MASTER EN SCIENCES INFORMATIQUES

Bases de données déductives : la méthode des événements internes

Delhez, Marie-Anne

Award date:
1991

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix,
Institut d'Informatique,
21 rue Grandgagnage,
5000 NAMUR

BASES DE DONNEES DEDUCTIVES :

La méthode des événements internes.

Marie-Anne DELHEZ

Promoteur : Roland LESUISSE.

Mémoire présenté en vue
de l'obtention du titre de
Licencié et Maître en
Informatique.

Année Académique 1990-1991

Résumé

Ce mémoire a pour sujet l'étude d'un nouveau type de bases de données : les bases de données déductives, également appelées bases de données logiques. Situées à mi-parcours entre les bases de données d'une part et la logique mathématique d'autre part, les bases de données déductives sont généralement peu connues et, bien que les nombreuses recherches entreprises depuis une quinzaine d'années fournissent des résultats engageants, les bases de données déductives risquent de ne jamais être commercialisées par manque de "publicité", ou tout simplement parce qu'elles seront "passées de mode".

Les bases de données déductives ont de nombreuses similitudes avec les bases de données relationnelles, elles offrent en plus la possibilité de gérer des "connaissances" exprimées sous forme de règles, appelées règles de déduction ou règles d'inférence. La présence de ces règles nécessite la mise au point de nouvelles méthodes de gestion ; la méthode des événements internes propose une solution particulière aux problèmes du contrôle de cohérence (lors de mises à jour diverses) et de gestion de mises à jour de faits dérivés.

Abstract

The aim of this thesis is to study a new kind of databases : deductive databases.

Placed between databases on one hand and mathematical logic on the other hand, deductive databases are generally badly known and, even if the researches undertaken during the last fifteen years provide very good results, deductive databases risk not to be commercialized by lack of advertising ; or simply because they'll be "out of fashion".

Deductive databases have a lot of similarities with relationnal databases ; futhermore, they offer the opportunity to handle knowledges expressed as rules, called deduction rules or inference rules. The presence of these rules makes it necessary to think up new management methods. The internal event method proposes a particular solution to the consistency problem (during updates) and to the view updating problem.

Je tiens à exprimer mes plus vifs remerciements à Monsieur le Professeur Lesuisse, sans son assistance et son appui, je n'aurais pu mener à bien ce travail.

Je remercie également tous les membres du département Langage et Système d'Information de l'UPC à Barcelone pour leur accueil chaleureux au sein de leur équipe. J'aimerais remercier plus particulièrement le Professeur A. Olive et le Professeur J. Sistac pour l'aide qu'ils m'ont apportée durant les 3 mois de stage, ainsi que le Professeur E. Teniente pour sa patience et sa disponibilité.

Ensuite, je tiens à remercier Monsieur le Professeur Hainaut pour ses précieux conseils.

J'exprime finalement ma reconnaissance à toutes les personnes qui, de près ou de loin, ont contribué à la réalisation de ce mémoire.

TABLE DES MATIERES

I. INTRODUCTION

I.1. Les bases de données déductives : angles d'approche possibles

Les bases de données déductives... Domaine intéressant, prometteur, mais également vaste et mal cerné.

Depuis une quinzaine d'années, un nombre considérable d'articles s'y rapportant sont publiés, alors que pratiquement aucun ouvrage n'y est consacré de manière globale, structurée. Peut-être est-ce là une première explication à la méconnaissance de ce domaine par le public, spécialisé ou non, enthousiasmé par tout ce qui est "orienté objet" depuis le début des années 90.

Comment, dès lors, présenter le sujet ?

Différents angles d'approche sont possibles.

Une première possibilité consiste à choisir un aspect spécifique des bases de données déductives et à le développer de manière très approfondie. Cette idée est fort attrayante dans le cadre d'un mémoire car elle permet un développement rigoureux et complet du sujet... Sujet fort centré et donc compréhensible et intéressant uniquement dans la mesure où tout son environnement est également maîtrisé et est source d'intérêt.

Une seconde optique, radicalement opposée, consiste à présenter la matière de façon plus globale. L'étude des différents points spécifiques est naturellement moins complète mais les caractéristiques principales des bases de données déductives sont présentées, et peut-être est-ce ce qui fait le plus défaut dans la littérature actuelle. C'est selon cette philosophie que ce mémoire a été rédigé.

Les développements théoriques formels seront intentionnellement limités au strict nécessaire pour 2 raisons :

a) La compréhension de ces développements formels requiert une maîtrise parfaite des notions mathématiques sous-jacentes, et elles sont nombreuses et complexes !

b) L'exposé complet que ces développements mathématiques exigeraient serait d'une longueur incompatible avec celle d'un travail pareil au nôtre.

De plus, l'objectif de ce mémoire n'est absolument pas de "spécialiser" un lecteur averti mais bien d'offrir à un lecteur ordinaire l'opportunité de découvrir ce que

sont les bases de données déductives. Seuls les principes de la méthode de résolution SLDNF (basée sur le principe de résolution de ROBINSON) [VanHent85] et du langage PROLOG [Clocksin] sont supposés connus.

I.2. Plan du mémoire

La matière étudiée est structurée de la manière suivante.

Le premier chapitre est consacré aux bases de données hiérarchiques et relationnelles. Afin de comprendre mieux les nouveautés relatives aux bases de données déductives, il est intéressant de connaître les caractéristiques de leurs prédécesseurs. Les bases de données hiérarchiques, bases de données de première génération, ne sont que très brièvement abordées, l'objectif est de montrer l'évolution gigantesque survenue en quelques années. Les bases de données relationnelles (bases de données de deuxième génération) étant plus proches des bases de données déductives, l'on s'y consacrera davantage afin de définir les concepts qui les caractérisent et d'étudier tous les aspects que l'on retrouvera, sous une forme plus ou moins semblable, dans les bases de données déductives.

Une fois ces rappels terminés, on abordera l'étude des bases de données déductives dans le chapitre 3. La compréhension de ce chapitre est fondamentale car il pose les bases théoriques nécessaires à la compréhension de tout raisonnement sur les bases de données déductives. Suite à cela, une définition des bases de données déductives est donnée, illustrée d'une présentation plus concrète et de divers exemples. Les aspects spécifiques des bases de données déductives seront étudiés, lorsque c'est possible, l'on procédera à des comparaisons avec les bases de données relationnelles.

Finalement, une méthode de gestion des bases de données déductives est présentée dans le chapitre 4. Cette méthode porte le nom de "méthode des événements internes", elle offre une solution aux problèmes spécifiques des bases de données déductives, problèmes qui auront été mis en lumière tout au long du travail.

Avant de conclure en résumant brièvement l'état des travaux sur les bases de données déductives et en donnant une idée de leurs perspectives, un cinquième chapitre sera consacré à la présentation de l'implémentation de la méthode des événements internes. Ce chapitre est intéressant dans la mesure où il donne une idée de l'approche pratique de la mise au point d'une nouvelle théorie.

II. LES MODELES HIERARCHIQUE ET RELATIONNEL

II.1. Introduction

Avant d'étudier le nouveau type de bases de données que sont les bases de données déductives, il est intéressant de rappeler les caractéristiques de leurs prédécesseurs.

Nous commencerons par un extrait de l'ouvrage de Gardarin et Valduriez "Les SGBD avancés" [Gard-Vald90] qui nous permettra de retracer un peu l'histoire des SGBD de première et deuxième générations.

Nous poursuivrons avec un bref aperçu des bases de données hiérarchiques. En étudier les détails ne nous serait d'aucune utilité dans le cadre d'un travail sur les bases de données déductives car les divergences de philosophie entre ces deux types de bases de données sont énormes.

Par contre, nous nous consacrerons plus longuement à l'étude des bases de données relationnelles qui, elles, nous seront fort utiles tout au long de ce travail. Ainsi, nous verrons quelles sont les caractéristiques, les facilités d'utilisation,... des bases de données relationnelles.

Un lecteur possédant de bonnes notions de bases de données peut sans problème passer directement au chapitre suivant.

II.2. Historique

L'extrait ci-dessous explique brièvement quelles sont les fonctionnalités fondamentales de tout système de gestion de base de données et relate les grandes lignes de leur évolution vers la situation actuelle ; il est intégralement repris de l'ouvrage de GARDARIN et VALDURIEZ "Les SGBD avancés" [Gard-Vald90].

"Un système de gestion de base de données (SGBD) est aujourd'hui un logiciel de base essentiel dans un système informatique de gestion. Intuitivement, il permet à des utilisateurs concurrents de manipuler (insérer, modifier et rechercher) efficacement des données contenues dans une base de données. Depuis l'apparition des premiers SGBD vers 1962, d'importants résultats théoriques et pratiques ont ponctué l'histoire de la

recherche en bases de données. La mise en oeuvre de ces résultats a permis de faciliter l'administration et la manipulation d'une base de données, et d'accroître ainsi la productivité des utilisateurs des bases de données (administrateurs, programmeurs d'applications et utilisateurs finals).

L'histoire des SGBD peut être résumée en distinguant trois générations. La première génération de SGBD s'appuie sur les modèles de données hiérarchiques et réseaux. Elle vise à étendre un système de fichiers (l'ancêtre des SGBD) par des possibilités de liaisons inter-fichiers matérialisées par des pointeurs. Les bases de données sont représentées au niveau des types d'articles par une hiérarchie ou un graphe. Un SGBD de première génération fournit une faible indépendance physique, compliquant ainsi l'administration et la manipulation des données. En particulier, son langage de manipulation de données navigationnel impose au programmeur de spécifier les chemins d'accès aux données en naviguant dans le graphe de la base.

La deuxième génération de SGBD est née vers 1970 avec l'apparition du modèle de données relationnel. Une dizaine d'années d'efforts de recherche et de développement furent nécessaires pour aboutir à la commercialisation des premiers SGBD relationnels. Depuis, la technologie des bases de données relationnelles a fait l'objet de progrès remarquables en termes de facilité d'usage et de performances. Aujourd'hui, tout SGBD relationnel offre un ensemble intégré d'outils basés sur un Langage de 4ème Génération (L4G) afin d'accroître la productivité des utilisateurs. Aussi, certains systèmes supportent efficacement les applications transactionnelles, très exigeantes en performances. En conséquence, le marché actuel des bases de données est couvert en grande majorité par les SGBD relationnels. Les premières et secondes générations de SGBD ont été conçues pour les applications de gestion classique, comme, par exemple, la gestion de stocks ou la comptabilité d'une entreprise..."

II.3. Le modèle hiérarchique

II.3.1. Caractéristiques d'une base de données hiérarchique

La structure de données de base du modèle hiérarchique est la structure d'arbre.

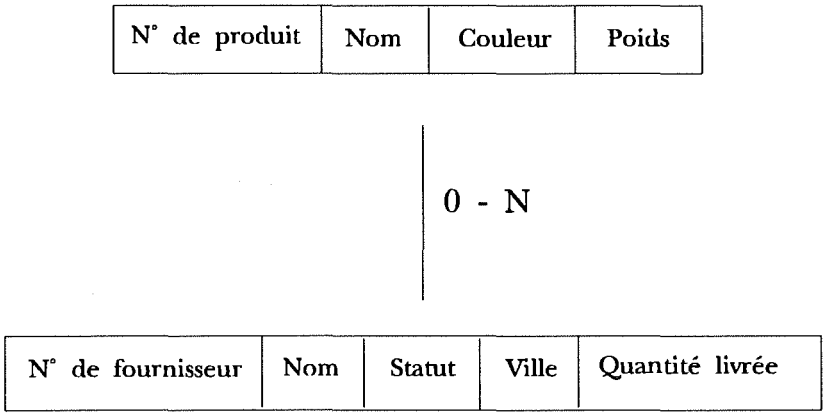
Un arbre est composé de 2 types de "records" : un record racine et des records dépendants. La racine est le record situé au sommet de l'arbre, elle peut avoir un nombre indéterminé de dépendants.

Comme son nom l'indique, un dépendant ne peut exister indépendamment de sa racine ; il peut à son tour avoir des dépendants de niveau inférieur, ce qui génère des arbres à différents niveaux de profondeur. Les termes *parent* et *enfant* sont également

utilisés pour représenter la hiérarchie.

Pour illustrer ces concepts, considérons le schéma d'une base de données fort simple reprenant une gamme de produits auxquels sont associés les fournisseurs qui les livrent. Un produit est caractérisé par son numéro, son nom, sa couleur et son poids. Dans les informations concernant un fournisseur figurent le numéro du fournisseur, son nom, son statut et sa ville. A ces informations, il faut encore ajouter le nombre d'unités du produit livrées.

Exemple :



A chaque produit est associé un arbre particulier qui représente une occurrence hiérarchique. Un arbre est donc associé à un produit et les fournisseurs de ce produit sont repris sous forme d'occurrences de records subordonnés. L'ensemble des occurrences de fournisseurs pour un produit donné peut être vide.

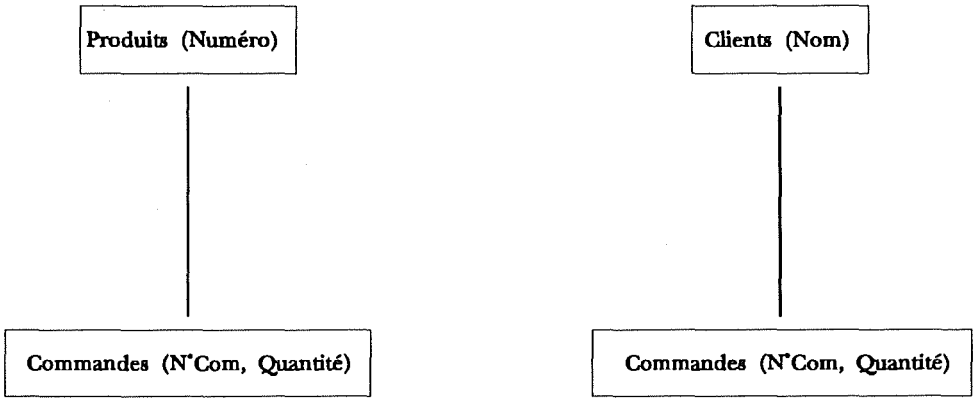
Tous les arbres sont généralement regroupés dans un seul fichier, fichier fort complexe à gérer puisqu'il peut contenir des records de plusieurs types différents ainsi que les liens entre ces différents records.

Dans le modèle hiérarchique, il est fondamental de ne considérer une occurrence d'un record que dans son contexte ; en effet, aucun record dépendant ne peut ne fut-ce qu'exister sans son supérieur. Pour exprimer une requête ou effectuer n'importe quelle opération sur la base de données, il faut par conséquent aussi spécifier le contexte, d'autant plus que dans une hiérarchie, les liens sont unidirectionnels : de la racine vers ses dépendants. Les requêtes seront par conséquent différentes selon qu'elles portent sur des records racines ou dépendants et l'écriture des requêtes sous forme de procédures est également fort complexe. Cette complexité des procédures provient du modèle lui-même et entraîne des complications dont l'utilisateur se passerait volontiers. En particulier,

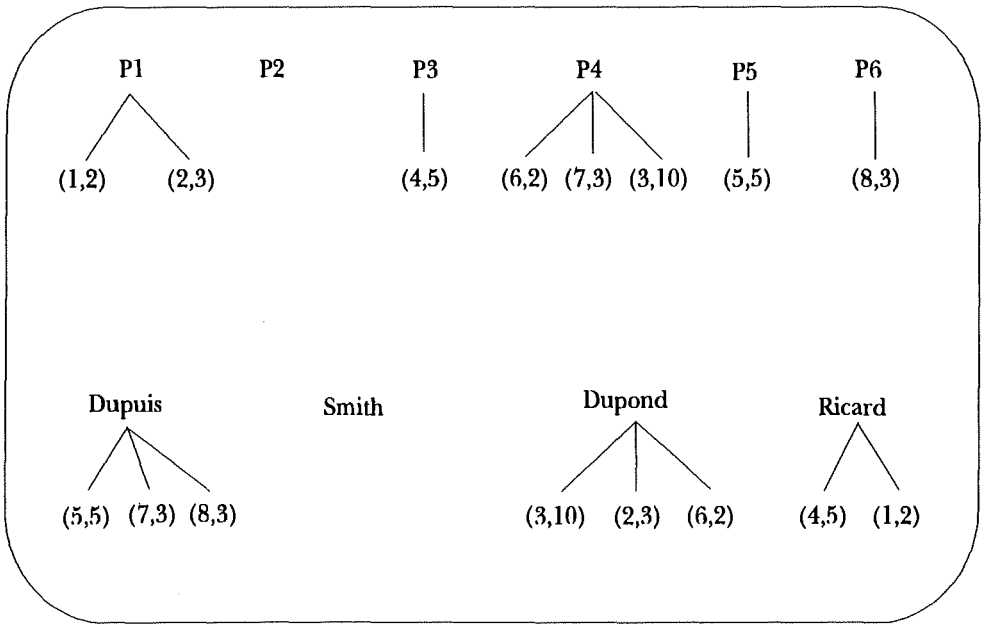
l'utilisateur est obligé de résoudre des problèmes introduits par le modèle lui-même plutôt que par la requête qu'il aimerait poser. Il est clair que la situation ne fera qu'empirer lorsque des types plus complexes de records seront introduits dans la structure et que la hiérarchie deviendra plus complexe.

Exemple :

Supposons le petit schéma de base de données suivant (inspiré de [ULL82]).

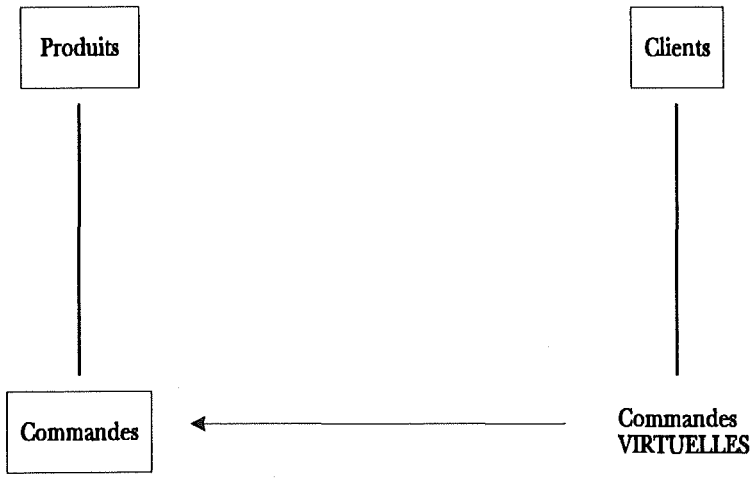


Et supposons que cette base de données contienne :

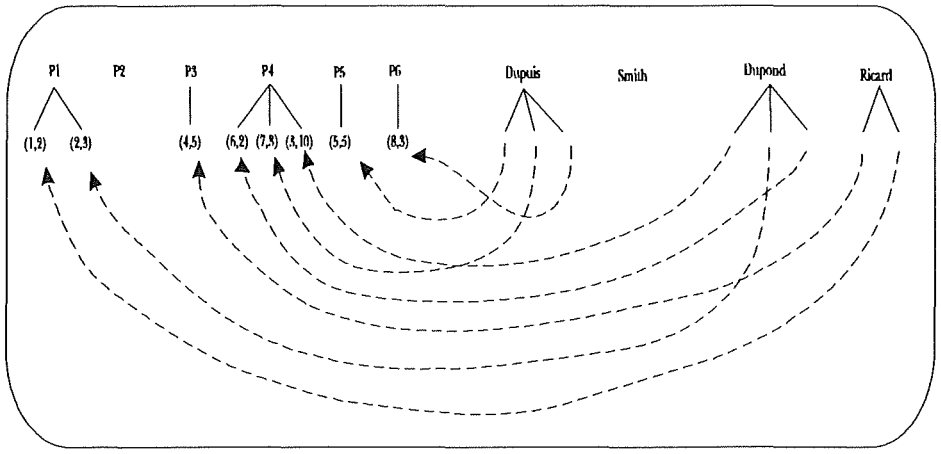


La redondance est évidente, les records dépendants des produits et des clients sont identiques, ils reprennent chaque fois le numéro de commande et la quantité associée. Cette situation n'est bien entendu pas souhaitable puisqu'une forte redondance entraîne inévitablement des difficultés supplémentaires pour le maintien de la cohérence de la base de données.

Une solution offerte par le modèle hiérarchique pour contourner ce problème de redondance est le recours aux types de records virtuels. D'un point de vue implémentation, un type de record virtuel T doit être interprété comme un pointeur vers le record physique de type T qu'il représente. Dans notre exemple, cela donne...



Lorsque l'on applique ce principe à l'exemple, l'illustration se complexifie considérablement et, bien que la base de données ne contienne que peu d'éléments, la lisibilité se détériore fortement.



Comme dans une hiérarchie, les liens sont unidirectionnels et vont de la racine vers ses descendants. Dans l'exemple, il est aisé de savoir quelles sont les quantités du produit P1 vendues puisque de P1, on accède directement à ses commandes ; par contre, si l'on désire savoir sur quoi portait la commande numéro 6, il n'y a pas de mécanisme direct et il revient au programmeur d'écrire toute la procédure donnant le résultat désiré. Cela signifie que les programmes sont plus compliqués que nécessaire ; l'écriture des programmes et leur maintenance demandent plus de temps de programmation qu'elles ne le devraient.

Si l'on se tourne maintenant vers les opérations de base que sont l'insertion, la suppression et la mise à jour de records, on voit que le modèle hiérarchique possède d'autres propriétés indésirables, directement dues au fait que le monde modélisé dans la base de données n'est pas une véritable hiérarchie, le lien entre la racine et ses descendants est de type "many-to-many" : un fournisseur livre plusieurs produits et un produit est livré par plusieurs fournisseurs. Quelques caractéristiques indésirables de la base de données fournisseurs-commandes servant d'exemple sont notamment qu'il n'est pas possible d'enregistrer des données concernant un nouveau fournisseur si ce dernier ne livre pas effectivement un produit ; apparaît ainsi un problème d'insertion de records dépendants.

De plus, puisque les données concernant un fournisseur se trouvent dans un type de record associé à une ligne de commande, lorsque l'on efface une ligne, l'occurrence correspondante du fournisseur est également supprimée. Si l'on supprime la seule ligne d'un fournisseur donné, on perd par la même occasion toutes les informations concernant ce fournisseur. Il y a par conséquent un problème lors de la suppression de records dépendants.

Un problème similaire se pose si l'on désire effacer un produit car supprimer une occurrence d'un record racine entraîne automatiquement la suppression de toutes les occurrences de ses records dépendants, en accord avec la philosophie hiérarchique.

Finalement, si l'on a besoin de changer la description d'un fournisseur, on doit soit analyser la base de données entière pour trouver toutes les occurrences de ce fournisseur, soit accepter une certaine incohérence de la base de données en admettant qu'un même fournisseur ait des valeurs d'attributs différentes selon les occurrences ; d'où un sérieux problème lors des mises à jour. Ce troisième problème est dû à la forte redondance des données enregistrées dans les bases de données hiérarchiques ; cette situation d'autant plus gênante que le contrôle d'intégrité est assez faible...

II.3.2. Contrôle d'intégrité dans une base de données hiérarchique

Le contrôle d'intégrité varie bien entendu d'un SGBD à l'autre ; dans le cas particulier du SGBD hiérarchique IMS d'IBM, on constate qu'il existe deux mécanismes qui peuvent être considérés comme mécanismes de gestion des contraintes d'intégrité [Ull82]. Le premier concerne l'unicité des valeurs de champs. Si lors de la déclaration d'un champ, la caractéristique "multi-valué" n'est pas spécifiée, IMS garantit qu'il n'y aura jamais deux occurrences identiques de ce champ ; s'il s'agit d'un record dépendant, la portée de la vérification se limite au contexte du parent. En d'autres termes, IMS rejette toute opération d'insertion qui tente d'introduire une duplication.

La seconde caractéristique est inhérente à la structure hiérarchique d'une base de données IMS, qui assure la vérification de certaines contraintes d'intégrité dans la structure de la base de données elle-même. Supposons, par exemple, que la base de données produit-fournisseur soit représentée par une hiérarchie dans laquelle les produits sont supérieurs aux fournisseurs (cas de l'exemple donné ci-avant) ; alors, de par la structure même de la base de données, il est impossible d'avoir un fournisseur qui livre un produit inexistant puisqu'un fournisseur est obligatoirement un descendant d'un produit. Par contre, rien ne garantit que le fournisseur d'un produit existe effectivement.

II.3.3. Les SGBD hiérarchiques : conclusion

Ces quelques pages consacrées aux bases de données hiérarchiques avaient pour objectif de donner l'idée sous-jacente à ce type de SGBD ; elles sont naturellement loin d'être complètes. Quoiqu'il en soit, une évidence se dégage de cette brève étude : l'utilisation d'une base de données hiérarchique requiert de bonnes connaissances de la structure et de l'organisation de la base de données de la part de l'utilisateur. De plus, la responsabilité de ce dernier est grande puisqu'il ne bénéficie que d'outils offerts par le SGBD relativement rudimentaires pour assurer la cohérence et l'exploitation de la base de données.

Les SGBD hiérarchiques font partie des SGBD dits "de première génération" ; fort importants à la fin des années 70, ils n'occupent actuellement plus qu'une faible part du marché des systèmes de gestion de base de données ; marché dominé par des SGBD de conception totalement différente les rendant nettement plus performants et agréables à utiliser...

II.4. Le modèle relationnel

II.4.1. Caractéristiques d'une base de données relationnelle

A nouveau, pour présenter les caractéristiques des bases de données relationnelles, nous reprendrons un extrait de l'ouvrage de Gardarin et Valduriez [Gard-Vald90] ; il donne une excellente synthèse du modèle relationnel. Certains points importants seront explicités par la suite.

"Le modèle relationnel, inventé par CODD [Codd70], peut être introduit informellement par les trois propriétés suivantes :

1 - Les structures de données sont simples et se construisent à partir de la théorie des ensembles. Ce sont des tables à deux dimensions dont chaque élément appartient à un ensemble de valeurs appelé domaine. Puisqu'elle définit une relation entre des domaines, une telle table est appelée relation. Une colonne d'une relation, appelée attribut, est définie sur un même domaine. Une ligne d'une relation, appelée n-uplet ou tuple, relie logiquement les éléments d'information fournis par les valeurs d'attributs. Toute l'information de la base de données doit être représentée explicitement par des valeurs dans des tables. En particulier, il n'y a pas de pointeurs inter-relations visibles par l'utilisateur.

2 - Un ensemble d'opérateurs appliqués à des relations, constituant l'algèbre relationnelle, permet la définition, la recherche et la mise à jour des données. Chaque opérateur prend une ou deux relations en argument et produit une relation. L'algèbre relationnelle comprend les opérateurs classiques de la théorie des ensembles (produit Cartésien, union, intersection, différence) et des opérateurs permettant de composer des sous-ensembles d'une ou deux relations (projection, sélection, jointure et division).

3 - Un ensemble de contraintes d'intégrité sémantiques définit les états cohérents de la base de données. Il existe au minimum deux contraintes d'intégrité structurelles, c'est-à-dire inhérentes au modèle relationnel. La première est l'unicité de clé, qui spécifie qu'un ou plusieurs attributs constituent la clé unique d'une relation. Par exemple, le numéro de sécurité sociale est la clé unique de la relation PERSONNE. La seconde est la contrainte de référence, qui permet de garantir que la valeur d'une donnée dans une relation est présente dans une autre relation. Par exemple, un produit commandé, référencé dans la relation COMMANDE, doit exister aussi dans la relation PRODUIT.

L'avantage majeur du modèle relationnel est sa faculté à assurer l'indépendance complète entre les descriptions de données logiques (en termes relationnels) et physiques (en termes de fichiers et liaisons inter-fichiers). Cette indépendance physique a permis le développement de langages de définition et de manipulation de données de haut niveau, appelés langages de requêtes, c'est-à-dire basés sur la logique des prédicats, et libèrent le programmeur de la spécification des chemins d'accès aux données. En conséquence, l'optimisation des requêtes de manipulation de données peut être entièrement automatisée. L'existence du langage de requêtes standard SQL [Ansi86] contribue d'ailleurs fortement à la promotion du modèle relationnel. SQL fournit une interface uniforme aux administrateurs, programmeurs d'applications, et utilisateurs finals, pour la définition, le contrôle et la manipulation de données...

La plupart des SGBD relationnels commercialisés supportent le langage SQL. De plus, ils offrent généralement un ensemble intégré d'outils L4G (générateur de code etc.) qui facilitent le développement des applications "base de données" et améliorent ainsi la productivité des utilisateurs. La réalisation de ces outils a été facilitée par la simplicité et la puissance du modèle relationnel."

Cette présentation, quoique concise, est très claire pour un lecteur avisé, elle fait cependant intervenir de nombreux éléments qui peuvent poser problème. Le point suivant est par conséquent consacré à l'étude plus détaillée des éléments particulièrement pertinents pour notre sujet.

II.4.2. Etude détaillée des éléments déterminants

Nous savons qu'une base de données a pour objectif de représenter et de maintenir une connaissance sur une certaine réalité appelée le monde modélisé. Deux types d'informations peuvent y être contenues : les informations élémentaires et les informations générales. Dans le cadre des base de données relationnelles, seules les informations élémentaires sont représentées ; les lois générales sont utilisées comme contraintes d'intégrité. Les informations élémentaires constituent alors un état de la base de données et les contraintes d'intégrité délimitent les états valides.

Pour comprendre comment le système procède pour évaluer une requête, il est nécessaire de connaître les hypothèses qui gouvernent leur évaluation ainsi que celle des contraintes d'intégrité. Ces hypothèses ont également été établies par CODD lorsqu'il a défini le modèle relationnel [Codd70] ; elles sont au nombre de 3.

1) L'hypothèse du monde fermé (Closed World Assumption (CWA)).

Cette hypothèse suppose une connaissance totale du monde modélisé ; ce qui permet d'affirmer que seuls les faits positifs doivent être effectivement représentés. Un fait négatif est implicitement présent si son homologue positif ne l'est pas. En résumé, sous l'hypothèse CWA, pour toute relation R(x1,...,xn) et pour tout tuple <a1,...,an>, soit R(a1,...,an) soit ¬R(a1,...,an) est vrai.

L'hypothèse du CWA a pour conséquence de fusionner les faits faux et les faits inconnus lorsque la connaissance du monde n'est pas totale.

2) L'hypothèse de fermeture du domaine (Domain Closure Assumption (DCA)).

Cette hypothèse établit qu'il n'y a pas d'autres individus que ceux présents dans la base de données. Elle est particulièrement importante lorsque l'on répond à des requêtes universellement quantifiées car elle donne la véritable sémantique du ∀. Ce que l'on désire, ce n'est pas d'avoir une réponse qui soit vraie dans tous les états possibles de la base de données mais bien dans son état actuel. La sémantique du ∀ est donc "Pour tout élément que l'on connait". Cette hypothèse de fermeture du domaine est également importante pour établir la réponse à des questions du genre "Quels sont les individus x1, x2,..., xn tels que <x1,x2,...,xn> n'appartient pas à la relation R ?". L'hypothèse de fermeture du domaine délimite l'ensemble des individus à prendre en considération, ce qui donne un sens à ce genre de questions.

3) L'hypothèse d'unicité des noms (Unique Name Assumption (UNA)).

Cette dernière hypothèse établit tout simplement que les éléments à noms distincts sont différents.

A présent que les hypothèses de CODD sont établies, revenons au point 2 de l'extrait dans lequel Gardarin et Valduriez parlent de l'algèbre relationnelle.

Dans le point 2, Gardarin et Valduriez citent l'existence d'opérateurs qui, appliqués à des relations, fournissent une relation résultante dont les éléments respectent certaines propriétés particulières. Pour permettre au lecteur de bien réaliser la puissance du modèle relationnel par rapport au modèle hiérarchique, ces opérateurs sont détaillés ci-dessous ; les définitions sont reprises de [Hainaut88].

- Produit cartésien : "Il s'agit d'un ensemble de n-uplets constitué de tous les arrangements différents obtenus en prenant une valeur dans le premier domaine, puis une dans le deuxième, et ainsi de suite".

- Union : "Un domaine (une relation) est l'union de deux ou plusieurs domaines (relations) compatibles si chacun de ses éléments appartient à au moins l'un de ces domaines (relations)".

- Intersection : "Un domaine (une relation) est l'intersection de deux ou plusieurs domaines (relations) compatibles si chacun de ses éléments appartient simultanément à tous ces domaines (relations)".

- Différence : "Un domaine (une relation) est la différence d'un premier domaine (relation) et d'un second domaine (relation) tous deux compatibles si ses éléments sont ceux du premier qui n'appartiennent pas au second".

- Projection : "La projection d'une relation sur certains de ses attributs est une relation obtenue en ne conservant de la relation initiale que les valeurs de ces attributs. L'extension de la relation étant un ensemble, certaines lignes sont amenées à disparaître".

- Sélection : "L'opération de sélection consiste à ne retenir d'une relation que les n-uplets dont les valeurs vérifient une condition du type : la valeur de tel attribut appartient à tel ensemble".

- Jointure : "La jointure est une opération qui permet de construire une relation en accouplant les n-uplets de deux relations arguments qui ont même valeur pour un ou plusieurs attributs compatibles".

- Division : "Etant donné les relations R(A,B,C) et S(C,D), la division de R par S sur C consiste à retenir les n-uplets (a,b) qui, dans R, sont associés à tous les C présents dans S. Les autres domaines de S (ici D) n'interviennent pas. A, B, C et D représentent un ou plusieurs attributs".

Ces 8 opérateurs ne sont bien entendu pas exclusifs, il est possible d'en définir d'autres, notamment à partir de ceux dont on dispose déjà (voir [Hainaut88]). D'ores et déjà, on constate cependant que les possibilités offertes par un SGBD relationnel ainsi que sa facilité d'utilisation sont nettement supérieures à celle d'un SGBD hiérarchique. Ceci est dû à ce que GARDARIN et VALDURIEZ appellent "l'avantage majeur des bases de données relationnelles" ; à savoir la nette séparation entre *base de données physique* et de *base de données conceptuelle*, dont voici les définitions.

On entend par *base de données physique* l'ensemble des fichiers, pointeurs et autres artifices qui permettent le rangement des données sur support physique. Par contre, lorsque l'on parle de *base de données conceptuelle*, on se situe à un niveau

d'abstraction totalement différent. Dans [Hainaut88], une base de données conceptuelle est définie comme étant constituée :

- d'un schéma : texte ou formalisme quelconque décrivant les propriétés générales stables des domaines (nom, type des valeurs,...) et des relations (nom, liste des attributs,...) ainsi que les contraintes ;
- des ensembles de valeurs des domaines et des ensembles des lignes des relations ; ces ensembles, contrairement au schéma, peuvent évoluer rapidement au cours du temps.

De plus, à chaque domaine et à chaque relation est associée une expression sémantique en langue naturelle qui en donne l'interprétation, ou signification par rapport au réel perçu.

Cette séparation *physique - conceptuel* facilite hautement la tâche de l'utilisateur qui, lorsqu'il désire créer ou modifier la structure d'une base de données, n'a pas à se soucier de préoccupations d'ordre plus terre à terre qui ne l'intéressent en général aucunement.

Avant de conclure notre rappel consacré aux bases de données relationnelles, il est nécessaire de développer quelque peu l'étude des contraintes d'intégrité dans ce type de bases de données (voir [Date86]).

II.4.3. Contrôle d'intégrité dans une base de données relationnelle

Assurer l'intégrité d'une base de données revient à assurer, dans la mesure du possible, que les données de la base sont correctes à tout moment. Un SGBD relationnel ne peut bien entendu pas vérifier l'exactitude des valeurs entrées individuellement mais il peut vérifier leur plausibilité (par exemple en vérifiant leur appartenance à un intervalle donné).

En général, à chaque relation d'une base de données relationnelle sera associé un ensemble de contraintes d'intégrité ; elles se trouvent dans le schéma conceptuel. Théoriquement, les contraintes peuvent être d'une complexité arbitraire mais pratiquement, les SGBD ne supportent que des contraintes assez simples. Une contrainte d'intégrité typique serait, par exemple, de fixer la limite supérieure de la valeur d'un attribut particulier dans une relation (par exemple : l'âge d'une personne doit être inférieur à 150 ans).

Lors de l'insertion d'une nouvelle contrainte dans la base de données, le SGBD doit être en mesure de vérifier si elle est satisfaite dans la base de données actuelle. Si

ce n'est pas le cas, l'insertion de la nouvelle contrainte sera normalement refusée [1].

Parmi les types de contraintes d'intégrité généralement vérifiées figure la contrainte relative à l'*identifiant* (appelé *clé unique* dans l'extrait de GARDARIN et VALDURIEZ). Par définition, l'identifiant de toute relation doit être unique : deux tuples différents d'une relation ne peuvent pas avoir deux valeurs d'identifiant égales. Ainsi, le SGBD doit rejeter toute tentative de générer un tuple dont la valeur d'identifiant est nulle ou partiellement nulle, ou encore qui est une duplication d'une clé existante.

De plus, si une relation contient d'autres clés *candidates* au statut d'identifiant et si le qualificatif *unique* est précisé dans le schéma conceptuel, le SGBD doit également veiller à ce qu'aucune duplication ne soit introduite.

Bon nombre de contraintes d'intégrité se rapportent également aux valeurs des attributs d'une relation. On citera notamment les contraintes d'intégrité qui font intervenir des *dépendances fonctionnelles*. Par définition [Hainaut88], une dépendance fonctionnelle d'un attribut A vers un attribut B dans une relation R exprime que chaque fois qu'une valeur de A apparaît dans R, il lui est associé la même valeur de B. Une dépendance fonctionnelle peut aussi exister d'un groupe d'attributs vers un autre. Pour assurer le respect de ces dépendances, le SGBD doit surveiller les opérations d'insertion et de mise à jour d'attributs.

Les dépendances fonctionnelles ne sont pas les seules contraintes imposables aux attributs d'une relation donnée. La valeur d'un attribut d'une relation peut devoir être inférieur à celle d'un autre, se trouver endéans certaines limites, appartenir à un ensemble de valeurs déterminées, ne pas appartenir à un ensemble de valeurs déterminées, la valeur d'un attribut peut aussi devoir se conformer à un format particulier, ... D'autres contraintes sont possibles, par exemple l'ensemble des valeurs d'une colonne particulière d'une relation doit être le même que l'ensemble, ou un sous-ensemble, des valeurs apparaissant dans une autre colonne, dans la même relation ou dans une relation différente, mais il ne faut pas perdre de vue que plus les contraintes sont nombreuses, plus la tâche du SGBD sera complexe et le temps d'exécution des opérations de base (insertion, suppression,...) sera long.

Tous les exemples donnés jusqu'à présent sont des exemples de contraintes *statiques* ; c'est-à-dire qu'elles spécifient des conditions qui doivent être vérifiées dans tout état de la base de données. Un autre type important de contraintes d'intégrité implique les *transitions* d'un état de base de données à un autre (par exemple, l'âge d'une personne ne peut qu'augmenter). Pour spécifier de telles contraintes, il faut disposer d'un moyen de référencer les anciennes valeurs et les nouvelles ; ce qui requiert

---

[1] L'action prise dépend du SGBD particulier, d'où le terme "normalement".

un mécanisme de gestion des contraintes plus complexe que pour les contraintes d'intégrité statiques, mécanisme dont l'existence (ou la non existence) dépend d'un SGBD à l'autre. Les contraintes d'intégrité de transition portent également le nom de contraintes d'intégrité *dynamiques*.

II.4.4. Les SGBD relationnels : conclusion

Tout au long de ce chapitre, nous avons mis en évidence les énormes avantages des SGBD relationnels par rapport aux SGBD de première génération auxquels appartiennent notamment les SGBD hiérarchiques. Le principal atout de ce progrès est la séparation physique-conceptuel qui a permis de faciliter fortement la tâche de l'utilisateur puisque ce dernier dispose à présent d'un langage de haut niveau grâce auquel il peut manipuler un certain nombre d'opérateurs prédéfinis qu'il lui suffit d'appliquer à des données représentées de façon homogène et uniforme. De plus, l'environnement relationnel a permis de développer différents outils dits "de 4ème génération" qui libèrent l'utilisateur de toute tâche qui ne soit pas directement inhérente à l'opération qu'il désire effectuer sur la base de données.

Quoique largement dominants sur le marché actuel des systèmes de gestion de bases de données (et à juste titre), les SGBD relationnels souffrent cependant de certaines limites telles que le support d'objets complexes, la gestion de données réparties et la gestion de connaissances. Ces 3 lacunes sont actuellement sujets de recherches actives. Pour notre part, nous nous intéresserons au problème de gestion des connaissances, problème auquel les SGBD déductifs tentent d'offrir une solution, comme nous allons le voir dans le chapitre suivant.

III. LES BASES DE DONNEES DEDUCTIVES

III.1. Introduction

Nous avons vu brièvement dans le chapitre précédent les caractéristiques dominantes des bases de données hiérarchiques et relationnelles, nous allons à présent nous consacrer aux caractéristiques fondamentales des bases de données déductives.

Après une courte présentation générale dans laquelle nous situerons les bases de données déductives par rapport aux bases de données relationnelles et nous adapterons les hypothèses de CODD aux bases de données déductives, nous nous consacrerons à leur aspect plus formel. De la sorte, nous montrerons pourquoi les bases de données déductives sont aussi appelées bases de données logiques. Dans cette partie figureront quelques rappels théoriques de la logique des prédicats du premier ordre. Les définitions de base seront explicitées et l'utilité de la logique lorsqu'elle est appliquée aux bases de données sera mise en lumière. Un bref exposé des optiques adoptées dans les bases de données relationnelles et déductives sera également donné.

Ce n'est qu'après avoir posé les bases théoriques que les différents composants des bases de données déductives seront présentés, suivis de quelques explications sur les potentialités des SGBD déductifs.

Nous terminerons par l'étude du problème des contraintes d'intégrité et du maintien de la cohérence lors de mises à jour diverses.

III.2. Présentation générale

Nous savons qu'une base de données a pour objectif de représenter et de maintenir une connaissance sur une certaine réalité appelée le monde modélisé. Deux types d'informations peuvent y être contenues : les informations élémentaires et les informations générales, également appelées lois générales. Nous avons vu que dans le cadre des bases de données relationnelles, les informations peuvent être représentées par un ensemble d'éléments reliés entre eux par des relations. Seules les informations élémentaires sont représentées ; les lois générales sont utilisées comme contraintes d'intégrité. Les bases de données déductives sont une extension des bases de données relationnelles, les potentialités de ces dernières sont donc présentes dans les bases de données déductives mais les bases de données déductives permettent en plus de

représenter des lois générales autres que les contraintes d'intégrité.

Avant de considérer la formalisation des bases de données et de centrer notre étude sur les spécificités des bases de données déductives, il est nécessaire d'adapter les hypothèses de CODD données dans le chapitre précédent (point II.4.2.) pour le modèle relationnel. Elles ont été reprises par REITER qui les a adaptées au modèle déductif [Reiter78]. Leur objectif est toujours de spécifier la représentation implicite des faits négatifs et de préciser l'univers de référence des requêtes mais dans une base de données déductive cette fois-ci. La modification apportée par REITER aux 3 hypothèses (CWA, DCA, UNA) pour les adapter aux bases de données déductives peut se résumer en une phrase : en plus des faits physiquement présents dans la base de données, il faut également considérés comme "vrais" les éléments qu'il est possible de déduire à partir du contenu de la base de données. Ainsi, la nouvelle formulation de la CWA pourrait être : les seules instances possibles d'une relation sont celles impliquées par (ou "que l'on peut déduire de") la base de données.

Nous allons à présent montrer comment, sous les hypothèses énoncées, une base de données (qu'elle soit relationnelle ou déductive) peut être exprimée en termes de la logique. En même temps, nous montrerons la parfaite compatibilité entre le monde des bases de données relationnelles dans lequel la notion de "vérité" est fondamentale et le monde des bases de données déductives qui, lui, est basé sur la notion de "preuve".

III.2.1. Logique et bases de données

Nous avons vu au cours du chapitre précédent qu'une condition sine qua non d'une utilisation aisée d'une base de données est la disponibilité d'un mécanisme de raisonnement et de représentation efficace et adapté, permettant de gérer de grandes quantités de données.

Lorsque l'on s'intéresse aux bases de données déductives, le problème prend un tout autre aspect puisqu'à partir de certaines données connues et de "définitions logiques" dont nous verrons la forme par la suite, le SGBD doit être en mesure de générer de nouvelles données. La complexité de l'environnement dans lequel on se situe nécessite le recours à des mécanismes de raisonnement formels ; la logique mathématique convient particulièrement bien dans ce contexte, comme nous allons le montrer dans le paragraphe suivant.

III.2.2. La logique du premier ordre : qu'est-ce ?

La logique du premier ordre, aussi appelée calcul des prédicats, est essentielle au développement et à la compréhension des bases de données déductives. Il s'agit d'un système formel utilisé pour représenter des relations entre objets et pour déduire de nouvelles relations à partir de relations existantes et vraies. Présentée de la sorte, la logique du premier ordre semble être le remède miracle qui offre un pouvoir absolu à l'utilisateur d'une base de données ! Il convient donc de clarifier la situation en définissant ce qu'est exactement un système formel, avec ses potentialités et ses limites dans le cadre des bases de données déductives.

Un système formel est défini comme étant composé d'un langage objet, d'un composant sémantique, d'une théorie de la preuve et d'un certain nombre d'axiomes logiques (bien que ces derniers soient très souvent implicites car repris dans le composant 'théorie de la preuve').

Schématiquement, les composants du calcul des prédicats du premier ordre se présentent donc ainsi [VanLams89].

Calcul des prédicats du premier ordre

=

Langage du premier ordre
+
Axiomes logiques
+
Sémantique
+
Théorie des preuves

Cette définition reste cependant lapidaire, c'est pourquoi nous allons reprendre chacun des éléments et les étudier pour le système formel particulier que constitue le calcul des prédicats du premier ordre.

III.2.2.1. Langage du premier ordre

Un langage du premier ordre a pour objectif de fixer la syntaxe autorisée dans le système utilisé, il peut varier car il est défini par la personne qui fixe le système formel.

Les différents éléments généralement utilisés dans un langage sont les suivants :

- variables : x, y, z, ...
- constantes : a, b, c, ...
- prédicats à n arguments (n fixe) : P, Q, R, ...
- connecteurs logiques : ->, ¬, V, &, <->
- fonctions à n arguments (n fixe) : f, g, h, ...
- quantificateurs : ∀, ∃.

A partir de ces éléments, on définit les concepts suivants :

- Un terme est soit :
  - une variable,
  - une constante,
  - le résultat de l'application d'une fonction à un terme.

- Une formule atomique ou atome :
  Si P est un prédicat à n arguments et t1, ..., tn sont des termes, alors $P(t_1, ..., t_n)$ est une formule atomique.

- Un atome "ground" :
  Un atome ne contenant que des variables auxquelles une valeur a été assignée est dit "ground".

- Une formule bien formée :
  - Toute formule atomique est une formule bien formée.
  - Si F1 et F2 sont des formules bien formées, alors F1 V F2, F1 & F2, ¬F1, F1 -> F2, F1 <-> F2 sont des formules bien formées.
  - Si F est une formule bien formée et x une variable libre (non quantifiée) dans F, alors ∃ x F et ∀ x F sont des formules bien formées.

- Une formule bien formée fermée :

  Une formule bien formée qui ne contient aucune variable libre, càd qui contient uniquement des variables quantifiées et des constantes est appelée une formule bien formée fermée.

- Une formule bien formée sous forme prénexe :

  Une formule bien formée dans laquelle les quantificateurs sont en tête de formule est dite sous forme prénexe.

- Un littéral :

  On appelle littéral soit une formule atomique, soit la négation d'une formule atomique.

- Une clause :

  On appelle clause une disjonction de littéraux dont toutes les variables sont implicitement quantifiées universellement.

$$\neg A_1 \text{ V } \neg A_2 \text{ V } \ldots \text{ V } \neg A_n \text{ V } B_1 \text{ V } \ldots \text{ V } B_m$$

$$<=>$$

$$A_1 \text{ \& } A_2 \text{ \& } \ldots \text{ \& } A_n \text{ -> } B_1 \text{ V } \ldots \text{ V } B_m$$

(Condition -> Conséquent)

- Une clause est dite 'de Horn' si m = 1.
- Une clause est dite 'ground' si elle ne contient aucune variable non instanciée (à laquelle une valeur n'a pas été assignée).
- Une clause est dite 'range restricted' si chaque variable figurant dans son conséquent figure également dans sa condition.

Le langage du premier ordre permet d'exprimer un certain nombre de formules bien formées spécifiques à un environnement, ces formules bien formées sont généralement connues sous le nom d'"axiomes non logiques". Ce sont ces connaissances, couplées à l'ensemble des composants du calcul des prédicats du premier ordre, qui permettent de définir une théorie du premier ordre particulière.

III.2.2.2. Axiomes logiques

Ces axiomes font partie de la définition du calcul des prédicats du premier ordre et sont utilisés pour effectuer les déductions. Ils ne sont pas particulièrement intéressants pour notre propos et seront donc juste énoncés. Ils sont au nombre de 8 [VanLams89].

Soient A et B des formules bien formées :

1 : A -> (B -> A)
2 : (A -> (B -> C)) -> ((A -> B) -> (A -> C))
3 : (¬B -> ¬A) -> ((¬B -> A) -> B)

Soient A et B des prédicats :

4 : ((∀ x) A(x)) -> A(t)
où A est un symbole de prédicat et t est un terme "libre de x" càd dans lequel x ne figure pas.
5 : (∀ x) (A -> B) -> (A -> (∀ x) (B)) si A n'est pas de forme libre par rapport à x.
6 : A -> (A V B)
7 : (A -> B) -> ((A V C) -> (B V C))
8 : ((A V B) -> C) <-> ((A -> C) & (B -> C))

III.2.2.3. Sémantique

Le langage nous donne les outils nécessaires à l'écriture de formules et autres expressions ; on dispose de moyens syntaxiques, ce n'est bien entendu pas suffisant. En effet, affirmer "cette formule est vraie et cette autre est fausse" n'a pas de sens comme tel ; pour donner une valeur de vérité à une ou plusieurs formules il faut avoir précisé dans quelle interprétation elles doivent être évaluées, une interprétation d'un ensemble de formules bien formées étant la spécification d'un ensemble non vide E (également appelé domaine) dans lequel les constantes et variables prennent leur valeur. Dans le cadre d'un système formel, la sémantique est donc l'élément qui définit les valeurs de vérité.

Dans une interprétation, une formule bien formée fermée est soit vraie, soit fausse, alors qu'une formule bien formée ouverte avec n variables libres ($n > 0$) détermine un ensemble de n-uplets (i.e. une relation) sur E^n. Si les composants de chacun de ces n-uplets sont substitués aux variables libres correspondantes dans la formule ouverte, on obtient une formule fermée et, dans l'interprétation fixée, la valeur de vérité de la nouvelle formule fermée est 'vrai'.

Dans une interprétation donnée, on peut définir un modèle de l'ensemble de formules bien formées. Par modèle, on entend une interprétation dans laquelle toutes les formules bien formées de l'ensemble sont vraies.

Par définition, on dira également qu'une formule bien formée w est une conséquence logique d'un ensemble W de formules bien formées si et seulement si w est vraie dans tous les modèles de W. On notera que w est une conséquence logique de W ainsi :

$$W \mid== w$$

Après ce rappel de quelques notions théoriques, essayons de les appliquer dans le contexte des bases de données, de type relationnel plus particulièrement. A cet effet, rappelons que le concept central de ce type de bases de données est la table relationnelle dans laquelle des informations élémentaires liées par des relations sémantiques peuvent être rangées. Or, à chaque relation d'arité n, on peut associer un symbole de prédicat n-aire, de sémantique équivalente, et à chaque élément d'un domaine, on peut associer une constante. Une base de données relationnelle peut donc être considérée comme une interprétation d'un langage du premier ordre. Les prédicats correspondent alors aux relations entre les objets, valeurs des domaines de la base de données. Les contraintes d'intégrité sont les seuls axiomes non logiques de la base de données puisque ce sont les seules lois générales prises en compte dans les bases de données relationnelles.

A titre de conclusion, on peut affirmer que les trois expressions *interprétation*, *modèle* et *structure relationnelle* sont fort proches. Un *modèle* est une *interprétation* qui rend tous les axiomes vrais ; travailler avec une base de données à structure relationnelle implique une évaluation des requêtes posées en supposant les éléments présents dans la base de données vrais. Tous ces termes sont liés à la définition sémantique de vérité, ils se réfèrent en général à l'expression plus globale ***théorie du modèle***.

Après avoir étudié les trois premiers composants du calcul des prédicats du premier ordre, voyons finalement ce que l'on entend par ***théorie des preuves***.

III.2.2.4. Théorie des preuves

La ***théorie des preuves*** regroupe l'ensemble des méthodes d'inférence dont on dispose, càd des méthodes permettant de déduire de nouvelles formules à partir d'un ensemble de formules connues comme satisfaites (vraies sur tout domaine auquel elles sont appliquées) et appelées axiomes.

Les règles d'inférence de base de la théorie des preuves sont :

- Modus Ponens : si p -> q et p alors q.
- Modus Tollens : si p -> q et ¬q alors ¬p.
- Chaînage : si p -> q et si q -> r alors p -> r.
- Spécialisation : si ∀ x P(x) alors P(a).
- Généralisation : si p sans occurrence de x alors ∀ x P.

A ces cinq règles fondamentales, on préfère en général la méthode d'inférence standard mise au point par ROBINSON en 1965 et appelée *méthode de résolution de ROBINSON* [Robi65] dont l'idée générale peut être résumée ainsi.

Soient 2 formules bien formées f1 et f2 sous forme clausale, à partir desquelles on aimerait dériver une troisième formule f3, appelée résolvante.

A titre d'exemple, on pourrait imaginer les deux clauses :

C1 : Q(x,y,z) V P(d,e)
C2 : ¬Q(a,b,c) V R(x,y)
où a,b,c,d,e représentent des constantes et x,y,z des variables.

Le procédé à suivre pour calculer la résolvante est composé de trois grandes étapes :

Etape 1 - Chercher une paire de littéraux complémentaires càd avec le même foncteur (le même "nom" [0]) dont l'un est positif et l'autre négatif. Dans l'exemple, Q(x,y,z) et ¬Q(a,b,c) sont complémentaires.

---

[0] Un foncteur est un symbole de fonction qui permet de définir une relation entre plusieurs éléments.

Etape 2 - Déterminer s'ils s'unifient càd déterminer s'il existe une substitution de variables telle que les prédicats atomiques puissent être rendu identiques symbole/symbole à une substitution de variables près. Dans l'exemple, la substitution est {x/a, y/b, z/c}.

Etape 3 - Si les formules sont unifiables, alors...

a) Effacer les littéraux unifiés dans les clauses. Dans l'exemple, on obtient les nouvelles clauses C1 : P(d,e) et C2 : R(x,y).

b) Former la disjonction des littéraux restant dans ces deux clauses. On obtient : P(d,e) V R(x,y).

c) Appliquer la substitution révélée par l'unification, càd remplacer les variables par les valeurs qui leur ont été assignées. La résolvante obtenue est P(d,e) V R(a,b).

Le principe de résolution de ROBINSON est intéressant parce qu'il préserve la satisfaisabilité, autrement dit, si les 2 clauses de départ C1 et C2 ont un même modèle commun, alors la clause C3 dérivée par la règle de résolution partage aussi ce même modèle. Cette règle a donc la propriété fort intéressante de préserver la sémantique. Un autre avantage de ce principe de résolution est qu'il s'agit d'une méthode d'inférence plus générale que les règles Modus Ponens, Modus Tollens,... Toutes les règles que l'on sait déduire à partir des cinq règles d'inférence de base peuvent aussi être déduites à partir de la règle de résolution ; l'inverse n'est pas vrai.

Le principe de résolution de Robinson est fondamental en théorie de la preuve puisque cette fois, l'objectif n'est pas de dire si une formule est vraie ou fausse mais bien de dire si elle est dérivable (prouvable) ou non.

Par définition, on dit qu'une formule w est dérivable dans une théorie T à partir d'un ensemble de formules W si et seulement si w est obtenue par un nombre fini d'applications des règles d'inférence de la théorie des preuves à des formules bien formées obtenues de T et de W. Dans le cas particulier où W est vide, w est un théorème de la théorie T. La notation utilisée pour exprimer que w est dérivable de W dans T est :

$$T,W \vdash w$$

L'approche est donc totalement différente de la théorie du modèle mais cela ne pose pas de problème car il est démontré que :

$$W \mid\!\!- w \quad <==> \quad W \mid== w$$

Cette équivalence peut s'expliquer par deux propriétés fondamentales des approches *sémantique* et *théorie de la preuve* : les propriétés de complétude et de consistance (ou cohérence).

Cohérence : $W \mid\!\!- w \quad ==> \quad W \mid== w$
Les seules formules prouvables sont des conséquences logiques.

Complétude : $W \mid\!\!- w \quad <== \quad W \mid== w$
Toutes les conséquences logiques sont dérivables.

==> Les mêmes résultats sont obtenus lorsque l'on travaille selon l'approche théorie du modèle ou selon l'approche théorie de la preuve.

Une conséquence directe de cette équivalence est que, bien que les bases de données relationnelles soient généralement exploitées à partir d'une vue théorie du modèle, elles peuvent aussi être considérées à partir d'une vue théorie de la preuve ; on peut donc affirmer qu'elles constituent un cas particulier des bases de données logiques.

III.2.3. Logique et base de données : conclusion

Une base de données peut être caractérisée d'un point de vue logique de 2 manières différentes [Gallaire84]: soit comme une ***interprétation***, soit comme une ***théorie***. En interprétation, les requêtes et les contraintes d'intégrité sont des formules qui doivent être évaluées dans l'interprétation donnée en utilisant la définition sémantique de vérité. En théorie de la preuve, les requêtes et les contraintes d'intégrité sont des théorèmes à prouver ("est vrai ce qui est prouvable"). Les points de vue interprétation et théorie de la preuve formalisent respectivement les concepts de bases de données relationnelle et de bases de données déductive.

L'environnement théorique est établi, nous pouvons à présent examiner de quoi se compose exactement une base de données déductive...

III.3. Composition

Une base de données déductive est composée de 3 ensembles finis :

- l'ensemble des faits, appelé ***base de données extensionnelle***,
- l'ensemble des règles de déduction, appelé ***base de données intentionnelle***,
- l'ensemble des ***contraintes d'intégrité***.

Dans la base de données extensionnelle, un fait a la forme d'une formule atomique dans laquelle les termes sont des constantes.

Dans la base de données intentionnelle, une règle de déduction a la forme d'une clause de Horn :

$$A <- L_1 \& \ldots \& L_n \text{ avec } n > 0$$

où A est un atome dénotant la conclusion (tête de la règle) et les Li sont des littéraux représentant les conditions (corps de la règle).

Toute variable dans A, $L_1,\ldots,L_n$ est supposée universellement quantifiée sur toute la formule.

Les prédicats qui composent les règles de déduction sont soit de base, soit dérivés. Un prédicat de base apparaît seulement dans la base de donnée extensionnelle et, éventuellement, dans le corps de certaines règles de déduction. Un prédicat dérivé apparaît seulement dans la base de données intentionnelle.

Les prédicats figurant en condition d'une règle de déduction peuvent être évaluables ou non-évaluables. Ces derniers sont des prédicats de base ou dérivés tandis que les prédicats évaluables peuvent, comme leur nom l'indique, être évalués sans accès à la base de données, c'est le cas par exemple d'une comparaison ou d'un prédicat arithmétique.

L'ensemble des contraintes d'intégrité enfin est composé de formules du premier ordre que la base de données doit satisfaire. Une contrainte d'intégrité a la forme :

$$Ic <- L_1 \& L_2 \& \ldots \& L_n \text{ avec } n > 0$$

où les L_i sont des littéraux et les variables sont quantifiées universellement sur toute la formule.

Les contraintes d'intégrité peuvent faire référence à des faits élémentaires et dérivés et donc leur évaluation peut nécessiter le recours aux règles de déduction qui définissent les faits dérivés. L'exemple suivant en donne une illustration.

Faits de base

travail(Smits,Marketing).
travail(Dupuis,Vente).
travail(Delcours,Vente).
"Si travail(X,D) alors X travaille dans le département D"

chef(Dupuis).
"Si chef(X) alors X a le statut de chef"

Contraintes d'intégrité

Ic1 <- resp(X1,D) & resp(X2,D) & X1 <> X2.
"Il y a inconsistance si X1 et X2 sont deux personnes différentes et sont toutes deux responsables du département D"

Ic2 <- supérieur(X,X).
"Il y a inconsistance si une personne est son propre supérieur"

Règles de déduction

resp(X,D) <- travail(X,D) & chef(X).
"Une personne X est responsable du département D si X travaille en D et si X a le statut de chef"

supérieur(X,Y) <- resp(X,D) & travail(Y,D) & X < > Y.
"Une personne X est un supérieur d'une personne Y si X est responsable du département dans lequel Y travaille et si X et Y sont 2 personnes différentes"

supérieur(X,Y) <- supérieur(X,Z) & supérieur(Z,Y).
"Une personne X est un supérieur d'une personne Y si X est supérieur d'une troisième personne Z, elle-même supérieur de Y"

Les deux premiers ensembles forment pratiquement une base de données relationnelle (il faudrait cependant adapter les contraintes d'intégrité car elles sont définies en termes des règles de déduction). Ainsi, l'équivalent de la première règle de déduction dans une base de données relationnelle dans laquelle les requêtes sont exprimées en SQL serait la vue suivante :

```
CREATE VIEW Resp( Nom,Departement )
        AS SELECT Chef.Nom, Travail.Departement
        FROM Travail, Chef
        WHERE (( Travail.Departement = D )
                AND ( Travail.Nom = Chef.Nom ))
```

Une règle de déduction peut donc être comparée à une vue d'un SGBD relationnel ; les règles de déduction sont cependant plus puissantes, elles permettent notamment d'exprimer des relations récursives. Ce gain en puissance d'expression ne sera bien entendu possible que si les règles de déduction sont harmonieusement intégrées dans l'ensemble du SGBD. Cette condition est d'autant plus importante que l'un des objectifs des bases de données déductives est de permettre à l'utilisateur une manipulation homogène des faits, qu'ils soient de base, càd physiquement présents dans la base de données, ou dérivés grâce aux règles de déduction. Au vu de cette considération, un problème se pose immédiatement lorsque l'on reconsidère l'exemple donné ci-dessus : puisque c'est permis, comment procéder pour insérer un nouveau responsable ?

Dans une base de données relationnelle, une vue ne peut en général être que consultée, l'insertion demandée est par conséquent interdite. Une telle restriction ne peut évidemment être imposée dans une base de données déductive puisqu'un fait dérivé doit pouvoir être modifié, inséré ou supprimé au même titre qu'un fait de base.

Par conséquent, bien qu'une base de données déductive puisse être définie comme étant une base de données relationnelle à laquelle une partie intentionnelle a été ajoutée, sa gestion est totalement différente de celle d'une base de données relationnelle.

En résumé, si l'on ne tient pas compte des contraintes d'intégrité qui représentent une forme de connaissance bien particulière, dans une base de données déductive, les connaissances sont divisées en 2 grandes parties. La première est la partie extensionnelle qui regroupe des faits "ground" dans des relations de base. Cette partie extensionnelle reprend les informations élémentaires dont on dispose à un moment donné à propos du monde modélisé. A cette première partie vient s'ajouter la partie intentionnelle, elle est composée d'un ensemble de règles de déduction qui représentent les vues perçues par l'utilisateur de la base de données à propos du monde modélisé. Dans le cadre d'une théorie formelle, elles sont considérées comme des mécanismes de raisonnement grâce auxquels de nouveaux faits peuvent être dérivés des connaissances disponibles.

Pour se resituer par rapport à ce qui a été explicité dans le paragraphe précédent, on peut dire que la partie intentionnelle et les contraintes d'intégrité correspondent aux axiomes non logiques de la théorie du premier ordre. Cet environnement permet d'inférer de nouveaux faits à partir de faits connus et de poser des questions dont les réponses sont plus "intelligentes" qu'une énumération de faits. Pour illustrer le type de réponses que l'on est en droit d'attendre, supposons une base de données contenant, parmi d'autres, le fait Socrate est un homme et la connaissance Tout homme est mortel. Le fait Socrate est mortel peut en être déduit et à la question Qui est mortel ?, la réponse Tout homme est produite.

En général, seules les règles de déduction sont rangées dans la base de données, les faits dérivés ne sont normalement pas conservés comme le sont les faits élémentaires. Ceci nécessite cependant quelques commentaires. En effet, actuellement, certains prototypes de bases de données déductives permettent, ou même obligent, le rangement des faits dérivés dans des tables similaires à celles des faits élémentaires ; pourquoi cela ?... La réponse est simple : pour des raisons de performances.

Nous ne verrons pas comment le SGBD procède pour répondre à une requête (voir [Bry90] [Bry-Manthey90] [Cholvy90] [Ross90] [Bancil-Ramak86]) mais nous pouvons néanmoins, par exemple en se remémorant comment fonctionne la méthode de résolution SLDNF (utilisée dans l'interpréteur PROLOG), subodorer un procédé assez complexe. Or, l'objectif principal des concepteurs de SGBD déductifs est de les rendre compétitifs sur le marché des bases de données, ce qui n'est concevable que s'ils offrent des temps de réponse aux requêtes équivalents ou supérieurs à ceux des SGBD relationnels. D'une part, on a donc des évaluateurs de requêtes qui n'ont pas encore

atteint leur meilleur niveau de performances, d'autre part figurent les SGBD relationnels dont les performances ne sont plus à démontrer. La réponse à notre "pourquoi" est maintenant évidente : si les faits dérivés sont rangés dans des tables relationnelles, on peut y accéder grâce aux techniques relationnelles, et par conséquent obtenir des performances excellentes tout en conservant les avantages relatifs à l'utilisation de règles de déduction. Cette solution est théoriquement très attrayante mais pratiquement, elle est d'une complexité énorme !

Le rangement de faits dérivés pose d'énormes problèmes lorsque l'on désire insérer ou supprimer des faits dérivés. En effet, on peut envisager soit d'insérer le fait dérivé dans sa table, soit de trouver les insertions ou suppressions de faits de base rendant le fait dérivé prouvable ; ces deux possibilités ont des effets sur la base de données totalement différents. De plus, il ne peut y avoir contradiction entre les faits dérivés effectivement grâce aux faits de base et aux règles de déduction et les faits dérivés rangés dans des tables, ces derniers sont-ils également dérivables ? Comment assurer efficacement la cohérence d'une telle base de données ? Il suffit d'imaginer quelques opérations possibles sur la base de données pour se rendre compte de l'énorme complexité qu'entraîne cette organisation.

Dans ce document, les bases de données déductives avec rangement dans des tables d'informations dérivées ne sont pas étudiées (en accord avec la littérature actuelle dans laquelle très peu d'articles étudient ce cas). Nous développerons l'approche dans laquelle l'assimilation d'une nouvelle information dérivée n'est pas effectuée par l'ajout explicite de cette information dans la base de données. Au lieu de cela, la nouvelle information est assimilée de façon implicite par l'addition explicite d'autres connaissances élémentaires. Indépendamment des remarques faites ci-avant, plusieurs arguments permettent de justifier ce choix.

Premièrement, l'insertion d'une nouvelle pièce d'information permet d'inférer d'autres informations implicites qui seraient perdues si la nouvelle information était seulement insérée explicitement dans la base de données. Par exemple, considérons une base de données sur les familles organisée principalement en termes de la relation parent. Alors, une nouvelle pièce d'information telle que Tom est le frère de Bob, si ajoutée explicitement comme un fait, n'apporte aucune information implicite telle que le fait que Tom et Bob ont les mêmes grand-parents. Par contre, si ceci est assimilé en ajoutant un parent commun à Bob et Tom, alors la base de données contient implicitement le fait qu'ils ont les mêmes grands-parents [Gallaire84].

Deuxièmement, différents utilisateurs peuvent avoir différents ensembles de règles intentionnelles avec lesquelles ils peuvent manipuler les données présentes dans la base de données extensionnelle commune. Quand une nouvelle information sur une relation dérivée arrive à un utilisateur particulier, elle est assimilée en changeant de façon appropriée la partie extensionnelle plutôt qu'en l'ajoutant explicitement comme un fait dérivé. De cette manière, différents utilisateurs partagent la nouvelle information reçue par chacun, et la structure de la base de données garde sa forme initiale.

Une dernière raison importante qui motive l'exécution d'une mise à jour de faits dérivés en changeant la partie extensionnelle est la suivante : les règles, qui représentent des lois générales, sont considérées comme complètes et indéfaisables, elles ne devraient donc pas être changées (la définition de la relation grand-parent est établie une fois pour toutes) ; par contre, la partie extensionnelle contient une connaissance incomplète et défaisable du monde, des données d'une relation de base peuvent être ajoutées ou supprimées en fonction des changements survenus dans le monde modélisé.

Avant de poursuivre notre étude de ce nouveau type de bases de données et, notamment, d'en préciser les potentialités, il est nécessaire de dire quelques mots à propos des caractéristiques du langage utilisé dans un SGBD déductif.

III.4. Caractéristiques du langage utilisé

Comme tout autre SGBD, un SGBD déductif doit posséder un langage de description de données ainsi qu'un langage de requêtes [Gard-Vald90]. En plus de cela, l'utilisateur doit pouvoir exprimer ses connaissances sous forme de règles de déduction, d'où la nécessité d'un troisième type de langage dit "de règles". En toute généralité, un langage de règles doit être, dans la mesure du possible, non procédural et ensembliste. En d'autres termes, si un utilisateur désire connaître toutes les personnes engagées après le premier janvier 1990 par exemple, il n'a ni à spécifier comment procéder pour obtenir le résultat, ni à préciser qu'il désire obtenir le premier, puis le suivant,... et terminer quand le système est arrivé à tel point. L'utilisateur doit uniquement fournir les informations strictement nécessaires, comme dans une requête SQL par exemple.

```
SELECT Engage.Nom
    FROM Engage
    WHERE Engage.Date > '01/01/90'
```

En résumé, tout ce qui est exprimable en SQL doit pouvoir être exprimé dans le langage utilisé dans un SGBD déductif (en particulier les opérations classiques du calcul relationnel), mais en plus de cela, il est également souhaitable de pouvoir exprimer la récursivité, la négation (ex.: rechercher toutes les personnes d'un département donné qui ne sont pas des chefs),... dans le but de tirer un profit maximum du pouvoir d'expression offert par les règles de déduction. DATALOG [1] est un de ces langages. Il peut être vu comme une variante de PROLOG avec une sémantique ensembliste, ce qui rend le résultat d'un programme indépendant de l'ordre d'apparition des clauses dans ce programme.

Une étude approfondie d'un de ces langages n'est pas un prérequis indispensable au travail sur les bases de données déductives ; en général, il est fait référence à PROLOG ou à une de ses versions améliorées. Les articles [Ross90] [Abiteboul90] sont de bonnes références ; faute de place, cette matière ne sera pas développée dans ce mémoire.

III.5. Potentialités des SGBD déductifs

Les SGBD déductifs offrent de nouvelles possibilités d'applications des bases de données puisqu'ils permettent une certaine forme de raisonnement sur les données disponibles.

La création d'un SGBD déductif est bien entendu fort complexe mais l'enjeu est à la mesure. En effet, si un maximum de connaissances sont intégrées dans la base de données intentionnelle, le développement d'applications nouvelles autour de la base de données en sera fortement facilité puisque le programmeur-utilisateur devra uniquement savoir ce dont il a besoin, par quel prédicat y accéder et, éventuellement, comment définir de nouvelles lois générales. De plus, chaque utilisateur peut avoir sa propre partie intentionnelle de la base de données, ce qui personnalise la vue de chacun à propos du monde modélisé sans pour autant rendre le système plus complexe.

Un avantage supplémentaire des bases de données déductives est qu'elles permettent d'éviter la duplication de programmes à fonctionnalités voisines. Ainsi, grâce à la règle de déduction `resp(X,D) <- travail(X,D) & chef(X)`, on peut retrouver qui est le responsable d'un département donné (en instanciant la variable D lors de la requête) mais

---

[1] Détails dans [Gard-Vald90].

également de quel département une personne donnée est responsable (en instanciant la variable X). Une autre illustration peut être donnée en considérant l'exemple traditionnel (déjà mentionné dans le point précédent) d'une base de données représentant les familles. Si on considère la relation parent(P,M,E) comme relation centrale, on peut définir les relations représentant les frère, grand-parent, cousin, oncle,... en une seule règle de déduction.

Ces quelques avantages ne sont bien entendu pas exhaustifs. En toute généralité, on peut dire que le principal atout d'un SGBD déductif est sa facilité d'utilisation quelles que soient les compétences de l'utilisateur ; et ce, grâce à la possibilité de raisonnement symbolique et grâce à l'emploi d'un langage formel fort proche du langage humain. Un utilisateur habitué à manipuler un SGBD relationnel classique risque cependant d'être quelque peu décontenancé dans un premier temps. Le mode de raisonnement est en effet différent puisque l'on se situe à un niveau d'abstraction plus élevé mais l'adaptation ne devrait pas poser de problème, la principale difficulté étant de penser "à la PROLOG".

Il est cependant bien connu qu'une utilisation aisée cache souvent une gestion interne au SGBD fort complexe. C'est le cas pour les SGBD déductifs ; nous nous limiterons à étudier les problèmes de gestion des contraintes d'intégrité et des opérations de mise à jour.

III.6. Gestion des contraintes d'intégrité

III.6.1. Introduction

Nous commencerons par une brève présentation de la situation, suivie de quelques explications se rapportant à ce que l'on entend par cohérence d'une base de données déductive.

Le problème du maintien de la cohérence d'une base de données lors de mises à jour est ensuite développé. Deux angles d'approche différents sont présentés pour les mises à jour de faits élémentaires.

Dans le premier, on se préoccupe des répercussions d'une mise à jour tandis que dans le second, les contraintes d'intégrité sont gérées comme des axiomes d'une théorie formelle et leur vérification est assurée par le biais de techniques de démonstration de théorèmes.

Nous poserons enfin le problème des mises à jour de faits dérivés, également appelées mises à jour de vues.

III.6.2. Présentation de la situation

Dans les SGBD commerciaux, une bonne gestion des contraintes est rarement implémentée, excepté pour certaines classes réduites de contraintes d'intégrité telles que l'identifiant. Pourquoi ?... Des techniques générales efficaces manquent !

Il existe néanmoins un grand effort dans le domaine du contrôle des contraintes d'intégrité. Des méthodes se développent, elles diffèrent selon le genre de base de données considérées (ex : relationnelle ou déductive), selon le genre de contraintes d'intégrité qu'elles permettent et garantissent, selon le genre de mises à jour qu'elles considèrent (exemple : simple insertion de faits ou mises à jour complexes) et, bien sûr, selon l'approche particulière suivie par chaque méthode (voir [Decker90] [Kakas-Manca90] [Pastor90] [Ullman90] [Nytro90]).

Le problème de la cohérence d'un ensemble de contraintes et/ou de règles reste cependant presqu'entièrement posé. Une gestion adaptée est pourtant bien nécessaire car rien ne permet de supposer que les contraintes sont très peu mises à jour. Une contrainte est partie intégrante de la base de données et doit, de ce fait, pouvoir être mise à jour sans restriction particulière.

Le contrôle d'intégrité peut être effectué de manière naïve et coûteuse en évaluant toutes les contraintes sur toute la base de données après chaque mise à jour. Les coûts d'évaluation résultants sont évidemment excessivement élevés. Une façon de réduire drastiquement ces coûts consiste à exploiter au maximum l'hypothèse selon laquelle la base de données est cohérente avant la mise à jour. En effet, seules les contraintes et la partie de la base de données concernées par la mise à jour doivent alors être examinées. Cette hypothèse est assez optimiste et demande beaucoup de rigueur d'implémentation car il est fort aisé (... et fréquent) d'obtenir une base de données incohérente. Par définition, les données et réponses ne contiennent jamais de contradiction. Seules les contraintes d'intégrité et les règles de déduction sont une source potentielle d'incohérence. Mais qu'est-ce que la cohérence lorsque l'on parle de bases de données déductives ?

III.6.3. Cohérence / Incohérence d'une base de données déductive

D'un point de vue formel, la cohérence est une propriété d'un ensemble d'axiomes (formules fermées). En théorie de la preuve, on dit qu'un ensemble A est cohérent s'il n'y a pas de formule F telle que F et ¬F sont des conséquences logiques de A. Tandis qu'en théorie du modèle, A est cohérent s'il a un modèle, càd une interprétation dans laquelle chaque axiome de A est vrai. L'équivalence de ces deux définitions a été démontrée par GOEDEL.

On peut distinguer deux formes différentes d'incohérence. Premièrement, les contraintes d'intégrité et les règles de déduction ne posent pas de problème, mais les faits ne satisfont pas les contraintes : soit des faits présents sont interdits par les contraintes d'intégrité, soit des faits requis par les contraintes d'intégrité sont manquants ; d'où un problème dit de ***satisfaction*** des contraintes.

Deuxièmement, les contraintes d'intégrité et/ou les règles sont mal conçues, les contraintes d'intégrité ne peuvent en principe être satisfaites. Tout fait inséré entraînerait une violation de contraintes ; d'où un problème dit de ***satisfaisabilité*** des contraintes.

Le problème de satisfaisabilité est plus général que le problème de satisfaction. En effet, si une contrainte d'intégrité est satisfaite à fortiori elle est satisfaisable mais le contraire n'est pas vrai : une contrainte d'intégrité peut être satisfaisable sans pour autant être satisfaite.

Les contraintes d'intégrité et les règles (sans les faits) peuvent être satisfaisables mais pas applicables pratiquement, dans ce cas le seul modèle qu'il est possible d'obtenir est le modèle vide. Contrairement à ce que l'on pourrait penser, une telle situation n'est pas rare, comme l'illustre l'exemple ci-dessous [Bry-Manthey90].

Soit la base de données suivante (pour les besoins de l'illustration, les contraintes d'intégrité n'ont pas la forme de clauses de Horn) :

Règles de déduction

1. travail_pour(X,Z) <- membre(X,D) & resp(Z,D) & X<>Z.

"Une personne X travaille pour une autre personne Z si X est membre du département D et si Z est responsable de ce même département"

2. membre(X,D) <- resp(X,D).

"Une personne X est membre d'un département D si X en est le responsable"

IV. LA METHODE DES EVENEMENTS INTERNES

IV.1. Introduction

Au cours de ce chapitre, nous verrons comment la méthode des événements internes permet de gérer le contrôle d'intégrité d'une base de données déductive, principalement lors de mises à jour diverses.

De très nombreux exemples seront donnés, pour en faciliter la compréhension, ils feront tous référence à la même base de données ; elle figure en début de ce chapitre.

Suite à cela, une présentation de la méthode des événements internes ainsi que de ses différents éléments spécifiques est donnée ; il est fondamental d'en comprendre parfaitement la signification et la raison d'être.

Le reste du chapitre explique comment la méthode des événements internes permet d'assurer le contrôle des contraintes d'intégrité statiques et dynamiques lors de mises à jour diverses. La solution que la méthode des événements internes offre au problème des mises à jour de vues est finalement présentée, suivie d'une brève conclusion.

IV.2. Base de données servant d'exemple

La base de données ci-dessous servira d'exemple de base tout au long de ce chapitre. Il est fortement recommandé au lecteur d'essayer de mémoriser la signification des contraintes d'intégrité et des règles de déduction car elles sont fréquemmment utilisées.

Faits de base

trav(Smits,Marketing).
trav(Dupuis,Vente).
trav(Delcours,Vente).
chef(Dupuis).

Contraintes d'intégrité

Ic1 <- resp(X1,D) & resp(X2,D) &X1 < > X2.

"2 personnes différentes ne peuvent pas être responsables d'un même département".

Ic2 <- sup(X,X).

"1 personne ne peut être son propre supérieur ".

Règles de déduction

resp(X,D) <- trav(X,D) & chef(X).

"Une personne X est responsable du département D si Xtravaille dans le département D et si X a le statut de chef".

sup(X,Y) <- resp(X,D) & trav(Y,D) & X < > Y.

"Une personne X est un supérieur d'une personne Y si X est responsable du département D, si Y travaille dans le département D et si X et Y sont deux personnes différentes".

sup(X,Y) <- sup(X,Z) & sup(Z,Y).

"Une personne X est un supérieur d'un personne Z si X est un supérieur de Z et si Z est un supérieur de Y".

IV.3. Présentation

La méthode des événements internes est due à OLIVE [Olive90] et est actuellement développée par son équipe à l'Université Polytechnique de Catalogne. Les travaux sont toujours en phase expérimentale. Le prototype est implémenté en PROLOG, la résolution SLDNF standard est utilisée et donc le problème de la réponse aux requêtes n'est pas abordé.

La méthode des événements internes permet de traiter les problèmes des contraintes d'intégrité et mises à jour de vues avec une relative simplicité une fois les notions de base acquises.

Partant d'une base de données déterminée, la méthode des événements internes ajoute à la base de données un certain nombre de règles particulières qui sont utilisées pour définir explicitement les insertions et suppressions de faits générées ou nécessaires pour effectuer une mise à jour donnée. Les paragraphes suivants sont consacrés à la présentation de ces règles. Avant de les aborder, il est nécessaire de s'attarder quelque peu sur le concept d'événement interne.

IV.4. Qu'est-ce qu'un événement interne ?

Les différents éléments dont on a besoin pour définir un événement interne sont les suivants.

Soit D, une base de données, U, une mise à jour et D', la base de données D mise à jour. U produit une transition de D, état courant, vers D', nouvel état.

Par simplification, supposons momentanément que U soit composé d'un ensemble de faits de base à insérer et/ou à supprimer.

De par les règles de déduction, il est évident que la mise à jour U, uniquement composée d'actions sur les faits de base, peut entraîner d'autres mises à jour sur certains prédicats dérivés.

Considérons P, un prédicat dérivé dans D et P', ce même prédicat évalué dans D'.

4 cas de figure sont possibles :

1 - P(K) est vrai dans D et P'(K) est vrai dans D'.
(K est un vecteur de constantes.)
2 - P(K) est vrai dans D et P'(K) est faux dans D'.
3 - P(K) est faux dans D et P'(K) est faux dans D'.
4 - P(K) est faux dans D et P'(K) est vrai dans D'.

Dans les situations 2 et 4, la valeur de vérité du prédicat dérivé P(K) est modifiée suite à la mise à jour ; il s'agit d'un événement interne à la base de données puisqu'il est "invisible" extérieurement et uniquement produit par des effets sur les faits de base, agissant indirectement sur les faits dérivés par l'intermédiaire des règles de déduction.

Dans le deuxième cas, on voit que la mise à jour U a pour effet de supprimer un fait évalué à vrai dans D. Par définition, lorsque lors d'une transition d'un état D vers un état D', un fait de la base de données initiale D change de valeur, en l'occurrence passe de la valeur "vrai" à la valeur "faux", on dit qu'un événement interne de suppression [1] a lieu, ce qui se note dP(K).

Dans le quatrième cas, la situation est similaire puisque la mise à jour U a pour effet d'insérer dans D' un fait évalué à faux dans D ; on dit alors qu'un événement interne d'insertion [2] se produit lors de la transition de D vers D', cela se note par iP(K).

Ces événements internes peuvent donc être formellement définis comme suit :

(1) $\forall$ x (iP(x) <-> P'(x) & $\neg$P(x))
(2) $\forall$ x (dP(x) <-> P(x) & $\neg$P'(x))
où x est un vecteur de variables.

De ceci, on peut déduire :

(3) $\forall$ x (P'(x) <-> (P(x) & $\neg d$P(x)) v iP(x))
(4) $\forall$ x ($\neg$P'(x) <-> ($\neg$P(x) & $\neg i$P(x)) v dP(x))

---

[1] En anglais : "deletion internal event"
[2] En anglais : "insertion internal event"

Des règles (1) et (2) ci-dessus, on peut également déduire :

(5) $\forall$ x (*i*P(x) -> $\neg$P(x)) et
(6) $\forall$ x (*d*P(x) -> P(x))

qui font également partie des prédicats de base. Ils se comprennent aisément dans la mesure où on ne peut insérer dans la base de données que des faits qui n'y sont pas encore et on ne peut en supprimer que des faits présents.

Si P est un prédicat de base, la situation est simple. Les faits '*i*P' représentent les insertions de faits de base qui ont lieu lors d'une mise à jour tandis que les faits '*d*P' représentent les suppressions de faits de base. Les faits dérivés grâce aux règles de déduction sont dépendants des faits de base, il est par conséquent possible d'exprimer une mise à jour de la base de données exclusivement en termes d'insertions et/ou de suppressions d'événements internes de prédicats de base. Pour cela, il faut cependant donner la transition entre un événement interne d'un prédicat dérivé et un ensemble d'événements internes de prédicats de base. Les règles de transition, définies dans le point suivant, ont pour fonction d'établir ce lien. Elles expliquent comment, à partir des règles de déduction et d'événements internes, un prédicat dérivé peut "changer de valeur" lors d'une transition d'un état de la base de données vers un autre.

IV.5. Les règles de transition

Soit P, un prédicat dérivé de la base de données.

L'ensemble des règles dans lesquelles P figure en conclusion ("en tête") définissent P. Soit m (m > 0) le nombre de règles de cet ensemble. Par souci de clarté, les conclusions de ces m règles sont renommées $P_1, \ldots, P_m$. On a donc un nouvel ensemble de clauses :

$$P <- P_i \ ; \ i=1..m$$

et $P_i <-> L_1 \ \& \ \ldots \ \& \ L_n$

où P_i est la tête, ou conséquent, de la règle et possède son propre vecteur de variables, et les L_j (j=1..n) sont les littéraux formant le corps, ou condition, de la règle et possèdent chacun leur vecteur de variables respectif.

L'ensemble des littéraux dont le vecteur de variables est inclus dans celui de P_i constitue la partie universelle de la règle et se dénote $U(P_i)$. En opposition, $E(P_i)$ est la partie existentielle de la règle et regroupe l'ensemble des littéraux dont au moins une variable ne figure pas parmi celles de P_i. Tout littéral du corps de la règle appartient soit à $U(P_i)$, soit à $E(P_i)$.

Exemple :

```
chef_potentiel(X) <- diplôme(X) & travail(X,Y) & Y > 5
U(chef_potentiel(X)) = { diplôme(X) }
E(chef_potentiel(X)) = { travail(X,Y), Y > 5 }
```

Soit $P_i(x) <-> L_1 \& \ldots \& L_n$, une des règles définissant P.

Lorsque cette même règle doit être évaluée dans l'état mis à jour, sa forme est $P'_i(x) <-> L'_1 \& \ldots \& L'_n$.

Si on remplace chaque littéral du corps par sa définition équivalente en termes de l'état "de départ" et des événements internes, on obtient une nouvelle règle, appelée **règle de transition**, qui définit le prédicat P'_i.

Plus précisément, supposons le littéral L'_j positif et de forme $Q'_j(x_j)$. En appliquant la règle (3), on peut le remplacer par :

$$(Q'_j(x_j) <-> (Q_j(x_j) \& \neg dQ_j(x_j)) \vee iQ_j(x_j))$$

Si L'_j est négatif et a la forme $\neg Q_j(x_j)$, en appliquant la règle (4), on peut le remplacer par :

$$(\neg Q'_j(x_j) <-> (\neg Q_j(x_j) \& \neg iQ_j(x_j)) \vee dQ_j(x_j))$$

Si L_j est un prédicat évaluable, aucune consultation de la base de données n'est nécessaire pour connaître sa valeur et L'_j est identique à L_j.

Pour éclaircir l'écriture des expressions résultantes, il est utile de recourir aux notations suivantes :

$O(L'_j)$ représente la partie de L'_j qui ne change pas.

"On avait ceci avant mise à jour, on veut le conserver donc on nie sa modification".
(O <=> Old).

$$\begin{aligned} O(L'_j) &= (Q_j(x_j) \;\&\; \neg dQ_j(x_j)) && \text{si } L'_j = Q'_j(x_j) \\ &= (\neg Q_j(x_j) \;\&\; \neg iQ_j(x_j)) && \text{si } L'_j = \neg Q'_j(x_j) \\ &= L_j && \text{si } L_j \text{ est évaluable.} \end{aligned}$$

$N(L'_j)$ représente la partie modifiée par rapport à la situation ancienne.

"Un fait était présent, on le supprime. Un fait était absent, on l'insère."
(N <=> New).

$$\begin{aligned} N(L'_j) &= iQ_j(x_j) && \text{si } L'_j = Q'_j(x_j) \\ &= dQ_j(x_j) && \text{si } L'_j = \neg Q'_j(x_j) \end{aligned}$$

Avec ces notations, on vérifie alors :

$$(7) \quad P'_i(x) <-> \;\&_{r=1}^{n} [O(L'_r) \vee N(L'_r) \mid O(L'_r)]$$

où la première option est prise si L'_j est un littéral non évaluable et la seconde dans le cas contraire.

Après distribution des "&" sur les "v", on obtient un ensemble équivalent de règles de transition ; elles représentent en fait toutes les combinaisons possibles d'états (O ou N) des littéraux. Ces règles ont la forme générale :

$$(8) \quad P'_{i,j}(x) <-> \;\&_{r=1}^{n} [O(L'_r) \mid N(L'_r)] \quad j = 1..2^k$$

où k est le nombre de littéraux non évaluables dans la règle P_i, et

(9) $P'_i(x) <- P'_{i,j}(x) \quad j = 1..2^k$

On peut souligner le cas particulier où "rien ne change", la valeur des littéraux n'est pas modifiée. (Ce cas sera utile par la suite).

(10) $P'_{i,1}(x) <-> O(L'_1) \& \ldots \& O(L'_n)$.

L'exemple ci-dessous illustre clairement la signification de tous ces formalismes, il donne l'ensemble des règles de transition associées aux règles de déduction de l'exemple de base de données déductive donnée en début de chapitre. Les règles de déduction sont reprises dans les trois premières lignes ; les lignes suivantes donnent les règles de transition établies par application des règles formelles explicitées ci-avant.

Pour bien comprendre le sens de ces règles, il faut absolument avoir à l'esprit qu'un prédicat P' représente le prédicat P dans la base de données mise à jour.

Intuitivement, la règle $resp'_{1,1}(X,D)$ peut s'expliquer comme suit. Elle établit que $resp'_{1,1}(X,D)$ est vrai (et donc resp' aussi) si :

1 - X travaille dans le département D avant la mise à jour (trav(X,D)) ;
2- le fait trav(X,D) n'est pas supprimé lors de la mise à jour ($\neg \delta$trav(X,D)); événement interne de base négatif) ;
3 - X est a le statut de chef avant la mise à jour (chef(X)) ;
4 - le fait chef(X) n'est pas supprimé lors de la mise à jour.

Toutes les règles ci-dessous ont une explication du même type.

resp(X,D) <- trav(X,D) & chef(X).
sup(X,Y) <- resp(X,D) & trav(Y,D) & X <> Y.
sup(X,Y) <- sup(X,Z) & sup(Z,Y).

resp'(X,D) <- $resp'_1(X,D)$.
$resp'_1(X,D)$ <- $resp'_{1,j}(X,D)$; j = 1..4.
sup'(X,Y) <- $sup'_1(X,Y)$.
sup'(X,Y) <- $sup'_2(X,Y)$.
$sup'_1(X,Y)$ <- $sup'_{1,j}(X,Y)$; j=1..4.
$sup'_2(X,Y)$ <- $sup'_{2,j}(X,Y)$, j=1..4.

$resp'_{1,1}(X,D)$ <- trav(X,D) & ¬*d*trav(X,D) & chef(X) & ¬*d*chef(X).
$resp'_{1,2}(X,D)$ <- trav(X,D) & ¬*d*trav(X,D) & *i*chef(X).
$resp'_{1,3}(X,D)$ <- *i*trav(X,D) & chef(X) & ¬*d*chef(X).
$resp'_{1,4}(X,D)$ <- *i*trav(X,D) & *i*chef(X).

$sup'_{1,1}(X,Y)$ <- resp(X,D) & ¬*d*resp(X,D) & trav(Y,D) & ¬*d*trav(Y,D) & X <> Y.
$sup'_{1,2}(X,Y)$ <- resp(X,D) & ¬*d*resp(X,D) & *i*trav(Y,D) & X <> Y.
$sup'_{1,3}(X,Y)$ <- *i*resp(X,D) & trav(Y,D) & ¬*d*trav(Y,D) & X <> Y.
$sup'_{1,4}(X,Y)$ <- *i*resp(X,D) & *i*trav(Y,D) & X <> Y.

$sup'_{2,1}(X,Y)$ <- sup(X,Z) & ¬*d*sup(X,Z) & sup(Z,Y) & ¬*d*sup(Z,Y).
$sup'_{2,2}(X,Y)$ <- sup(X,Z) & ¬*d*sup(X,Z) & *i*sup(Z,Y).
$sup'_{2,3}(X,Y)$ <- *i*sup(X,Z) & sup(Z,Y) & ¬*d*sup(Z,Y).
$sup'_{2,4}(X,Y)$ <- *i*sup(X,Z) & *i*sup(Z,Y).

Si les événements internes sont de base, leurs effets sont immédiats ; il suffit d'insérer ou de supprimer le fait correspondant dans la base de données. Par contre, s'il s'agit d'événements internes dérivés, comme *i*sup(X,Y) par exemple, intuitivement, leur signification est évidente, mais concrètement, comment réaliser l'insertion ou la suppression d'un fait dérivé ?...

IV.6. Règles d'événements internes d'insertion

On a vu que les règles de transition permettent de définir la valeur de vérité d'un prédicat dérivé dans une base de données mise à jour en utilisant uniquement les règles de déduction et des événements internes. Il s'avère que les règles de transition peuvent également être utilisées pour donner les conditions à satisfaire pour insérer une instance d'un prédicat dérivé dans la base de données ; elles sont exprimées sous forme de règles de déduction et portent le nom de **règles d'événements internes d'insertion.**

Soit P, un prédicat dérivé. Les événements internes d'insertion pour P ont été définis comme suit :

$$\forall\ x\ (iP(x) <-> P'(x) \ \&\ \neg P(x))$$

Si m règles définissent le prédicat P :

$$P'(x) <-> P'_i(x) \; ; \; i = 1..m,$$

les règles d'insertion de P ont la forme :

$$iP(x) <- P'_i(x) \; \& \; \neg P(x) \; ; \; i = 1..m$$

Par définition des règles de transition, ces formes sont équivalentes à :

$$(11) \quad iP(x) <- P'_{i,j}(x) \; \& \; \neg P(x) \; ; \; i = 1..m, \; j = 1..2^k$$

selon la définition donnée en (9).

Ces règles sont appelées règles d'événements internes d'insertion du prédicat P. Elles permettent de savoir quelles insertions induites (ou faits *i*P) ont lieu lors d'une transition.

On peut supprimer certaines de ces règles et, dans certains cas, on peut les simplifier. Par exemple, il est aisé de prouver que lorsque j = 1, cas où tous les littéraux du corps de la règle restent dans un état inchangé, il ne saurait pas y avoir insertion et ¬P(x) simultanément. Des règles (11), on peut donc supprimer le cas où j = 1 et on obtient :

$$(11') \; iP(x) <- P'_{i,j}(x) \; \& \; \neg P(x) \; ; \; i = 1..m, \; j = 2..2^k$$

A partir de ces règles, on peut écrire l'algorithme (trivial) qui détermine l'ensemble des règles définissant les insertions implicites d'un prédicat P.

```
Pour i = 1..m
    Pour j = 2..2^k
        iP(x) <- P'i,j(x) & ¬P(x)
    FinPour
FinPour
```

L'exemple ci-dessous donne les règles d'insertion correspondant aux règles de déduction de la base de données servant d'exemple tout au long de ce chapitre. Elles sont dans un premier temps données sous forme opérationnelle ; ensuite, une d'entre elle est reprise sous une forme plus aisément compréhensible, accompagnée de quelques explications.

*i*resp(X,D) <- resp'$_{1,2}$(X,D) & ¬resp(X,D).
*i*resp(X,D) <- resp'$_{1,3}$(X,D) & ¬resp(X,D).
*i*resp(X,D) <- resp'$_{1,4}$(X,D) & ¬resp(X,D).

*i*sup(X,Y) <- sup'$_{1,2}$(X,Y) & ¬sup(X,Y).
*i*sup(X,Y) <- sup'$_{1,3}$(X,Y) & ¬sup(X,Y).
*i*sup(X,Y) <- sup'$_{1,4}$(X,Y) & ¬sup(X,Y).

*i*sup(X,Y) <- sup'$_{2,2}$(X,Y) & ¬sup(X,Y).
*i*sup(X,Y) <- sup'$_{2,3}$(X,Y) & ¬sup(X,Y).
*i*sup(X,Y) <- sup'$_{2,4}$(X,Y) & ¬sup(X,Y).

La forme longue de la règle *i*resp(X,D) <- resp'$_{1,2}$(X,D) & ¬resp(X,D). est obtenue en remplaçant resp'$_{1,2}$(X,D) par sa définition, cela donne :

*i*resp(X,D) <- trav(X,D) & ¬*d*trav(X,D) & *i*chef(X) & ¬resp(X,D).

La signification de cette règle est fort simple si l'on a bien compris la signification des différents éléments qui la composent. En français, on pourrait dire : "pour insérer dans la base de données le fait resp(X,D) (X est responsable du département D), il faut que :
1 - X travaille dans le département D (trav(X,D)) ;
2 - cette situation ne soit pas modifiée par une mise à jour quelconque (¬*d*trav(X,D)) ;
3 - X soit inséré comme étant un nouveau chef (*i*chef(X)) ;
4 - et, bien entendu, que X ne soit pas déjà responsable du département D, auquel cas l'insertion n'a pas de sens (¬resp(X,D)).

IV.7. Règles d'événements internes de suppression

Le principe et la signification des règles d'événements internes de suppression sont tout à fait analogues à ceux des règles d'événements internes d'insertion. La seule différence est que cette fois, leur fonctionalité est de donner les conditions à satisfaire pour supprimer une occurrence d'un prédicat dérivé de la base de donnée.

Soit P, un prédicat dérivé. Les événements internes de suppression pour P ont été définis comme suit :

$$\forall\ x\ (d\mathrm{P}(x) <-> \mathrm{P}(x)\ \&\ \neg \mathrm{P}'(x))$$

Si m règles définissent le prédicat P, alors

$$d\mathrm{P}(x) <- \mathrm{P}_i(x)\ \&\ \neg \mathrm{P}'(x)\ ;\ i = 1..m\ ;$$

$$\text{or}\ \ \mathrm{P}'(x) <-> \mathrm{P}'_1(x)\ v\ \ldots\ v\ \mathrm{P}'_m(x)\ ;$$

d'où on obtient :

$$d\mathrm{P}(x) <- \mathrm{P}_i(x)\ \&\ \neg \mathrm{P}'_1(x)\ \&\ \ldots\ \&\ \neg \mathrm{P}'_i(x)\ \&\ \ldots\ \&\ \neg \mathrm{P}'m(x)\ ;\ i = 1..m$$

dans lesquelles $\mathrm{P}'_i(x) <-> \mathrm{P}'_{i,j}(x)\ ;\ j = 1..2^k$ d'après la règle (9).

L'algorithme ci-dessous permet de déterminer l'ensemble des règles qui définissent les suppressions implicites du prédicat P induites par une mise à jour de la base de données.

L'algorithme est :

Pour i = 1..m

Si $U(P_i) = L_1 \& \ldots \& L_q$

Pour j = 1..q

(17) dP(x) <- $L_1 \& \ldots \& L_{j-1} \& [dQ_j(x_j) | iQ_j(x_j)] \& L_{j+1} \& \ldots \& L_q \& E(P_i) \&$

$\neg$P'$_1$(x) & ... & $\neg$P'$_{i-1}$(x) & $\neg$P'$_{i+1}$(x) & ... & $\neg$P'm(x)

où dQ$_j(x_j)$ est choisi si L_j est positif et iQ$_j(x_j)$ si L_j est négatif ;

FinPour

FinSi

Si $E(P_i) = L_{q+1} \& \ldots \& L_n$

Pour j = q+1..n

(18) dP(x) <- $U(P_i) \& L_{q+1} \& \ldots \& L_{j-1} \& [dQ_j(x_j) | iQ_j(x_j)] \& L_{j+1} \& \ldots \& L_n \&$

$\neg$E(P'$_{i,1}$) & ... & $\neg$E(P'$_{i,2}$k) & $\neg$P'$_1$(x) & ... & $\neg$P'$_{i-1}$(x) &

$\neg$P'$_{i+1}$(x) & ... & $\neg$P'$_m$(x)

où dQ$_j(x_j)$ est choisi si L_j est positif et iQ$_j(x_j)$ si L_j est négatif ;

FinPour

FinSi

FinPour

Les règles 17 et 18 sont appelées règles d'événements internes de suppression du prédicat P. Elles permettent de savoir quels faits dP, suppressions induites, ont lieu lors d'une transition.

Cet algorithme s'explique intuitivement ainsi :

m règles définissent le prédicat P. Chacune d'entre elles, de par le contenu de son corps, donne plusieurs possibilités de supprimer P. Par souci d'efficacité, seules les solutions minimales sont retenues ; il est clair que toute combinaison de situations expliquées ci-dessous conduirait également à la suppression de P.

Tour à tour, chacune des règles définissant P est considérée. Soit P_i, une de ces règles. Parmi l'ensemble des littéraux dont elle est composée, on peut distinguer les littéraux universels d'une part et les littéraux existentiels d'autre part.

Soit Lu, un littéral universel quelconque figurant dans le corps de P_i. Si Lu est positif (négatif), il suffit alors de le nier (l'affirmer) pour que la condition ne soit plus satisfaite et donc P_i n'est plus vérifié. Ainsi, dans la règle de déduction

chef_potentiel(X) <- diplôme(X) & travail(X,Y,T) & T > 5, diplôme(X) est un littéral universel, il suffit par conséquent de le nier pour que la preuve de chef_potentiel(X) soit impossible.

Pour un littéral existentiel, la situation est légèrement différente. Procéder comme pour un littéral universel ne suffit pas. En effet, si dans l'exemple chef_potentiel(X) <- diplôme(X) & travail(X,Y,T) & T > 5, on se contente de nier travail(X,Y,T), rien ne dit que X ne travaille pas aussi en Z depuis une durée T1 par exemple et que T1 est également supérieur à 5 ans. On aurait alors chef_potentiel(X) <- diplôme(X) & travail(X,Z,T1) & T1 > 5 et la condition est toujours bien vérifiée. Cette situation ne peut être évitée que si l'on nie les parties existentielles de toutes les règles de transition de P_i. Enfin, il est évidemment nécessaire de nier toutes les règles définissant P ("... $\neg$P'1(x) & ... & $\neg$P'i-1(x) & $\neg$P'i+1(x) & ... & $\neg$P'm(x)") car il suffit qu'une seule soit satisfaite pour que P soit prouvable. Par exemple, si la règle de déduction chef_potentiel est définie par chef_potentiel(X) <- diplôme(X) & travail(X,Y,T) & T > 5 et par chef_potentiel(X) <- fils_patron(X), rien ne sert d'assurer l'échec de la première si l'on ne peut également assurer l'échec de la seconde.

L'exemple ci-dessous donne les règles d'événements internes de suppression associées aux règles de déduction données dans l'exemple présenté en tête de chapitre.

```
resp(X,D) <- trav(X,D) & chef(X).
sup(X,Y) <- resp(X,D) & trav(Y,D) & X <> Y.
sup(X,Y) <- sup(X,Z) & sup(Z,Y).
```

$$\delta resp(X,D) \leftarrow \delta trav(X,D) \,\&\, chef(X).$$
$$\delta resp(X,D) \leftarrow trav(X,D) \,\&\, \delta chef(X).$$

$$\delta sup(X,Y) \leftarrow \delta resp(X,D) \,\&\, trav(Y,D) \,\&\, \neg sup'_{1,1}(X,Y) \,\&\, \neg sup'_{1,2}(X,Y) \,\&\, \neg sup'_{1,3}(X,Y) \,\&\, \neg sup'_{1,4}(X,Y) \,\&\, \neg sup'_{2}(X).$$

$$\delta sup(X,Y) \leftarrow resp(X,D) \,\&\, \delta trav(Y,D) \,\&\, \neg sup'_{1,1}(X,Y) \,\&\, \neg sup'_{1,2}(X,Y) \,\&\, \neg sup'_{1,3}(X,Y) \,\&\, \neg sup'_{1,4}(X,Y) \,\&\, \neg sup'_{2}(X).$$

$$\delta sup(X,Y) \leftarrow \delta sup(X,Z) \,\&\, sup(Z,Y) \,\&\, \neg sup'_{2,1}(X,Y) \,\&\, \neg sup'_{2,2}(X,Y) \,\&\, \neg sup'_{2,3}(X,Y) \,\&\, \neg sup'_{2,4}(X,Y) \,\&\, \neg sup'_{1}(X).$$

$$\delta sup(X,Y) \leftarrow sup(X,Z) \,\&\, \delta sup(Z,Y) \,\&\, \neg sup'_{2,1}(X,Y) \,\&\, \neg sup'_{2,2}(X,Y) \,\&\, \neg sup'_{2,3}(X,Y) \,\&\, \neg sup'_{2,4}(X,Y) \,\&\, \neg sup'_{1}(X).$$

L'utilisation du concept d'événement interne nous permet de savoir comment et quand une occurrence d'un prédicat dérivé peut être insérée ou supprimée. Cette propriété est fort importante, comme nous allons le voir dans les deux points suivants, relatifs aux contraintes d'intégrité et aux mises à jour de vues.

IV.8. La méthode des événements internes pour le contrôle d'intégrité dans les bases de données déductives

La méthode des événements internes est applicable aussi bien pour les contraintes d'intégrité statiques que pour les contraintes d'intégrité dynamiques (de transition). L'approche est uniforme, les 2 types de contraintes (statiques et dynamiques) peuvent y être définis et vérifiés.

IV.8.1. Gestion des contraintes d'intégrité statiques

Pour rappel, une contrainte d'intégrité statique est une formule du premier ordre fermée Ic <- L1 & ... & Ln avec n>0, où les Li sont des littéraux dont les variables sont supposées universellement quantifiées sur toute la formule. Dans le but d'assurer l'uniformité d'expression des règles de déduction et des contraintes d'intégrité, à chaque contrainte d'intégrité est associé un **prédicat d'inconsistance Icn**, avec ou sans terme. Les règles associées aux prédicats d'inconsistance sont appelées **règles d'intégrité**.

L'objectif étant de gérer les contraintes d'intégrité comme les règles de déduction, les règles de transition sont également définies, suivant le procédé "classique" exposé précédemment (point IV.5.).

En ce qui concerne les règles d'événements internes d'insertion, de forme iP(x) <- P'i,j(x) & ¬P(x) pour i=1..m et j=2..2^k, lorsque P est un prédicat d'inconsistance, elles peuvent être légèrement simplifiées. La condition ¬P(x) peut en effet être retirée puisque la base de données est supposée cohérente avant mise à jour : ¬P(x) est vérifié dans tous les cas. Si iP(x) a lieu lors d'une transition, cela signifie qu'une occurrence du prédicat d'inconsistance P(x) est insérée dans la base de données et une contrainte d'intégrité est violée.

Corollairement, les règles d'événements internes de suppression n'ont pas de sens pour les contraintes d'intégrité. En effet, la suppression d'un prédicat P n'est concevable que si ce prédicat est vrai dans l'état avant mise à jour, c'est-à-dire si P(x) est prouvable. Or, si P correspond à un prédicat d'inconsistance, il ne peut être vrai dans l'état avant mise à jour puisque cela signifierait que la base de données était incohérente ; définir des règles de suppression de prédicats d'inconsistance n'est par conséquent pas

pertinent.

Les règles données ci-dessous sont à nouveau reprises de la base de données qui figure en début de chapitre.

Exemple :

Les contraintes d'intégrité de la base de données servant d'exemple sont :

Ic1 <- resp(X1,D) & resp(X2,D) & X1 <> X2.
Ic2 <- sup(X,X).

Grâce aux procédés expliqués tout au long de ce chapitre, on peut déterminer les règles de transition et d'événements internes correspondantes. Les quatres premières règles sont les règles de transition de la première contrainte d'intégrité, les deux suivantes sont celles de la seconde contrainte. Enfin, les deux dernières lignes donnent les quatres règles d'événements internes d'insertion associées aux contraintes d'intégrite.

$Ic'_{1,1}$ <- resp(X1,D) & ¬*d*resp(X1,D) & resp(X2,D) & ¬*d*resp(X2,D) & X1 <> X2.
$Ic'_{1,2}$ <- resp(X1,D) & ¬*d*resp(X1,D) & *i*resp(X2,D) & X1 <> X2.
$Ic'_{1,3}$ <- *i*resp(X1,D) & resp(X2,D) & ¬*d*resp(X2,D) & X1 <> X2.
$Ic'_{1,4}$ <- *i*resp(X1,D) & *i*resp(X2,D) & X1 <> X2.

$Ic'_{2,1}$ <- sup(X,X) & ¬*d*sup(X,X).
$Ic'_{2,2}$ <- *i*sup(X,X).

$\textit{i}c_1$ <- $Ic'_{1,j}$; j=2..4
$\textit{i}c_2$ <- $Ic'_{2,2}$.

Ces règles de transition et d'événements internes d'un prédicat d'inconsistance peuvent être utilisées directement pour vérifier qu'une mise à jour ne produit pas d'incohérence. Supposons une mise à jour U constituée d'un ensemble d'événements internes de base. Si U mène à une incohérence, alors certaines règles d'insertion de prédicats d'inconsistance sont vérifiées lors de la transition. Par la résolution SLDNF utilisée dans l'interpréteur PROLOG, on montre que U viole la contrainte d'intégrité Icj si l'objectif '<-*i*Icj' est prouvable à partir de U et de la base de données D augmentée (A(D)), c'est-à-dire à laquelle sont ajoutées les règles de transition et d'événements internes associées à ses règles de déduction et à ses contraintes d'intégrité. Si '<-*i*Icj'

n'est pas prouvable, alors la mise à jour U ne viole pas la contrainte Icj. La base de données n'est mise à jour qu'après vérification du respect de toutes les contraintes.

Différentes sortes de mises à jour sont possibles, elles peuvent être regroupées en cinq classes :

- insertion et suppression de faits de base,
- insertion et suppression de faits de base qualifiés,
- insertion et suppression de règles de déduction,
- insertion et suppression de contraintes d'intégrité,
- mises à jour multiples.

Les 5 points suivants sont consacrés à une étude plus détaillée de chacune de ces classes.

IV.8.1.1. Insertion et suppression d'un fait de base

La mise à jour consiste en une insertion ou une suppression d'un fait de base, dénotée par U={*i*Q(K)} ou U={*d*Q(K)} respectivement, où Q est un prédicat de base et K un vecteur de constantes. La mise à jour est simple à réaliser, ce qui nous intéresse, c'est de savoir si elle provoque une violation de contrainte. L'exemple ci-dessous explique le procédé. Une seule branche de l'arbre de résolution, choisie pour les besoins de l'illustration, y est développée. Dans un cas réel, toutes les possibilités sont évidemment étudiées.

Exemple :

La base de données servant d'exemple contient déjà :

```
trav( Smits,Marketing ).
trav( Dupuis,Vente ).
trav( Delcours,Vente ).
chef( Dupuis ).
```

Soit l'insertion du fait chef(Delcours),
U = {*i*chef(Delcours)}.

<- *i*c1.

<- $ic'_{1,2}$.

<- resp(X1,D) & ¬*d*resp(X1,D) & *i*resp(X2,D) & X1 <> X2.

<- trav(X1,D) & chef(X1) & ¬*d*resp(X1,D) & *i*resp(X2,D) & X1 <> X2.

<- trav(X1,D) & chef(X1) & ¬*d*resp(X1,D) & trav(X2,D) & ¬*d*trav(X2,D) & *i*chef(X2)
& X1 <> X2.
(X2 = Delcours, D = Vente)

<- trav(X1,Vente) & chef(X1) & ¬*d*resp(X1,Vente) & trav(Delcours,Vente) &
¬*d*trav(Delcours,Vente) & *i*chef(Delcours) & X1 <> Delcours.

<- trav(X1,Vente) & chef(X1) & ¬*d*resp(X1,Vente).
(X1 = Dupuis)

<- trav(Dupuis,Vente) & chef(Dupuis) & ¬*d*resp(Dupuis,Vente).

<- ¬*d*resp(Dupuis,Vente).

L'objectif final correspond à un événement interne dérivé négatif, il est impossible de connaître directement sa valeur de vérité. Par contre, on sait qu'il est prouvable si et seulement si *d*resp(Dupuis,Vente) ne l'est pas. Le nouvel objectif, développé dans l'arbre auxiliaire ci-dessous, est par conséquent *d*resp(Dupuis,Vente).

Arbre auxiliaire :

<- *d*resp(Dupuis,Vente).

PREMIERE BRANCHE

<- *d*travail(Dupuis,Vente) & chef(Dupuis).

Echec.

<- *d*resp(Dupuis,Vente).

DEUXIEME BRANCHE

<- travail(Dupuis,Vente) & *d*chef(Dupuis).

Echec.

Dans les deux branches de l'arbre auxiliaire, l'échec est immédiat. Dans la première, on ne supprime pas le fait que Dupuis travaille dans le département Vente, ce qui provoque l'échec de l'événement interne de base *d*travail(Dupuis,Vente). Dans la seconde, c'est l'événement interne de base *d*chef(Dupuis) qui échoue puisque l'on ne supprime pas non plus le fait que Dupuis soit un chef. Les deux branches de l'arbre auxiliaire mènent donc à un échec, ce qui entraîne une réussite de l'arbre principal.

En conclusion, on voit que l'insertion du fait chef(Delcours) induit l'insertion du prédicat d'inconsistance correspondant à la première contrainte d'intégrité, à savoir : si l'on insère chef(Delcours), alors on aura deux responsables pour un même département, ce qui est interdit.

On notera que dans la majorité des bases de données réelles, le nombre de faits est normalement beaucoup plus grand que le nombre d'événements internes produits par une transition ; il est par conséquent préférable de sélectionner en premier les événements internes, une fois complètement instanciés s'ils sont négatifs. Cette dernière restriction est une condition d'applicabilité de la méthode de résolution SLDNF, des détails peuvent être trouvés dans [Caved-Deck90] ; elle est intuitivement compréhensible si l'on recourt à un exemple. Ainsi, essayer de prouver un prédicat ¬trav(X,Vente) dans la base de données servant d'illustration est impossible dans la mesure où la valeur de la variable X n'est pas connue.

IV.8.1.2. Insertion et suppression de faits de base qualifiés

Ce genre de mise à jour ajoute (ou supprime) de la base de données un ensemble de faits de base qui satisfont une condition donnée. On l'exprime par une règle *i*P(x) <- L1 & ... & Ln, où P est un prédicat de base et L1,...,Ln sont des conditions qui définissent l'ensemble des faits P à insérer. L'exemple suivant montre que ces mises à jour peuvent être vérifiées comme dans le cas simple.

Exemple :

Soit la mise à jour U = {*d*trav(X,Vente) <- trav(X,Vente)}. Le procédé est semblable à celui illustré dans le point A. Tous les espaces de recherche ayant comme clause de départ { <-*i*Ic1 } et { <-*i*Ic2 } échouent, par conséquent tous les faits trav(X,Vente) peuvent être supprimés.

IV.8.1.3. Insertion et suppression de règles de déduction

Supposons maintenant que la mise à jour consiste en l'insertion d'une nouvelle règle de déduction : P(x) <- L1 & ... & Ln. Suite à l'insertion de cette nouvelle règle, la base de données mise à jour va probablement contenir de nouveaux faits, implicites, qui pourraient violer certaines contraintes. De telles violations peuvent aussi être détectées par la méthode des événements internes. La seule nécessité est de déterminer quels faits *i*P sont produits lors de la transition. Une fois ces faits connus, les contraintes d'intégrité peuvent être analysées comme dans le cas précédent. Si toutes les contraintes d'intégrité sont satisfaites, la nouvelle règle est acceptée, la base de données mise à jour et les règles de transition et d'événements internes modifiées selon besoins.

Si P est un nouveau prédicat dans la base de données, les événements internes d'insertion produits par la mise à jour sont donnés par *i*P(x) <-> P'(x). En transformant P'(x) en son ensemble équivalent de règles, on obtient directement les règles d'insertion.

Si P est un prédicat existant, il est renommé par P_k dans la conclusion de la règle de déduction, où :

k = 1 si P est un prédicat de base
k = m+1 si P est un prédicat dérivé à m règles.

Les faits *i*P produits par la mise à jour sont donnés par :

$$iP(x) \leftarrow P'_k(x) \;\&\; \neg P(x)$$

et transformant $P'_k(x)$ en son ensemble équivalent de règles de transition, on obtient les règles d'insertion.

Si une mise à jour supprime une règle de déduction existante, on procède de manière similaire, mais ce sont évidemment les règles de suppression qui sont dérivées.

Exemple :

Supposons que la base de données contienne les faits de base supplémentaires :

```
engage( Smits, '10/08/90' ).
engage( Dupuis, '02/06/89' ).
engage( Delcours, '01/01/88' ).
engage( Roffin, '25/01/91' ).
```

Et soit la nouvelle règle de déduction :

```
ancien(X) <- engage(X,Date) & Date < '01/01/85'.
```

Les règles de transition sont :

$ancien'_{1,1}(X)$ <- engage(X,Date) & ¬*d*engage(X,Date) & Date < '01/01/85'.
$ancien'_{1,2}(X)$ <- *i*engage(X,Date) & Date<'01/01/85'.

Les contraintes d'intégrité étant totalement indépendantes de ce nouveau prédicat, tous les arbres de résolution de but " <-*i*Icj" échouent et la base de données reste parfaitement cohérente.

IV.8.1.4. Insertion et suppression de contraintes d'intégrité

Les contraintes d'intégrité sont traitées comme des règles de déduction classiques, le procédé est donc similaire à celui expliqué ci-dessus à l'exception d'une simplification dans le cas d'une suppression.

Il est clair en effet que la suppression d'une contrainte d'intégrité ne peut causer une incohérence et la vérification des contraintes d'intégrité pour une mise à jour qui supprime une contrainte n'est pas nécessaire.

Exemple :

La base de données actuelle est :

```
trav( Smits,Marketing ).
trav( Dupuis,Vente ).
trav( Delcours,Vente ).
chef( Dupuis ).
engage( Smits, '10/08/90' ).
engage( Dupuis, '02/06/89' ).
engage( Delcours, '01/01/88' ).
engage( Roffin, '25/01/91' ).

Ic1 <- resp(X1,D) & resp(X2,D) & X1 <> X2.
Ic2 <- sup(X,X).

resp(X,D) <- trav(X,D) & chef(X).
sup(X,Y) <- resp(X,D) & trav(Y,D) & X <> Y.
sup(X,Y) <- sup(X,Z) & sup(Z,Y).
ancien(X) <- engage(X,Date) & Date<'01/01/85'.
```

Dans cette base de données, soit la nouvelle contrainte Ic3 <- engage(X,Date) & ¬trav(X,D) signifiant que toute personne engagée travaille obligatoirement dans un département.

Les règles de transition et d'événements internes d'insertion sont :

```
Ic'3,1 <- engage(X,Date) & ¬dengage(X,Date) & ¬trav(X,D) & ¬itrav(X,D).
Ic'3,2 <- engage(X,Date) & ¬dengage(X,Date) & dtrav(X,D).
Ic'3,3 <- iengage(X,Date) & ¬trav(X,D) & ¬itrav(X,D).
Ic'3,4 <- iengage(X,Date) & dtrav(X,D).

iIc3 <- Ic'3,j ; j=1..4 (car il s'agit d'une nouvelle contrainte).
```

L'arbre de résolution suivant montre que la base de données actuelle viole cette contrainte.

<- Ic3

<- $\imath$Ic'$_{3,1}$

<- engage(X,Date) & ¬δengage(X,Date) & ¬trav(X,D) & ¬$\imath$trav(X,D).

<- engage(X,Date) & ¬trav(X,D).
(X = Roffin ; Date = '25/01/91'))

<- ¬trav(Roffin,D).
[]

Cet objectif mène à une réussite car Roffin ne travaille dans aucun département. L'insertion de Ic3 entraîne une incohérence de la base de données.

IV.8.1.5. Mises à jour multiples

Lorsque plusieurs mises à jour doivent être satisfaites, on détermine premièrement les événements internes produits par chaque mise à jour, ensuite on analyse la satisfaction des contraintes d'intégrité. Si l'ensemble des mises à jour est accepté, la base de données et, éventuellement, les règles de transition et d'événements internes sont mises à jour.

Exemple :

La base de données est :

```
trav( Smits,Marketing ).
trav( Dupuis,Vente ).
trav( Delcours,Vente ).
chef( Dupuis ).
engage( Smits, '10/08/90' ).
engage( Dupuis, '02/06/89' ).
engage( Delcours, '01/01/88' ).
```

```
engage( Roffin, '25/01/91' ).

Ic1 <- resp(X1,D) & resp(X2,D) & X1 <> X2.
Ic2 <- sup(X,X).
Ic3 <- engage(X,Date) & ¬trav(X,D).

resp(X,D) <- trav(X,D) & chef(X).
sup(X,Y) <- resp(X,D) & trav(Y,D) & X <> Y.
sup(X,Y) <- sup(X,Z) & sup(Z,Y).
ancien(X) <- engage(X,Date) & Date<'01/01/85'.
```

Dans cette base de données, soit l'ensemble de mises à jour suivant :

```
E : { Insert dept( Marketing ),
      Insert dept( Vente ),
      Insert dept( Comptabilité ),
      Insert Ic4 <- trav(X,D) & ¬dept(D) }
```

Les 3 premières insertions sont des faits de base, 'dept' représente 'département'. La quatrième insertion ajoute une nouvelle contrainte d'intégrité : *"une personne ne peut travailler que dans un département présent dans la base de données."*

Les événements internes correspondant à cet ensemble de mises à jour sont :

```
idept(Marketing).
idept(Vente).
idept(Comptabilité).
iIc4 <- trav(X,D) & ¬dtrav(X,D) & ¬dept(D) & ¬idept(D).
iIc4 <- trav(X,D) & ¬dtrav(X,D) & ddept(D).
iIc4 <- itrav(X,D) & ¬dept(D) & ¬idept(D).
iIc4 <- itrav(X,D) & ddept(D).
```

Toutes les tentatives de preuve ayant comme clause de départ { <- *i*Icj } ; j = 1..4, échouent. Seules celles correspondant à la nouvelle contrainte sont développées ci-dessous.

Examinons en détail le développement d'une de ces branches :

<- *i*c4

PREMIERE BRANCHE

<- trav(X,D) & ¬*d*trav(X,D) & ¬dept(D) & ¬*i*dept(D).

(X = Smits, D = Marketing)

<- ¬*d*trav(Smits,Marketing) & ¬dept(Marketing) & ¬*i*dept(Marketing).

Echec

On constate que l'objectif aboutit à un échec car Insert dept(Marketing) figure dans la transaction et donc ¬*i*dept(Marketing) échoue.

Le principe est identique dans les 3 autres cas. L'événement interne se rapportant au département échoue chaque fois.

L'étude des différents types de mises à jour a montré la parfaite applicabilité de la méthode des événements internes pour le maintien de la cohérence d'une base de données lorsque l'on se limite aux contraintes d'intégrité statiques, le point suivant a pour but de montrer que le principe peut sans problème être étendu aux contraintes d'intégrité de transition.

IV.8.2. Gestion des contraintes d'intégrité de transition

Jusqu'à présent, seules des contraintes d'intégrité statiques ont été traitées. La méthode des événements internes peut cependant aussi gérer les contraintes d'intégrité de transition. Rappelons qu'une contrainte d'intégrité de transition est une formule du premier ordre fermée que les transitions de la base de données doivent satisfaire. On ne considère que les transitions entre 2 états successifs de la base de données.

Dans la méthode des événements internes, les mises à jour d'un prédicat P sont représentées explicitement au moyen des prédicats *i*P et *d*P. Ceci suppose que, comme pour une contrainte d'intégrité statique, une contrainte d'intégrité de transition est une

condition Tic <- L1 & ... & Ln ; n > 0 où les littéraux peuvent être soit des prédicats de base soit des prédicats d'événements internes. Les prédicats de base figurant dans la condition doivent être évalués dans l'état courant de la base de données, et pas dans l'état résultant de la mise à jour. Les contraintes d'intégrité de transition ne doivent donc pas être transformées, contrairement aux contraintes d'intégrité statiques, et les prédicats associés à des contraintes d'intégrité de transition ont la nature d'un événement interne.

En conséquence, une contrainte d'intégrité de transition Ticn est une règle dont la forme générale est :

*i*Ticn <- L1 & ... & Ln avec n>0

Exemple :

Considérons une contrainte d'intégrité de transition stipulant qu'une personne ne peut changer de statut et de salaire que si ce changement correspond à une promotion valable, connue. Exprimée sous forme de règle, cette restriction donne :

*i*Tic1 <- stat(P,St,Sal) & *d*stat(P,St,Sal) & *i*stat(P,St',Sal') & ¬promotion(St,Sal,St',Sal').

Il faut remarquer que dans la méthode, une mise à jour est composée d'une suppression et d'une insertion dans la même transition. Cependant, étant donné l'implication *d*P(x) -> P(x) (point IV.7.), on peut retirer le premier littéral dans la règle ci-dessus et la règle devient :

*i*Tic1 <- *d*stat(P,St,Sal) & *i*stat(P,St',Sal') & ¬promotion(St,Sal,St',Sal').

On pourrait également imaginer une seconde règle empêchant la suppression de toute personne engagée avant une date donnée...

*i*Tic2 <- *d*engage(X,Date) & Date < '01/01/76'.

La vérification des contraintes d'intégrité de transition n'est pas différente de celle des contraintes d'intégrité statiques. Une mise à jour U violera une contrainte d'intégrité de transition Ticj si le but '<-*i*Ticj' est atteint à partir de l'ensemble de départ composé de la base de données D augmentée et de U. Si chaque branche de l'arbre de recherche SLDNF est une branche d'échec, alors U ne viole pas la contrainte

de transition Ticj.

Jusqu'à présent, nous avons étudié comment la méthode des événements internes peut être utilisée pour résoudre les problèmes des contraintes d'intégrité, le reste de ce chapitre explique comment le problème des mises à jour de vues (de faits dérivés) peut également être résolu.

IV.9. La méthode des événements internes pour la mise à jour de vues dans les bases de données déductives

Comme nous l'avons vu précédemment, dans une base de données déductive, une vue est assimilée à une règle de déduction. Le problème de mise à jour de vues peut donc être reformulé ainsi : "comment insérer ou supprimer un fait dérivé dans la base de données ?". Partant d'une demande d'insertion (par exemple) d'un fait dérivé, l'objectif est de trouver une combinaison d'actions sur les faits de base ayant le même effet sur la base de données que la demande initiale.

Tout comme pour le contrôle d'intégrité, ce sont les règles de transition et d'événements internes qui sont utilisées pour transformer les demandes de mise à jour de vues en une combinaison de mises à jour élémentaires. La compréhension du processus étant nettement plus aisée lorsqu'il est accompagné d'un exemple, de nombreuses illustrations seront données à partir de la base de données utilisée tout au long de ce chapitre. Pour éviter toute confusion, voici son contenu suite à toutes les mises à jour effectuées dans les paragraphes précédents.

Faits de base

```
trav( Smits,Marketing ).
trav( Dupuis,Vente ).
trav( Delcours,Vente ).
trav( Roffin,Marketing ).
chef( Dupuis ).
engage( Smits, '10/08/90' ).
engage( Dupuis, '02/06/89' ).
engage( Delcours, '01/01/88' ).
engage( Roffin, '25/01/91' ).
dept( Marketing ).
```

dept(Vente).

dept(Comptabilité).

Contraintes d'intégrité

Ic1 <- resp(X1,D) & resp(X2,D) & X1<>X2.

Ic2 <- sup(X,X).

Ic3 <- engage(X,Date) & ¬trav(X,D).

Ic4 <- trav(X,D) & ¬dept(D).

Règles de déduction

resp(X,D) <- trav(X,D) & chef(X).

sup(X,Y) <- resp(X,D) & trav(Y,D) & X <> Y.

sup(X,Y) <- sup(X,Z) & sup(Z,Y).

ancien(X) <- engage(X,Date) & Date < '01/01/85'.

Règles de transition

resp'(X,D) <- $resp,_1$(X,D).

$resp'_1$(X,D) <- $resp'_{1,j}$(X,D) ; j = 1..4.

sup'(X,Y) <- sup'_1(X,Y).

sup'(X,Y) <- sup'_2(X,Y).

sup'_1(X,Y) <- $sup'_{1,j}$(X,Y) ; j=1..4.

sup'_2(X,Y) <- $sup'_{2,j}$(X,Y), j=1..4.

ancien'(X) <- $ancien'_1$(X).

$ancien'_1$(X) <- $ancien'_{1,j}$(X) ; j=1..2.

Ic' <- Ic'1.

Ic' <- Ic'2.

Ic' <- Ic'3.

Ic' <- Ic'4.

Ic'1 <- $Ic'_{1,j}$; j = 1..4.

Ic'2 <- $Ic'_{2,j}$; j = 1..2.

Ic'3 <- $Ic'_{3,j}$; j = 1..4.

Ic'4 <- $Ic'_{4,j}$; j = 1..4.

$resp'_{1,1}(X,D)$ <- trav(X,D) & ¬*d*trav(X,D) & chef(X) & ¬*d*chef(X).

$resp'_{1,2}(X,D)$ <- trav(X,D) & ¬*d*trav(X,D) & *i*chef(X).

$resp'_{1,3}(X,D)$ <- *i*trav(X,D) & chef(X) & ¬*d*chef(X).

$resp'_{1,4}(X,D)$ <- *i*trav(X,D) & *i*chef(X).

$sup'_{1,1}(X,Y)$ <- resp(X,D) & ¬*d*resp(X,D) & trav(Y,D) & ¬*d*trav(Y,D) & X <> Y.

$sup'_{1,2}(X,Y)$ <- resp(X,D) & ¬*d*resp(X,D) & *i*trav(Y,D) & X <> Y.

$sup'_{1,3}(X,Y)$ <- *i*resp(X,D) & trav(Y,D) & ¬*d*trav(Y,D) & X <> Y.

$sup'_{1,4}(X,Y)$ <- *i*resp(X,D) & *i*trav(Y,D) & X <> Y.

$sup'_{2,1}(X,Y)$ <- sup(X,Z) & ¬*d*sup(X,Z) & sup(Z,Y) & ¬*d*sup(Z,Y).

$sup'_{2,2}(X,Y)$ <- sup(X,Z) & ¬*d*sup(X,Z) & *i*sup(Z,Y).

$sup'_{2,3}(X,Y)$ <- *i*sup(X,Z) & sup(Z,Y) & ¬*d*sup(Z,Y).

$sup'_{2,4}(X,Y)$ <- *i*sup(X,Z) & *i*sup(Z,Y).

$ancien'_{1,1}(X)$ <- engage(X,Date) & ¬*d*engage(X,Date) & Date < '01/01/85'.

$ancien'_{1,2}(X)$ <- *i*engage(X,Date) & Date<'01/01/85'.

$Ic'_{1,1}$ <- resp(X1,D) & ¬*d*resp(X1,D) & resp(X2,D) & ¬*d*resp(X2,D) & X1<>X2.

$Ic'_{1,2}$ <- resp(X1,D) & ¬*d*resp(X1,D) & *i*resp(X2,D) & X1<>X2.

$Ic'_{1,3}$ <- *i*resp(X1,D) & resp(X2,D) & ¬*d*resp(X2,D) & X1<>X2.

$Ic'_{1,4}$ <- *i*resp(X1,D) & *i*resp(X2,D) & X1<>X2.

$Ic'_{2,1}$ <- sup(X,X) & ¬*d*sup(X,X).

$Ic'_{2,2}$ <- *i*sup(X,X).

$Ic'_{3,1}$ <- engage(X,Date) & ¬*d*engage(X,Date) & ¬trav(X,D) & ¬*i*trav(X,D).

$Ic'_{3,2}$ <- engage(X,Date) & ¬*d*engage(X,Date) & *d*trav(X,D).

$Ic'_{3,3}$ <- *i*engage(X,Date) & ¬trav(X,D) & ¬*i*trav(X,D).

$Ic'_{3,4}$ <- *i*engage(X,Date) & *d*trav(X,D).

$Ic'_{4,1}$ <- trav(X,D) & ¬*d*trav(X,D) & ¬dept(D) & ¬*i*dept(D).

$Ic'_{4,2}$ <- trav(X,D) & ¬*d*trav(X,D) & *d*dept(D).

$Ic'_{4,3}$ <- *i*trav(X,D) & ¬dept(D) & ¬*i*dept(D).

$Ic'_{4,4}$ <- *i*trav(X,D) & *d*dept(D).

Règles d'événements internes d'insertion

$i$resp(X,D) <- resp'$_{1,2}$(X,D) & ¬resp(X,D).
$i$resp(X,D) <- resp'$_{1,3}$(X,D) & ¬resp(X,D).
$i$resp(X,D) <- resp'$_{1,4}$(X,D) & ¬resp(X,D).

$i$sup(X,Y) <- sup'$_{1,2}$(X,Y) & ¬sup(X,Y).
$i$sup(X,Y) <- sup'$_{1,3}$(X,Y) & ¬sup(X,Y).
$i$sup(X,Y) <- sup'$_{1,4}$(X,Y) & ¬sup(X,Y).

$i$sup(X,Y) <- sup'$_{2,2}$(X,Y) & ¬sup(X,Y).
$i$sup(X,Y) <- sup'$_{2,3}$(X,Y) & ¬sup(X,Y).
$i$sup(X,Y) <- sup'$_{2,4}$(X,Y) & ¬sup(X,Y).

$i$ancien(X) <- ancien'$_{1,2}$(X) & ¬ancien(X).

$i$lc1 <- lc'$_{1,j}$; j=2..4
$i$lc2 <- lc'$_{2,2}$.
$i$lc3 <- lc'$_{3,j}$; j=2..4
$i$lc4 <- lc'$_{4,j}$; j=2..4

Règles d'événements internes de suppression

$d$resp(X,D) <- $d$trav(X,D) & chef(X).
$d$resp(X,D) <- trav(X,D) & $d$chef(X).

$d$sup(X,Y) <- $d$resp(X,D) & trav(Y,D) & ¬sup'$_{1,1}$(X,Y) & ¬sup'$_{1,2}$(X,Y) & ¬sup'$_{1,3}$(X,Y) & ¬sup'$_{1,4}$(X,Y) & ¬sup'$_{2}$(X).

$d$sup(X,Y) <- resp(X,D) & $d$trav(Y,D) & ¬sup'$_{1,1}$(X,Y) & ¬sup'$_{1,2}$(X,Y) & ¬sup'$_{1,3}$(X,Y) & ¬sup'$_{1,4}$(X,Y) & ¬sup'$_{2}$(X).

$d$sup(X,Y) <- $d$sup(X,Z) & sup(Z,Y) & ¬sup'$_{2,1}$(X,Y) & ¬sup'$_{2,2}$(X,Y) & ¬sup'$_{2,3}$(X,Y) & ¬sup'$_{2,4}$(X,Y) & ¬sup'$_{1}$(X).

$d$sup(X,Y) <- sup(X,Z) & $d$sup(Z,Y) & ¬sup'$_{2,1}$(X,Y) & ¬sup'$_{2,2}$(X,Y) & ¬sup'$_{2,3}$(X,Y) & ¬sup'$_{2,4}$(X,Y) & ¬sup'$_{1}$(X).

*d*ancien(X) <- *d*engage(X,Date) & Date < '01/01/85'.

Voyons à présent comment, à l'aide des règles d'insertion et de suppression dont on dispose, il est possible de trouver une combinaison d'actions sur les faits de base qui permette de satisfaire une demande d'insertion et/ou de suppression de faits dérivés.

IV.9.1. Présentation et définitions des concepts utilisés

Une demande de mise à jour M d'un prédicat dérivé P est soit une demande d'insertion, soit une demande de suppression d'un fait P(K).

Par définition, une **translation** de M, notée T, est un ensemble d'insertions et/ou de suppressions de faits de base tels que si M est une demande d'insertion (resp. de suppression), P'(K) est (resp. n'est pas) une conséquence logique de l'exécution de la mise à jour de la base de données en fonction de T.

La translation T à elle seule ne suffit cependant pas à satisfaire la demande de mise à jour. En effet, elle représente uniquement les opérations qui doivent être effectuées pour satisfaire la demande de mise à jour ; ces opérations sont nécessaires mais pas suffisantes : il faut également préciser les opérations qui ne peuvent pas être effectuées dans le cadre de la mise à jour. On définit dès lors un nouvel ensemble, celui des conditions, noté C.

Par définition, C est l'ensemble des insertions et/ou suppressions de faits de base qui ne peuvent pas être exécutées pour que la demande de mise à jour soit satisfaite.

Comment obtenir les ensembles T et C ?

IV.9.2. Procédé a suivre

IV.9.2.1. Explication

Les éléments dont on dispose sont A(D), la base de données D augmentée, et M, la mise à jour exprimée sous forme d'un prédicat d'événement interne *i*P(K) (ou *d*P(K)). Une autre formulation du problème pourrait être : comment prouver (au sens théorie formelle) *i*P(K) à partir de A(D) et en ayant recours à la méthode de résolution SLDNF ?

Intuitivement, le raisonnement est le suivant. L'objectif de départ est {<- *i*P(K)}. Cet objectif correspond à une règle de déduction de A(D), il peut donc être affiné selon sa définition en différentes branches, chacune ayant la forme d'une conjonction de littéraux L1,L2,...,Ln plus élémentaires. Parmi ces littéraux, différents cas de figure sont possibles.

Soit Li un littéral.

Si Li correspond à un prédicat dérivé, il est 'remplacé' par sa définition ; ce qui engendre éventuellement plusieurs sous-branches plus élémentaires.

A l'opposé, le cas le plus simple est celui dans lequel Li correspond à un prédicat de base (ou la négation d'un prédicat de base), il suffit alors de consulter la base de données pour pouvoir évaluer Li, s'il est faux, la branche peut être abandonnée.

Li peut également être un prédicat d'événement interne de base, par exemple *i*Q(x). Pour que la branche aboutisse à une réussite, il faut obligatoirement *i*Q(x), on l'insère donc dans T, conformément à la définition de cet ensemble. Par contre, si Li est négatif, donc de forme ¬*i*Q(x), cela signifie que *i*Q(x) ne peut absolument pas avoir lieu sous peine d'échec de la branche, il faut par conséquent l'insérer dans C.

Les différentes situations possibles, ainsi que l'ordre dans lequel les littéraux doivent être évalués, seront étudiés davantage par la suite mais on peut déjà déduire de l'exemple que la translation T est obtenue en forçant la réussite de certaines branches de l'arbre de dérivation qui échouaient initialement à partir de l'ensemble de départ composé de A(D) et de {<- M}. En d'autres termes, l'ensemble T doit être tel que {<- M} soit prouvable à partir de A(D) et de T.

L'ensemble des conditions C est obtenu de façon similaire mais il contient les événements internes de base négatifs sélectionnés durant la dérivation.

Généralement, la réussite de plusieurs branches du même arbre peut être obtenue ; plusieurs combinaisons différentes peuvent donc répondre à la demande de mise à jour ; elles sont toutes générées mais seules les solutions minimales sont retenues, c'est-à-dire celles dont aucun sous-ensemble n'est lui-même solution. Le problème du choix parmi les différentes solutions n'est pas étudié dans ce document, des explications peuvent être trouvées dans [Kakas-Manca90]

Les exemples ci-dessous illustrent l'approche, le premier correspond à une demande d'insertion d'un fait dérivé et le second à une demande de suppression d'un fait dérivé.

IV.9.2.2. Exemple d'insertion

Supposons que la demande de mise à jour soit l'insertion du fait dérivé resp(Smits,Marketing). Voici l'arbre obtenu, seules ses branches principales sont développées.

<- *i*resp(Smits,Marketing).

PREMIERE BRANCHE

<- resp'1,2(Smits,Marketing) & ¬resp(Smits,Marketing).

<- resp'1,2(Smits,Marketing).

<- trav(Smits,Marketing) & ¬*d*trav(Smits,Marketing) & *i*chef(Smits).

<- ¬*d*trav(Smits,Marketing) & *i*chef(Smits).

Doit réussir = >

T = {*i*chef(Smits)} C = {*d*trav(Smits,Marketing)}

<- *i*resp(Smits,Marketing).

DEUXIEME BRANCHE

<- resp'1,3(Smits,Marketing) & ¬resp(Smits,Marketing).

<- resp'1,3(Smits,Marketing).

<- *i*trav(Smits,Marketing) & chef(Smits) & ¬*d*chef(Smits).

ECHEC

<- *i*resp(Smits,Marketing).

TROISIEME BRANCHE

<- resp'1,4(Smits,Marketing) & ¬resp(Smits,Marketing).

<- resp'1,4(Smits,Marketing).

<- *i*trav(Smits,Marketing) & *i*chef(Smits).

ECHEC

La première branche atteint le but <- ¬*d*trav(Smits,Marketing) & *i*chef(Smits)., uniquement composé de prédicats d'événements internes de base. Ce but est satisfait si l'on suppose que l'événement *i*chef(Smits) a lieu durant la transition, ce qui équivaut à l'insérer dans l'ensemble de translation T. Par contre, l'événement *d*trav(Smits,Marketing) ne peut pas avoir lieu pour que la dérivation réussisse, il est donc inséré dans l'ensemble de conditions C.

De par les significations opposées des ensembles T et C, avant d'insérer un événement dans T (resp. dans C), il est nécessaire de vérifier qu'il n'appartient pas à la partie de C (resp. de T) déjà déterminée. Dans le cas contraire, il est impossible d'aboutir à une réussite de la branche courante car il y a contradiction entre les contenus de T et de C, aucune combinaison valide ne peut en être obtenue. Dans l'exemple, aucune contradiction n'est trouvée. Le résultat obtenu est T={*i*chef(Smits)}, il satisfait la demande de mise à jour de vue initiale *i*resp(Smits,Marketing).

Dans la seconde branche, on aboutit à l'objectif <- *i*trav(Smits,Marketing) & chef(Smits) & ¬*d*chef(Smits). Ce but échoue entre autres parce que le fait chef(Smits) n'est pas présent dans la base de données actuelle. Comme il n'est pas possible d'obtenir un succès à partir de cette branche, aucune translation ne peut en être obtenue.

De façon similaire, la troisième branche mène à un objectif impossible à satisfaire. En effet, dans <- *i*trav(Smits,Marketing) & *i*chef(Smits) le premier littéral ne peut être accepté dans notre base de données car le fait trav(Smits,Marketing) y est déjà présent.

IV.9.2.3. Exemple de suppression

Les demandes de suppression sont gérées de la même manière que les demandes d'insertion. Le processus est cependant légèrement plus difficile à comprendre car les règles de suppression d'événements internes ont une forme plus complexe.

Supposons que la demande de mise à jour soit la suppression de sup(Dupuis,Delcours). L'arbre ci-dessous illustre le procédé pour une des quatres règles de suppression de sup(Dupuis,Delcours).

<- *d*sup(Dupuis,Delcours).

<- *d*resp(Dupuis,D) & trav(Delcours,D) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_2$(Dupuis,Delcours).

{ D = Vente }

<- dresp(Dupuis,Vente) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_2$(Dupuis,Delcours).

Comment satisfaire cet objectif ?

Parmi les 6 littéraux dont il est composé, le premier est un événement interne dérivé positif et peut par conséquent être remplacé par le contenu du corps des règles le définissant. Dans la base de données, on constate que deux règles sont associées à dresp ; deux branches différentes sont donc nécessaires.

<- *d*resp(Dupuis,Vente) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

PREMIERE BRANCHE

<- *d*trav(Dupuis,Vente) & chef(Dupuis) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

<- dtrav(Dupuis,Vente) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

<- *d*resp(Dupuis,Vente) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

DEUXIEME BRANCHE

<- trav(Dupuis,Vente) & *d*chef(Dupuis) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

<- dchef(Dupuis) & ¬sup'$_{1,1}$(Dupuis,Delcours) & ¬sup'$_{1,2}$(Dupuis,Delcours) & ¬sup'$_{1,3}$(Dupuis,Delcours) & ¬sup'$_{1,4}$(Dupuis,Delcours) & ¬sup'$_{2}$(Dupuis,Delcours).

Dans un premier temps, on s'intéressera aux premiers littéraux de chacun des objectifs finaux. Ce sont des événements internes de base positifs ; ils peuvent être satisfaits si l'on suppose que *d*trav(Dupuis,Vente) et *d*chef(Dupuis) ont respectivement lieu durant la transition ; ce qui équivaut à les inclure dans T. Jusqu'à présent, on a deux possibilités : T1 = {*d*trav(Dupuis,Vente)} et T2 = {*d*chef(Dupuis)}. Il reste à vérifier la réussite des 5 littéraux restants.

Si P est un prédicat, on sait qu'une réussite de ¬P équivaut à un échec de P. Dans notre cas, sup'$_{1,j}$(Dupuis,Delcours) ; j = 1..4, et sup'$_{2}$(Dupuis,Delcours) sont 5 prédicats dérivés, il faut par conséquent recourir à des arbres auxiliaires pour tester la valeur de leur négation. En voici une illustration.

<- sup'$_{1,1}$(Dupuis,Delcours)

T1 = {dtrav(Dupuis,Vente)}

T2 = {dchef(Dupuis)}

<- resp(Dupuis,D) & ¬*d*resp(Dupuis,D) & trav(Delcours,D) & ¬*d*trav(Delcours,D) & Delcours < > Dupuis.

{ D = Vente }

<- resp(Dupuis,Vente) & ¬*d*resp(Dupuis,Vente) & trav(Delcours,Vente) & ¬*d*trav(Delcours,Vente).

<- travail(Dupuis,Vente) & chef(Dupuis) & ¬*d*resp(Dupuis,Vente) & trav(Delcours,Vente) & ¬*d*trav(Delcours,Vente).

Echec

Dans ce premier arbre, l'objectif <- travail(Dupuis,Vente) & chef(Dupuis) & ¬*d*resp(Dupuis,Vente) & trav(Delcours,Vente) & ¬*d*trav(Delcours,Vente) échoue pour la transaction T1 car le littéral trav(Dupuis,Vente) ne peut être satisfait puisque *d*trav(Dupuis,Vente) fait partie de T1. La réussite de ¬sup'$_{1,1}$(Dupuis,Delcours) est immédiate.

Le principe est identique pour la transaction T2 car satisfaire *d*chef(Dupuis) et chef(Dupuis) entraîne une contradiction flagrante.

L'arbre associé à ¬sup'$_{1,2}$(Dupuis,Delcours) échoue de façon tout à fait similaire. Il reste l'étude de ¬sup'$_{1,3}$(Dupuis,Delcours) et de ¬sup'$_{1,4}$(Dupuis,Delcours). Considérons dans un premier temps l'arbre auxiliaire associé à ¬sup'$_{1,3}$(Dupuis,Delcours).

<- $sup'_{1,3}$(Dupuis,Delcours).

T1 = {*d*trav(Dupuis,Vente)}
T2 = {*d*chef(Dupuis)}

<- *i*resp(Dupuis,D) & trav(Delcours,D) & ¬*d*trav(Delcours,D) & Delcours <> Dupuis.

{ D = Vente }

<- *i*resp(Dupuis,Vente) & trav(Delcours,Vente) & ¬*d*trav(Delcours,Vente).

<- *i*resp(Dupuis,Vente) & ¬*d*trav(Delcours,Vente).

ECHEC

Cet arbre échoue également car, si l'on développait les 3 branches associées à *i*resp(Dupuis,Vente), on verrait qu'il est impossible d'aboutir à une réussite de ces branches puisque dans la base de données actuelle, resp(Dupuis,Vente) est déjà prouvable. L'arbre correspondant à $sup'_{1,4}$(Dupuis,Delcours) échoue pour la même raison.

Actuellement, on est donc certain de la réussite de ¬$sup_{1,j}$(Dupuis,Delcours) ; j=1..4. Il reste à démontrer la réussite de ¬sup'_2(Dupuis,Delcours). Pour ce faire, il faut à nouveau prouver l'échec de sup'_2(Dupuis,Delcours).

<- sup'_2(Dupuis,Delcours).

PREMIERE BRANCHE

<- $sup'_{2,1}$(Dupuis,Delcours).

<- sup(Dupuis,Z) & ¬*d*sup(Dupuis,Z) & sup(Z,Delcours) & ¬*d*sup(Z,Delcours).

<- sup(Dupuis,Z) & ¬*d*sup(Dupuis,Z) & resp(Z,D) & trav(Delcours,D) & Z <> Delcours & ¬*d*sup(Z,Delcours).

{ D = Vente }

<- sup(Dupuis,Z) & ¬*d*sup(Dupuis,Z) & trav(Z,Vente) & chef(Z) & Z <> Delcours & ¬*d*sup(Z,Delcours).

ECHEC

Cette branche mène à un échec car seul Dupuis a le statut de chef dans la base de données et Z ne peut pas être instancié à Dupuis car on aurait sup(Dupuis,Dupuis).

<- sup'_2(Dupuis,Delcours).

DEUXIEME BRANCHE
<- $sup'_{2,2}$(Dupuis,Delcours).

<- sup(Dupuis,Z) & ¬*d*sup(Dupuis,Z) & *i*sup(Z,Delcours).

<- resp(Dupuis,D) & trav(Z,D) & Dupuis < > Z & ¬*d*sup(Dupuis,Z) & *i*sup(Z,Delcours).

<- trav(Dupuis,D) & chef(Dupuis) & trav(Z,D) & Dupuis < > Z & ¬*d*sup(Dupuis,Z) & *i*sup(Z,Delcours).

{D = Vente}

<- trav(Z,Vente) & Dupuis < > Z & ¬*d*sup(Dupuis,Z) & *i*sup(Z,Delcours).

ECHEC

L'échec est immédiat car lorsque l'on consulte la base de données, on voit que la seule instanciation possible de Z telle que trav(Z,Vente) et Dupuis < > Z soient vérifiés est Z = Delcours. Or, pour que la branche réussisse, il faut aussi pouvoir insérer sup(Z,Delcours), ce qui sera refusé car la cohérence de la base de données ne sera plus respectée.

<- sup'_2(Dupuis,Delcours).

TROISIEME BRANCHE
<- $sup'_{2,3}$(Dupuis,Delcours).

<- *i*sup(Dupuis,Z) & sup(Z,Delcours) & ¬*d*sup(Z,Delcours).

<- *I*sup(Dupuis,Z) & resp(Z,D) & trav(Delcours,D) & Z <> Delcours &
¬*d*sup(Z,Delcours).

{D = Vente}

<- *i*sup(Dupuis,Z) & resp(Z,Vente) & Z <> Delcours & ¬*d*sup(Z,Delcours).

<- *i*sup(Dupuis,Z) & trav(Z,Vente) & chef(Z) & Z <> Delcours &
¬*d*sup(Z,Delcours).

{Z = Dupuis}

<-*i*sup(Dupuis,Dupuis) & ¬*d*sup(Dupuis,Delcours).

ECHEC

L'échec provient de l'impossibilité d'insérer Dupuis comme supérieur de Dupuis.

<- sup'_2(Dupuis,Delcours).

QUATRIEME BRANCHE
<- $sup'_{2,4}$(Dupuis,Delcours).

<- *i*sup(Dupuis,Z) & *i*sup(Z,Delcours).

... **ECHEC**

Montrer l'échec de cette branche en l'illustrant au moyen de toutes les branches qu'elle génère est un travail fastidieux et pas réellement justifié dans ce cas-ci. Pour que Dupuis soit supérieur de Z et que Z soit supérieur de Delcours, il faut que Dupuis soit responsable du département dans lequel Z travaille et que Z soit responsable du département dans lequel Delcours travaille. Or Delcours travaille dans le département Vente dont Dupuis est responsable... Par la première contrainte d'intégrité, on sait que Z ne trouvera pas d'instanciation.

En conclusion, pour supprimer resp(Dupuis,Vente), deux solutions sont acceptables : T1={*d*trav(Dupuis,Vente)} et T2={*d*chef(Dupuis)}. Dans ce cas-ci, les deux solutions sont minimales. Il est cependant intéressant de remarquer que des solutions non minimales sont parfois générées car tant que le processus de dérivation n'est pas achevé, rien ne dit qu'une solution minimale ne sera pas invalidée suite à l'obtention d'une contradiction entre les contenus de T et de C. Dans ce cas, une solution non minimale pourrait devenir solution minimale car seule acceptable.

IV.9.2.4. Synthèse du procédé

Initialement, la mise à jour a la forme d'un prédicat d'événements internes d'insertion ou de suppression dérivé et les ensembles T et C sont vides. La première étape de la résolution est toujours de remplacer l'objectif *i*P(K) (ou *d*P(K)) par une conjonction de littéraux L1 & L2 & ... & Lk équivalente. Pour ce faire, il suffit de consulter la règle définissant *i*P(K) dans la base de données augmentée.

Les grandes étapes du procédé sont données dans l'algorithme ci-dessous.

```
Debut
    T = {}
    C = {}
    Objectif = "L1 & L2 & ... & Lk"
    Tant qu'il reste des littéraux Li dans l'objectif
        Faire
            Procédure 1 : Sélectionner le littéral Lj  le plus prioritaire
            Procédure 2 : "Traiter" le littéral Lj càd
                              - Adapter le contenu de T
                              - Adapter le contenu de C
                              - Définir le nouvel objectif
        Fin Faire
Fin
```

Examinons à présent le contenu du corps de la boucle de manière plus détaillée.

Avant de déterminer le contenu de la première procédure, il faut se rappeler que, dans une base de donnée augmentée, 5 sortes de prédicats sont possibles, à savoir les prédicats de base (ex. trav(Smits,Marketing)), les prédicats dérivés (ex. resp(X,D)), les prédicats d'événements internes de base (ex. *i*chef(Delcours)), les prédicats d'événements internes dérivés (ex. *i*resp(Roffin,Marketing)) et enfin les prédicats de transition (ex. $sup'_{1,2}(X,Y)$). Tous ces prédicats peuvent bien entendu être soit positifs, soit négatifs.

L'argument en entrée de la procédure 1 a la forme d'une conjonction de littéraux "L1 & L2 &...& Lk" ; cette procédure donne comme résultat le littéral Li ayant la plus haute priorité selon l'ordre de priorité descendant établi ci-dessous :

- un prédicat de base positif ;
- un prédicat de base négatif ground, càd dans lequel les variables sont instanciées ;
- un prédicat dérivé positif ;
- un prédicat dérivé négatif ground ;
- un prédicat de transition positif ;
- un prédicat d'événement interne dérivé positif ;
- autres (principalement tous les prédicats dont l'évaluation nécessite l'utilisation d'un arbre auxiliaire).

Une fois le littéral sélectionné, il faut savoir comment le traiter et quels sont les effets de ce traitement sur T, C et sur l'objectif résultant ; ce qui correspond à la deuxième procédure.

La procédure 2 reçoit comme arguments le littéral Li à traiter, l'objectif (sous forme de conjonction de littéraux) et les ensembles T et C déjà déterminés.

L'opération à effectuer pour adapter les contenus de T et de C varie en fonction de la nature du littéral sélectionné et des contenus déjà déterminés des ensembles T et C. Nous allons dans un premier temps analyser les différents cas de figure possibles dans un arbre de dérivation principal, c'est-à-dire dont on essaie d'obtenir la réussite.

1) Si Lj est positif et correspond à un prédicat soit de base, soit dérivé, soit de transition, soit d'événement interne dérivé alors la procédure applique simplement la résolution SLDNF pour atteindre la définition du nouvel objectif, T et C restent

inchangés. Ainsi, si Lj est un prédicat dérivé par exemple, le nouvel objectif sera identique à l'ancien mais Lj y est remplacé par le corps de la règle le définissant.

2) Si Lj est un prédicat de base ou dérivé négatif "¬P" et s'il peut être prouvé, càd si l'objectif <-P échoue, alors le nouvel objectif est "L1 &...& Lj-1 & Lj+1 &...& Lk", T et C sont inchangés.

3) Si Lj est un événement interne de base positif *i*P, Lj n'appartient pas à C et le fait P n'est pas prouvable dans la base de données actuelle, alors le nouvel objectif est "L1 &...& Lj-1 & Lj+1 &...& Lk", Lj est inséré dans T puisqu'il doit avoir lieu pour que l'objectif réussisse, et C est inchangé.

4) Si Lj est un événement interne de base négatif "¬*i*P" (resp "¬*d*P") dans lequel toutes les variables sont instanciées, *i*P (resp *d*P) n'appartient pas à T et le fait P n'est pas (resp est) prouvable dans la base de données actuelle, alors le nouvel objectif est "L1 &...& Lj-1 & Lj+1 &...& Lk", T est inchangé et *i*P (resp *d*P) est inséré dans C puisqu'il ne peut pas avoir lieu pour que l'objectif réussisse.

5) Si Lj est un prédicat de transition ou d'événement interne dérivé négatif "¬P", alors la situation est un peu plus compliquée car il faut essayer de prouver l'échec de P en ayant recours à un arbre auxiliaire. L'objectif initial de cet arbre auxiliaire est "<-P" mais cette fois, c'est l'échec de l'arbre que l'on doit essayer d'obtenir de sorte que "¬P" réussisse. Les ensembles T et C déjà déterminés sont donnés en input de cette deuxième dérivation.

Les différents cas de figure possibles dans un arbre auxiliaire sont les suivants.

A1) Si Lj est positif et correspond à un prédicat soit de base, soit dérivé, soit de transition, soit d'événement interne dérivé alors il suffit d'appliquer la résolution SLDNF pour atteindre la définition du nouvel objectif, T et C sont inchangés.

A2) Si Lj est un prédicat de base ou dérivé négatif "¬P" et s'il peut être prouvé en démontrant l'échec de P, alors le nouvel objectif est "L1 &...& Lj-1 & Lj+1 &...& Lk", T et C sont inchangés.

A3) Si Lj est un événement interne de base positif appartenant à C, alors l'échec de la branche est assuré puisqu'il y a contradiction : d'un côté Lj devrait avoir lieu pour assurer la réussite de la branche, de l'autre, il ne peut pas avoir lieu puisqu'il appartient

à C.

A4) Si Lj est un événement interne de base positif appartenant à T mais pas à C, alors le nouvel objectif à prouver est "L1 &...& Lj-1 & Lj+1 &...& Lk", T et C sont inchangés.

A5) Si Lj est un événement interne de base positif n'appartenant ni à T ni à C, et si le fait P n'est pas prouvable dans la base de données actuelle, alors Lj est inséré dans C ; ce qui assure l'échec de cette branche de l'arbre auxiliaire car Lj devrait avoir lieu pour que la réussite soit possible, or il est inséré dans C...

A6) Si Lj est un événement interne de base négatif "$\neg i$P" (resp "$\neg d$P") dont toutes les variables sont instanciées et si iP (resp. dP) appartient à T, alors l'échec de la branche est immédiat.

A7) Si Lj est un événement interne de base négatif "$\neg i$P" (resp. "$\neg d$P") dont toutes les variables sont instanciées et si iP (resp dP) n'appartient pas à T mais appartient à C, alors le nouvel objectif est "L1 &...& Lj-1 & Lj+1 &...& Lk", T et C sont inchangés.

A8) Si Lj est un événement interne de base négatif "$\neg i$P" (resp "$\neg d$P") dont toutes les variables sont instanciées et si iP (resp dP) n'appartient ni à T, ni à C et le fait P n'est pas (resp est) prouvable dans la base de données actuelle alors iP (resp dP) est inséré dans T, ce qui assure l'échec de la branche.

A9) Si Lj est un prédicat d'événement interne dérivé ou de transition négatif "$\neg$P", alors il faut essayer de prouver l'échec de P en ayant à nouveau recours à un arbre auxiliaire. Les ensembles T et C déjà déterminés sont donnés en input de cette troisième dérivation ; le procédé est identique.

Une fois la dérivation terminée, les ensembles de translation obtenus correspondent aux différentes possibilités de satisfaire la demande de mise à jour initiale. Si aucun ensemble de translation T n'est trouvé, des changements dans la partie extensionnelle de la base de données ne suffisent pas à satisfaire la demande. Dans ce cas, des changements dans la base de données intentionnelle doivent être opérés ; ce problème n'est pas développé dans ce chapitre.

Le traitement de mises à jour multiples, composées d'insertions et de suppressions diverses, est également possible. Ces mises à jour ont la forme 'insert(P1) et ... et insert(Pn) et delete(Q1) et ... et delete(Qm)' ; ce qui correspond à l'objectif racine '<- *i*P1 & ... & *i*Pn & *d*Q1 & ... &*d*Qm'. Le procédé à suivre est exactement le même, la seule différence est la taille nettement plus grande de l'arbre de résolution.

Avant de clôturer ce chapitre, quelques précisions doivent encore être données à propos de la satisfaction des contraintes d'intégrité. En effet, dans l'exemple de *d*sup(Dupuis,Delcours), l'on s'est contenté de dire qu'une action donnée était refusée "parce qu'elle violait une contrainte"... Voyons à présent comment le respect de la cohérence est assuré.

IV.9.4. Satisfaction des contraintes d'intégrité lors de mises à jour

Certaines des translations obtenues peuvent être invalidées par les contraintes d'intégrité car elles sont composées d'événements internes provoquant l'insertion de faits "Icn" durant la transition de l'état courant de la base de données à l'état mis à jour. En suivant la méthode, le contrôle d'intégrité peut être effectué dynamiquement durant le processus de dérivation. Ainsi, seules des translations qui satisfont aussi bien la demande de mise à jour que les contraintes d'intégrité sont obtenues.

L'objectif racine doit être modifié dans le but d'incorporer le contrôle d'intégrité dans la demande de mise à jour ; il est défini comme {<- M & ¬*i*Ic}, où M détermine la demande de mise à jour et ¬*i*Ic indique qu'aucune contrainte d'intégrité ne peut être violée. Pour rappel, la définition du prédicat iIc est *i*Ic <- *i*Ic1,..., *i*Ic <- *i*Icn, où n est le nombre de contraintes dans la base de données.

Le procédé qui permet de trouver toutes les mises à jour qui satisfont la demande initiale et les contraintes d'intégrité est exactement le même que celui qui a été expliqué précédemment. Les arbres de dérivation sont cependant plus complexes et il est difficile de les représenter clairement sur papier avec une base de données telle que celle qui nous sert d'exemple. Le lecteur intéressé pourra cependant trouver un exemple complet dans [Teniente91] où la base de données a été spécialement élaborée à cet escient.

IV.10. Conclusion

Tout au long de ce chapitre, nous avons vu comment la méthode des événements internes offre une solution adéquate aux problèmes du contrôle de cohérence et des mises à jour de vues. Le procédé est simple et ne pose pas de problèmes particuliers lors de l'implémentation puisque tous les artifices utilisés sont formulés sous forme de règles de déduction, directement manipulables par l'interpréteur PROLOG. Cette facilité de gestion représente un avantage considérable.

L'on pourrait cependant reprocher à la méthode d'augmenter considérablement le nombre de règles nécessaires dans la base de données. Ce reproche est malheureusement fondé mais il perd quelque peu de sa crédibilité lorsque l'on sait que dans une base de données réelle, le nombre de faits est normalement nettement supérieur au nombre de règles.

V. IMPLEMENTATION DE LA METHODE DES EVENEMENTS INTERNES

V.1. Introduction

Nous avons vu au cours du chapitre précédent que la méthode des événements internes offre une solution satisfaisante au contrôle de cohérence d'une base de données déductive ainsi qu'au problème des mises à jour, qu'elles soient de base ou dérivées.

L'ensemble du chapitre reposait uniquement sur des raisonnements théoriques, illustrés de quelques exemples élémentaires, plus pédagogiques que réalistes... Quoique ces raisonnements théoriques fassent la joie d'un lecteur passionné par les démonstrations mathématiques et autres raisonnements formels, il est fort probable qu'un lecteur ordinaire se soit senti frustré face à un tel flot de théories relativement difficiles à saisir intuitivement... L'objectif des quelques pages qui suivent est de montrer que la méthode est réalisable pratiquement puisqu'un prototype existe et est toujours en cours de développement.

V.2. Objectif

La théorie des événements internes est le fruit des recherches du Professeur Olive ; prenant de plus en plus d'importance et suscitant un intérêt sans cesse grandissant lors de présentations à diverses conférences, le Professeur Olive a estimé que la suite logique de l'évolution de ses recherches serait le test pratique de la pertinence de sa nouvelle méthode. Pour ce faire, il a confié au Professeur Teniente, un de ses jeunes collaborateurs, la tâche de réaliser le premier prototype implémentant la méthode des événements internes ; mais de quoi s'agit-il exactement ?

Avant tout, une petite mise au point s'impose. Il est en effet fondamental de bien réaliser que l'objectif n'est absolument pas de mettre au point le prototype d'un système de gestion de base de données déductif ; une telle entreprise serait vouée à un échec certain ! Créer un SGBD relationnel "classique", dont tous les aspects sont à présent bien maîtrisés, est déjà une tâche fort complexe nécessitant d'importants moyens financiers. Que dire alors des ressources nécessaires à la création d'un SGBD déductif ?

L'investissement aussi bien en capital humain que financier est non seulement énorme, mais en plus il est risqué. Les chercheurs butent en effet sur un réel problème

d'évaluation des performances du système dans un environnement d'exploitation normal, et il est impératif de savoir si oui ou non les temps de réponse aux requêtes restent dans un ordre de grandeur acceptable lorsque le nombre d'informations contenues dans la base augmente.

Actuellement, aucun SGBD déductif "digne de ce nom" n'est commercialisé, quelques prototypes existent (voir notamment [Ling-Goh90] [Vaghani90]), d'après leurs concepteurs, ils offrent de "très bonnes performances", "d'excellents temps de réponse",... Certains les disent même "compétitifs par rapport aux SGBD relationnels classiques". Bien que ces expressions soient vraisemblablement utilisées à bon escient, il ne faut pas perdre de vue les hypothèses sur lesquelles elles reposent !

Les composants d'un SGBD déductif sont nombreux, le vrai problème est donc de trouver la bonne combinaison. Supposons par exemple qu'un chercheur affirme, chiffres à l'appui, que la méthode d'évaluation de requêtes qu'il vient d'implémenter donne les meilleurs résultats atteints jusqu'à présent. Bien que ne mettant pas en doute la bonne foi du chercheur, on peut se demander si ces beaux résultats ne deviendraient pas catastrophiques si la base de données était de taille double, triple, ou simplement de la taille d'une base de données en exploitation réelle. Le doute ne vient pas uniquement du nouveau module, mais bien de l'intégration de ce module dans son environnement, dont la maîtrise n'est pas, par définition, strictement dépendante des travaux du chercheur en question.

De nombreuses personnes se consacrent à l'implémentation de systèmes de gestion de bases de données déductifs. En général, leur motivation première est le test d'une nouvelle théorie qu'ils croient prometteuse. Ils utilisent donc d'autres outils déjà développés ! Ceci explique partiellement la difficulté de compréhension du comportement d'un système de gestion et de ses différentes performances. En effet, comparer deux techniques particulières n'a de sens que si les hypothèses sont identiques, encore faut-il que ces dernières ne favorisent pas davantage une technique par rapport à l'autre.

En quoi consiste alors le prototypage de la méthode des événements internes ?

L'objectif initial était tout simplement de savoir si, étant donné une base de données déductive quelconque augmentée de ses règles de transition et d'événements internes, et étant donné une mise à jour portant sur une occurrence d'un fait dérivé, le prototype serait en mesure de produire les différentes combinaisons d'actions élémentaires satisfaisant la demande de mise à jour initiale tout en assurant le respect de

la cohérence de la base de données. Le prototypage est donc uniquement limité à ce qui a été explicité dans le chapitre précédent : si la méthode est effectivement pertinente, la théorie et les règles exposées auparavant sont strictement suffisantes à l'implémentation d'un prototype opérationnel... Et c'est le cas !

L'implémentation de cette première étape de développement a mené à une réussite assez immédiate. Bien entendu, cette implémentation ne fait pas intervenir une véritable base de données. Le prototype consulte en fait un fichier PROLOG composé de 2 parties distinctes :

- la base de faits : elle contient l'équivalent des faits élémentaires de la base de données augmentée ; dans une situation d'exploitation réelle, ces faits se trouveraient dans des tables relationnelles ;
- la base de règles : elle contient les règles de déduction, les contraintes d'intégrité exprimées sous forme de règles d'intégrité, les règles de transition et les règles d'événements internes de suppression et d'insertion.

Dans de telles conditions, il a suffit de quelques pages de programmes PROLOG pour aboutir à l'objectif escompté. Bien que le listing d'un programme ne soit dans notre contexte que d'une utilité fort discutable, il m'a néanmoins semblé intéressant de le reprendre [1]. Pourquoi cela ?... Tout simplement parce qu'un lecteur maîtrisant la programmation en PROLOG sera en mesure d'y retrouver, sous forme de procédures, l'ensemble des raisonnements donnés dans le chapitre précédent.

Quelques indications sont cependant nécessaires, ainsi :

- "Goal" représente la mise à jour à résoudre.
- "L" représente la liste de mises à jour élémentaires dont l'effet est équivalent à la demande de mise à jour initiale, à savoir "Goal".
- Lorsque le foncteur est "solve", l'objectif est d'aboutir à une réussite de l'objectif alors que lorsque le foncteur est "do_fail", cela signifie que l'on se trouve dans un arbre auxiliaire que l'on essaie de faire échouer.

---

[1] Il s'agit d'une version simplifiée (on n'y gère pas l'ensemble de conditions C), plus aisée à comprendre.

```
solve(Goal) :-
        solve(Goal,[],L),
        write(L), nl,
        fail.


solve(true,X,X) :- !.

solve((A,B),L1,L) :-
        !, solve(A,[],L2), solve(B,[],L3),
        append(L1,L2,Lint), append(Lint,L3,L).



solve(not(ins(A)),X,[not(ins(A))|X]) :-
        not(A), !.

solve(not(ins(A)),X,X) :-
        !.



solve(not(del(A)),X,[not(del(A))|X]) :-
        A, !.

solve(not(del(A)),X,X) :-
        !.



solve(not(A),X,L) :-
        do_fail(A,[],Y),
        append(X,Y,L).



solve(ins(A),X,X) :-
        ins(A), !.

/* A és un predicat que té algun dels seus membres sense instanciar, caldrà
doncs demanar-li els valors a l'usuari */

/* A is a predicate that has some of its members non instantiated, the user
must be asked for some value */

solve(ins(A),X,[ins(Res)|X]) :-
        A=..[T1|L], hi_ha_termes_no_instanciats(L), !,
        nl, write('Assigna valors a les variables de '),
        write(A), write(' per inserció '),
        llegir(L), Res =.. [T1|L], not(Res), asserta(ins(Res)),
        on_backtracking(retract(ins(Res))).

/* Ja tenim tots els valors a inserir, no cal que li demanem res a l'usuari */

/* We have all values to insert, we don't have to ask the user */

solve(ins(A),X,[ins(A)|X]) :-
        !, not(A), asserta(ins(A)),
        on_backtracking(retract(ins(A))).



solve(del(A),X,X) :-
        del(A), !.

solve(del(A),X,[del(A)|X]) :-
        !, A, asserta(del(A)),
        on_backtracking(retract(del(A))).
```

```
solve(A,X,L) :-
        clause(A,B),
        solve(B,X,L).



/* Allowedness condition ensures that all negative literals will be fully
instantiated when they have to be evaluated */


do_fail((not(ins(A)),B),X,X) :-
        ins(A), !.

do_fail((not(ins(A)),B),X,[ins(A)|X]) :-
        not(A), asserta(ins(A)),
        on_bactracking(retract(ins(A))).

/* en la següent regla es retracta ins(A) ja que reconsiderem, i cal retractar
el ins(A) corresponent que s'ha inserit a la regla anterior */

/* In next rule, ins(A) inserted in previous rule is retracted because we
reconsider */

/* do_fail((not(ins(A)),B),X,L) :-
        not(A), retract(ins(A)), !, do_fail(B,X,L).*/

do_fail((not(ins(A)),B),X,L) :-
        !, do_fail(B,X,L).

do_fail(not(ins(A)),X,X):-
        ins(A), !.

do_fail(not(ins(A)),X,[ins(A)|X]):-
        not(A), !,
        asserta(ins(A)),
        on_backtracking(retract(ins(A))).

/* en la següent regla es retracta ins(A) ja que reconsiderem, i cal retractar
el ins(A) corresponent que s'ha inserit a la regla anterior */

/* In next rule, ins(A) inserted in previous rule is retracted because we
reconsider */

/* do_fail(not(ins(A)),X,[ins(A)|X]):-
        not(A), !, retract(ins(A)), fail. */



do_fail((not(del(A)),B),X,X):-
        A, del(A), !.

do_fail((not(del(A)),B),X,[del(A)|X]):-
        A, asserta(del(A)),
        on_backtracking(retract(del(A))).

/* en la següent regla es retracta del(A) ja que reconsiderem, i cal retractar
el del(A) corresponent que s'ha inserit a la regla anterior */

/* In next rule, del(A) inserted in previous rule is retracted because we
reconsider */

/* do_fail((not(del(A)),B),X,L) :-
        A, retract(del(A)), !, do_fail(B,X,L). */

do_fail((not(del(A)),B),X,L) :-
        !, do_fail(B,X,L).

do_fail(not(del(A)),X,X):-
        A, del(A), !.
```

```
do_fail(not(del(A)),X,[del(A)|X]):-
        A, !,
        asserta(del(A)),
        on_backtracking(retract(del(A))).

/* en la següent regla es retracta del(A) ja que reconsiderem, i cal retractar
el del(A) corresponent que s'ha inserit a la regla anterior */

/* In next rule, del(A) inserted in previous rule is retracted because we
reconsider */

/* do_fail(not(del(A)),X,X):-
        A, !, retract(del(A)), fail. */



do_fail((ins(A),B),X,L) :-
        findall(B,clause(ins(A),true),Llista),
        !, fer_fallar_llista(Llista,Y),
        append (X,Y,L).

do_fail((ins(A),B),X,L) :-
        not(A),
        do_fail(B,X,L).

do_fail((ins(A),B),X,[not(ins(A))|X]) :- not(A), !.

do_fail((ins(A),B),X,X) :- !.

do_fail(ins(A),X,[not(ins(A))|X]) :- not(A), !.

do_fail(ins(A),X,X) :- !.



do_fail((del(A),B),X,L) :-
        findall(B,clause(del(A),true),Llista),
        !, fer_fallar_llista(Llista,Y),
        append (X,Y,L).

do_fail((del(A),B),X,L) :-
        A,
        do_fail(B,X,L).

do_fail((del(A),B),X,[not(del(A))|X]) :- A, !.

do_fail((del(A),B),X,X) :- !.

do_fail(del(A),X,[not(del(A))|X]) :- A, !.

do_fail(del(A),X,X) :- !.



do_fail((not(A),B),X,X) :-  /* A és basic i és a la BD */
        clause(A,true), !.

do_fail((not(A),B),X,L) :-  /* A és derivat */
        clause(A,C),
        solve(A,[],Y),
        append(X,Y,L).

do_fail((not(A),B),X,L) :-  /* A és derivat */
        clause(A,C),
        !, do_fail(A,[],Y),
        do_fail(B,[],Z),
        append(X,Y,Lint),
        append(Lint,Z,L).

do_fail((not(A),B),X,L) :-  /* A és basic i NO és a la BD */
        !, do_fail(B,[],Y),
        append(X,Y,L).
```

```
/* Es podria afegir també:
                do_fail(not(A),X,X) :- clause(A,true), !.
        per diferenciar entre fets bàsics i derivats, potser és més eficient */



do_fail(not(A),X,L) :-
        !, solve(A,[],Y),
        append(X,Y,L).



do_fail((A,B),X,L) :-  /* A és fet bàsic i és a la BD */
        findall(B,clause(A,true),Llista),
        !, fer_fallar_llista(Llista,Y),
        append(X,Y,L).

do_fail((A,B),X,L) :-  /* A és derivat */
        clause(A,C),
        solve(C,[],Y),
        do_fail(B,[],Z),
        append(X,Y,Lint), append(Lint,Z,L).

do_fail((A,B),X,L) :-  /* A és derivat */
        clause(A,C),
        findall(C,clause(A,C),Llista),
        !, fer_fallar_llista(Llista,Y),
        append(X,Y,L).

do_fail((A,B),X,X) :- !.  /* A és fet basic i NO és a la BD */

do_fail(A,X,X) :-  /* A és bàsic i és a la BD */
        clause(A,true),!, fail.

/* A és derivat. Cal buscar totes les clàusules C, del tipus A:-C,
i fer fallar C per cadascuna d'elles */

/* A is derived. We have to look for all clauses C, of the kind A:-C, and
make them fail */

do_fail(A,X,L):-
        findall(C,clause(A,C),Llista),
        !, fer_fallar_llista(Llista,Y),
        append(X,Y,L).




/* A partir d'aqui es trben els predicats intermedis necessaris pel metode */

/* From now on we find all auxiliary predicates necessary for the method */

fer_fallar_llista([],[]).
fer_fallar_llista([H|T],L) :-
        !, do_fail(H,[],X),
        fer_fallar_llista(T,Y),
        append(X,Y,L).


append([],L,L) :- !.
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).

findall(X,G,_) :- asserta(found(mark)), call(G), asserta(found(X)), fail.
findall(_,_,L) :- collect_found([],M), !, L=M.
collect_found(S,L) :- getnext(X), !, collect_found([X|S],L).
collect_found(L,L).
getnext(X) :- retract(found(X)), !, X \== mark.
```

```
/* El predicat "llegir" s'usa per llegir (instanciar) els termes d'un atom que
no estan instanciats quan s'ha d'inserir o esborrar aquest atom. O sigui cal
posar l'atom instanciat a la llista de solucions T(u) */

/* "llegir" predicate is used in order to read (instantiate) all terms non
instantiated of an atom when it must be inserted or deleted */

llegir([Term]) :- var(Term), read(Term), !.
llegir([Term]).
llegir([T1|T2]) :- llegir([T1]), llegir(T2), !.


/* El predicat hi_ha_termes_no_instanciats(L) comprova si algun element de la
llista L és una variable */

/* hi_ha_termes_no_instanciats(L) predicate checks if some element of list L
is a variable */

hi_ha_termes_no_instanciats([Term|_]) :- var(Term).
hi_ha_termes_no_instanciats([_|L1]) :- hi_ha_termes_no_instanciats(L1).


/* Predicate retract_int_ev retracts all base internal events that have been
asserted because they belong to T(u). This allows the program to find a new
translation T(u) */

retract_int_ev :- retract(ins(A)), fail.
retract_int_ev :- retract(del(A)), fail.
retract_int_ev.

llista_int_ev :- ins(A), write(ins(A)), nl, fail.
llista_int_ev :- del(A), write(del(A)), nl, fail.
llista_int_ev.


/*on_backtracking(X) predicate is used in order to execute X while backtrack */

on_backtracking(X).
on_backtracking(X) :-
        X, !, fail.
```

Bien entendu, ce programme n'est que la première étape de l'implémentation. Beaucoup de développements restaient (et restent toujours) à faire, mais très vite un problème pratique s'est posé : le test du programme n'est possible que si les arguments en "input" sont la demande de mise à jour initiale accompagnée de la base de données à mettre à jour augmentée de ses règles de transition et d'événements internes. Ceci signifie qu'avant de pouvoir exploiter une base de données déductive, il fallait avant tout en calculer la base de données augmentée, et ce travail est fastidieux à faire "manuellement". De plus, toutes ces règles devaient encore être encodées... A titre indicatif, si le corps d'une règle de déduction est composé de 4 littéraux non évaluables, alors les règles de transition sont au nombre de 16 (càd 2^4), les règles d'événements internes d'insertion au nombre de 15, et enfin les règles d'événements internes de suppression au nombre de 4 mais leur forme est nettement plus complexe. L'exemple de base de données augmentée figurant dans le chapitre précédent au point IV.9. donne une bonne illustration de l'ampleur que peut atteindre une base de données augmentée.

Pour pouvoir tester efficacement le prototype, il était par conséquent absolument indispensable de disposer d'un *générateur de règles de transition et d'événements internes*. Après proposition du Professeur Teniente et en accord avec tous les autres membres de l'équipe, j'ai décidé d'accepter cette réalisation comme premier objectif pratique. Cette entreprise s'est révélée très fructueuse.

Dans un premier temps, cela m'a permis de m'intégrer davantage encore au sein de leur équipe puisque je quittais mon "poste d'observation" pour réaliser un travail vraiment utile.

Ensuite, la réalisation de ce *générateur de règles de transition et d'événements internes* nécessitait une maîtrise parfaite de l'ensemble de la théorie sous-jacente ; cette maîtrise, je ne l'ai acquise qu'en résolvant les nombreux problèmes auxquels j'ai été confrontée.

Finalement, après quelques semaines de travail, le générateur s'est avéré tout à fait opérationnel et a été incorporé intégralement dans le prototype existant.

Ainsi, les tests ont pu être effectués sur des bases de données nettement plus complexes à gérer, avec des mises à jour multiples,... Les résultats obtenus correspondaient effectivement à ceux escomptés mais un nouveau problème s'est posé : la piètre qualité des temps de réponse.

Il est bien évident que l'objectif des Professeurs Olive et Teniente n'est absolument pas d'obtenir des temps de réponse à la hauteur de ceux d'un SGBD relationnel. Leur seul objectif est de tester la méthode des événements internes et de convaincre les chercheurs travaillant sur les bases de données déductives de l'originalité et de l'efficacité de cette méthode, ces derniers pouvant éventuellement l'intégrer dans un prototype de SGBD de plus grande envergure. La qualité des temps de réponse n'est donc pas d'une importance capitale (d'autant plus que la véritable origine du problème réside dans le mode de fonctionnement de l'interpréteur PROLOG) mais il s'avère que l'ordre des littéraux dans le corps d'une règle est un facteur influençant fortement les performances de l'interpréteur PROLOG. Il a par conséquent été décidé d'ajouter un nouveau module au prototype. Sa fonctionnalité est d'engendrer, à partir d'une base de données augmentée, une base de données augmentée équivalente mais dans laquelle l'ordonnancement des littéraux dans le corps des différentes règles est optimal. La réalisation de ce nouveau module m'a à nouveau été confiée, ce qui m'a permis de me spécialiser davantage en programmation PROLOG.

L'ordre d'ordonnancement des littéraux souhaité dans le corps des règles était le suivant :

1 - les littéraux universels ;

2 - les littéraux positifs "simples", càd qui ne correspondent pas à un prédicat d'événement interne ;

3 - les littéraux positifs qui correspondent à un prédicat d'événement interne dérivé à paramètres non universels ;

4 - les littéraux positifs qui correspondent à un prédicat d'événement interne de base à paramètres non universels ;

5 - les littéraux négatifs "simples", càd qui ne correspondent pas à un prédicat d'événement interne ;

6 - les littéraux négatifs qui correspondent à un prédicat d'événement interne dérivé à paramètres non universels ;

7 - les littéraux négatifs qui correspondent à un prédicat d'événement interne de base à paramètres non universels.

Cet ordre n'est pas définitif puisqu'il est étroitement lié au fonctionnement de l'interpréteur PROLOG, d'où la décision d'opter pour un module séparé, facilement adaptable à d'éventuelles modifications.

V.3. Le prototype : conclusion

Ces quelques pages consacrées à l'implémentation de la méthode des événements internes ont montré que la méthode représente bien davantage que de simples raisonnements mathématiques : directement implémentable sous forme de procédures PROLOG, elle pourrait parfaitement être intégrée à un véritable système de gestion de bases de données déductif.

Bien entendu, la méthode est nouvelle, de nombreuses améliorations et extensions sont nécessaires, elles sont actuellement en cours de développement.

VI. CONCLUSION

Durant tout ce mémoire consacré aux bases de données déductives et à une de leur méthode de gestion particulière qu'est la méthode des événements internes, nous avons vu que les bases de données déductives, bien que comportant certaines similitudes avec les bases de données relationnelles, ont été conçues selon une philosophie tout à fait innovatrice : les tables relationnelles sont toujours présentes mais elles sont accompagnées de règles de déduction de sorte que l'ensemble contienne toutes les informations dont on dispose à propos du monde modélisé. Cet ensemble d'informations est exploité à l'aide de méthodes d'inférence diverses qui offrent les outils nécessaires à une certaine forme de raisonnement sur les données.

Les bases de données déductives répondent donc non seulement à toutes les fonctionnalités des bases de données relationnelles mais en plus, elles facilitent la tâche de l'utilisateur puisqu'elles lui permettent d'exprimer son raisonnement dans un formalisme fort proche du langage humain... Que souhaiter de plus ? Tout simplement des résultats pratiques.

Les chercheurs qui étudient différents problèmes des bases de données déductives ont beau prôner, à juste titre, la qualité des résultats obtenus, aucun SGBD déductif n'est à ce jour disponible sur le marché ; pourquoi cela ?

Différents facteurs interviennent dans l'explication de l'état d'avancement des quelques prototypes existants ([Vaghani90] [Ling-Goh90]). Le plus déterminant de ces facteurs est sans aucun doute la réelle difficulté que représente la conception d'un SGBD déductif. Nous avons vu quelques uns des problèmes spécifiques à un SGBD déductif, il en existe bien d'autres.

De plus, une question qu'il peut être intéressant de se poser est de savoir si les SGBD déductifs répondent toujours, à l'heure actuelle, à une réelle demande du marché. Une telle question risque de surprendre le lecteur car tout au long de ce mémoire, les bases de données déductives ont été présentées comme suite logique des bases de données relationnelles. Leur intérêt n'est bien entendu pas à mettre en doute, les bases de données déductives offrent nettement plus de possibilités que les bases de données relationnelles !... Peut-être, mais il ne faut pas oublier que les bases de données relationnelles ont évolué elles-aussi, autours du SGBD proprement dit se sont développés de nombreux outils aux fonctionnalités diverses, libérant l'utilisateur de toute tâche de programmation fastidieuse.

Les SGBD relationnels ne cessent d'évoluer pour répondre à de nouveaux besoins et les SGBD déductifs, dans leur tentative de se substituer aux SGBD relationnels sur un marché déjà bien établi, risquent de rester toujours une étape en retard : à quand les premiers SGBD déductifs bénéficiant de mécanismes de reprise après incident, supportant les attributs multivalués, ... ?

Je pense, mais ceci est un avis personnel, que si les SGBD déductifs veulent répondre un jour à un réel besoin, ils doivent avant tout se spécialiser. Les opportunités de développement des SGBD déductifs sont réelles mais dans des domaines spécifiques tels que l'intelligence artificielle où l'intérêt est davantage porté sur la notion de connaissances et de gestion de connaissances, plutôt que sur la gestion de grandes quantités de données à consulter en une fraction de seconde...

VII. BIBLIOGRAPHIE

[Abiteboul90] : S.Abiteboul, "Toward a deductive object-oriented database language". Data and Knowledge engineering 5, 1990.

[Ansi86] : American National Standard for Information Systems, "Database Language SQL", ANSI X3.135-1986, Octobre 1986.

[Bancil-Ramak86] : F.Bancilhon, R.Ramakrishnan, "An amateur's introduction to recursive query processing strategies", Proc. ACM SIGMOD Int. Conf. on Management of data. Washington DC, May 1986.

[Barbic90] : F.Barbic, R.Maiocchi, B.Pernici, "Automatic deduction of temporal information", Int. Workshop on the Deductive Approach to Information Systems and Databases, Catalogne, Octobre 1990.

[Blanco-Illa90] : J.M.Blanco, A.Illarramendi, "Semantic query optimization : from DBMS to DKB". International Workshop on the Deductive Approach to Information Systems and Databases, S'Agaro (Catalogne), Octobre 1990.

[Bry-Manthey90] : F.Bry, R.Manthey : "Deductive Databases". Publicaton interne de l'European Computer-Industry Research Centre (ECRC), LPSS'1990.

[Bry90] : F.Bry, "Query evaluation in recursive databases : bottom-up and top-down reconciled". Data and Knowledge engineering, 5, 1990.

[Caved-Deck90] : L.Cavedon, H.Decker, "A Weak Allowedness Condition that Ensures Completeness of SLDNF-Resolution". Int Workshop on the Deductive Approach to Information Systems and Databases, Catalogne, Octobre 1990.

[Cholvy89] : L.Cholvy, "Mises à jour dans les bases de connaissances (Une approche syntaxique)". Journées nationales de PRC_BD3. Genève, Septembre 1989.

[Cholvy90] : L.Cholvy, "Answering queries addressed to a rule base". Revue de l'Intelligence Artificielle, vol.1, 1990.

[Clocksin] : W-F.Clocksin, C.S.Mellish, "Programming in Prolog", 3ème édition, Livre, Springer-Verlag.

[Codd70] : E.F.Codd, "A Relational Model for Large Shared Data Banks". Communications of the ACM, Vol.13, N°6, Octobre 1970.

[Date84] : C.J.Date, "Database : A Primer", Livre, 4ème édition, Addison Wesley Readings, Août 1984.

[Date86] : C.J.Date, "Relational Databases : selected writings", Livre, Addison Wesley, 1986.

[Date86] : C.J.Date, "An Introduction to Database Systems", Livre, Vol.1, 4ème édition, Addison Wesley Readings, 1986.

[Decker88] : H.Decker, "Domain-independent and Range-restricted Formulas and Deductive Databases". Séminaire de programmation en logique, 25-27 Mai, Trégastel, France 1988.

[Decker90] : H.Decker, "Drawing Updates From Derivations". Publication interne ECRC (projet ESPRIT). March 1990.

[Gallaire84] : H.Gallaire, J.Minker, J-M.Nicolas : "Logic and Databases : A Deductive Approach". Computing Surveys, Vol. 16, N°2, June 1984.

[Gall-Nicol] : H.Gallaire, J-M.Nicolas, "Logic and Databases : An Assessment" ; publication de l'ECRC, Munich.

[Gard-Vald] : G.Gardarin, P.Valduriez, "Relational databases and knowledge bases", Livre, Addison-Wesley.

[Gard-Vald90] : G.Gardarin, P.Valduriez : "SGBD Avancés : Bases de données objets, déductives, réparties". Livre, Eyrolles 1990.

[Hainaut86] : J-L.Hainaut, "Conception assistée des applications informatiques - 2. Conception de la base de données", Masson, Presse universitaire de Namur, 1986.

[Hainaut88] : J-L.Hainaut, "Introduction à la théorie relationnelle des bases de données", syllabus Institut d'Informatique, FUNDP, Mars 1988.

[Helman-Veroff] : P.Helman, R.E.Veroff, "Designing deductive databases", University of Mexico, Technical Report N°CS86-5.

[Kakas-Manca90] : A.C.Kakas, P.Mancarella, "Databases Updates Through Adbuction". Proceedings of the 16th VLDB conference, Brisbane, Australia, 1990.

[Kakas90] : A.C.Kakas, "Belief Revision For Deductive Databases". Int. Workshop on the Deductive Approach to Information Systems and Databases, Octobre 1990.

[Kifer86] : M.Kifer, "A Framework for an Efficient Implementation of Deductive Databases". State university of New York, technical report 86/4.

[Ling-Goh90] : T-W.Ling, C-H.Goh, "Yet Another Deductive Database Implementation", Technical Report, National University of Singapore, Discs publication N°TRA7/90, July 1990.

[Minker86] : J.Minker, "Perspectives in Deductive Databases". The journal of logic programming, N°5, 1988.

[Nytro90] : O.Nytro, "Optimizing deductive databases using integrity constraints", Int. Workshop on the Deductive Approach to Information Systems and Databases, Catalogne, Octobre 1990.

[Olive90] : A.Olive, "The internal events method for integrity checking in deductive databases", publication interne UPC, Barcelone, March 1990.

[Pastor90] : J.A.Pastor, "The internal events method for integrity constraints enforcement in deductive databases". Int. Workshop on the Deductive Approach to Information Systems and Databases, Catalogne, Octobre 1990.

[Przymusinski] : C.Przymusinski, "On the Declarative Semantics of Deductive Databases and Logic Programs", Departement of Mathematical Science, University of Texas, El Paso, TX (Référence précise inconnue).

[Reiter78] : R.Reiter, "On closed world databases", Logic and Databases, Gallaire et Minker, Eds Plenum, New York, p56-76, 1978.

[Robi65] : J.A.Robinson, "A machine oriented logic based on the resolution principle", Journal of the ACM 12, 1965.

[Ross90] : K.A.Ross, "Modular Stratification and Magic Sets for DATALOG Programs with Negation". ACM 089791-352-3/90/0004/0161. 1990.

[Sancho90] : M.R.Sancho, "Deriving an internal events model from a deductive conceptual model". Int. Workshop on the Deductive Approach to Information Systems and Databases, Catalogne, Octobre 1990.

[Sripada88] : S.M.Sripada, "A logical framework for temporal deductive databases". Proceedings of the 14th VLDB Conference, Los Angeles, California 1988.

[Teniente-Olive90] : E.Teniente, A.Olive, "The internal events method for view updating in deductive databases", publication interne UPC, département LSI, Barcelone, Septembre 1990.

[Teniente91] : E.Teniente, "The internal events methods for view updating in deductive databases". Publication interne, UPC, département LSI, Barcelone, 1991.

[Tomasic88] : A.Tomasic, "View update annotation in definite deductive databases", Proc. ICDT88, Springer, 1988.

[Ull82] : J.D. Ullman, "Principles of database systems", Computer Science press Rockville, 2ème édition, 1982.

[Vaghani90] : J.Vaghani, K.Ramamohanarao, D.B.Kemp, Z.Somogyi, P.J.Stuckey, "Design Overview of the Aditi deductive database system". Proc. of the Far-East Workshop on Future DBMS, Melbourne, April 1990.

[VanLams89] : Note du cours d'intelligence artificielle du Professeur A. van Lamsweerde, 2ème licence, FUNDP (institut d'informatique).

[vandeRiet90] : R.P.van de Riet, "Introduction to the Special Issue on deductive and object-oriented databases". Data and Knowledge Engineering 5, 1990.

[Vanhent85] : P. Vanhentenryck, "Logique et bases de données", mémoire de fin d'études, FUNDP, Institut d'informatique, 1985.