dans le langage courant. Les *données* d'une mémoire relationnelle sont des ensembles finis d'uplets et les uplets sont des applications dans le domaine des *valeurs*.

⁵ On dit aussi *instance*, *extension* ou *interprétation*.

correspondant et chacune de ses lignes représente un uplet. Ainsi, la première ligne de la table du symbole *marque* représente-t-elle l'uplet $\langle \text{twingo}, \text{renault} \rangle$. La réalisation du symbole Ω est le singleton $\{\langle \omega \rangle\}$. Le symbole *possède* stocke un ensemble d'uplets $\langle x, y, z \rangle$ dont la sémantique correspond à l'énoncé *l'individu dont le n° de sécurité sociale est x et le nom est y possède une voiture z* tandis que le symbole *marque* stocke un ensemble d'uplets $\langle x, y \rangle$ qui correspondent à l'énoncé *la marque de la voiture x est y*.

$\Delta(\Omega)$	$\Delta(\text{possède})$			$\Delta(\text{marque})$	
1	1	2	3	1	2
ω	1 63 ...	martin	twingo	twingo	renault
	1 63 ...	martin	206	206	peugeot
	1 46 ...	dupont	xantia	xantia	citroën
	...			...	
	<i>n°SS</i>	<i>nom</i>	<i>voiture</i>	<i>voiture</i>	<i>constructeur</i>

Figure 3. Réalisation du schéma $\{\Omega, \text{possède}, \text{marque}\}$

Exemple 2.2. Jeu mathématique. Paul connaît le produit de deux nombres entiers de l'intervalle $[2,99]$. Serge connaît leur somme. Tous deux cherchent à connaître les deux nombres. S'ensuit le dialogue :

Paul : « Je ne connais pas les deux nombres. » ;

Serge : « Je le savais » ;

Paul : « Maintenant, je les connais » ;

Serge : « Moi aussi ».

Que ce sont ces nombres ?

$\Delta(\text{add})$	$\Delta(\text{mult})$
2 2 4	2 2 4
2 3 5	2 3 6
...	...

Figure 4. Réalisations de *add* et *mult*

La solution « par les bases de données » de ce divertissement (cf. Exemple 3.7) a ceci de surprenant qu'elle ne réclame aucune connaissance sur les nombres entiers sauf celle, brute, des opérations arithmétiques. La base de données utile à la solution est définie sur une mémoire relationnelle dont le domaine $\mathbf{D}$ des valeurs est l'ensemble des entiers naturels. Son schéma est à deux symboles relationnels *add* et *mult* d'arité 3. Ils stockent respectivement les tables d'addition et de multiplication des entiers – celles des litanies de l'Ecole Primaire ! – limitées à l'intervalle $[2,99]$ et dans lesquelles le premier opérande est inférieur ou égal au second :

$$\Delta(\text{add}) = \{ \langle x, y, s \rangle \mid x \in [2,99], y \in [2,99], x \leq y, s = x + y \} ;$$

$$\Delta(\text{mult}) = \{ \langle x, y, p \rangle \mid x \in [2,99], y \in [2,99], x \leq y, p = x \times y \} \text{ (cf. Figure 4).}$$

Commentaire 2.1. Symboles relationnels d'arité 0. Selon les définitions précédentes, une mémoire relationnelle dispose de symboles d'arité 0. Leur réalisation est soit le singleton $\{\langle \rangle\}$ soit l'ensemble vide. Elle n'a pas de représentation tabulaire puisque la table correspondante n'a pas de colonne. Dans la pratique, les logiciels de bases de données n'offrent pas toujours la possibilité de ces symboles. Il est pourtant naturel de les introduire. Ils permettent en particulier le calcul propositionnel à partir d'une machine relationnelle (Cf. Annexe). Toutefois, tous les exposés, formels ou non, du

modèle relationnel ne les présentent pas. Ils le sont dans [Abiteboul *et al.*, 1995, p 31-32] mais ne le sont pas dans [Delobel, Adiba, 1982] ou dans [Ullman, 1999].

Commentaire 2.2. Mémoire relationnelle et mémoire de fichiers d'enregistrements. Dans le but de fixer les idées, il est fréquent d'assimiler la réalisation d'un symbole relationnel à un fichier d'enregistrements, outil traditionnel pour le stockage d'informations. Cette assimilation est abusive, même si elle est parfois commode – on l'utilise d'ailleurs dans l'indication de la preuve du théorème 3.H. En effet, un fichier d'enregistrements est une suite finie d'uplets alors qu'une réalisation est un ensemble fini d'uplets. La réalisation d'un symbole relationnel ne contient donc pas de doublons, i.e. deux fois le même uplet, contrairement à un fichier d'enregistrements. Cette propriété d'unicité, qui va de soi en théorie des ensembles, est au contraire volontariste en informatique où la présence des doublons est « monnaie courante » et où leur suppression réclame une algorithmique spécifique, en général coûteuse en temps-machine. Formellement, un ensemble fini de fichiers d'enregistrements correspond à un stockage dans une mémoire (F, D^{sf}) dans laquelle F est un ensemble dénombrable, celui des noms de fichiers, et D^{sf} désigne l'ensemble des suites finies d'uplets de même arité sur D . Les données ne sont donc pas les mêmes dans une mémoire d'enregistrements et dans une mémoire relationnelle, suites finies d'uplets pour celle-là, ensembles finis d'uplets pour celle-ci.

Commentaire 2.3. Approche numérotée et approche nommée. La définition des mémoires relationnelles présentée ici constitue ce qu'on appelle communément leur approche *numérotée*. Ceci pour signifier que les colonnes de la représentation tabulaire d'une base de données sont numérotées (cf. Figures 3 et 4). Il existe une approche complémentaire dans laquelle les colonnes sont nommées. Elle s'appuie sur la donnée d'un ensemble dénombrable A , appelé ensemble des *attributs*. Une *sorte* sur A est un sous-ensemble fini de A et une *sorte ordonnée* la donnée d'une sorte sur A et d'un ordre total sur elle. En conservant les notations précédentes, une mémoire relationnelle est alors définie comme un quintuplet (R, A, Ω, SO, D) dans lequel SO est une application de R dans l'ensemble des sortes ordonnées sur A telle que, pour toute sorte ordonnée so , l'ensemble $SO^{-1}(so)$ est infini. L'application SO remplace l'application *arité* qui désormais s'en déduit par $arité(R) = |SO(R)|^6$, $\forall R \in R$. Dès lors, schéma et réalisation peuvent se définir comme précédemment. Dans l'approche nommée, chaque colonne de la représentation tabulaire des réalisations est identifiable par un attribut. C'est la raison pour laquelle l'approche est dite *nommée*. Dans la Figure 3, les attributs sont indiqués au bas de chaque colonne. La sorte ordonnée associée au symbole *possède* est constituée de la sorte $\{n^{\circ}SS, nom, voiture\}$ et de l'ordre $n^{\circ}SS < nom < voiture$.

Les passages entre les approches numérotée et nommée sont canoniques. Si le quadruplet $(R, \Omega, arité, D)$ est une mémoire « numérotée », alors le quintuplet $(R, \mathbb{N}, \Omega, SO, D)$, dans lequel $\mathbb{N}$ est l'ensemble des entiers naturels et $SO(R)$ est l'intervalle $[1, arité(R)]$, est une mémoire « nommée ». Le passage inverse est tout aussi simple. Le choix entre l'une ou l'autre des deux approches dépend des besoins et des usages. L'approche numérotée est en général utilisée dans les développements formels, l'approche nommée dans les applications car celles-ci utilisent parfois des symboles relationnels de grande arité, par exemple plus de 100 colonnes. Dans ces conditions, l'approche nommée se révèle en pratique indispensable pour identifier plus

⁶ Pour un ensemble fini A , $|A|$ désigne le nombre de ses éléments.

commodément les colonnes. Le langage SQL est construit sur l'approche nommée tandis que le langage Datalog l'est sur l'approche numérotée. On n'utilisera dans cet article que l'approche numérotée.

Commentaire 2.4. Structure du 1^{er} ordre. Une base de données relationnelle est une structure purement relationnelle du 1^{er} ordre⁷. Il s'agit-là du carcan que le modèle relationnel impose et les informations à stocker doivent être mises sous cette forme. En particulier, les uplets sont des applications à valeur dans le domaine $\mathbf{D}$ des valeurs, pas dans l'ensemble des parties finies de $\mathbf{D}$. Par conséquent, le modèle n'autorise pas d'uplet de la forme $\langle 163 \dots, \text{martin}, \{ \text{twingo}, 206 \} \rangle$ qui traduirait en une seule « ligne » toutes les voitures que martin possède, alors que la réalisation du symbole *possède* dans l'exemple 3.1 y consacre deux « lignes » (voir Figure 3). Aussi, le stockage relationnel peut-il se révéler peu économe de l'espace-disque, argument qui a perdu de son importance avec le considérable accroissement de la capacité des disques durs et l'abaissement de leur coût.

On souligne aussi que, contrairement aux langages du 1^{er} ordre, une mémoire relationnelle ne dispose pas de symboles de fonctions dont les réalisations seraient des applications de $\mathbf{D}^{[n]}$ dans $\mathbf{D}$. Leur absence fait partie des limites du modèle relationnel, dont on dit parfois, par raccourci, qu'il est la logique du 1^{er} ordre sans symbole de fonctions. Elle s'explique par le fait qu'on demande à une base de données soit de restituer sélectivement soit de réorganiser les informations qu'elle stocke et pas d'en calculer de nouvelles. Ceci étant, la règle peut être partiellement contournée. En effet, toute fonction de $\mathbf{D}^{[n]}$ dans $\mathbf{D}$ est un ensemble d'uplets d'arité $n + 1$. Ainsi, par exemple, une fonction f de $\mathbf{D}^{[1]}$ dans $\mathbf{D}$ est un ensemble infini d'uplets d'arité 2, à savoir $\{ \langle x, f(x) \rangle \mid x \in \mathbf{D} \}$. Mise à part son infinitude, une fonction est donc de même nature que la réalisation d'un symbole d'arité 2. Dès lors, il est possible qu'au sein d'un schéma, un symbole particulier d'arité 2 stocke une partie finie de la fonction f . La base de données dispose ainsi, au moins partiellement, de ladite fonction. Cette méthode, décrite dans [Ebbinghaus, Flum, 1999], est celle utilisée, dans l'exemple 2.2, pour l'addition et la multiplication : leurs opérandes sont limités à l'intervalle $[2, 99]$. On souligne enfin que, si les fonctions sont absentes du formalisme relationnel strict, les logiciels relationnels disposent de certaines pour des raisons pratiques. Ainsi, le langage SQL dispose-t-il des opérations arithmétiques [Ullman, 1999, p. 214 ; Merret, 1984, p. 217-218].

Commentaire 2.5. Mémoire relationnelle et mémoire de programmation. Si on ignore le symbole Ω , au rôle purement technique, les définitions du début de chapitre distinguent quatre notions qui s'organisent selon l'énoncé suivant : dans le *réservoir* dénombrable $\mathbf{R}$ de symboles dont dispose une mémoire relationnelle, chacun étant caractérisé par une *arité*, un *schéma* précise les symboles utilisés dans une application particulière et une *réalisation* leur associe une donnée conformément à leur arité, i.e. un ensemble fini d'uplets de l'arité considérée. Cette situation est semblable à celle de la programmation classique et incite à considérer ici les SGBD relationnels comme des machines programmables. En effet, un langage de programmation, par exemple C++, dispose d'une *infinité dénombrable de variables* (tout du moins d'un nombre très grand), chacune étant caractérisée par son *type*. Tout programme n'utilise qu'un *nombre fini* d'entre elles – elles sont déclarées dans le programme – et une application appelée

⁷ Une structure purement relationnelle est la donnée d'un ensemble $\mathbf{A}$ non vide et d'une famille de relations n -aires sur $\mathbf{A}$ [Bridge, 1977] (cf. Annexe §A.1).

valuation leur associe une donnée conformément à leur type. Le parallèle des énoncés conduit naturellement aux mises en vis-à-vis suivantes : *réservoir de symboles/ensemble des variables possibles*, *arité/type*, *schéma/ensemble des variables déclarées et réalisation/valuation*. Ainsi, les symboles relationnels ont-ils un statut semblable à celui des variables de la programmation classique. Date [1998] les nomme d'ailleurs, p. 99, *variables relationnelles*. Comme toute variable, leur valeur peut être modifiée par l'opération algorithmique d'affectation, dont le paragraphe 2.2 examine la forme.

On termine ce paragraphe en introduisant une notion importante pour le chapitre suivant. On appelle *domaine attesté* d'une base de données (S, Δ) le sous-ensemble :

$$\{ a \in \mathbf{D} \mid \exists R \in S, \exists t \in \Delta(R), \exists i \in [1, \text{arité}(R)] \text{ tels que } t(i) = a \}$$

de $\mathbf{D}$. On le note $\mathbf{a}(\Delta)$, parfois $\mathbf{a}$ en l'absence d'ambiguïté. Le domaine attesté regroupe les éléments du domaine $\mathbf{D}$ des valeurs qui sont attestés dans la base⁸.

LEMME 2.A. Le domaine attesté de toute base de données est fini et non vide.

Démonstration : fini, car les réalisations sont des ensembles finis d'uplets. Non vide, puisque la réalisation du symbole Ω n'est, par définition, jamais vide.

Commentaire 2.6. Le symbole Ω assure que le domaine attesté d'une base n'est jamais vide, c'est son unique rôle. Les rôles de cet invariant sont examinés dans l'annexe.

2.2 HISTOIRE D'UNE MÉMOIRE RELATIONNELLE

À l'instar de toute mémoire informatique, le contenu d'une mémoire relationnelle M doit pouvoir changer. Aussi est-il naturel de considérer des séquences finies de bases de données sur M . Et, parmi elles, celles telles que le passage du rang i au rang $i + 1$ corresponde à une des opérations définies maintenant. On désigne par B une base de données (S, Δ) , par $\mathbf{r}$ un symbole relationnel d'arité $n \geq 0$, appartenant à S et différent de Ω , par T un ensemble *fini* d'uplets d'arité n sur $\mathbf{D}$ et, finalement, par $\underline{\mathbf{r}}$ un symbole relationnel n'appartenant pas à S . On appelle :

- *insertion* l'application qui associe au triplet $(B, \mathbf{r}, T)$ la base de donnée (S, Δ') telle que $\forall R \in S, R \neq \mathbf{r}, \Delta'(R) = \Delta(R)$ et $\Delta'(\mathbf{r}) = \Delta(\mathbf{r}) \cup T$;
- *suppression* l'application qui associe au triplet $(B, \mathbf{r}, T)$ la base de donnée (S, Δ') telle que $\forall R \in S, R \neq \mathbf{r}, \Delta'(R) = \Delta(R)$ et $\Delta'(\mathbf{r}) = \Delta(\mathbf{r}) - T$;
- *introduction* l'application qui associe au couple $(B, \underline{\mathbf{r}})$ la base de donnée (S', Δ') telle que $S' = S \cup \{ \underline{\mathbf{r}} \}$ et $\forall R \in S, \Delta'(R) = \Delta(R)$ et $\Delta'(\underline{\mathbf{r}}) = \emptyset$;
- *effacement* l'application qui associe au couple $(B, \mathbf{r})$ la base de donnée (S', Δ') telle que $S' = S - \{ \mathbf{r} \}$ et $\forall R \in S', \Delta'(R) = \Delta(R)$.

L'insertion (resp. la suppression) consiste à augmenter (resp. diminuer) la réalisation d'un symbole d'un ensemble fini d'uplets, l'introduction à augmenter le schéma d'une base d'un nouveau symbole, enfin l'effacement à supprimer un symbole du schéma. L'insertion et la suppression relèvent de l'opérateur algorithmique d'affectation, l'introduction et l'effacement de la gestion des variables d'un programme.

⁸ Codd [1970] introduisit l'adjectif *active* : *active domain*, que la littérature en français traduit par *actif*. Compte tenu de ce qu'il exprime, on adopte ici l'adjectif *attesté*, plus explicite.

On appelle *histoire* d'une mémoire relationnelle M toute suite finie B_i , $0 \leq i \leq n$, de bases de données sur M telle que pour tout i , $0 \leq i < n$, la base B_{i+1} est la base image de la base B_i par une insertion, une suppression, une introduction ou un effacement. Le premier élément d'une histoire est la base *originelle*, le dernier la base *terminale*. Dans la suite, on écrira $(B : \uparrow \underline{r})$ au lieu de *introduction*($B, \underline{r}$) et $(B : \underline{r} \Leftarrow T)$ au lieu de *insertion*($B, \underline{r}, T$). De plus, en l'absence d'ambiguïtés, on ne mentionnera pas la base B et on écrira $\uparrow \underline{r}$ et $\underline{r} \Leftarrow T$.

Exemple 2.3. La suite de bases de données $(B_i, 0 \leq i \leq 4)$, telle que :

B_0 est la base de schéma vide (sauf Ω toujours présent et de réalisation non vide) ;

$B_1 = (B_0 : \uparrow \text{possède})$

$B_2 = (B_1 : \text{possède} \Leftarrow$

$\{ \langle 1\ 63 \dots, \text{martin}, \text{twingo} \rangle, \langle 1\ 63 \dots, \text{martin}, 206 \rangle, \langle 1\ 46 \dots, \text{dupont}, \text{xantia} \rangle \})$

$B_3 = (B_2 : \uparrow \text{marque})$

$B_4 = (B_3 : \text{marque} \Leftarrow \{ \langle \text{twingo}, \text{renault} \rangle, \langle 206, \text{peugeot} \rangle, \langle \text{xantia}, \text{citroën} \rangle \})$

est une histoire dont la base terminale B_4 est celle de la Figure 3.

Les quatre applications décrites assurent la dynamique des mémoires relationnelles. On souligne que, rapportée à une machine programmable, chacune d'elles peut être regardée comme une instruction. Un programme est une séquence de ces instructions et une histoire est le journal de l'exécution séquentielle dudit programme à partir d'une base originelle. Le programme correspondant à l'exemple 2.3 serait :

$(\uparrow \text{possède}) ; (\text{possède} \Leftarrow \{ \dots \}) ; (\uparrow \text{marque}) ; (\text{marque} \Leftarrow \{ \dots \})$.

2.3 ENTITÉS, MÉMOIRE RELATIONNELLE ET CLÉS SUCCÉDANÉES

Parce que la notion d'*entité* est très fréquemment et très spontanément utilisée dans la pratique des bases de données, ce paragraphe examine son rapport avec les mémoires relationnelles. On expose, en particulier, les raisons pour lesquelles celles-ci ne peuvent pas être considérées comme étant basées sur les entités. On présente alors comment l'artifice technique dit des clés succédanées permet de les y adapter.

Une entité est un objet du « monde extérieur » doué d'une unité matérielle⁹ et possédant une identité qui le distingue des autres [Delobel, Adiba, 1982, p. 18]. Il n'existe pas de définition plus formelle [Ullman, 1999, p. 34]. De plus, les entités sont porteuses d'informations, ce qui les relie aux bases de données. Date [1998, p. 12] écrit qu'une entité est « tout objet sur lequel on souhaite stocker des informations ». C'est d'ailleurs une manière de maîtriser la complexité des informations provenant d'une réalité donnée que de les rapporter aux entités identifiées de ladite réalité. La notion d'entité est donc, entre autres, un outil de description. Ainsi, pour concevoir une base de données pour un hôpital, on commencera par identifier les entités constitutives : un médecin est une entité, de même qu'un patient, une intervention, un lit ou encore un dossier médical. Pour chaque entité, ou chaque catégorie d'entités, on déterminera ensuite les informations à retenir et stocker : le nom pour un patient, la date pour une intervention et ainsi de suite.

⁹ « Entité : objet considéré comme un être doué d'une unité matérielle », *Le petit Robert*, 2001.

On schématise par $E \rightarrow I$ le fait, déjà mentionné, que les entités (E) portent ($\rightarrow$) des informations (I). On fait remarquer que ce schéma peut se formaliser comme un stockage (cf. Chapitre 1) dans lequel les entités seraient les supports et les informations les données. Certains modèles de données reproduisent structurellement le schéma $E \rightarrow I$. À ce propos, cf. l'article Chen [1976], fondateur du modèle entité-association ainsi que la littérature sur les bases de données de type « objet » [Adiba, Collet, 1993 ; Bouzeghoub, Gardarin, Valduriez, 1998]. Le modèle relationnel est tout autre. Ullman [1999] écrit qu'il est basé sur les valeurs (*value-oriented*), qu'on peut paraphraser ici par « basé sur les informations ». Autrement dit, du schéma $E \rightarrow I$, le modèle relationnel ne conserve que le I. C'est une de ses originalités. Non qu'il soit impossible de reproduire le schéma $E \rightarrow I$ avec une mémoire relationnelle, mais qu'on sort alors de son usage naturel. On s'appuie sur un exemple qui montre qu'en reproduisant le schéma $E \rightarrow I$, la base de données relationnelle obtenue perd l'essentiel de ses intérêts pratiques. L'exemple permet aussi d'introduire la solution alternative des clés succédanées.

L'exemple est celui des entités « patient » d'un hôpital. On veut stocker leurs symptômes. Un premier patient p_1 souffre de maux de tête et de douleurs au coude, un second p_2 de maux de tête et de douleurs au dos, et ainsi de suite. Si on se donne un ensemble fini P de patients et un ensemble fini S de symptômes, la situation peut se formaliser de deux manières : (a) par une application s de P dans l'ensemble 2^S des parties de S : $s(p_1) = \{\text{tête, coude}\}$ et $s(p_2) = \{\text{tête, dos}\}$, (b) par une correspondance c de P vers S , i.e. un sous-ensemble du produit $P \times S$: $c = \{(p_1, \text{tête}), (p_1, \text{coude}), (p_2, \text{tête}), (p_2, \text{dos})\}$. Étant donnée une mémoire relationnelle $M = (\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$, ces deux formalisations conduisent à des bases de données relationnelles très différentes, celles que montrent les Figures 5(a) et 5(b). La base de données (a) est issue de la formalisation (a). Son schéma

Figure 5. Deux stockages relationnels

contient autant de symboles que de patients *via* une injection $id_{\mathbf{R}}$ qui associe à chaque patient un symbole relationnel d'arité 1 désigné sur la figure par un entier précédé d'une « * » : $id_{\mathbf{R}}(p_1) = *1$, $id_{\mathbf{R}}(p_2) = *2$, ... ; $*1, *2, \dots \in \mathbf{R}$. Pour chaque patient p , la réalisation $\Delta(id_{\mathbf{R}}(p))$ du symbole associé est $\{ \langle v \rangle \mid v \in s(p) \}$. Elle regroupe tous les maux du patient p . La base de données (b) est issue de la formalisation (b). Son schéma contient un seul symbole, *maux*, d'arité 2. Une injection implicite $id_{\mathbf{D}}$ associe à chaque patient un élément du domaine $\mathbf{D}$ des valeurs, désigné sur la figure par un entier précédé d'un « # » : $id_{\mathbf{D}}(p_1) = \#1$, $id_{\mathbf{D}}(p_2) = \#2$, ... ; $\#1, \#2, \dots \in \mathbf{D}$. La réalisation du symbole *maux* est $\{ \langle id_{\mathbf{D}}(p), v \rangle \mid (p, v) \in c \}$.

Un lecteur habitué au modèle relationnel aura déjà jugé « hérétique » la base de données de la Figure 5(a). Elle l'est en effet. Elle présente cependant des avantages certains. Au premier rang desquels le fait de reproduire structurellement le schéma $E \rightarrow I$: entités et informations sont formellement distinguées puisque les premières sont assimilées, *via* l'injection $id_{\mathbf{R}}$, à des symboles relationnels tandis que les secondes aux données que ces symboles stockent. Il en résulte l'avantage qu'une entité « patient »

peut apparaître dans la base de données même lorsqu'on ne dispose d'aucune information sur elle – il suffit de lui associer un symbole relationnel de réalisation vide. Dans la solution (b), au contraire, les entités « patient » sont assimilées, *via* l'injection id_D , aux valeurs #1, #2, ... et confondues avec les autres valeurs – elles sont en quelque sorte « valeur-isées » et dé-réifiées. Et ce n'est que par un geste volontariste et externe au modèle que ces valeurs sont considérées comme particulières. Par ailleurs, pour qu'un patient soit dans la base (b), il faut connaître au moins un de ses symptômes, sinon il manque des informations et le patient ne peut être recensé¹⁰. En conclusion, pour ce qui est « du fond des choses », la base (a), parce qu'elle reproduit la situation d'entités porteuses d'informations, est la plus satisfaisante. Cela étant, elle n'est pas la « bonne » manière d'utiliser une mémoire relationnelle.

La raison se révèle si on s'intéresse à la question *quels sont les patients qui souffrent de la tête ?*. La réponse est le sous-ensemble $r = \{p_1, p_2\}$. Rapportée aux bases de données (a) et (b), la réponse est, pour la base (a), l'image $r^{(a)} = id_R(r) = \{^*1, ^*2\}$ et, pour la base (b), l'image $r^{(b)} = id_D(r) = \{\#1, \#2\}$. Le point important est qu'il n'est pas possible de stocker, dans la mémoire relationnelle M , la réponse $r^{(a)}$ tandis qu'il est possible de stocker la réponse $r^{(b)}$. En effet, la réponse $r^{(b)}$ correspond à l'ensemble $\{<\#1>, <\#2>\}$ d'uplets sur D d'arité 1, lequel peut être stocké par un symbole relationnel d'arité 1 choisi à cet effet. En revanche, rien de tel n'est possible avec la réponse $r^{(a)}$. Certes, elle correspond à l'ensemble $\{<^*1>, <^*2>\}$ de tuples d'arité 1 sur R . Mais, par définition, une mémoire relationnelle ne dispose pas de supports pour stocker des ensembles d'uplets sur R , donc pour stocker la réponse $r^{(a)}$. Il en résulte qu'une machine BD dont la mémoire est relationnelle ne peut, quelle que soit la puissance de son langage, évaluer la réponse $r^{(a)}$ puisqu'elle ne peut la stocker¹¹. Le constat se généralise et sont ainsi écartées de la base (a) toutes les questions de la forme *quels sont les patients tels que...?*. L'utilité pratique de la base (a) s'en trouve fortement limitée¹² et cette solution doit être rejetée. Reste la solution (b). Elle est fondée sur l'injection id_D . Codd [1979] appelle une telle injection *clé succédanée* (*surrogate key*) car sa valeur pour chaque entité remplace, ou encore représente, l'entité en question. Dans l'exemple, l'injection id_D est abstraite, elle pourrait être plus concrète, comme, par exemple, le n° de Sécurité Sociale. Dans tous les cas, il s'agit d'un élément extérieur au modèle relationnel, qui code chaque entité par une valeur. Grâce aux clés succédanées, le stockage d'informations sur des entités est « simulable » dans une mémoire relationnelle. Elles sont très utilisées en pratique.

¹⁰ Cette limitation relève de ce qu'on nomme usuellement la *gestion de l'information incomplète*. Pour la lever, des extensions du modèle relationnel ont été proposées et certaines mises en œuvre. Elles ne sont pas envisagées dans cet article (cf. [Codd, 1979 ; Abiteboul *et al.*, 1995 ; Date, 1998]).

¹¹ On insiste sur le lien de nécessité entre stockage et calcul : celui-là est nécessaire (mais non suffisant) à celui-ci. Ce lien tient à la définition même de la calculabilité *via* les machines de Turing puisque le « résultat » d'une machine est défini comme l'état de son ruban, autrement dit comme ce que le ruban stocke, une fois qu'elle s'est arrêtée.

¹² Dans le SGBD relationnel ORACLE, version 8.1, certaines questions ayant un « goût » de celle de l'exemple sont possibles, telle celle-ci : *quels sont les symboles relationnels dont la réalisation contient 3 uplets*. Il s'agit d'une commodité contingente, basée sur un artifice et qui sort du cadre relationnel strict.

Les mémoires relationnelles ne permettent donc qu'en apparence de reproduire le schéma $E \rightarrow I$. Aussi, n'est-il pas possible de considérer qu'elles sont basées sur les entités. En revanche, la technique nécessaire des clés succédanées, parce ce qu'elle réclame la « valeur-isation » des entités, corrobore l'assertion d'Ullman, citée *supra*, que les mémoires relationnelles sont basées sur les valeurs.

CHAPITRE 3

3 DE LA LOGIQUE DU 1^{er} ORDRE VERS UN LANGAGE « BASES DE DONNÉES »

Les opérations d'*insertion* et de *suppression* modifient l'état courant d'une mémoire à partir d'un ensemble fini T d'uplets de même arité (cf. §.2.2). Dans l'exemple 2.3, les ensembles T d'uplets sont en extension. On s'intéresse dans ce chapitre à la manière de les exprimer en compréhension, c'est-à-dire à partir d'une propriété. Par exemple, la réponse à la question *quels sont les noms des personnes qui possèdent un véhicule* correspond à l'ensemble des uplets $\langle x \rangle$ tels que x est le nom d'une personne qui possède un véhicule. Les travaux qui ont suivi

l'article fondateur de Codd [1970], ont proposé et développé l'idée que le langage que la logique du 1^{er} ordre associe à toute structure relationnelle était bien adapté au questionnement des bases de données relationnelles, bien qu'il n'ait pas (ou parce qu'il n'a pas) toute la puissance de la calculabilité. Partant de la logique du 1^{er} ordre, ce chapitre présente la construction d'un langage de requêtes pour le questionnement des bases de données. La construction est basée sur deux restrictions

successives : sémantique puis syntaxique. Des exigences propres aux bases de données conduisent à restreindre le langage des requêtes du 1^{er} ordre à un sous-langage dit des requêtes indépendantes, première restriction. Or, ce sous-langage n'est pas récursif, en d'autres termes, la restriction visée n'est pas programmable. Cependant, ce sous-langage peut, à son tour, être restreint *sans perte de pouvoir d'expression* à un sous-langage récursif, seconde restriction. La Figure 6 schématise cette situation. L'ovale extérieur est celui des requêtes du 1^{er} ordre. L'ovale médian est caractérisé à partir de la propriété sémantique d'indépendance, qui n'est pas calculable. L'ovale intérieur est une restriction du précédent à partir d'une caractérisation syntaxique calculable. Il permet de formuler les mêmes questions que l'ovale médian. La possibilité d'une approche syntaxique de l'indépendance est le fondement théorique de langages pratiques comme SQL ou Datalog.

Figure 6. Requêtes du 1^{er} ordre

Le chapitre est en trois étapes. Le paragraphe 3.1 définit syntaxiquement ce que sont les formules et les requêtes du 1^{er} ordre, puis présente leur sémantique. Il s'inspire de [Kreisel, Krivine, 1967, chap. 2]. Le paragraphe 3.2 introduit les requêtes indépendantes et explique pourquoi il est nécessaire de s'y limiter. Il présente aussi les notions d'interrogation et de pouvoir d'expression. Le paragraphe 3.3 montre que l'ensemble des requêtes indépendantes n'est pas récursif, puis établit l'existence d'un

sous-ensemble récursif – qui n’est d’ailleurs pas unique (cf. Chapitre 4) – des requêtes indépendantes dont le pouvoir d’expression est le même que celui des requêtes indépendantes.

3.1 FORMULES ET REQUÊTES : SYNTAXE ET SÉMANTIQUE

Étant donnés une mémoire relationnelle $(\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$, un ensemble dénombrable $\mathbf{V}$, appelé *ensemble des variables*, l’ensemble $\{ \&, \neg, \exists \}$ des symboles logiques et enfin le singleton $\{ eg \}$ pour le symbole de l’égalité, ces trois derniers constituants étant disjoints deux à deux et disjoints de $\mathbf{R}$ et de $\mathbf{D}$, on définit le langage $\mathcal{F}$ comme le plus petit langage tel que :

- si R est un élément de $\mathbf{R}$ d’arité $n = 0$, alors $R \in \mathcal{F}$;
- si R est un élément de $\mathbf{R}$ d’arité $n > 0$ et si $v_1, v_2, \dots, v_n$ sont des variables, alors $R v_1 v_2 \dots v_n \in \mathcal{F}$;
- si v_1 et v_2 sont des variables, alors $eg v_1 v_2 \in \mathcal{F}$;
- si $F \in \mathcal{F}$ alors $\neg F \in \mathcal{F}$;
- si $F \in \mathcal{F}$ et $F' \in \mathcal{F}$, alors $\& F F' \in \mathcal{F}$;
- si $F \in \mathcal{F}$ et v est une variable, alors $\exists v F \in \mathcal{F}$.

Les éléments de $\mathcal{F}$ sont appelés *formules*. *Exemples* : soient R_0, R_1 et R_2 des symboles relationnels d’arité 0, 1 et 2, soient x et y des variables, alors $R_0, R_2 x y$ et $\neg \& R_1 y \neg R_2 x y$ sont des formules, de même que $\exists x eg x y$. Grâce aux règles usuelles du parenthésage, on écrira les formules sous leur forme plus habituelle et lisible, à savoir $R_0()$, $R_2(x, y)$, $\neg (R_1(y) \& \neg R_2(x, y))$, $\exists x eg(x, y)$. Enfin, le langage $\mathcal{F}$ peut aisément s’augmenter, selon les équivalences habituelles, de tout autre opérateur logique tels que la disjonction $\vee$ et l’implication $\Rightarrow$ ainsi que du quantificateur universel $\forall$.

Étant donnée une formule F , on désigne par $sch F$ le schéma constitué de l’ensemble des symboles relationnels que la formule utilise et du symbole distingué Ω ; il est appelé *schéma intrinsèque* de la formule. *Exemples* : le schéma intrinsèque de la formule $\exists y R_2(x, y)$ est $\{ \Omega, R_2 \}$ et celui de $eg(v_1, v_2)$ est $\{ \Omega \}$. Une formule F est dite *sur un schéma* S si $S \supseteq sch F$. De plus, on note $var F$ l’ensemble des variables qu’une formule F utilise et, si $\langle v_1, v_2, \dots, v_n \rangle$ est un uplet sur l’ensemble $\mathbf{V}$ des variables, on note $var \langle v_1, v_2, \dots, v_n \rangle$ son image, i.e. l’ensemble des variables qu’il utilise. Enfin, on désigne par *libre* F l’ensemble des variables dites *libres* de F et défini comme suit. Si F est de la forme :

- $R()$, $R \in \mathbf{R}$ et d’arité 0, alors $libre F = \emptyset$;
- $R(v_1, v_2, \dots, v_n)$, $R \in \mathbf{R}$ et d’arité $n > 0$, alors $libre F = var \langle v_1, v_2, \dots, v_n \rangle$;
- $eg(v_1, v_2)$, alors $libre F = var \langle v_1, v_2 \rangle$;
- $\neg F'$, alors $libre F = libre F'$;
- $F' \& F''$, alors $libre F = libre F' \cup libre F''$;
- $\exists v F'$, alors $libre F = libre F' - \{ v \}$.

Une *requête* est un couple $[\nu ; F]$ dans lequel ν est un uplet sur l’ensemble $\mathbf{V}$ des variables et F un élément de $\mathcal{F}$. L’uplet ν est appelé l’*extracteur* de la requête, la formule F son *filtre*. L’*arité* d’une requête est celle de son extracteur. *Exemple* : $[\langle x \rangle ;$

$\exists y R_2(x, y)$ est une requête dont l'extracteur est $\langle x \rangle$, l'arité 1 et le filtre $\exists y R_2(x, y)$. Le schéma *intrinsèque* $sch \rho$ d'une requête ρ est celui de son filtre. Une requête est dite *sur un schéma* S si $S \supseteq sch \rho$. On note $\mathcal{R}$ l'ensemble des requêtes.

La sémantique des formules et des requêtes est liée à leur interprétation sur une base de données relativement à un sous-domaine des valeurs. Étant donné un sous-ensemble $\mathbf{d}$ du domaine $\mathbf{D}$ des valeurs, chaque application de $\mathbf{V}$ dans $\mathbf{d}$, i.e. chaque élément de $\mathbf{d}^{\mathbf{V}}$, est appelée *valuation* sur $\mathbf{d}$ des variables. Soit un schéma S . Selon le théorème des schémas fonctionnels [Kreisel, Krivine, 1967, p. 3], il existe une unique application $eval$ qui associe à tout sous-ensemble $\mathbf{d}$ du domaine $\mathbf{D}$, à toute formule F sur S et à toute base de données Δ sur S , un ensemble $eval(\mathbf{d}, F, \Delta)$ de valuations sur $\mathbf{d}$ tel que si F est de la forme :

$$(3.1a) \quad R(), R \in S \text{ et d'arité } 0, \text{ alors } eval(\mathbf{d}, F, \Delta) = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \delta_{\circ \langle \rangle} \in \Delta(R) \};$$

$$(3.1b) \quad R(v_1, v_2, \dots, v_n), R \in S \text{ et d'arité } n > 0, \forall i, 1 \leq i \leq n, v_i \in \mathbf{V}, \text{ alors}$$

$$eval(\mathbf{d}, F, \Delta) = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \delta_{\circ \langle v_1, v_2, \dots, v_n \rangle} \in \Delta(R) \};$$

$$(3.1c) \quad eg(v_1, v_2), \text{ alors } eval(\mathbf{d}, F, \Delta) = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \delta(v_1) = \delta(v_2) \};$$

$$(3.1d) \quad \neg F', \text{ alors } eval(\mathbf{d}, F, \Delta) = \mathbf{d}^{\mathbf{V}} - eval(\mathbf{d}, F', \Delta);$$

$$(3.1e) \quad F' \& F'', \text{ alors } eval(\mathbf{d}, F, \Delta) = eval(\mathbf{d}, F', \Delta) \cap eval(\mathbf{d}, F'', \Delta);$$

$$(3.1f) \quad \exists v F', \text{ alors } eval(\mathbf{d}, F, \Delta) = (eval(\mathbf{d}, F', \Delta) \downarrow (\mathbf{V} - \{v\})) \uplus \mathbf{d}^{\{v\}}.$$

LEMME 3.A. Pour tout schéma S , toute formule F sur S et toute base Δ sur S , $eval(\mathbf{d}, F, \Delta) = eval(\mathbf{d}, F, \Delta \downarrow sch F)$, $\Delta \downarrow sch F$ étant la restriction de Δ à $sch F$.

Eu égard au lemme 3.A, l'évaluation $eval(\mathbf{d}, F, \Delta)$ ne dépend que de la partie « utile », au regard de la formule F , de la base Δ . Ainsi, si S et S' sont des schémas tels $S \cap S' \supseteq sch F$, et si Δ et Δ' sont des bases respectivement sur S et sur S' qui coïncident sur $sch F$, alors $eval(\mathbf{d}, F, \Delta) = eval(\mathbf{d}, F, \Delta')$. Lorsque le domaine $\mathbf{d}$ est fixé, on note $\underline{\mathbf{d}}$ l'application définie par $\underline{\mathbf{d}}(F, \Delta) = eval(\mathbf{d}, F, \Delta)$. On l'appelle *évaluation sur le domaine* $\mathbf{d}$ et on écrira $\underline{\mathbf{d}}_{\Delta} F$ au lieu de $\underline{\mathbf{d}}(F, \Delta)$. L'évaluation est dite *naturelle*, notée $\underline{\mathbf{D}}$, lorsque $\mathbf{d} = \mathbf{D}$ et *vide*, notée $\emptyset$, lorsque $\mathbf{d} = \emptyset$. Enfin, l'évaluation *attestée* est l'application $\mathbf{att}$ définie par $\mathbf{att}(F, \Delta) = eval(\mathbf{a}(\Delta), F, \Delta)$ – on écrira $\mathbf{att}_{\Delta} F$ –, dans laquelle le domaine d'évaluation est le domaine attesté $\mathbf{a}(\Delta)$ de la base (cf. Commentaire 3.1), donc $\mathbf{att}_{\Delta} F = \mathbf{a}(\Delta)_{\Delta} F$. Lorsqu'une seule base de données sera considérée, on omettra parfois sa mention et on écrira $\underline{\mathbf{d}} F$ au lieu de $\underline{\mathbf{d}}_{\Delta} F$ (donc $\underline{\mathbf{D}} F$ et $\emptyset F$ au lieu de $\underline{\mathbf{D}}_{\Delta} F$ et de $\emptyset_{\Delta} F$) et $\mathbf{att} F$ au lieu de $\mathbf{att}_{\Delta} F$.

Le Lemme 3.B ci-dessous reformule de deux manières la règle (3.1f) et énonce celle pour le quantificateur universel. Le Lemme 3.C examine le cas de l'évaluation vide. Enfin, le Lemme 3.D explicite le rôle central des variables libres.

LEMME 3.B.

$$(A) \quad \underline{\mathbf{d}}_{\Delta} \exists v F' = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \exists \delta' \in \underline{\mathbf{d}}_{\Delta} F' \text{ tq } \delta \text{ et } \delta' \text{ coïncident sur } \mathbf{V} - \{v\} \};$$

(B) $\underline{\mathbf{d}}_{\Delta} \exists v F' = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \exists a \in \mathbf{d} \text{ tq } \delta_{v/a} \in \underline{\mathbf{d}}_{\Delta} F' \}^{13}$;

(C) $\underline{\mathbf{d}}_{\Delta} \forall v F' = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \forall a \in \mathbf{d} \text{ tq } \delta_{v/a} \in \underline{\mathbf{d}}_{\Delta} F' \}$.

LEMME 3.C. Pour tout schéma S , toute formule F sur S et toute base de données Δ sur S , $\underline{\mathcal{Q}}_{\Delta} F = \emptyset$.

Démonstration. L'évaluation $\underline{\mathcal{Q}}_{\Delta} F$ est un sous-ensemble de $\emptyset^{\mathbf{V}}$, qui est vide puisque $\mathbf{V}$ n'est pas vide.

LEMME 3.D. Pour tout schéma S , pour toute formule F sur S , toute base de données Δ sur S et tout domaine $\mathbf{d}$, $\underline{\mathbf{d}}_{\Delta} F = (\underline{\mathbf{d}}_{\Delta} F \downarrow \text{libre } F) \uplus \mathbf{d}^{\mathbf{V} - \text{libre } F}$.

Démonstration. Vrai pour $\mathbf{d} = \emptyset$, sinon par récurrence sur le nombre d'opérateurs logiques de F .

COROLLAIRE. Pour tout schéma S , toute formule F sur S sans variable libre, toute base de données Δ sur S et tout domaine $\mathbf{d}$, $\underline{\mathbf{d}}_{\Delta} F = \emptyset$ ou $\underline{\mathbf{d}}_{\Delta} F = \mathbf{d}^{\mathbf{V}}$.

L'évaluation s'étend aux requêtes de la manière suivante :

$$\underline{\mathbf{d}}_{\Delta} [\nu ; F] = \{ \delta \circ \nu \mid \delta \in \underline{\mathbf{d}}_{\Delta} F \} = (\underline{\mathbf{d}}_{\Delta} F) \circ \nu.$$

D'après le Lemme 3.C, l'évaluation $\underline{\mathcal{Q}}_{\Delta} [\nu ; F]$ d'une quelconque requête sur le domaine vide est vide.

Exemple 3.1. Soient la requête $\rho = [\langle x \rangle ; \exists y R_2(x, y)]$ et la base de données $\Delta(R_2) = \{ \langle a, b \rangle, \langle c, d \rangle \}$, $a, b, c, d \in \mathbf{D}$. On pose $K = \underline{\mathbf{D}} R_2(x, y)$. Soient δ et δ' les applications de $\{x, y\}$ dans $\mathbf{D}$ dont les graphes sont respectivement $\{ (x, a), (y, b) \}$ et $\{ (x, c), (y, d) \}$. On a $K = \{ \delta, \delta' \} \uplus \mathbf{D}(\mathbf{V} - \{x, y\})$ (règle 3.1b et lemme 3.D). Donc

$$\begin{aligned} \underline{\mathbf{D}} \rho &= ((K \downarrow (\mathbf{V} - \{y\})) \uplus \mathbf{D}\{y\}) \circ \langle x \rangle \text{ (règle 3.1f)} \\ &= ((\{ \{ (x, a) \}, \{ (x, c) \} \} \uplus \mathbf{D}(\mathbf{V} - \{x, y\})) \circ \langle x \rangle = \{ \langle a \rangle, \langle c \rangle \}. \end{aligned}$$

Exemple 3.2. Soient la requête $\rho = [\langle x \rangle ; R_0()]$, R_0 d'arité 0, la base $\Delta(R_0) = \{ \langle \rangle \}$ et $\mathbf{d} \subseteq \mathbf{D}$. On a : $\underline{\mathbf{d}} R_0() = \mathbf{d}^{\mathbf{V}}$ et $\underline{\mathbf{d}} \rho = \mathbf{d}^{\mathbf{V}} \circ \langle x \rangle = \mathbf{d}^{[1]}$.

Commentaire 3.1. Sur l'évaluation attestée. Comme il a déjà été dit (lemme 3.A), l'évaluation d'une requête ρ sur un domaine $\mathbf{d}$ ne dépend que de la réalisation des symboles du schéma intrinsèque $\text{sch } \rho$ de ρ . Elle ne dépend pas de la base tout entière. Il n'en est pas de même de l'évaluation attestée puisque, dépendant du domaine attesté, elle dépend de la base tout entière. *Exemple :* l'évaluation attestée de la requête $\rho = [\langle x \rangle ; \text{eg}(x, x)]$ sur toute base de données Δ est égale à l'ensemble $\mathbf{a}(\Delta)^{[1]}$ des uplets d'arité 1 sur le domaine attesté de Δ ; elle dépend de la base tout entière.

¹³ Étant donné un élément δ de $\mathbf{d}^{\mathbf{V}}$, une variable v et un élément a de $\mathbf{d}$, on note $\delta_{v/a}$ la valuation qui coïncide avec δ sur $(\mathbf{V} - \{v\})$ et telle que $\delta_{v/a}(v) = a$. Autrement dit, $\delta_{v/a} = (\delta \downarrow (\mathbf{V} - \{v\})) \uplus \{ (v, a) \}$.

LEMME 3.E. Pour tout schéma S , toute requête $[\nu; F]$ sur S , toute variable x telle que $x \notin \text{var } \nu$, toute base Δ sur S et tout domaine $\mathbf{d} \subseteq \mathbf{D}$, $\underline{\mathbf{d}}_{\Delta}[\nu; \exists x F] = \underline{\mathbf{d}}_{\Delta}[\nu; F]$.

Ce lemme permet de supprimer, dans le filtre des requêtes, certaines occurrences du quantificateur $\exists$. Ainsi, l'évaluation naturelle de l'exemple 3.1 est égale à celle de la requête $[< x >; R_2(x, y)]$. Le lemme explique, en partie, pourquoi les langages de bases de données limitent l'usage du quantificateur $\exists$. C'est le cas du langage Datalog – qui ne l'utilise pas – et du langage SQL.

Commentaire 3.2. Universalité des définitions (3.1). Les définitions (3.1) sont libres de toute restriction syntaxique. Aussi, présentent-elles l'avantage que toute requête est évaluable, y comprises celles spontanément ressenties comme « pathologiques », telles $[< x >; \exists x R_1(y)]$ ou $[< x >; R_0()]$. Très fréquemment, les auteurs procèdent différemment et ne proposent pas une définition universelle de l'évaluation. L'approche présentée par Abiteboul *et al.* [1995, p. 77] est de ce type (cf. Annexe). Elle s'appuie sur la notion syntaxique de variables libres. Toutes les formules sont évaluables, mais pas toutes les requêtes. Seules sont évaluables les requêtes $[\nu; F]$ pour lesquelles $\text{var } \nu \subseteq \text{libre } F$. Certes, cette manière de faire élimine de l'interprétation les « étranges » requêtes données ci-dessus en exemple. Mais *a priori* et de façon volontariste. Par ailleurs, elle n'élimine pas tous les cas « pathologiques », en particulier ceux que le paragraphe 3.2 va éliminer. L'universalité des définitions (3.1) a l'avantage de dépasser l'intuition. Le rôle des variables libres y demeure essentiel mais *a posteriori* (cf. Lemme 3.D).

On termine ce paragraphe par l'énoncé d'une propriété algorithmique de l'évaluation, utile pour le paragraphe suivant.

LEMME 3.F. Pour tout sous-ensemble fini $\mathbf{d}$ de $\mathbf{D}$, l'application $\underline{\mathbf{d}}$ est calculable.

Indication. Soit une requête $[\nu; F]$. Soit $\mathbf{d}$ fini. Si $\mathbf{d}$ est vide alors l'évaluation $\underline{\mathbf{d}}_{\Delta}[\nu; F]$ est vide, donc calculable. Sinon, pour toute base de données Δ ,

$$\underline{\mathbf{d}}_{\Delta}[\nu; F] = (\underline{\mathbf{d}}_{\Delta} F) \circ \nu = ((\underline{\mathbf{d}}_{\Delta} F \downarrow \text{libre } F) \uplus \mathbf{d}^{\mathbf{V} - \text{libre } F}) \circ \nu \text{ (lemme 3.D).}$$

L'ensemble $(\text{var } F \cup \text{var } \nu)$ incluant l'ensemble libre F , on a :

$$\underline{\mathbf{d}}_{\Delta}[\nu; F] = ((\underline{\mathbf{d}}_{\Delta} F \downarrow (\text{var } F \cup \text{var } \nu)) \uplus \mathbf{d}^{\mathbf{V} - (\text{var } F \cup \text{var } \nu)}) \circ \nu$$

L'ensemble $\mathbf{V} - (\text{var } F \cup \text{var } \nu)$ étant disjoint de $\text{var } \nu$, on a :

$$\underline{\mathbf{d}}_{\Delta}[\nu; F] = (\underline{\mathbf{d}}_{\Delta} F \downarrow (\text{var } F \cup \text{var } \nu)) \circ \nu.$$

Il en résulte que l'évaluation $\underline{\mathbf{d}}_{\Delta}[\nu; F]$ est incluse dans $\mathbf{d}(\text{var } F \cup \text{var } \nu) \circ \nu$. Or, le domaine $\mathbf{d}$ est fini par hypothèse, donc aussi l'ensemble $\mathbf{d}(\text{var } F \cup \text{var } \nu)$. Il est simple de concevoir un algorithme qui énumère les éléments de $\mathbf{d}(\text{var } F \cup \text{var } \nu)$ et de montrer que la question de savoir si un élément de $\mathbf{d}(\text{var } F \cup \text{var } \nu)$ appartient à $\underline{\mathbf{d}} F \downarrow (\text{var } F \cup \text{var } \nu)$ est décidable. D'où une construction, élément par élément, de l'évaluation $\underline{\mathbf{d}}_{\Delta}[\nu; F]$.

Commentaire 3.3. On fait remarquer que le mérite de l'algorithme décrit dans la preuve du lemme 3.F est uniquement théorique. Car, en pratique, l'algorithme est irréaliste. Il est usuel que le domaine attesté d'une base de données contienne, par exemple, 10^7 éléments. Si l'ensemble $(\text{var } F \cup \text{var } \nu)$ contient 4 variables, le calcul de l'évaluation attestée, pour ne considérer qu'elle, conduirait à devoir énumérer $10^{4 \times 7}$ valuations !

3.2 REQUÊTES INDÉPENDANTES ET INTERROGATIONS

Au regard de ce qu'on entend naturellement par questionnement d'une base de données, les outils formels du paragraphe précédent se révèlent trop généraux et, de ce fait, inadaptés. La raison est que l'évaluation $\underline{d}_{\Delta}\rho$ d'une requête ρ dépend à la fois du domaine $\underline{d}$ et de la base de données Δ . Or, la dépendance vis à vis du domaine est contraire à l'idée de questionnement. Quand on consulte le bottin à la recherche du numéro de téléphone d'un individu x , le domaine d'évaluation n'intervient pas : le processus ne dépend que de x (la question) et du contenu du bottin (la base de données). Pour « coller » à l'idée de questionnement, l'évaluation doit donc être « libérée » de la dépendance vis à vis du domaine. Pour ce faire, on commence par écarter les domaines $\underline{d}$ qui sont inclus dans le domaine attesté, i.e. $\underline{d} \subseteq \underline{a}(\Delta)$. En effet, ces domaines introduisent un filtrage sur les données qui se surajoute artificiellement à celui qu'opère déjà le filtre de la requête, le cas le plus extrême de ces filtrages surnuméraires étant celui où $\underline{d} = \emptyset$ puisque tout est éliminé (cf. Lemme 3.C). On ne considère donc dorénavant que des évaluations telles que $\underline{a}(\Delta) \subseteq \underline{d} \subseteq \underline{D}$.

Cette limitation posée, toutes les requêtes ne deviennent pas pour autant acceptables. En effet, questionnant une base de données, on s'attend d'une part à ce que les réponses soient à valeur dans le domaine attesté de la base, en d'autres termes, ne contiennent que des valeurs présentes dans la base, et d'autre part qu'elles soient en nombre fini. Or, certaines requêtes ne satisfont pas ces exigences. C'était déjà le cas de la requête de l'exemple 3.2 dont l'évaluation $\underline{d}^{[1]}$ est infinie avec $\underline{d}$, c'est aussi celui de la requête $[< nom > ; \neg personne(n^{os}, nom, véhicule)]$ sur le schéma de l'exemple 2.1. Son évaluation sur un domaine $\underline{d} \subseteq \underline{a}(\Delta)$ est égale à $(\underline{d}^V - K) \circ < nom >$, en posant $K = \underline{d} \text{ personne}(n^{os}, nom, véhicule)$. Il est simple d'établir qu'elle est encore égale à $(\underline{d} - \underline{a}(\Delta))^{[1]} \cup (\underline{a}(\Delta)^{[1]} - K \circ < nom >)$, qui dépend du domaine $\underline{d}$. Si $\underline{d}$ est infini, l'évaluation l'est aussi. Or, à partir d'une base de données, nul ne s'attend à une réponse infinie et on comprend que la requête ne soit pas acceptable. La notion ci-après d'indépendance appréhende cette exigence. Elle est introduite par le Lemme 3.G

LEMME 3.G. Soit une requête ρ . S'il existe un schéma S , $sch \rho \subseteq S$, tel que

$$(3.2) \quad \forall \Delta \in \mathcal{B}(S), \forall \underline{d}, \underline{a}(\Delta) \subseteq \underline{d} \subseteq \underline{D}, \text{ alors } \underline{d}_{\Delta}\rho = \underline{att}_{\Delta}\rho,$$

alors la propriété (3.2) est vérifiée par tout schéma S' , $sch \rho \subseteq S'$.

Démonstration. Soit une requête ρ pour laquelle il existe un schéma S , $sch \rho \subseteq S$, vérifiant (3.2). Soit un schéma S' , $sch \rho \subseteq S'$, et une base de données Δ' sur S' .

Cas 1 : $S \subset S'$. Soit $\underline{d}$, $\underline{a}(\Delta') \subseteq \underline{d} \subseteq \underline{D}$. On pose $\Delta = \Delta' \downarrow S$. On a :

$$\underline{d}_{\Delta'}\rho \stackrel{(*)}{=} \underline{d}_{\Delta}\rho \stackrel{(**)}{=} \underline{att}_{\Delta}\rho \stackrel{(***)}{=} \underline{a}(\Delta')_{\Delta}\rho \stackrel{(*)}{=} \underline{a}(\Delta')_{\Delta'}\rho = \underline{att}_{\Delta'}\rho ;$$

(*) d'après le Lemme 3.A ; (**) d'après l'hypothèse ; (***) d'après l'hypothèse et $\underline{a}(\Delta) \subseteq \underline{a}(\Delta')$.

Cas 2 : $S' \subset S$. Il suffit de considérer la base de données Δ sur S telle que si $R \in S'$ alors $\Delta(R) = \Delta'(R)$ sinon $\Delta(R) = \emptyset$. On a : $\mathbf{a}(\Delta) = \mathbf{a}(\Delta')$. Soit $\mathbf{d}, \mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$. On a : $\underline{\mathbf{d}}_{\Delta'} \rho =^{(*)} \underline{\mathbf{d}}_{\Delta} \rho =^{(**)} \mathbf{att}_{\Delta} \rho =^{(****)} \mathbf{att}_{\Delta'} \rho$; $^{(****)}$ puisque $\mathbf{a}(\Delta) = \mathbf{a}(\Delta')$.

Il résulte du Lemme 3.G que la propriété (3.2) ne dépend que de la requête, pas du schéma sur laquelle elle est définie. Dès lors, une requête ρ sera dite *indépendante du domaine d'évaluation* – plus simplement *indépendante* – si elle vérifie la propriété (3.2) appliquée à son schéma intrinsèque – ou à tout schéma qui l'inclut. Si une requête ρ est indépendante, alors, pour tout schéma S , $\text{sch } \rho \subseteq S$, toute base de données Δ sur S et tout domaine $\mathbf{d}, \mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$, l'évaluation $\underline{\mathbf{d}}_{\Delta} \rho$ ne dépend que de Δ , on l'appellera *évaluation* de ρ sur Δ – sans préciser le domaine – et notera $\mathbf{e}_{\Delta} \rho$ ou $\mathbf{e} \rho$; on a : $\mathbf{e}_{\Delta} \rho = \mathbf{att}_{\Delta} \rho$. Elle est toujours finie et ses éléments sont des uplets à valeur dans le domaine attesté de la base. On note $\mathcal{R}^{ind}$ l'ensemble des requêtes indépendantes.

Commentaire 3.4. Indépendance et évaluation vide. La définition de l'indépendance impose que l'évaluation sur tout sur-ensemble du domaine attesté $\mathbf{a}(\Delta)$ d'une base de données soit égale à l'évaluation attestée. Or, le domaine attesté $\mathbf{a}(\Delta)$ n'est jamais vide (Lemme 2.A). Autrement dit, l'évaluation vide n'intervient pas dans la propriété d'indépendance. Une requête peut donc être indépendante même si $\emptyset_{\Delta} \rho \neq \mathbf{att}_{\Delta} \rho$.

Exemple 3.3. La requête $\rho = [\langle x \rangle ; \exists y R_2(x, y)]$ est indépendante. En effet, soient un schéma S , $\{ \Omega, R_2 \} \subseteq S$, une base Δ sur S et $\mathbf{d} \supseteq \mathbf{a}(\Delta)$. On a :

$$\underline{\mathbf{d}}_{\Delta} \rho = \underline{\mathbf{d}}[\langle x \rangle ; R_2(x, y)] \text{ (Lemme 3.E) } = \underline{\mathbf{d}} R_2(x, y) \circ \langle x \rangle = \mathbf{att} R_2(x, y) \circ \langle x \rangle.$$

Exemple 3.4. La requête $\rho = [\langle x \rangle ; \forall y (R_1(y) \Rightarrow R_2(x, y))]$ n'est pas indépendante. En effet, soient un schéma S , $\{ \Omega, R_1, R_2 \} \subseteq S$, une base Δ sur S et $\mathbf{d} \supseteq \mathbf{a}(\Delta)$. Posant $K(x, y) = (R_1(y) \& \neg R_2(x, y))$, on a : $\underline{\mathbf{d}} \rho = [\langle x \rangle ; \neg \exists y K(x, y)]$

$$\begin{aligned} &= (\mathbf{d}^{\mathbf{V}} - \underline{\mathbf{d}} \exists y K(x, y)) \circ \langle x \rangle \\ &= (\mathbf{d}^{\mathbf{V}} - ((\underline{\mathbf{d}} K(x, y) \downarrow (\mathbf{V} - \{y\}) \uplus \mathbf{d}^{\{y\}})) \circ \langle x \rangle. \end{aligned}$$

Par ailleurs, $\underline{\mathbf{d}} K(x, y) = \underline{\mathbf{d}} R_1(y) \cap \underline{\mathbf{d}} \neg R_2(x, y)$. Si $\Delta(R_1)$ est vide, alors $\underline{\mathbf{d}} R_1(y)$ est vide, donc $\underline{\mathbf{d}} K(x, y)$ aussi et $\underline{\mathbf{d}} \rho = \mathbf{d}^{\mathbf{V}} \circ \langle x \rangle = \mathbf{d}^{[1]}$, qui dépend de $\mathbf{d}$.

Exemple 3.5. La requête $[\langle x \rangle ; \mathbf{eg}(x, y)]$ est indépendante tandis que $[\langle x \rangle ; \neg \mathbf{eg}(x, y)]$ ne l'est pas. Soit une base de données Δ sur un schéma S quelconque et $\mathbf{d} \supseteq \mathbf{a}(\Delta)$. On a $\underline{\mathbf{d}} [\langle x \rangle ; \mathbf{eg}(x, y)] = \{ \delta \in \mathbf{d}^{\mathbf{V}} \mid \delta(x) = \delta(y) \} \circ \langle x \rangle$. Or, $\mathbf{a} \neq \emptyset$ (Lemme 2.A), donc $\mathbf{d} \neq \emptyset$. Dès lors, $\underline{\mathbf{d}} [\langle x \rangle ; \mathbf{eg}(x, y)] = \{ \langle x \rangle \}$ ne dépend pas de $\mathbf{d}$ et $[\langle x \rangle ; \mathbf{eg}(x, y)]$ est indépendante. En revanche, si $|\mathbf{a}(\Delta)| = 1$, alors $\mathbf{att} [\langle x \rangle ; \neg \mathbf{eg}(x, y)] = \emptyset$ tandis que si $|\mathbf{d}| > 1$, alors $\underline{\mathbf{d}} [\langle x \rangle ; \neg \mathbf{eg}(x, y)] = \{ \langle x \rangle \}$.

Exemple 3.6. La requête $[\langle y \rangle ; R_1(x) \& \neg R_1(x)]$ est indépendante puisque son évaluation est vide quels que soient le domaine d'évaluation et la base de données. L'exemple montre que l'inclusion $\text{var } \nu \subseteq \text{libre } F$ n'est pas nécessaire à l'indépendance d'une requête (cf. Lemme A.B.(B) dans l'Annexe).

On appelle *interrogation* d'arité $n \geq 0$ portant sur un schéma S toute application de l'ensemble $\mathcal{B}(S)$ des bases de données sur S dans $\mathbf{2}^D$ qui associe à chaque base Δ sur S un sous-ensemble *fini* de l'ensemble $\mathbf{a}(\Delta)^{[n]}$ des uplets d'arité n sur le domaine attesté $\mathbf{a}(\Delta)$. Étant donnés une requête indépendante ρ d'arité $n \geq 0$ et un schéma S , $S \supseteq \text{sch } \rho$, on considère l'interrogation f_ρ portant sur le schéma S définie par $f_\rho(\Delta) = \mathbf{e}_{\Delta\rho}$, $\forall \Delta \in \mathcal{B}(S)$. L'arité de f_ρ est n . On dit que ρ *dénote* l'interrogation f_ρ portant sur S . Pour chaque schéma S , $S \supseteq \text{sch } \rho$, une requête ρ dénote une unique interrogation. Aussi, en dénote-t-elle une infinité, autant que de schémas contenant $\text{sch } \rho$. Si on note int_ρ celle portant sur le schéma intrinsèque de ρ , alors $f_\rho(\Delta) = \text{int}_\rho(\Delta \downarrow \text{sch } \rho)$. Deux requêtes indépendantes ρ et ρ' sont *équivalentes* si elles dénotent la même interrogation sur le schéma $\text{sch } \rho \cup \text{sch } \rho'$. *Exemple* : $[\langle x \rangle ; R(x)]$ et $[\langle x \rangle ; R(x) \ \& \ (R'(y) \vee \neg R'(y))]$ sont indépendantes et équivalentes. L'infinité d'interrogations qu'une requête dénote est appelé son *pouvoir d'expression*. Par extension, celui d'un ensemble de requêtes est l'union, sur les éléments dudit ensemble, de leur pouvoir d'expression. On note $\mathcal{E}^{ind}$ le pouvoir d'expression de l'ensemble $\mathcal{R}^{ind}$ des requêtes indépendantes.

THÉORÈME 3.H. Étant donnés une requête indépendante ρ et un schéma S , $S \supseteq \text{sch } \rho$, l'interrogation portant sur S que ρ dénote est une application calculable.

Indication. Soient une requête indépendante ρ , un schéma S , $S \supseteq \text{sch } \rho$, et f l'interrogation portant sur S que ρ dénote. Pour toute base de données Δ sur S , on a $f(\Delta) = \mathbf{att}_{\Delta\rho} = \text{eval}(\mathbf{a}(\Delta), \Delta, F)$. D'après le lemme 3.F, eval est calculable pour les domaines finis. L'interrogation f est donc calculable si l'application qui associe à toute base de données Δ son domaine attesté $\mathbf{a}(\Delta)$ l'est. Or, s'appuyant sur l'analogie qui assimile la réalisation d'un symbole relationnel à un fichier d'enregistrements (cf. Commentaire 2.2), il est aisé de concevoir un algorithme qui calcule la liste de tous les éléments du domaine attesté : en parcourant tous les enregistrements de chaque fichier, on accumule dans une liste les valeurs de tous les champs. La suppression des doublons de la liste conduit au domaine attesté.

Eu égard au Théorème 3.H, les requêtes indépendantes ne sont plus uniquement une manière d'exprimer des questions, elles permettent aussi d'évaluer les réponses, puisque les interrogations qu'elles dénotent sont calculables. L'algorithme de leur évaluation, qui combine les algorithmes décrits dans les preuves du Lemme 3.F et du Théorème 3.H, est canonique – même si son mérite est uniquement formel car sa complexité en temps est « inabordable » (cf. Commentaire 3.3), (cf. Chapitre 4 pour une approche plus performante). Dès lors, les requêtes indépendantes ne relèvent plus seulement de la logique, elles sont aussi une écriture algorithmique. La solution « base de données » ci-dessous du jeu de l'exemple 2.2 est fondée sur cette remarque. En « posant » les requêtes d'un problème – comme on pose ses équations – on dispose *ipso facto*, pour peu que les requêtes soient indépendantes, de ses solutions.

Exemple 3.7 (suite de l'exemple 2.2) : soient $p^\sim$ le nombre que Paul connaît, $s^\sim$ celui que Serge connaît, $x^\sim$ et $y^\sim$ les nombres cherchés et B_0 la base de données décrite dans l'exemple 2.2. On a $\langle x^\sim, y^\sim, s^\sim \rangle \in \Delta(\text{add})$ et $\langle x^\sim, y^\sim, p^\sim \rangle \in \Delta(\text{mult})$. Paul et Serge sont supposés disposer des tables *add* et *mult* de B_0 . Rapporté à la table *mult* et sans faire

intervenir de considérations liées à l'arithmétique, l'énoncé de Paul *je ne connais pas les nombres* n'est possible que si $p^\sim$ apparaît au moins deux fois dans la dernière colonne de cette table, plus formellement s'il existe, dans la réalisation $\Delta(mult)$, deux uplets $\langle x, y, p^\sim \rangle$ et $\langle x', y', p^\sim \rangle$ différents. On considère alors la requête

$$\rho_0 = [\langle p \rangle ; mult(x, y, p) \ \& \ mult(x', y', p) \ \& \ \neg(eg(x, x') \ \& \ eg(y, y'))].$$

Elle est indépendante et $\langle p^\sim \rangle \in \mathbf{ep}_0$. L'évaluation $\mathbf{ep}_0$ limite donc le champ des valeurs possibles de $p^\sim$. Il s'agit d'un ensemble d'uplets d'arité 1 qui peut être stocké dans la mémoire relationnelle. Pour cela, on introduit un nouveau symbole π_1 d'arité 1 dans le schéma de la base B_0 à qui l'on affecte l'ensemble $\mathbf{ep}_0$, ce dont rend compte formellement l'histoire $(B_0, (\uparrow \pi_1), (\pi_1 \Leftarrow \mathbf{ep}_0))$.

De même, la réplique de Serge *je le savais* n'est possible que si pour tout uplet de la forme $\langle x, y, s^\sim \rangle$ dans la réalisation $\Delta(add)$ et pour tout uplet de la forme $\langle x, y, p \rangle$ dans la réalisation $\Delta(mult)$ alors $\langle p \rangle \in \Delta(\pi_1)$. On considère alors la requête :

$$\rho_1 = [\langle s \rangle ; add(x', y', s) \ \& \ \forall p \forall x \forall y ((add(x, y, s) \ \& \ mult(x, y, p)) \Rightarrow \pi_1(p))].$$

Il est intéressant d'y souligner le rôle essentiel de la sous-formule $add(x', y', s)$. Elle impose à la variable s de n'être valuée que dans le domaine attesté et, de ce fait, assure l'indépendance de la requête ρ_1 , qui ne le serait pas en son absence (cf. Exemple 3.4). On a : $\langle s^\sim \rangle \in \mathbf{ep}_1$. L'évaluation $\mathbf{ep}_1$ limite le champ des valeurs possibles de $s^\sim$. On la stocke dans la mémoire relationnelle grâce à un nouveau symbole σ_1 d'arité 1.

Enfin, en suivant la même démarche, on vérifie aisément que la seconde réplique de Paul conduit à la requête indépendante $\rho_2 = [\langle p \rangle ; \exists! s \ \pi_1(p) \ \& \ mult(x, y, p) \ \& \ add(x, y, s) \ \& \ \sigma_1(s)]^{14}$ et que $\langle p^\sim \rangle \in \mathbf{ep}_2$. On stocke $\mathbf{ep}_2$ grâce au nouveau symbole π_2 . De même, la seconde réplique de Serge conduit à la requête indépendante $\rho_3 = [\langle s \rangle ; \exists! p \ \sigma_1(s) \ \& \ add(x, y, s) \ \& \ mult(x, y, p) \ \& \ \pi_2(p)]$ et $\langle s^\sim \rangle \in \mathbf{ep}_3$. On stocke $\mathbf{ep}_3$ grâce au nouveau symbole σ_2 . Finalement, les entiers $s^\sim$ et $p^\sim$ étant liés par le fait qu'ils sont la somme et le produit des mêmes nombres $x^\sim$ et $y^\sim$, l'uplet $\langle x^\sim, y^\sim \rangle$ appartient nécessairement à l'évaluation de la requête indépendante $\rho_4 = [\langle x, y \rangle ; \sigma_2(s) \ \& \ \pi_2(p) \ \& \ add(x, y, s) \ \& \ mult(x, y, p)]$. La séquence des calculs décrits est représentée par l'histoire ci-dessous :

$$(B_0, (\uparrow \pi_1), (\pi_1 \Leftarrow \mathbf{ep}_0), (\uparrow \sigma_1), (\sigma_1 \Leftarrow \mathbf{ep}_1), (\uparrow \pi_2), (\pi_2 \Leftarrow \mathbf{ep}_2), \\ (\uparrow \sigma_2), (\sigma_2 \Leftarrow \mathbf{ep}_3), (\uparrow res), (res \Leftarrow \mathbf{ep}_4))$$

On souligne que la procédure suivie ne fait jamais appel à des considérations arithmétiques. La base initiale B_0 pourrait être quelconque. Lorsqu'elle est celle

¹⁴ La quantificateur $\exists!$ est le quantificateur existentiel avec unicité. L'écriture $\exists! s F(s)$ est une abréviation de la formule $\exists s (F(s) \ \& \ (\forall s' F(s') \Rightarrow eg(s, s'))$.

présentée dans l'exemple 2.2, la réalisation du symbole *res* ne contient que l'uplet $\langle 4, 13 \rangle$, qui est la solution $\langle x, y \rangle$ cherchée¹⁵.

Commentaire 3.5. Homogénéité entre réalisation et évaluation. L'évaluation $e_{\Delta\rho}$ d'une requête indépendante ρ d'arité $n \geq 0$ est un ensemble fini d'uplets d'arité n . Sa nature est donc homogène à celle des réalisations $\Delta(R)$ qui sont aussi des ensembles d'uplets de même arité. Dès lors, l'évaluation $e_{\Delta\rho}$ est stockable dans la mémoire relationnelle. Cette possibilité a été largement utilisée dans l'exemple 3.7. L'homogénéité entre évaluation des requêtes et réalisation est la clef de voûte des machines relationnelles puisqu'elle permet l'opération informatique d'affectation, opération constitutive des machines à mémoire.

Commentaire 3.6. Absence des constantes. Les requêtes telles qu'elles ont été définies, donc aussi les requêtes indépendantes, ne contiennent pas de constantes alors qu'en pratique elles sont indispensables. Eu égard aux définitions présentées, aucune requête ne peut exprimer la question *quelle est l'heure de départ du train pour Vierzon ?*, car Vierzon est une constante. Or, il est bien entendu essentiel qu'elle puisse l'être et, d'ailleurs, tous les langages BD autorisent les constantes. Toutefois, leur présence masque un aspect du rôle des questions que l'approche sans constante, développée ici, vise à faire apparaître. Lorsqu'une question utilise une constante, par exemple Vierzon, elle apporte une information, à savoir ladite constante. Autrement dit, dans le « tandem » que forment une base de données et une question, les informations ne sont pas uniquement du côté de la base mais aussi de la question, pour une part certes minime mais réelle et dont le Chapitre 5 montrera l'importance. Cette remarque se formalise comme ceci. Si la question utilise n constantes $c_1, c_2, \dots, c_n$, $n \geq 1$, on considère, à partir de la base originelle B_0 , l'histoire $(B_0, (\uparrow\uparrow Q), (Q \Leftarrow \{ \langle c_1, c_2, \dots, c_n \rangle \}))$. Elle introduit un nouveau symbole Q qui stocke les constantes de la question¹⁶. Sur la base terminale, la question avec constantes se transcrit par une requête sans constantes, puisqu'elles sont accessibles *via* le symbole Q . Dans l'exemple des trains pour Vierzon, le symbole Q serait d'arité 1 et stockerait le singleton $\{ \langle \text{Vierzon} \rangle \}$. La question se transcrirait par la requête sans constante $[\langle h \rangle ; \text{départ}(d, h) \ \& \ Q(d)]$. Une question avec constantes se transcrit donc par une requête sans constante sur une base qui intègre la base originelle et les constantes de la question. Les constantes se révèlent alors inutiles, tout du moins formellement.

3.3 APPROCHE SYNTAXIQUE DU POUVOIR D'EXPRESSION DES REQUÊTES INDÉPENDANTES

Le paragraphe 3.2 a défini sémantiquement les requêtes indépendantes comme étant la catégorie de requêtes naturellement adaptée au questionnement des bases de données. Dans ce paragraphe, on commence par montrer qu'elles n'ont pas de définition algorithmique (Théorème 3.K), ce qui les écarte de pouvoir être une abstraction

¹⁵ Le code SQL du jeu est téléchargeable à l'adresse : <http://cams-atid.ivry.cnrs.fr/bd/bd.html>

¹⁶ On pourrait tout aussi bien réserver un symbole relationnel d'arité 1 par constante : $(Q_1 \Leftarrow \{ \langle c_1 \rangle \}), (Q_2 \Leftarrow \{ \langle c_2 \rangle \}), \dots, (Q_n \Leftarrow \{ \langle c_n \rangle \})$

calculable (cf. §0.5). Toutefois, ce résultat négatif est contournable puisqu'à sa suite, on établit l'existence d'un sous-ensemble de l'ensemble des requêtes indépendantes qui est récursif et a le *même* pouvoir d'expression (Théorème 3.O). Ce sous-ensemble n'est d'ailleurs pas unique : l'objectif du présent paragraphe est uniquement de s'assurer de son existence sans se préoccuper de son efficacité algorithmique ou de son ergonomie (pour ces aspects, cf. Chapitre 4).

Étant donné un schéma S , on désigne par n l'arité de ses symboles de plus grande arité. Du fait du symbole distingué Ω , $n \geq 1$. Si x, v_1, v_2 et v_{n-1} sont n éléments différents de $\mathbf{V}$, on désigne alors par $\alpha_S(x, v_1, v_2, \dots, v_{n-1})$ la formule obtenue comme suit. Si le schéma contient, par exemple, un symbole R_3 d'arité 3, on associe à R_3 la disjonction :

$$\exists v_1 \exists v_2 R_3(x, v_1, v_2) \vee \exists v_1 \exists v_2 R_3(v_1, x, v_2) \vee \exists v_1 \exists v_2 R_3(v_1, v_2, x)$$

dans laquelle la variable x est libre et occupe successivement toutes les places de R_3 . La même opération peut être réalisée pour tous les symboles relationnels d'arité $\neq 0$ d'un schéma S . Chacun d'eux conduit à une disjonction. La disjonction de ces disjonctions est la formule $\alpha_S(x, v_1, v_2, \dots, v_{n-1})$ visée, dont x est la seule variable libre et dont S est le schéma intrinsèque.

LEMME 3.I. Étant donnés un schéma S , n l'arité de ses symboles de plus grande arité et n variables x, v_1, v_2 et v_{n-1} toutes différentes, alors,

(3.3a) pour toute base Δ sur S et tout domaine $\mathbf{d}$, $\mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$,

$$\underline{\mathbf{d}}_{\Delta} \alpha_S(x, v_1, v_2, \dots, v_{n-1}) = \mathbf{a}(\Delta)^{\{x\}} \uplus \mathbf{d}(\mathbf{V} - \{x\}) ;$$

(3.3b) la requête $[< x > ; \alpha_S(x, v_1, v_2, \dots, v_{n-1})]$ est indépendante et son évaluation sur une base de données Δ sur S est égale à $\mathbf{a}(\Delta)[1]$.

D'après la propriété (3.3b), étant donnée une base Δ sur un schéma S , l'évaluation $\mathbf{e}_{\Delta}[< v > ; \alpha_S(x, v_1, v_2, \dots, v_{n-1})]$ est égale au domaine attesté de la base Δ sous la forme d'un ensemble d'uplets d'arité 1.

LEMME 3.J. Étant donnée une requête ρ , la question de savoir si :

$$(3.4) \quad \forall \Delta \in \mathcal{B}(sch \rho), \forall \mathbf{d}, \mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}, \underline{\mathbf{d}}_{\Delta} \rho = \emptyset$$

n'est pas décidable.

Démonstration. La preuve s'appuie sur le résultat suivant. Étant donnée une correspondance de Post π , il existe une requête indépendante ρ_{π} telle que π a une solution ssi il existe une base Δ sur le schéma intrinsèque $sch \rho_{\pi}$ de ρ_{π} telle que $\underline{\mathbf{d}}_{\Delta} \rho_{\pi} \neq \emptyset$. Pour une construction explicite de ρ_{π} , voir la preuve du théorème 6.3.1 dans [Abiteboul *et al.*, 1995, p. 123-125]. On suppose alors que la question (3.4) est décidable et on applique l'algorithme de décision à ρ_{π} . Si (3.4) est vraie pour ρ_{π} , alors, compte tenu du lemme cité, π n'a pas de solution. Si (3.4) est fausse, alors $\exists \Delta \in \mathcal{B}(sch \rho_{\pi}), \exists \mathbf{d}, \mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}, \underline{\mathbf{d}}_{\Delta} \rho_{\pi} \neq \emptyset$. Or, ρ_{π} est indépendante, donc $\underline{\mathbf{d}}_{\Delta} \rho_{\pi} = \underline{\mathbf{d}}_{\Delta} \rho_{\pi}$. Il en résulte que $\exists \Delta \in \mathcal{B}(sch \rho_{\pi}), \underline{\mathbf{d}}_{\Delta} \rho_{\pi} \neq \emptyset$, autrement dit, que π a une solution. Ce qui contredit l'indécidabilité du problème de correspondance de Post.

THÉOREME 3.K. Étant donnée une requête quelconque, la question de savoir si elle est indépendante n'est pas décidable.

Démonstration. Soient une formule F , S son schéma intrinsèque et la requête $\rho = [\nu; F \& \neg G]$, dans laquelle G est une formule de la forme $\alpha_S(x, v_1, v_2, \dots, v_{n-1})$, la variable x n'apparaissant ni dans le filtre F ni dans l'extracteur ν . On montre qu'une cns pour que la requête ρ soit indépendante est que l'évaluation $\underline{d}[\nu; F]$ de la requête $[\nu; F]$ sur toute base Δ sur S et pour tout domaine $\mathbf{d}$, $\forall \mathbf{d} \supseteq \mathbf{a}(\Delta)$, est vide. Il en résultera le théorème, à partir du Lemme 3.J.

La condition est suffisante. Inversement, on suppose ρ indépendante. Soient Δ une base de données sur S et $\mathbf{a}$ son domaine attesté. On montre que $\underline{d}F = \emptyset$, $\forall \mathbf{d} \supseteq \mathbf{a}$. On a $\underline{d}\rho = (\underline{d}F \cap \underline{d}(\neg G)) \circ \nu$. Par ailleurs, $\underline{d}(\neg G) = \mathbf{d}^{\mathbf{V}} - (\mathbf{a}^{\{x\}} \cup \mathbf{d}^{\mathbf{V}} - \{x\})$, d'après la propriété (3.3a), donc, pour $\mathbf{d} = \mathbf{a}$, $\underline{d}(\neg G) = \emptyset$. D'où il résulte que $\text{att}\rho = \emptyset$. Pour que ρ soit indépendante, il faut donc que $\underline{d}\rho = \emptyset$, $\forall \mathbf{d} \supset \mathbf{a}$, ou encore que $\underline{d}F \cap \underline{d}(\neg G) = \emptyset$, $\forall \mathbf{d} \supset \mathbf{a}$. Puisque $x \notin \text{var } F$, $\underline{d}F$ et $\underline{d}(\neg G)$ se factorisent respectivement en :

$$\underline{d}F = (\underline{d}F \downarrow \text{var } F) \cup \mathbf{d}^{\{x\}} \cup \mathbf{d}^{\mathbf{V}} - (\text{var } F \cup \{x\});$$

$$\underline{d}(\neg G) = \mathbf{d} \text{ var } F \cup (\mathbf{d} - \mathbf{a})^{\{x\}} \cup \mathbf{d}^{\mathbf{V}} - (\text{var } F \cup \{x\}).$$

D'après la règle (1.1b), $\underline{d}F \cap \underline{d}(\neg G) = (\underline{d}F \downarrow \text{var } F) \cup (\mathbf{d} - \mathbf{a})^{\{x\}} \cup \mathbf{d}^{\mathbf{V}} - (\text{var } F \cup \{x\})$. Or, $(\mathbf{d} - \mathbf{a})^{\{x\}} \neq \emptyset$ puisque $\mathbf{d} \supset \mathbf{a}$ et $\mathbf{d}^{\mathbf{V}} - (\text{var } F \cup \{x\}) \neq \emptyset$. Pour que $\underline{d}F \cap \underline{d}(\neg G) = \emptyset$, il est nécessaire que $(\underline{d}F \downarrow \text{var } F) = \emptyset$, donc, d'après la règle (1.1a), que $\underline{d}F = \emptyset$.

Il résulte du théorème précédent que l'ensemble $\mathcal{R}^{ind}$ des requêtes indépendantes ne peut être le langage d'une machine relationnelle puisque le test d'indépendance n'est pas programmable. On montre maintenant que son pouvoir d'expression $\mathcal{E}^{ind}$ peut toutefois être atteint par un de ses sous-ensembles récurrents. Pour cela, étant donné un schéma S et $x \in \mathbf{V}$, on note $\underline{\alpha}_S(x)$ l'ensemble des formules $\alpha_S(x, v_1, v_2, \dots, v_{n-1})$ dans lesquelles les variables $v_i, 1 \leq i \leq n-1$, sont toutes différentes et différentes de x . L'ensemble $\underline{\alpha}_S(x)$ est un ensemble infini de formules dont les évaluations sont toutes égales d'après la propriété (3.3a). La syntaxe des formules s'étend naturellement à des ensembles de formules. Ainsi, si K est un ensemble de formules, alors K est l'ensemble $\{\neg F \mid F \in K\}$ de formules. De même, la notation $[\nu; K]$ désigne l'ensemble $\{[\nu; F] \mid F \in K\}$ de requêtes. Enfin, si L est un ensemble fini de variables, $\underline{\alpha}_S(L)$ désigne la conjonction des $\underline{\alpha}_S(l)$, $l \in L$. On considère alors l'application attesté_S de l'ensemble $\mathcal{F}(S)$ des formules sur un schéma S dans l'ensemble des parties de $\mathcal{F}(S)$ qui associe à toute formule F l'ensemble $\text{attesté}_S F$ tel que si F est de la forme :

$$(3.5a) \quad R(), R \in S \text{ d'arité } 0, \text{ alors } \text{attesté}_S F = \{F\};$$

$$(3.5b) \quad R(v_1, v_2, \dots, v_n), R \in S \text{ d'arité } n > 0, \text{ alors } \text{attesté}_S F = \{F\};$$

$$(3.5c) \quad \text{eg}(v', v''), v' \in \mathbf{V}, v'' \in \mathbf{V}, \text{ alors } \text{attesté}_S F = \underline{\alpha}_S(v') \& \underline{\alpha}_S(v'') \& F;$$

$$(3.5d) \quad F' \text{ et libre } F' \neq \emptyset, \text{ alors } \text{attesté}_S F = \underline{\alpha}_S(\text{libre } F') \& \neg \text{attesté}_S F';$$

$$(3.5e) \quad F' \text{ et libre } F' = \emptyset, \text{ alors } \text{attesté}_S F = \neg \text{attesté}_S F';$$

$$(3.5f) \quad F' \& F'', \text{ alors } \text{attesté}_S F = \text{attesté}_S F' \& \text{attesté}_S F'';$$

$$(3.5g) \quad \exists v F', \text{ alors } \text{attesté}_S F = \exists v (\underline{\alpha}_S(v) \& \text{attesté}_S F').$$

Exemple 3.8. $\text{attesté}_S \neg \neg R(x) = \underline{\alpha}_S(x) \& \neg(\underline{\alpha}_S(x) \& \neg R(x))$.

On note $\mathcal{F}^{att}$ l'ensemble des formules égal à l'union des images $attesté_S F$ sur tout schéma S et toute formule F sur S . L'application $attesté_S$ s'étend aux requêtes de la manière suivante. Si une requête ρ est de la forme :

- $[<> ; F]$, alors $attesté_S \rho = [<> ; attesté_S F]$;
- $[v ; F]$, $v \neq <>$, alors $attesté_S \rho = [v ; \underline{\alpha}(var\ v) \ \& \ attesté_S F]$.

Exemple 3.9. $attesté_S [<y> ; \neg\neg R(x)] = [<y> ; \underline{\alpha}_S(y) \ \& \ \underline{\alpha}_S(x) \ \& \ \neg(\underline{\alpha}_S(x) \ \& \ \neg R(x))]$.

On désigne par $\mathcal{R}^{att}$ l'union des images $attesté_S \rho$ sur tout schéma S et toute requête ρ sur S . Les éléments de $\mathcal{R}^{att}$ sont appelés *requêtes attestées*. Intuitivement, une requête attestée impose aux variables de son extracteur et de son filtre d'être valuées dans le domaine attesté de la base (cf. Lemme 3.M). On note $\mathcal{E}^{att}$ le pouvoir d'expression de l'ensemble $\mathcal{R}^{att}$ des requêtes attestées. On montre successivement que l'ensemble des requêtes attestées est récursif (Lemme 3.L), qu'elles sont indépendantes (Corollaire 1 du Lemme 3.N) et, finalement, que leur pouvoir d'expression est égal à celui des requêtes indépendantes (Théorème 3.O).

LEMME 3.L. L'ensemble $\mathcal{R}^{att}$ des requêtes attestées est récursif.

Indication. Soient une requête de la forme $[<> ; F]$ et k le nombre d'opérateurs logiques de F . La requête est attestée s'il existe une formule F' sur $sch\ F$ telle que $F \in attesté_{sch\ F} F'$. Si une telle formule F' existe alors, d'après les règles (3.5),

- (1) le nombre de ses opérateurs logiques est $\leq k$,
- (2) $sch\ F' \subseteq sch\ F$ et
- (3) $var\ F' \subseteq var\ F$.

Les requêtes candidates F' peuvent donc être réduites à l'ensemble de celles vérifiant les propriétés (1), (2) et (3). Cet ensemble est fini et il est simple de concevoir un algorithme combinatoire qui énumère tous ses éléments F' et évalue pour chacun d'eux le test $F \in attesté_{sch\ F} F'$. La méthode s'étend aisément aux requêtes $[v ; F]$, $v \neq <>$.

LEMME 3.M. Pour tout schéma S , toute formule F sur S , toute base de données Δ sur S , tout domaine $\mathbf{d}$, $\mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$, et tout $G \in attesté_S F$,

$$(3.6) \quad \underline{\mathbf{d}}_\Delta G = (\mathbf{att}_\Delta F \downarrow libre\ F) \uplus \mathbf{d}^{\mathbf{V}} - libre\ F.$$

Démonstration. Par récurrence sur le nombre k d'opérateurs logiques de la formule F . On ne présente de la preuve que les cas où la non vacuité du domaine attesté est nécessaire. Soient un schéma S , $sch\ F \subseteq S$, une base de données Δ sur S , $\mathbf{a}$ son domaine attesté et $\mathbf{d} \supseteq \mathbf{a}$. Pour $k = 0$, on se limite au cas où la formule F est de la forme $R()$. D'après (3.5a), $attesté_S R()$ est le singleton $\{R()\}$. Il s'agit donc de montrer que $\underline{\mathbf{d}}R() = (\mathbf{att}\ R() \downarrow \emptyset) \uplus \mathbf{d}^{\mathbf{V}}$. Si $\Delta(R) = \emptyset$, alors $\underline{\mathbf{d}}R() = \mathbf{att}\ R() = \emptyset$ et l'égalité visée est vérifiée. Si $\Delta(R) \neq \emptyset$, alors $\Delta(R) = \{<>\}$, donc $\underline{\mathbf{d}}R() = \mathbf{d}^{\mathbf{V}}$ et $\mathbf{att}\ R() = \mathbf{a}^{\mathbf{V}}$ (cf. Exemple 3.2). Or $\mathbf{a} \neq \emptyset$ (Lemme 2.A), donc $\mathbf{d} \neq \emptyset$. Il en résulte que $\mathbf{a}^{\mathbf{V}} \neq \emptyset$ et $\mathbf{d}^{\mathbf{V}} \neq \emptyset$, d'où l'égalité visée.

On suppose que toute formule qui contient $k \geq 0$ opérateurs logiques vérifie la propriété (3.6). Soit F ayant $k + 1$ opérateurs logiques. On se limite au cas où F est de la forme (3.5e), i.e. $F = \neg F'$ et F n'a pas de variables libres, donc F' aussi. Soit $G \in attesté_S F$. La formule G est de la forme $\neg G'$, $G' \in attesté_S F'$. Posant $K = (\mathbf{att}\ \neg F' \downarrow \emptyset) \uplus \mathbf{d}^{\mathbf{V}}$, il s'agit

de montrer que $\underline{d} \neg G' = K$. Selon l'hypothèse de récurrence $\underline{d} G' = (\text{att } F' \downarrow \emptyset) \uplus \mathbf{d}^V$. Donc, $\underline{d} \neg G' = \mathbf{d}^V - \underline{d} G' = \mathbf{d}^V - ((\text{att } F' \downarrow \emptyset) \uplus \mathbf{d}^V)$. Puisque F' est sans variables libres, $\text{att } F'$ est soit vide soit égal à $\mathbf{a}^V$. Si $\text{att } F'$ est vide, alors $\underline{d} \neg G'$ et K sont égaux à $\mathbf{d}^V$. Si $\text{att } F' = \mathbf{a}^V$, alors $\underline{d} \neg G'$ est vide car $\mathbf{a}$ étant $\neq \emptyset$ (lemme 2.A), $\mathbf{a}^V \neq \emptyset$. De même K est vide. Dans tous les cas, $\underline{d} \neg G' = K$.

Le lemme suivant est la conséquence du précédent pour les requêtes.

LEMME 3.N. Pour tout schéma S , toute requête ρ sur S , toute base de donnée Δ sur S , tout domaine $\mathbf{d}$, $\mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$, et tout $\rho' \in \text{attesté}_S \rho$, $\text{att}_{\Delta} \rho = \underline{d}_{\Delta} \rho'$.

COROLLAIRE 1. Toute requête attestée est indépendante.

Démonstration. Soit $\rho \in \mathcal{R}^{\text{att}}$. Il existe une requête ρ' telle que $\rho \in \text{attesté}_{\text{sch}} \rho \rho'$. D'après le lemme 3.N, l'évaluation $\underline{d}_{\Delta} \rho$ est égale à $\text{att}_{\Delta} \rho'$. Elle ne dépend pas de $\mathbf{d}$.

COROLLAIRE 2. Pour tout schéma S , toute requête indépendante ρ sur S , toute base de donnée Δ sur S et tout $\rho' \in \text{attesté}_S \rho$, $\mathbf{e}_{\Delta} \rho = \mathbf{e}_{\Delta} \rho'$.

THÉORÈME 3.O. Le pouvoir d'expression $\mathcal{E}^{\text{att}}$ de l'ensemble des requêtes attestées est égal à celui $\mathcal{E}^{\text{ind}}$ de l'ensemble des requêtes indépendantes.

Démonstration. On a $\mathcal{R}^{\text{att}} \subseteq \mathcal{R}^{\text{ind}}$ (corollaire 1 du Lemme 3.N), donc $\mathcal{E}^{\text{att}} \subseteq \mathcal{E}^{\text{ind}}$. Inversement, soient $f \in \mathcal{E}^{\text{ind}}$, S le schéma sur lequel f porte, Δ une base de données sur S et $\mathbf{a}$ son domaine attesté. Il existe une requête indépendante ρ qui dénote f , i.e. $f(\Delta) = \mathbf{e}_{\Delta} \rho$. Soit $\rho' \in \text{attesté}_S \rho$. D'après le corollaire 2 du Lemme 3.N, $\mathbf{e}_{\Delta} \rho = \mathbf{e}_{\Delta} \rho'$. Donc $f(\Delta) = \mathbf{e}_{\Delta} \rho'$, autrement dit, ρ' dénote f , d'où $\mathcal{E}^{\text{ind}} \subseteq \mathcal{E}^{\text{att}}$.

CONCLUSION DE LA PREMIÈRE PARTIE

À partir de la logique du 1^{er} ordre, cette première partie a posé les fondements d'une machine BD relationnelle. La mémoire de cette machine est une structure qui contraint à stocker les informations sous forme de tables. Outre les opérations classiques d'affectation et de gestion de variables, le langage de la machine est une adaptation du langage du 1^{er} ordre au questionnement des bases de données. Au regard de l'objectif d'abstraction calculable développé dans l'introduction (cf. §0.5), une partie du chemin est couvert. Les requêtes indépendantes, *sous leur forme attestée*, constituent une manière déclarative de questionner une base de données et on dispose d'algorithmes pour (1) les reconnaître et (2) les évaluer sur une base de données.

On serait donc, d'ores et déjà, en mesure de programmer un interpréteur des requêtes attestées, autrement dit de réaliser une version logicielle de la machine. Toutefois, ni les requêtes attestées, ni les algorithmes présentés ne sont « crédibles ». En « forçant » leurs variables à être évaluées sur le domaine attesté, les premières sont une manière naturelle d'imposer l'indépendance mais l'accumulation des formules $\alpha_S(x, v_1, v_2, \dots, v_{n-1})$ qu'elles exigent rend leur usage trop compliqué et peu ergonomique. Quant aux seconds, leur exécution ne peut être qu'excessivement lente. Le mérite des deux est uniquement formel. Cependant, l'objectif d'abstraction calculable n'est pas, on le rappelle, que formel, il est aussi pratique. Il réclame des langages simples et une algorithmique efficace. La seconde partie couvre le reste du chemin. Le Chapitre 4

présente une autre approche syntaxique du pouvoir d'expression des requêtes indépendantes, son algorithmique canonique et les techniques de son optimisation. Enfin, le Chapitre 5 introduit des éléments pour la caractérisation du pouvoir d'expression des machines relationnelles.

Remerciements. En premier lieu, l'auteur remercie Philippe Courrège pour les échanges fréquents sur les thèmes de l'article. Il exprime aussi sa reconnaissance envers Vincent Berment, Marie-Anne Moreaux, Ana et Michet Simonet, Bruno Leclerc, Olivier Hudry David Douglas, Mathieu Roger et Xavier Delannoy pour leurs conseils, suggestions, discussions ou relectures.

In memoriam. Edgar Franck Codd fut l'initiateur du modèle relationnel. Il est décédé en avril 2003, pendant la rédaction de cet article. Le Mercury News a rapporté ce propos de Don Chamberlin, informaticien : « He had a vision about data that was considered radical at the time. » (Sa vision des données fut tenue pour révolutionnaire à l'époque).

BIBLIOGRAPHIE

- ABITEBOUL S., HULL R., VIANU V., *Foundations of databases*, Addison-Wesley Publishing Company, 1995.
- ADIBA M., COLLET C., *Objets et bases de données, le SGD O₂*. Hermès, 1993.
- AHO A. V., ULLMAN J. D., "Universality of data retrieval languages", *Actes de ACM Symposium on Principles of Programming Languages*, 1979, p. 110-117.
- ARSAC J., *Jeux et casse-tête à programmer*. Dunod, 1985.
- ATKINSON M., BUNEMAN P., "Types and persistence in database programming languages", *ACM Computing Surveys*, Vol. 19, n° 2, 1987, p. 105-190.
- BOUZEGHOUB M., GARDARIN G., VALDURIEZ P., *Les objets*, Eyrolles, 1998.
- BRIDGE J., *Beginning model theory, The completeness theorem and some consequences*, Oxford, Clarendon Press, 1977.
- CHEN P. P., "The entity-relationship model : toward a unified view of data", *ACM Transactions on Databases Systems*, Vol. 1, n° 1, 1976, p. 9-36.
- CODD E. F., "A relational model of data for large shared databanks", *CACM*, Vol. 13, n° 6, 1970, p. 377-87.
- CODD E. F., "Extending the database relational model to capture more meaning", *ACM Transactions on Databases Systems*, Vol. 4, n° 4, 1979, p. 397-434.
- CORIR., LASCAR D., *Logique mathématique*, Masson, 1994, [2^e tirage corrigé, vol 1].
- DATE C. J., *Introduction aux bases de données*, Vuibert, 1998, [6^e édition].
- DELOBEL C., ADIBA M., *Bases de données et systèmes relationnels*. Dunod, 1982.
- EBBINGHAUS H.-D., FLUM J., *Finite model theory*, Springer-Verlag, 1999, [seconde édition révisée].
- GARDARIN G., *Bases de données, les systèmes et leurs langages*, Eyrolles, 1983.
- ISRAEL G., *La mathématisation du réel*, Éditions de Seuil, 1996.
- G. KREISEL, KRIVINE J. L., *Eléments de logique mathématique*. Dunod, 1967.
- MERRET, T. H., *Relational information systems*, Reston Publishing Co, 1984.
- PRIEUR M., *Comparatif : 27 disques IDE 7200 tpm*, 2002.
www.hardware.fr/articles/448/page1.html

QUASS D., RAJARAMAN A., SAGIV Y., ULLMAN J. D., WIDOM J., “Querying Semistructured Heterogeneous Information”, Actes de la 4^e conférence internationale *Deductive and Object-Oriented Databases*, Décembre 1995.

ROBBIN J. W., *Mathematical logic : a first course*, New York, W. A. Benjamin, 1969.

ULLMAN J. D., *Principles of database and knowledgebase systems*, Computer Science Press, 1988, [9^e édition, 1999].

VAN GELBER A., TOPOR R. W., “Safety and translation of relational calculus queries”, *ACM Transactions on Databases Systems*, Vol. 16, n° 2, 1991, p. 235-278.

ANNEXE

RÔLES DE LA CONTRAINTE- Ω

Il a déjà été dit (cf. §2.1) que la contrainte- Ω , dont l’effet est d’imposer au domaine attesté des bases de données de ne pas être vide (cf. Lemme 2.A), était inhabituelle dans les formalisations du modèle relationnel. Elle est introduite dans cet article pour deux raisons : (1) parce qu’elle met en conformité la formalisation du modèle relationnel avec la notion de modèle en logique et (2) parce qu’elle assure l’indépendance des requêtes dites propositionnelles. On examine successivement ces deux raisons. À des fins de comparaison, on présente ensuite l’approche développée par Abiteboul *et al.* [1995], représentative des approches qui n’imposent pas la contrainte- Ω .

A.1 BASES DE DONNÉES ET LOGIQUE

Les bases de données et la logique n’ont pas les mêmes préoccupations bien qu’elles soient basées sur le même langage $\mathcal{F}$ des formules du 1^{er} ordre et la notion de réalisation. La logique utilise le langage $\mathcal{F}$ pour étudier ce qui est vrai (ou faux), avec la validité comme notion centrale. Les bases de données utilisent le langage $\mathcal{F}$ pour la programmation des machines BD, avec l’indépendance comme notion centrale. L’objectif de ce paragraphe est de montrer comment la contrainte- Ω met en conformité validité et indépendance (cf. Lemme A.B). On commence en rappelant brièvement les notions utiles de logique.

Étant donné l’ensemble $\mathbf{R}$ de symboles et l’application *arité* tels qu’ils ont déjà été définis (cf. §2.1), on appelle *structure* tout couple $(\mathbf{E}, \Gamma)$, dans lequel $\mathbf{E}$ est un ensemble *non vide*, appelé *base* de la structure, et Γ une application qui associe à tout élément R de $\mathbf{R}$ un sous-ensemble $\Gamma(R)$ de l’ensemble $\mathbf{E}^{[\text{arité}(R)]}$ des uplets sur $\mathbf{E}$ d’arité celle de R . L’évaluation d’une formule relativement à une structure $(\mathbf{E}, \Gamma)$ est le sous-ensemble, noté $\underline{\mathbf{E}}_{\Gamma} F$, de $\mathbf{E}^{\mathbf{V}}$ défini selon les règles (3.1) en y remplaçant Δ par Γ et en fixant $\mathbf{d} = \mathbf{E}$. Une structure $(\mathbf{E}, \Gamma)$ est un *modèle* d’une formule F si $\underline{\mathbf{E}}_{\Gamma} F = \mathbf{E}^{\mathbf{V}}$, on bien encore si $\underline{\mathbf{E}}_{\Gamma} \neg F = \emptyset$ [Kreisel, Krivine, 1967, p. 16]. Si une structure $(\mathbf{E}, \Gamma)$ est modèle d’une formule F (resp. $\neg F$), on dit aussi que F est *vraie* (resp. *fausse*) *dans* $(\mathbf{E}, \Gamma)$. Enfin, une formule F est dite *valide* si elle est vraie dans toute structure.

Par définition, la base $\mathbf{E}$ d’une structure $(\mathbf{E}, \Gamma)$ n’est pas vide, donc $\mathbf{E}^{\mathbf{V}} \neq \emptyset$. Il en résulte qu’il n’existe pas de structure dans laquelle une formule soit à la fois vraie et fausse, ce qui ne serait plus le cas si $\mathbf{E}$ était vide puisque $\mathbf{E}^{\mathbf{V}}$ et $\emptyset$ seraient alors égaux (cf. Chapitre 1 des préliminaires). Ainsi, les ensembles $\mathbf{E}^{\mathbf{V}}$ et $\emptyset$, dès lors qu’ils ne sont pas égaux, permettent-ils de distinguer « matériellement » les notions de vrai et de faux.

Autrement dit, pour distinguer le vrai et le faux, il faut au moins un élément dans le base $\mathbf{E}$ des structures¹⁷.

LEMME A.A. Étant donnés une mémoire relationnelle $M = (\mathbf{R}, \Omega, \text{arité}, \mathbf{D})$, une base de données (S, Δ) sur M et un domaine $\mathbf{d}$ tel que $\mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$, le couple $(\mathbf{d}, \Gamma)$ dans lequel Γ est l'application telle que $\Gamma(R) = \Delta(R)$ si $R \in S$ et $\emptyset$ si $R \in \mathbf{R} - S$, est une structure. De plus, pour toute formule F , $\underline{\mathbf{d}}_{\Gamma} F = \underline{\mathbf{d}}_{\Delta} F$.

Démonstration. Puisque $\mathbf{a}(\Delta) \neq \emptyset$ (Lemme 2.A).

LEMME A.B.

(A) Pour toute formule valide F , la requête $[\langle \rangle ; F]$ est indépendante et son évaluation est $\{ \langle \rangle \}$.

(B) Pour toute formule F dont la négation $\neg F$ est une formule valide, toute requête $[\nu ; F]$, y compris pour $\nu = \langle \rangle$, est indépendante et son évaluation est $\emptyset$.

Démonstration. Soient une base (S, Δ) et un domaine $\mathbf{d}$ tel que $\mathbf{a}(\Delta) \subseteq \mathbf{d} \subseteq \mathbf{D}$.

(A) Puisque F est valide, $\underline{\mathbf{d}}_{\Delta} \neg F = \mathbf{d}^{\mathbf{V}}$ (lemme A.A). Or, $\mathbf{a}(\Delta) \neq \emptyset$, donc $\mathbf{d}^{\mathbf{V}} \neq \emptyset$. Il en résulte que $\underline{\mathbf{d}}_{\Delta} [\langle \rangle ; F] = \{ \langle \rangle \}$, qui ne dépend pas de $\mathbf{d}$.

(B) (B) $\underline{\mathbf{d}}_{\Delta} [\nu ; F] = (\mathbf{d}^{\mathbf{V}} - \underline{\mathbf{d}}_{\Delta} \neg F)_{\circ} \nu$. Puisque $\neg F$ est valide, $\underline{\mathbf{d}}_{\Delta} \neg F = \mathbf{d}^{\mathbf{V}}$. Donc, $\underline{\mathbf{d}}_{\Delta} [\nu ; F] = \emptyset$, qui ne dépend pas de $\mathbf{d}$.

A.2 BASES DE DONNÉES ET REQUÊTES PROPOSITIONNELLES

Une *requête propositionnelle* est une requête de la forme $[\langle \rangle ; F_0]$ dans laquelle l'extracteur est vide tandis que le filtre F_0 n'utilise que des symboles d'arité 0 et n'utilise pas le quantificateur $\exists$, donc $\text{var } F_0 = \emptyset$. *Exemple* : $[\langle \rangle ; \neg R() \ \& \ R'()]$. Chaque symbole d'arité 0 est assimilable à une variable propositionnelle puisque ses deux réalisations possibles, à savoir $\emptyset$ et $\{ \langle \rangle \}$, sont respectivement assimilables aux valeurs *faux* et *vrai*. Les requêtes propositionnelles sont le correspondant « bases de données » du calcul propositionnel – pour le calcul propositionnel, voir [Cori, Lascar, 1994, Vol 1].

LEMME A.C. Les requêtes propositionnelles sont indépendantes.

Démonstration. Par récurrence sur le nombre k d'opérateurs logiques dans le filtre F_0 . On se limite à montrer le lemme dans le cas $k = 0$, car il fait apparaître l'exigence de non vacuité du domaine attesté. Pour $k = 0$, la requête est de la forme $[\langle \rangle ; R()]$ dans laquelle R est d'arité 0. Soient Δ une réalisation de R et $\mathbf{d} \supseteq \mathbf{a}(\Delta)$. On distingue deux cas, selon que $\Delta(R)$ est vide ou non. Si $\Delta(R) = \emptyset$, alors $\underline{\mathbf{d}}_{\Delta} R() = \emptyset$, donc $\underline{\mathbf{d}} [\langle \rangle ; R()] = \emptyset$. Si $\Delta(R) \neq \emptyset$, alors $\Delta(R) = \{ \langle \rangle \}$, donc $\underline{\mathbf{d}}_{\Delta} R() = \mathbf{d}^{\mathbf{V}}$ et $\underline{\mathbf{d}} [\langle \rangle ; R()] = \mathbf{d}^{\mathbf{V}}_{\circ} \langle \rangle$. Or $\mathbf{a}(\Delta) \neq \emptyset$ (Lemme 2.A), donc $\mathbf{d}^{\mathbf{V}} \neq \emptyset$. Il en résulte que $\underline{\mathbf{d}} [\langle \rangle ; R()] = \{ \langle \rangle \}$. Dans les deux cas, $\underline{\mathbf{d}} [\langle \rangle ; R()]$ ne dépend pas de $\mathbf{d}$.

D'après le Lemme A.C, le calcul propositionnel est « programmable » sur une machine BD (Théorème 3.H). Le contraire aurait semblé « étrange ». Pour quelles raisons en effet introduire les symboles relationnels d'arité 0 s'ils ne peuvent être

¹⁷ Cori, Lascar [1994, vol 1, p. 161] sont parmi les rares à insister sur le fait que « l'ensemble de base d'une structure ne peut être qu'un ensemble non vide ». Ils n'en développent cependant pas la raison.

utilisés seuls ? Mais cela exige que le domaine attesté des bases de données ne soit pas vide. Cette condition est *sine qua non*, la contrainte- Ω permet de la satisfaire.

A.2 APPROCHE D'ABITEBOUL *et al.* [1995]

On nommera AHV cette approche du modèle relationnel. Elle se distingue de l'approche de l'article en n'imposant pas la contrainte- Ω et en définissant différemment l'évaluation des formules et des requêtes. On s'intéresse ici aux conséquences de ces différences. Après l'exposé des principales définitions AHV, on précise le rapport entre les deux approches (Lemme A.D) puis on montre que, dans l'approche AHV,

- (1) le Lemme A.B n'est plus valide tandis que
- (2) le Lemme A.C le reste (Lemme A.E).

Dans ce paragraphe, le glossaire, les notations et, parfois, les définitions sont les mêmes que ceux de l'article, mais avec le sens AHV.

Une *mémoire relationnelle* M est un triplet $(\mathbf{R}, \text{arité}, \mathbf{D})$ dont les composants désignent ce qu'ils désignaient au paragraphe 2.1 et une *base de données* sur une mémoire M est un couple (S, Δ) dans lequel S est un sous-ensemble fini de $\mathbf{R}$ tel que, pour tout R dans S , $\Delta(R)$ est un élément de $\mathbf{2}^{\mathbf{D}}$ dont l'arité est celle de R . L'ensemble S est le *schéma* de la base. Le domaine attesté $\mathbf{a}(\Delta)$ d'une base (S, Δ) se définit de la même manière que dans l'article (voir §2.1). On souligne qu'il peut maintenant être vide. Les définitions qui suivent sont une réécriture dans le formalisme de cet article des définitions dans [Abiteboul *et al.*, 1995, p. 77]. Étant donné un sous-ensemble X de l'ensemble des variables $\mathbf{V}$ et un sous-ensemble $\mathbf{d}$ de $\mathbf{D}$, on appelle *valuation* de X sur $\mathbf{d}$, toute application de X dans $\mathbf{d}$. Étant donné un schéma S , on appelle *évaluation* l'unique application qui associe à toute formule F sur S , à toute base Δ sur S et à tout sous-ensemble $\mathbf{d}$ du domaine $\mathbf{D}$, l'ensemble, noté $\underline{\mathbf{d}}_{\Delta}F$, des valuations de *libre* F sur $\mathbf{d}$ tel que si F est de la forme :

$$(A.1a) \ R(), R \in S \text{ et d'arité } 0, \text{ alors } \underline{\mathbf{d}}_{\Delta}F = \{ \delta \in \mathbf{d}^{\emptyset} \mid \delta_{\circ} \in \Delta(R) \};$$

$$(A.1b) \ R(v_1, v_2, \dots, v_n), R \in S \text{ et d'arité } n > 0, \forall i, 1 \leq i \leq n, v_i \in \mathbf{V}, \text{ alors,}$$

$$\underline{\mathbf{d}}_{\Delta}F = \{ \delta \in \mathbf{d}^{var \langle v_1, v_2, \dots, v_n \rangle} \mid \delta_{\circ} \in \Delta(R) \};$$

$$(A.1c) \ \text{eg}(v_1, v_2), \text{ alors, } \underline{\mathbf{d}}_{\Delta}F = \{ \delta \in \mathbf{d}^{var \langle v_1, v_2 \rangle} \mid \delta(v_1) = \delta(v_2) \};$$

$$(A.1d) \ F', \text{ alors } \underline{\mathbf{d}}_{\Delta}F = \mathbf{d}^{libre F'} - \underline{\mathbf{d}}_{\Delta}F';$$

$$(A.1e) \ F' \ \& \ F'', \text{ alors}$$

$$\underline{\mathbf{d}}_{\Delta}F = (\underline{\mathbf{d}}_{\Delta}F' \uplus \mathbf{d}^{libre F'' - libre F'}) \cap (\underline{\mathbf{d}}_{\Delta}F'' \uplus \mathbf{d}^{libre F'' - libre F'});$$

$$(A.1f) \ \exists v F', \text{ alors } \underline{\mathbf{d}}_{\Delta}F = (\underline{\mathbf{d}}_{\Delta}F') \downarrow (libre F' - \{v\}).$$

La différence entre les définitions (A.1) et (3.1) tient au fait que, selon (A.1), $\underline{\mathbf{d}}_{\Delta}F$ est un sous-ensemble de $\mathbf{d}^{libre F}$ tandis que, selon (3.1), $\underline{\mathbf{d}}_{\Delta}F$ est un sous-ensemble de $\mathbf{d}^{\mathbf{V}}$. Le lemme suivant exprime les rapports entre l'évaluation selon (A.1) et celle selon (3.1), qu'on notera, pour cet énoncé, $\underline{\mathbf{d}}^+_{\Delta}F$.

LEMME A.D. Soient une base de données Δ , un domaine $\mathbf{d}$ et une formule F . On a :

$$(A.2a) \quad \underline{d}^+_{\Delta} F = \underline{d}_{\Delta} F \uplus \mathbf{d}^V\text{-libre } F ;$$

$$(A.2b) \quad \text{si } \mathbf{d} \neq \emptyset, \text{ alors } \underline{d}_{\Delta} F = \underline{d}^+_{\Delta} F \downarrow \text{libre } F.$$

On fait remarquer que la propriété (A.2b) réclame la non vacuité du domaine $\mathbf{d}$. En effet, lorsque $\mathbf{d} = \emptyset$, l'évaluation $\underline{d}^+_{\Delta} F$ est vide (cf. Lemme 3.C), mais pas nécessairement l'évaluation AHV $\underline{d}_{\Delta} F$ (cf. preuve du Lemme A.E). Les requêtes $[\nu; F]$ se définissent comme dans l'article. Si une requête vérifie $\text{var } \nu \subseteq \text{libre } F$, son évaluation AHV est définie par $\underline{d}_{\Delta}[\nu; F] = (\underline{d}_{\Delta} F) \circ \nu$. Pour les autres requêtes, elle n'est pas définie. L'évaluation AHV n'est donc pas universelle (cf. Commentaire 3.2). L'évaluation attestée est celle sur le domaine attesté de la base. On la note $\mathbf{att}_{\Delta} \rho$. Il est simple de montrer que le Lemme 3.G est encore vrai dans le cadre de l'approche AHV. Une requête ρ sur un schéma S est *indépendante* si elle vérifie la propriété (3.2). On fait remarquer que l'évaluation $\emptyset$ sur le domaine vide intervient dans la propriété d'indépendance AHV (voir Commentaire 3.4).

En ce qui concerne le lien validité-indépendance dans le cadre AHV, le lemme A.B n'est plus vrai. En effet, pour la formule valide $F \equiv (\forall v R(v)) \Rightarrow \exists v R(v)$, les requêtes $[\langle \rangle; F]$ et $[\langle \rangle; \neg F]$ ne sont pas indépendantes, tandis que pour la formule valide $G \equiv \exists v R(v) \vee \neg \exists v R(v)$, les requêtes $[\langle \rangle; G]$ et $[\langle \rangle; \neg G]$ sont indépendantes.

Démonstration pour la formule valide F . $\underline{d}_{\Delta}[\langle \rangle; F] = \underline{d}_{\Delta}(\exists v R(v) \vee \exists v \neg R(v)) \circ \langle \rangle$

$$= (\underline{d}_{\Delta}(\exists v R(v)) \cup \underline{d}_{\Delta}(\exists v \neg R(v))) \circ \langle \rangle$$

$$= ((\underline{d}_{\Delta} R(v) \downarrow \emptyset) \cup (\underline{d}_{\Delta} \neg R(v) \downarrow \emptyset)) \circ \langle \rangle \text{ (règle A.1.f)}$$

$$= (K \downarrow \emptyset) \cup (\mathbf{d}^{\{v\}} - K) \downarrow \emptyset \circ \langle \rangle \text{ (règle A.1.d et posant } K = \underline{d}_{\Delta} R(v) \text{)}$$

$$= (K \cup (\mathbf{d}^{\{v\}} - K)) \downarrow \emptyset \circ \langle \rangle = (K \cup \mathbf{d}^{\{v\}}) \downarrow \emptyset \circ \langle \rangle$$

$$= (\mathbf{d}^{\{v\}} \downarrow \emptyset) \circ \langle \rangle \text{ (puisque } K \subset \mathbf{d}^{\{v\}} \text{)}$$

$$= (\text{si } \mathbf{d} = \emptyset \text{ alors } \emptyset \text{ sinon } \{\langle \rangle\}). \text{ La requête } [\langle \rangle; F] \text{ n'est donc pas indépendante.}$$

$$\underline{d}_{\Delta}[\langle \rangle; \neg F] = (\mathbf{d}^{\emptyset} - \underline{d}_{\Delta} F) \circ \langle \rangle = (\mathbf{d}^{\emptyset} - \mathbf{d}^{\{v\}} \downarrow \emptyset) \circ \langle \rangle = (\{\langle \rangle\} - \mathbf{d}^{\{v\}} \downarrow \emptyset) \circ \langle \rangle = (\text{si } \mathbf{d} = \emptyset \text{ alors } \{\langle \rangle\} \text{ sinon } \emptyset). \text{ La requête } [\langle \rangle; \neg F] \text{ n'est donc pas indépendante}$$

LEMME A.E. Les requêtes propositionnelles sont indépendantes.

Démonstration. Par récurrence sur le nombre k d'opérateurs logiques. On expose uniquement le cas $k = 0$. La requête est de la forme $[\langle \rangle; R()]$ dans laquelle R est d'arité 0. Si $\Delta(R) = \emptyset$, alors $\forall \mathbf{d}, \mathbf{d} \subseteq \mathbf{D}, \underline{d}_{\Delta}[\langle \rangle; R()] = \emptyset$. Sinon, $\Delta(R) = \{\langle \rangle\}$, donc, $\forall \mathbf{d}, \mathbf{d} \subseteq \mathbf{D}, \underline{d}_{\Delta} R() = \{\delta \in \mathbf{d}^{\emptyset} \mid \delta \circ \langle \rangle \in \{\langle \rangle\}\} = \mathbf{d}^{\emptyset}$. Or, $\forall \mathbf{d}, \mathbf{d} \subseteq \mathbf{D}, \mathbf{d}^{\emptyset} = \{\langle \rangle\}$. Donc, $\underline{d}_{\Delta}[\langle \rangle; R()] = \{\langle \rangle\}$. D'où l'indépendance de $[\langle \rangle; R()]$.

Dans l'approche AHV, l'indépendance des requêtes propositionnelles n'est pas due à la contrainte- Ω – elle n'est pas imposée ! –, elle vient de ce que l'évaluation $\underline{d}_{\Delta} F$ d'une formule F est un sous-ensemble de $\mathbf{d}^{\text{libre } F}$ (voir preuve du lemme A.E) alors que l'évaluation de l'article est un sous-ensemble de $\mathbf{d}^V$.