THESE

presentée a

l'université des sciences et de la technologie Houari Boumediene

pour obtenir

le grade de magister en informatique

par
BOUKRA ABDELMADJID
sujet de la thése

CONTRIBUTION A L'ELABORATION D'UN S.G.B.D DEDUCTIF

soutenue le 26 mars 1989 devant la commission composée de :

Mr C. B. BEN YELLES	professeur a l'USTHB	president
Mr S. A. LARIBI	professeur a l'USTHB	rapporteur
Mr A. AINOUCHE	professeur a l'USTHB	
Melle F. AZROU	chercheur au CERIST	examinateurs
Melle A. MOKHTARI	chargée de cours a l'USTHB	

REMERCIEMENTS

Je remercie :

Monsieur C.B. Ben yelles professeur à l'U.S.T.H.B. , de m'avoir fait l'honneur de présider ce jury,

Monsieur S.A. Laribi professeur à l'U.S.T.H.B , qui m'a accueilli dans son équipe (SIGOAD) et qui m'a dirigé tout au long de ce travail,

Monsieur A. Ainouche professeur à l'U.S.T.H.B.,
mademoiselle A.Mokhtari chargée de cours à l'U.S.T.H.B.
et mademoiselle F.Azrou chercheur au CERIST, pour avoir accépté de juger mon travail,

Mademoiselle N.Madani chargée de cours à l'U.S.T.H.B. ,
Messieurs J.M. Nicolas professeur à ECRC (Munchen) ,
A. Guessoum chercheur à l'université de Bristol,
N. Djeddi chercheur à Toulouse ,
M. Kara chercheur à Leeds,
et M. Alia chercheur à Toulouse, pour l'aide qu'ils m'ont apporté en particulier dans la recherche bibliographique.

Je remercie enfin tous les membres de l'équipe SIGOAD pour leurs fructueuses discussions tout au long de mes exposés.

RESUME:

Cette recherche concerne l'intégration de la déduction dans un SGBD conventionel . Cette déduction peut intervenir à deux niveaux. Elle peut intervenir à la consultation de la base (on parle alors de dérivation), comme elle peut intervenir lors des mises à jours (on parlera de génération). Nous nous sommes intéressés à cette dernière.

Le composant déductif élaboré travaille donc par génération et veuille à ce que la base de données soit à tout instant un modèle pour l'ensemble des formules de la base (clauses de Horn constituant les connaissances de la base).

MOTS CLES

Base de données déductive , clause de Horn , génération , dérivation , interprétation , compilation , récursivité , déduction .

SOMMAIRE

INTRODUCTION

Le domaine des bases de données est l'un de ceux qui ont rapidement bénéficié des développements de l'intelligence artificielle. En effet actuellement, on essaie d'apprendre aux SGBD à raisonner et déduire. Les concepteurs de SGBD s'attaquent désormais au problème de l'extension des SGBD à des fonctions de déduction. Les deux approches envisageables sont respectivement l'extension des travaux effectués en intelligence artificielle, et plus précisément en systèmes experts, à des fonctions de stockage et gestion de gros volumes de données, et l'extension des SGBD au traitement des règles déductives. Nous nous intéressons dans cette thèse à la seconde approche .

La réalisation du processus de déduction pose principalement deux problèmes.

-1) La détermination des conditions d'arrêt pour tout chemin de derivation, et

-2) L'architecture de l'implantation d'une base de données déductive.

Plusieurs solutions on étés proposées au premier problème [CHANG.81 - CHANG.86 - GARD.86 - BANC.85a-b], mais elles ne sont pas satisfaisantes [BANC.86] car elles imposent des contraintes sur les règles de déduction et diminuent ainsi leur puissance d'expression. Aussi, avons nous opté dans cette thèse pour l'approche par génération qui atténue le problème de la condition d'arrêt et diminue le nombre de contraintes sur les règles de déduction. Le processus de déduction dans cette approche est activé au moment des mises à jour pour déduire et stocker les nouvelles informations.

Au second problème ont été proposées plusieurs solutions. Certains concepteurs partent d'un SGBD conventionnel et d'un langage de programmation logique et batissent une interface d'appel du SGBD depuis le langage [KOU.86-MARQ.84], d'autres essaient d'intégrer les fonctions de déduction dans un SGBD conventionnel, il existe une troisième approche qui consiste à étendre un langage de programmation logique (PROLOG en général) avec un SGF puis un SGBD.

L'approche la plus prometteuse du point de vue performances en déduction est sûrement l'intégration, car l'utilisation d'un langage de programmation logique (PROLOG), qui lui même n'est pas performant entraine un SGBD peu performant. Nous avons donc opté pour l'intégration dans l'architecture.

Malgré la solidité des fondements théoriques des bases de données déductives, aucun SGBD déductif n'a encore été commercialisé , néanmoins , plusieurs prototypes expérimentaux ont été élaborés tels que DEDUC2 (IBM San José) [CHANG.78], BDGEN (CERT Toulouse) [NICO.83], PROSQL (IBM York Town Heights). Le domaine est donc encore ouvert.

Le présent document est composé de six chapitres.

Dans le premier chapitre, nous passons en revue quelques notions de la logique mathématique ainsi que la formalisation des bases de données par la logique mathématique.
Le chapitre 2 donne une définition formelle aux bases de données déductives et cite les différentes approches architecturales possibles. La mise en oeuvre d'un SGBD déductif fait quand à elle, l'objet du chapitre 3. Les problèmes des cycles et de la récursivité , sont discutés au chapitre 4.
Le chapitre 5 décrit quelques algorithmes proposés par différents auteurs pour répondre à des requêtes en présence de règles récursives .
Enfin, le chapitre 6 présente la méthode adoptée et les détails de l'implémentation.

CHAPITRE 1
LOGIQUE ET BASES DE DONNEES

1.1. QUELQUES NOTIONS DE LA LOGIQUE MATHEMATIQUE

Comme pour tout système formel, la logique mathématique repose sur un langage objet, une sémantique ou interprétation des formules dans ce langage et une théorie de la preuve .[GAL.84]

1.1.1. LANGAGE

Nous utiliserons pour langage objet, un langage du premier ordre comme celui du calcul des prédicats du premier ordre .
Les symboles primitifs sont :

* les symboles de variables, de constantes, de fonctions et de prédicats ,
* les connecteurs logiques usuels ,
* et les quantificateurs .
* et les parenthèses

Pour représenter les constantes, nous utiliserons les lettres minuscules du début de l'alphabet (a,b,c ...).
Pour représenter les variables , nous utiliserons les lettres minuscules de la fin de l'alphabet (u,v,w...) et nous utiliserons les lettres telles que (f,g,h ...) pour représenter les fonctions

1.1.1.1.Terme

Un terme est défini de la manière suivante :

* une variable est un terme,
* une constante est un terme,
* si f est un symbole de fonction d'arité n et si $t_1,t_2,...t_n$ sont des termes alors $f(t_1,t_2,...,t_n)$ est un terme .

Remarque :

En général, un terme dans le contexte des bases de données est soit une constante, soit une variable.

1.1.1.2. Formule atomique

Si P est un symbole de prédicat d'arité n et si $t_1, t_2, \ldots t_n$ sont des termes alors $P(t_1, t_2, \ldots, t_n)$ est une formule atomique .

Remarque :

Une formule atomique et sa négation sont appelées littéraux.

1.1.1.3. Formule bien formée (F.B.F.)

Une formule bien formée est définie de la manière suivante:

* une formule atomique est une F.B.F,
* si W_1 et W_2 sont deux F.B.F. alors :
  $\neg W_1$, W_1 & W_2 , W_1 v W_2 , $W_1 \longrightarrow W_2$, $W_1 \longleftrightarrow W_2$
  sont des F.B.F.

1.1.1.4. F.B.F fermée

Une F.B.F. est dite fermée si elle ne contient aucune variable libre (c.a.d. contient uniquement des constantes ou des variables quantifiées)

1.1.1.5. F.B.F. en forme normale prénexe (F.N.P)

Une formule est en F.N.P si tous les quantificateurs sont au début de la F.B.F.

Exemple :

La F.B.F correspondant a la phrase suivante " Tout enseignant a un diplome ", est :

(1) $\forall x \forall y$ enseign(x,y) $\longrightarrow \exists z$ diplome(x,z)

Il est possible de placer tous les quantificateurs au début de la formule pour former la F.N.P.[CHANG.73]

Quand ceci est fait on obtient la formule (2)

(2) $\forall x \forall y \exists z$ $\neg$enseign(x,y) V diplome(x,z)

1.1.1.6. Forme normale de Skolem, fonction de Skolem, constante de Skolem

Une F.N.P. est en F.N. de Skolem lorsque tous les quantificateurs existentiels sont éliminés en remplaçant les variables qu'ils quantifient par des fonctions arbitraires dont les variables sont celles qui précédent le quantificateur dans la formule. Ces fonctions sont appelées les fonctions de Skolem. Une fonction à zéro arguments est appelée constante de Skolem.

La forme normale de skolem de (2) est :

$$(5)\quad \forall x\ \forall y\ \neg\ \text{enseign}(x,y)\ v\ \text{diplome}(x,f(x,y))$$

où les variables quantifiées existentiellement sont remplacées par une fonction de Skolem.

Quand une F.B.F fermée est en F.N. de Skolem , tous les quantificateurs au début de la F.B.F peuvent être éliminés car toutes les variables sont universellement quantifiées. La formule (5) peut être écrite sous la forme :

$$(7)\quad \neg\ \text{enseign}(x,y)\ v\ \text{diplome}(x,f(x,y))$$

1.1.1.7. Clause

Une clause est une disjonction de littéraux dont les variables sont implicitement quantifiées universellement .

Exemple:

$\neg A_1\ v\ \ldots\ v\ \neg A_m\ v\ B_1\ v \ldots\ v\ B_n$ où A_i et B_j sont des littéraux positifs est une clause qui peut s'écrire :

$$A_1\ \&\ \ldots\ \&\ A_m \longrightarrow B_1\ v\ \ldots\ v\ B_n$$

1.1.1.8. Clause de Horn

Dans une clause de la forme :

$$A_1\ \&\ \ldots\ \&\ A_m \longrightarrow B_1\ V\ \ldots\ V\ B_n$$

chaque fois que n=0 ou 1, la clause est dite de Horn.

Si m et n sont égaux à zéro , la clause est appelée clause vide Une clause (un littéral) dans laquelle aucune variable n'apparait est appelée clause "ground" (littéral "ground").

1.1.2. LA SEMANTIQUE : INTERPRETATION ET MODELE

En sémantique nous nous intéressons à l'interprétation, où une interprétation d'un ensemble de F.B.F. consiste en la spécification d'un ensemble non vide E (ou domaine) à partir duquel on puise les valeurs des variables et des constantes:

* à chaque symbole de fonction d'arité n, on fait correspondre une fonction de E^n ——> E.

* à chaque symbole de prédicat d'arité n, on fait correspondre une relation sur E^n

1.1.2.1. Interprétation d'une F.B.F fermée et ouverte

Dans une interprétation sur un domaine E, une F.B.F fermée est soit vraie soit fausse , par contre une F.B.F ouverte avec n variables libres (n>1) , détermine un ensemble de n tuples (c.a.d. une relation) sur E^n. Chacun de ces n tuples est tel que lorsqu'on substitue ses composants aux variables libres de la F.B.F. ouverte , la F.B.F. fermée obtenue est vraie .

Si l'ensemble des n tuples est vide , alors la F.B.F. est dite fausse , et si l'ensemble des n tuples coincide avec E^n, alors la F.B.F. est dite vraie.

1.1.2.2. Evaluation de la valeur de vérité d'une F.B.F fermée

Si R est la relation associée au symbole de prédicat P, d'arité n, alors $P(e_1,\ldots,e_n)$ est évalué à vrai si $\langle e_1 , \ldots , e_n\rangle \in R$ sinon il est évalué à faux .

Si W_1 et W_2 sont deux F.B.F. fermées alors :

* $\neg W_1$ est vrai si W_1 est faux sinon , il est faux .
* W_1 & W_2 est vrai si W_1 et W_2 sont vrais sinon il est faux.
* W_1 ——>W_2 est vrai si , W_1 est faux ou W_2 est vrai , sinon, il est faux .

Si x est une variable dans W1 ,alors :

Vx $W_1(x)$ est évalué à vrai si pour tout élément e de E $W_1(e)$ est vrai , sinon , il est évalué à faux .

Ix $W_1(x)$ est évaluée à vrai, s'il existe un élément e dans E, tel que $W_1(e)$ est vrai , sinon , il est évalué à faux

1.1.2.3. Modèle d'un ensemble de F.B.F.

Un modèle d'un ensemble de F.B.F. est une interprétation dans laquelle toutes les F.B.F. de cet ensemble sont vraies .

1.1.3. SYNTAXE : THEORIE DU PREMIER ORDRE

Le calcul des prédicats est un système formel qui a comme langage objet un langage du premier ordre , un ensemble de schémas d'axiomes (axiomes logiques) et deux règles d'inférence:

* le modus ponens
* la généralisation

1.1.3.1. Théorie du premier ordre

D'autres F.B.F. peuvent être ajoutées aux axiomes . Les nouveaux axiomes sont appelés axiomes non logiques ou propres . Une théorie du premier ordre est caractérisée par ses axiomes propres [GAL.84]

Un ensemble d'axiomes propres peut être par exemple :

$$\text{Homme}(Ali)$$
$$\forall x\ \text{Homme}(x) \longrightarrow \text{Mortel}(x) .$$

Un modèle d'une théorie est une interprétation dans laquelle tous les axiomes sont vrais , les axiomes logiques sont en fait choisis vrais dans toute interprétation .
Pour la théorie ci dessus , l'interprétation suivante donne un modèle :

$$\text{Homme}(Ali) = \text{vrai}$$
$$\text{Mortel}(Ali) = \text{vrai}$$

1.1.3.2. Dérivabilité d'une F.B.F

Une F.B.F. w est dérivable d'un ensemble W de F.B.F. dans une théorie T (noté $W \vdash w$), si et seulement si w est déductible de W et des axiomes de T en appliquant un nombre fini de fois les régles d'inférences .
En utilisant la régle d'inférence du modus ponens , on obtient de la théorie ci dessus la dérivation suivante :

$$\text{Mortel}(Ali)$$

Si W est vide , alors w est un théorème de T (noté T $\vdash$ w)

Des régles d'inférence autres que le modus ponens et la généralisation peuvent être utilisées pour dériver des théoremes. En fait beaucoup de techniques de démonstrateurs de théorémes sont basées sur la régle d'inférence du principe de résolution de Robinson [ROB.65] qui s'applique aux F.B.F. sous forme clausale

1.1.3.3. Principe de résolution de Robinson.

Le principe de résolution de ROBINSON , est une régle d'inférence qui permet de dériver une nouvelle clause à partir de deux clauses données , et de plus , la clause dérivée est satisfaisable si les deux clauses sont satisfaisables .

Dans le contexte des bases de données , le principe peut être décrit comme suit :

de C_1 : $\neg P(a,b,c) \vee Q(d,e)$ (c.a.d. $P(a,b,c) \rightarrow Q(d,e)$) et
de C_2 : $P(x,y,z) \vee R(a,b)$

On peut dériver la clause

C_3 : $Q(d,e) \vee R(a,b)$

La clause C_3 est trouvée en considérant les littéraux des deux clauses qui ont le même nom de prédicat , mais l'un négatif ,l'autre non. Le seul prédicat de ce type est P . On regarde si P(a,b,c) et P(x,y,z) peuvent être rendus identiques par une substitution des variables . On trouve la substitution suivante

$$\{a/x , b/y , c/z\} .$$

On élimine ensuite les deux littéraux rendus identiques par la substitution des deux clauses (on dit qu'on a unifié ces deux littéraux) , on obtient ensuite une disjonction des littéraux qui restent. On applique ensuite la substitution de l'unification à ces littéraux et on obtient la clause dérivée C_3. [CHAN.73 - LOVE.78]

1.2. LES BASES DE DONNEES VUES A TRAVERS LA LOGIQUE MATHEMATIQUE

1.2.1. Hypothèses gouvernant l'évaluation des requêtes.

Avant de considérer le formalisme des bases de données en termes de logique, mentionnons les hypothèses qui gouvernent

l'évaluation des requêtes dans une base de données .
Il y a trois suppositions :

* 1 . Hypothèse du monde fermé.

Cette hypothèse est aussi connue sous le nom " convention pour les informations négatives ". Elle dit que si des faits ne sont pas connus comme étant vrais , alors ils sont supposés faux (c.a.d. $\neg R(e_1,\ldots,e_n)$ est supposé vrai ssi le tuple $\langle e_1,\ldots,e_n\rangle$ ne se trouve pas dans la relation R).

* 2 . Hypothèse des noms uniques.

Cette hypothèse dit que des individus avec des noms différents , sont différents

* 3 . Hypothèse de la fermeture du domaine .

Cette hypothèse dit qu'il n'existe pas d'autres individus que ceux de la base de données .

Le processus d'évaluation dans un SGBD travaille implicitement sous ces hypothèses . Ces suppositions sont rendues explicites à l'aide de la formalisation logique .

1.2.2. Les deux façons de voir les bases de données du point de vue logique .

Une base de données peut être considérée du point de vue logique de deux manières différentes :[GAL.78]

a / comme une interprétation d'une théorie du premier ordre .
b / comme une théorie du premier ordre .

Quand on la voit du point de vue interprétation , les requêtes et les contraintes d'intégrité sont des formules qui sont évaluées sur l'interprétation en utilisant la définition sémantique de la vérité.

Du point de vue théorie , les requêtes et les contraintes sont considérées comme des théorémes à démontrer.

1.2.2.1. Vue théorie du modèle .[REIT.84]

Soit B une instance d'une base de données relationnelle . B consiste en un ensemble de relations (c.a.d. une relation R pour chaque schéma de relation $R(A_1,...,A_n)$) et un ensemble de contraintes d'intégrité CI.

Soit D l'union des domaines de tous les attributs qui apparaissent dans le schéma de relation . Définissons un langage de premier ordre L :

* Un symbole de prédicat R à n places pour chaque relation d'arité n dans B,

* Un ensemble de symboles de constantes , un pour chaque élément dans D ,

* Le langage est supposé ne pas contenir de symboles de fonction.

B peut être vue comme une interprétation des formules du langage de premier ordre du calcul des prédicats du premier ordre et les formules de L peuvert être évaluées dans cette interprétation comme suit :

* Les variables varient sur le comaine D ,
* $R(e_1, ... ,e_n)$ est vraie ss. $< e_1,...,e_n> \in R$.

Le langage peut être étendu pour inclure les opérateurs de comparaison arithmétiques (>,<,= ...) comme des symboles de prédicats particuliers , on leur associe leur interprétation usuelle.

Si les contraintes d'intégrité CI sont exprimées comme des formules de L , alors la base de données B sera dans un état valide ssi chaque contrainte dans CI est évaluée à VRAI dans B (c.a.d. B est un modèle de CI)

L'évaluation d'une formule logique (sur une interprétation) est faite en accord avec les hypothèses suivantes :

* Hypothèse du monde fermé ,
* Hypothèse de l'unicité des noms,
* Hypothèse de la fermeture du domaine .

1.2.2.2. Vue théorie de la preuve.

La vue théorie de la preuve d'une base de données est obtenue en construisant une théorie T qui admet B comme unique modèle, alors pour toute F.B.F w dans T, T $\vdash$ w ssi w est vraie dans B.

Le processus définissant T consiste à préciser ses axiomes propres qui sont de trois types [REIT.84]

a) Les assertions

Pour n'importe quelle relation R dans B et n'importe quel tuple $< e_1,\ldots,e_n> \in R$ on a un axiome $R(e_1,..,e_n) \in T$

b) Les axiomes de particularisation

b.1 . Les axiomes de complétude

Il y a un tel axiome pour toute relation R dans B .

Si $< e'_1,\ldots,e^{\wedge}_1>$, $\ldots,<e'_p, \ldots , e^{\wedge}_p >$ sont des tuples dans R alors :

$$\forall x_1,\ldots,x_n \; (R(x_1,\ldots,x_n) \longrightarrow (x_1=e'_1 \;\&\; \ldots \;\&\; x_n=e^{\wedge}_1) \; V \; \ldots \; .. V \; (x_1=e'_p \;\&\; \ldots \;\&\; x_n=e^{\wedge}_p)$$

Cet axiome dit que les seules valeurs de tuples que peut avoir la relation R sont :

$$<e'_1,\ldots,e^{\wedge}_1>, \; \ldots \; , \; <e'_p,\ldots,e^{\wedge}_p>$$

b.2 . L'axiome du nom unique.

Si $e_1,\ldots e_q$ sont tous des individus dans B , les axiomes du nom unique sont :

$$(e_1<>e_2),\ldots,(e_1<>e_q),(e_2<>e_3),\ldots,(e_{q-1}<>e_q).$$

b.3 . l'axiome de fermeture du domaine .

$$\forall x \; ((x=e_1) \; v \; (x=e_2) \; v \; \ldots \; v \; (x=e_q) \;).$$

c) Les axiomes de l'égalité

Ces axiomes spécifient les propriétés usuelles de l'égalité

* réfléxivité

$$\forall x \ (x=x)$$

* symétrie

$$\forall x \ \forall y \ (\ (x=y) \longrightarrow (y=x) \ )$$

* transitivité

$$\forall x \ \forall y \ \forall z \ (\ (x=y) \ \& \ (y=z) \longrightarrow (x=z) \ )$$

* principe de substitution des termes égaux .

$$\forall x_1, \ldots, \forall y_1, \ldots$$

$$(P(x_1, \ldots x_n) \ \& \ (x_1=y_1) \ \& \ldots \& \ (x_n=y_n) \longrightarrow P(y_1, \ldots, y_n))$$

CHAPITRE 2
BASES DE DONNEES DEDUCTIVES

2.1. INTRODUCTION .

Une base de données déductive est une base de données dans laquelle de nouveaux faits peuvent être dérivés à partir de faits qui étaient introduits explicitement. Nous considérons ici les bases de données du point de vue théorie de la preuve comme une théorie du premier ordre

2.2. DEFINITION DES BASES DE DONNEES DEDUCTIVES .

En général, nous considérons qu'une base de données déductive consiste en un ensemble fini de constantes et un ensemble de clauses de premier ordre sans symbole de fonction.

Il est important de constater qu'en général , une clause est considérée sans symbole de fonction et ce pour deux raisons principales :

1) lors du processus d'unification , si la fonction n'est pas bijective , on ne saura pas quelle valeur affecter aux variables

Exemple:

soient les schémas de relation

R_1(A , B , C)
et R_2(D , f(E,G))

et la règle

$R_1(X , Y , Z) \longrightarrow R_2(X , f(Y,Z))$

Si on a le fait $R_2(a,b)$ et qu'on veuille faire un chainage arrière (il en est de même pour le chainage avant), on doit chercher $f^{-1}(b)$ pour trouver les valeurs de Y et Z et si la fonction f n'est pas bijective , on ne saura pas quelle valeur affecter à Y et à Z

2) même si la fonction est bijective , rien ne nous assure que les valeurs calculées par l'inverse de f , appartiennent au domaine des attributs.

La forme générale des clauses qui représenteront les faits et les lois déductives est :

$$P_1 \& P_2 \& \ldots \& P_k \text{ ---> } R_1 \vee \ldots \vee R_q$$

ce qui est équivalent à:

$$\neg P_1 \vee \ldots \vee \neg P_k \vee R_1 \vee \ldots \vee R_q$$

La conjonction des P_i est appelée membre gauche de la clause et la disjonction des R_j est appelée membre droit.
Les termes qui sont les arguments de P_i et P_j sont soit des constantes, soit des variables (puisque les clauses que nous considérons sont sans symbole de fonction)

Nous discuterons rapidement les différents types de clauses en fonction des valeurs de k et q :[MINK.83].

* type-1:

k=0 et q=1
Les clauses ont la forme

$$\text{————>} P(t_1,\ldots,t_m)$$

a) si les ti sont des constantes, on aura une assertion (ou un fait) dans la base de données .
   L'ensemble de toutes ces assertions pour un prédicat P correspond à une table dans la base de données relationnelle

b) quand quelques uns (ou tous) les t_i sont des variables, la clause correspond à un état général dans la base de données

   Exemple:

   Ancêtre (Adam,x) qui signifie que Adam est un ancêtre de tout individu dans la base de données

* type-2

k=1 et q=0
Les clauses ont la forme :

$$P(t_1,\ldots,t_m) \text{ ——>}$$

a) quand les t_i sont des constantes, on a un fait négatif
b) si quelques uns des t_i sont des variables, ceci peut être vu comme une contrainte d'intégrité.

* type-3

k>1 et q=0
Les clauses sont de la forme

$$P_1 \& \ldots \& P_k \longrightarrow$$

De telles clauses peuvent être vues comme des contraintes d'intégrité

Exemple :

$$\text{père}(x,y) \ \& \ \text{mére}(x,y) \longrightarrow$$

cette clause signifie qu'un individu ne peut pas être pére et mère en même temps.

* type-4

k≥1 et q=1
les clauses ont la forme

$$P_1 \ \& \ \ldots \& \ P_k \longrightarrow R_1$$

Cette clause peut être considérée soit comme une contrainte d'intégrité soit comme une définition du prédicat R1 en terme des prédicats $P_1 \ldots P_k$ (c'est une loi déductive)

* type-5

k=0 et q>1
Les clauses ont la forme

$$\longrightarrow R_1 \ v \ \ldots \ v \ R_q$$

Si les arguments des R_i sont des constantes, alors nous avons une assertion (c.a.d. toute combinaison des R_i est vraie mais on ne sait pas lesquelles sont vraies).

* type-6

k≥1 et q>1
les clauses ont la forme :

$$P_1 \ \& \ \ldots \ \& \ P_k \longrightarrow R_1 \ v \ \ldots \ v \ R_q$$

Cette clause peut être interprétée soit comme une contrainte d'intégrité soit comme une définition d'une donnée indéfinie

Exemple :

La contrainte d'intégrité qui dit que chaque individu a au plus deux parents peut être écrite comme :

$P(y_1,x_1)$ & $P(y_2,x_1)$ & $P(y_3,x_1)$ ——> $(y_1=y_2)$ v $(y_1=y_3)$ v $(y_2=y_3)$

Comme règle générale de déduction, on peut avoir :

parent (x,y) ——> mére(x,y) v pére(x,y)

Cette loi générale peut être interprétée aussi comme une contrainte d'intégrité

Enfin nous appellerons clause définie, une clause qui ne contient qu'un atome dans son membre droit .

2.3. DIFFERENTS TYPES DE BASES DE DONNEES DEDUCTIVES .

Nous distinguons deux types de bases de données déductives

* les bases de données déductives définies,
* les bases de données déductives indéfinies.

Les premières n'admettent pas de clauses du type 5 et 6, alors que les secondes les admettent.

2.3.1. Les bases de données déductives définies (BDDD) .

2.3.1.1. Définition formelle d'une BDDD.

Une base de données déductive définie consiste en :

1) Une théorie T dont les axiomes propres sont :

* axiomes 1:(axiomes de particularisation)

  - L'axiome de fermeture du domaine,
  - L'axiome du nom unique ,
  - Les axiomes de l'égalité,
  - Les axiomes de complétude .

* axiomes 2: (Les faits élémentaires)

C'est un ensemble de formules atomiques définies par des clauses du type

$$\longrightarrow P(c_1,\ldots,C_m)$$

* axiomes 3: (Les lois déductives)

C'est un ensemble de clauses du type

$$P_1 \;\&\; \ldots \;\&\; P_k \longrightarrow R_1$$

ou bien

$$\longrightarrow \text{ancêtre (Adam,x)}$$

2) Un ensemble de contraintes d'intégrité CI qui consiste en n'importe quelle formule fermée.

L'axiome de complétude pour un prédicat P n'est pas construit seulement à partir des faits relatifs à P (qui apparaissent dans l'axiome 2), mais aussi de la partie manquante " seulement si" de la clause définie définissant P (qui apparait dans l'axiome 3)

Exemple :

* Soit P un prédicat ayant les assertions suivantes dans T

$$P(c_i,c_j)$$
$$P(c_p,c_q)$$

et soient les deux lois suivantes

$$Q(x,y) \;\&\; R(y,z) \longrightarrow P(x,z)$$
$$S(x,y) \longrightarrow P(x,y)$$

L'axiome de complétude de P est:

$$P(x,y) \longrightarrow ((x=c_i) \;\&\; (y=c_j)) \vee ((x=c_p) \;\&\; (y=c_q)) \vee (Q(x,y) \;\&\; R(y,z)) \vee (S(x,y))$$

Un tel axiome de complètude permet de dériver le fait négatif $\neg P(d,e)$ à chaque fois que P(d,e) n'est ni dans l'axiome 2 ni dérivable par les axiomes 3. Donc de la base de données spécifiée on peut dériver $\neg p(c_i,c_p)$

Une réponse à la requête $W(x_1, \ldots, x_p)$ où $x_1 \ldots x_p$ sont des variables libres dans W , est l'ensemble des tuples.

$$< c_{i1}, \ldots, c_{ip} > \text{ tel que } T \vdash W(c_{i1},\ldots,c_{ip}).$$

Une base de données déductive obéit aux contraintes d'intégrité dans CI ssi pour toute formule F dans CI $T \vdash F$

Les lois déductives (dans les axiomes 3) qui impliquent une relation R fournissent une définition étendue de R .
Les tuples $<C_{i1}, \ldots, C_{im}>$ qui satisfont R ne sont pas seulement ceux tels que $R(C_{i1}, \ldots, C_{im})$ est un fait dans l'axiome 2 , mais aussi ceux tels que $R(C_{i1}, \ldots, C_{im})$ est dérivable à partir des lois déductives

Les relations qui sont définies conjointement par des lois déductives et des faits élémentaires dans une B.D.D. ,appelées relations dérivées, constituent une généralisation des relations définies comme vues dans les bases de données conventionnelles

Une vue est une relation non enregistrée dans la base de données, elle est définie en terme de relations de la base de données ou d'autres vues par une expression algébrique relationnelle. Une relation dérivée se réduit à une vue quand :

1) il n'y a pas de faits élémentaires dans les axiomes (2) qui sont relatifs à cette relation .

2) aucune loi déductive récursive ou cycle n'apparait parmi les lois déductives qui impliquent cette relation dans les axiomes (3).

Les axiomes de particularisation (Axiome-1) sont très longs à mettre en oeuvre et conduiraient, s'ils étaient implémentés tels quels, à des SGBD déductifs définis inefficaces .
La solution adoptée pour l'implémentation est de substituer aux axiomes de particularisation (axiomes 1), une méta règle adéquate; ceci conduit à une définition opérationnelle des B.D.D.D [CLAR.78]

2.3.1.2. Définition opérationnelle des B.D.D.D.

Les axiomes de particularisation peuvent être éliminés de la B.D.D.D et remplacés par une méta-règle de négation par échec.

La méta-régle de négation par échec dit que pour tout littéral positif P , $\vdash\!\!\!-\!\!\!- \neg P$ ssi $\vdash\!\!\!-\!\!\!\!/ P$ c.a.d. , l'échec de la preuve de P permet d'inférer $\neg P$.

Maintenant, une B.D.D.D. consiste en :

* Un ensemble d'axiomes
  - Axiomes 2 (faits élémentaires)
  - Axiomes 3 (lois déductives)

* Un ensemble de contraintes d'intégrité CI

* Une méta-règle : négation par échec

Remarque

La négation par échec fini généralise dans le cas des B.D.D, les faits négatifs dans les bases de données conventionnelles (Hypothèse du monde fermé)

2.3.2. Bases de données déductives indéfinies.(B.D.D.I)

Une B.D.D indéfinie différe de la B.D.D définie (en se référant à la définition opérationnelle) dans les axiomes 3 et de la méta-règle de la négation par échec.
La différence dans les axiomes 3 quoiqu'en apparence mineure , peut être substentielle. L'axiome 3' est défini comme suit

* Axiome 3': un ensemble de clauses définies ou indéfinies sans symbole de fonction.

Les méthodes et resultats de la B.D.D. définie ne sont plus utilisables ici.

En effet, considérons la B.D. formée d'un seul fait

oiseau(titi)

et de l'unique loi de déduction :

oiseau(x) ——> noir(x) v blanc(x)

puisque noir(titi) et blanc(titi) ne peuvent pas être prouvés, en appliquant la méta-règle de négation par échec on a:

⊢—— ¬blanc(titi)

⊢—— ¬noir(titi)

Alors que l'ensemble

{oiseau(x)--->noir(x) v blanc(x) , oiseau(titi) , ¬blanc(titi) , ¬noir(titi) } est inconsistant. [GAL.84]

Donc la négation par échec ne peut pas être utilisée , elle peut être étendue à la négation par échec généralisée comme suit:

Soit E l'ensemble de toutes les clauses positives non prouvables (éventuellement vide)alors :

¬P(x) ssi P(x) v C est non prouvable ∀ C ∈ E.[GALL.84]

Une base de donneés déductive indéfinie, consiste en :

1) Un ensemble d'axiomes T , ou T =(axiomes 2) U (axiomes 3')
2) Un ensemble de contraintes d'intégrité.
3) Une méta-règle :négation par échec généralisée décrite ci dessus.

2.4. Les approches architecturales.

Parmi les approches existantes, nous pouvons distinguer trois types :[BOCC.86] [JARK.84]

1/ Le premier type consiste à partir d'un SGBD classique et d'un langage de programmation logique à tendre vers un SGBD déductif.

2/ Le deuxième type consiste à partir d'un langage de programmation logique tel que PROLOG et à l'étendre tout d'abord avec un système de gestion de fichiers puis vers un SGBD.

3/ le troisième type consiste à intégrer les fonctions de déduction dans un SGBD classique .

L'extension d'un langage de programmation logique à un SGBD est une tache assez fastidieuse étant donné que la simple réalisation d'un SGBD classique nécessite beaucoup de temps. Aussi nous ne considérerons que les différentes approches du premier et troisième type.

2.4.1. Premier type

a/ Couplage faible.

Cette approche consiste à prendre un SGBD relationnel d'une part, un langage de programmation logique d'autre part (PROLOG en général) et à batir une interface d'appel du SGBD depuis le langage. L'utilisateur écrit ses programmes en PROLOG et invoque le SGBD en utlisant des prédicats prédéfinis du langage de programmation logique.

Exemple:

SELECT (liste d'arguments)
SELECT appellera l'ordre correspondant de SQL(par exemple)
liste d'arguments sera les paramétres de la requête SQL

b/ Couplage fort.

Cette approche consiste à modifier ou compléter un interpréteur de règles (l'interpréteur de PROLOG) de sorte à rendre invisible le SGBD. Le programmeur voit les tuples des relations stockées dans la base de données, comme des faits définis dans son programme .

L'interpréteur travaille alors comme une couche au dessus d'un SGBD existant et sait retrouver les faits pertinents dans la base de données .
Deux types d'implantation sont possibles .

b.1/ Couplage fort statique [BOC.86]

Ce couplage permet de ne pas modifier l'interprèteur PROLOG. On procède par une preévaluation pour déterminer les instances des prédicats bases de données nécessaires à l'exécution , ceci permet d'ajouter au programme les faits de la base permettant son exécution réelle .

b.2/ Couplage fort dynamique .

Ce couplage nécessite la modification de l'interpréteur, mais réalise une seule évaluation . Lorsqu'un prédicat base de données est invoqué , une question est envoyée au SGBD afin d'instancier le prédicat .

2.4.2. TROISIEME TYPE (l'intégration)

Cette approche consiste en l'intégration complète des mécanismes de déduction au SGBD. Ceci nécessite de maitriser le code du SGBD (ou de tout refaire), on modifie le SGBD en incluant la gestion d'une base de régles , un évaluateur de règles de nouvelles techniques d'évaluation des questions et des opérateurs spécialisés pour réaliser les opérations coûteuses (récursion)

Une telle approche permet d'espérer des interfaces integrées et de bonnes performances de déduction . Le langage externe offert à l'utilisateur n'est pas forcement PROLOG ou ses dérivés , il peut s'agir d'un langage orienté base de données.

CHAPITRE 3
MISE EN OEUVRE D'UN SGBD DEDUCTIF

Un SGBD déductif est un système qui permet de dériver de nouvelles informations à partir de celles introduites explicitement dans la base par l'utilisateur et ce au moyen des règles de production ou lois générales .

3.1.CONTRAINTES D'INTEGRITE ET REGLES DE DEDUCTION

Les lois déductives et les contraintes d'intégrité correspondent à la connaissance générale du monde modélisé par la base de données .

Etant donnée une loi générale , l'un des problèmes est de décider de mettre cette loi comme contrainte d'intégrité ou comme loi déductive .

Il n'y a pas de réponse finale à cette question .
Nous pouvons donner quelques suggestions proposées par Nicolas et Gallaire dans [GAL.78]

1) Nous avons vu que les clauses utilisées commes lois déductives étaient sans symbole de fonction , donc les connaissances générales qui contiennent des symboles de fonction seront traitées comme contraintes d'intégrité.

2) Pour éviter l'inconsistance avec l'hypothèse du monde fermé (CWA) , on retient comme lois déductives seulement les connaissances générales qui correspondent aux clauses définies

3) Puisque les clauses négatives (clauses de type P1 & ...Pn ⟶) ne produisent jamais de faits (sous la CWA) ,elles sont utilisées comme contraintes d'intégrité.

4) Les connaissances générales qui impliquent une relation qui est complètement définie indépendemment de la règle , ne produisent aucun nouveau fait utile (si elles sont utilisées comme lois déductives) et seront utilisées comme contraintes d'intégrité.

Exemple:

"L'age de toute personne est < 150"

$(\forall X \quad \forall Y) \quad (Age(X,Y) \longrightarrow (Y<150))$

Si on utilise cette connaissance comme loi déductive , elle nous donnera des faits erronés (ex : 180 < 150) ou redondants (ex: 35<150)

Dans une BD conventionnelle, toutes les lois générales (ou connaissances) sont utilisées comme contraintes d'intégrité pour vérifier la validité de l'état de la base de données. Par contre, dans les B.D.D., certaines de ces lois sont utilisées comme règles de déduction pour dériver de nouveaux faits des faits explicitement introduits dans la base de données .

Exemple :

Considérons une base de données qui contient les deux relations

PERE et GRAND-PERE

et la loi générale suivante:

"le pére d'un pére est un grand pére "

qui s'écrit :

L: PERE(x,y) & PERE(y,z) ——> GRAND-PERE(x,z)

Supposons qu'à un moment donné l'état de la base soit le suivant :

PERE

P	F
Ali	Omar
Omar	Reda

GRAND-PERE

GP	PF
Samir	Khaled

figure 2

Si la règle L est considérée comme une contrainte d'intégrité, l'état précédent de la base est consideré comme non valide, car la relation GRAND-PERE ne contient pas le tuple < Ali , Reda >.

Dans ce cas on a supposé que les tuples qui satisfont la relation GRAND-PERE sont exactement ceux qui apparaissent dans son extension .

En supprimant cette hypothèse , on peut considérer que les tuples qui satisfont la relation GRAND-PERE ne sont pas seulement ceux qui apparaissent dans son extension mais aussi les tuples qui peuvent être déduits de L et de la relation PERE , où L est considérée maintenant comme une règle de déduction .

Ces règles constituent donc une définition des relations dites déduites .

Une relation déduite peut ainsi être définie par une ou plusieurs règles , chaque règle étant formée d'une ou plusieurs requêtes bases de données .

La définition d'une relation peut faire intervenir des relations initiales , d'autres relations déduites ou la relation déduite elle même. Ceci permet entre autres des définitions récursives . Se pose alors le probléme de l'utilisation des règles par le système . Il peut être résolu par génération ou par dérivation.

3.2.UTILISATION DES REGLES PAR LE SYSTEME

3.2.1. La génération .[NICO.83a-b]

Cette méthode consiste à utiliser les règles de déduction lors de la phase de mise à jour des informations afin de générer puis de stocker des informations déductibles, on peut dire que cette méthode réalise la maintenance automatique de contraintes d'intégrité complexes.

La mise en oeuvre de cette méthode en présence de régles récursives ne pose pas de problémes particuliers .

3.2.2. La dérivation.

Cette méthode consiste à mettre en oeuvre les règles de déduction lors de la phase d'interrogation , afin de générer temporairement les informations nécessaires à l'évaluation de la requête.

Cette méthode pose le problème de la mise en oeuvre de l'arrêt du processus d'inférence. En effet, les techniques usuelles d'inférence utilisées en intelligence artificielle peuvent mener au parcours de chemins de dérivation potentiellement infinis.
Cette approche conduit à deux classes de méthodes : l'interprétation et la compilation

3.2.2.1. Méthode interprétée.[MINK.75-78]

Dans cette approche, les processus de résolution et de recherche des informations dans la base de faits sont intimement liés. La réponse à une requête mettra en jeu de façon simultanée, les axiomes et les faits impliqués .

Cette méthode provenant des techniques utilisées en intelligence arificielle est similaire à celle suivie dans les démonstrateurs de théorèmes et est également celle adoptée par le langage PROLOG.

Le problème initial est décomposé par chainage arrière en sous problèmes qui seront résolus par des accés successifs à la base de données et dont la réunion des réponses fournies , formera la réponse complète à la requête .

3.2.2.2. Méthode compilée [CHANG-81]

Cette méthode transforme une requête dont l'évaluation met en jeu un ensemble de relations implicites (fictives), en un programme de requêtes n'invoquant que des relations explicitement stockées dans la base de données.

Cette approche permet de dissocier totalement les processus de résolution et d'évaluation des requêtes . Ceci autorise l'utilisation du processeur d'optimisation de questions des SGBD relationnels classiques , les accès à la base de données pouvant être différés et regroupés en un seul accés .

Les programmes engendrés doivent vérifier deux critères d'efficacité

-Eviter la redondance : Les tuples ne doivent pas être générés deux fois par la même règle.

-Eviter la génération de tuples inutiles : Ceci correspond à effectuer les opérations de sélection le plus tôt possible afin de ne générer que les tuples utiles aux phases ultérieures.

Les méthodes compilées se divisent en deux catégories :

- Les méthodes compilées en chainage avant.
- Les méthodes compilées en chainage arrière.

a) **Méthodes compilées en chainage avant.**

Dans ces méthodes , nous pouvons distinguer deux goupes de méthodes:

a.1. Méthode naive

Ces méthodes compilent les règles qui dérivent la question en un programme itératif pour former la relation fictive puis poser la question sur cette relation

Le processus de compilation utilise le concept de graphe de relation [ULLM.85-BANC.86]. Une fois ce graphe obtenu, on procède à son évaluation . Cette évaluation se fait en deux parcours . Le premier permet la création de la relation temporaire, le second effectue l'évaluation proprement dite.

La méthode naive fournit le résultat souhaité , mais la constante appartenant à la requête n'est utilisée qu'à la fin , cela entraine la génération de tuples inutiles . De plus , lorsqu'on forme la relation déduite, on utilise à chaque étape tout les tuples générés , ce qui entraine une duplication .

La dernière remarque illustre la mauvaise performance de l'évaluation naive . Pour pallier à cet inconvénient, la méthode semi naive a été proposée.

a.2. Méthodes semi naives

Ces méthodes sont une variante optimisée de l'évaluation naive. A chaque itération , on ne considère plus tous les tuples de la relation virtuelle , mais seulement , ceux qui ont été générés à l'étape précédente.

b. Méthode compilée en chainage arrière.

Cette méthode consiste à transformer une requête contenant une relation virtuelle , en un programme itératif d'opérateurs de l'algébre relationnelle . Ce programme sera exécuté lors de l'évaluation de la requête .

Il existe des méthodes interprétées qui utilisent le chainage avant et le chainage arrière (par exemple , la méthode "ALEXANDRE" [ROHM.85]) .

L'idée générale étant de déterminer en chainage arrière , un ensemble de règles pertinentes pour une question donnée et de repondre à l'aide de ces régles en chainage avant .

Ceci permet d'une part de restreindre le domaine de recherche , d'autre part , de garantir la terminaison des requêtes récursives.

Pour illustrer les deux méthodes d'utilisation des régles de deduction (génération , dérivation), considérons l'exemple suivant :

Soit une base de données contenant deux relations (figure 3)

PERE

P	F
Ali Reda	Omar Farid

GRAND-PERE

GP	PF
Samir	Khaled

figure 3

Soit la loi générale :

" le père d'un père est grand père "

qui s'écrit :

(L) PERE (x,y) & PERE(y,z) ——> GRAND-pere (x,z)

Si la règle L est utilisée comme règle de dérivation , l'insertion du nouveau tuple < Omar , Raouf> , dans la relation PERE n'aura aucun effet explicite . Cependant lorsqu'une requête impliquant la relation GRAND-PERE est invoquée , son évaluation par le processus de déduction retournera le tuple< Ali , Raouf >.

Si d'un autre coté, L est utilisée comme une règle de génération, l'insertion du tuple < Omar, Raouf > , aura un effet explicite qui est l'insertion du tuple déduit < Ali , Raouf > dans la relation GRAND-PERE. Maintenant , puisque tout les faits sont explicites , l'évaluation de la requête peut être faite comme dans les bases de données conventionnelles .

Notons que dans une telle base de données , un tuple peut être soit explicite , (inseré directement par une requête), soit déduit (généré), soit hybride.

Nous illustrons maintenant , la différence entre l'interprétation et la compilation à laide de l'exemple suivant :

Soit une base de données constituées des relations

P, M, MARI

et supposons cette base dans l'état suivant (figure 4)

P

e	b1
e	b2
e	b3

M

c	e
c	f
c	g
g	d

MARI

a	c

figure 4

Soient les lois déductives :

A_1) M(x,y) & M(y,z) ——> GM(x,z)

A_2) M(x,y) & P(y,z) ——> GM(x,z)

A_3) GM(z,y) & MARI(x,z) ——> GP(x,y)

La relation GM est dite définie intentionnellement en termes des relations extensionnelles P ,M.

La relation GP est intentionnellement définie en termes des relations MARI et GM.

Nous illustrons l'apprôche interprétée en partant de la requête :

1) GP(a,y)

qui cherche les descendants de "a" (figure 5)

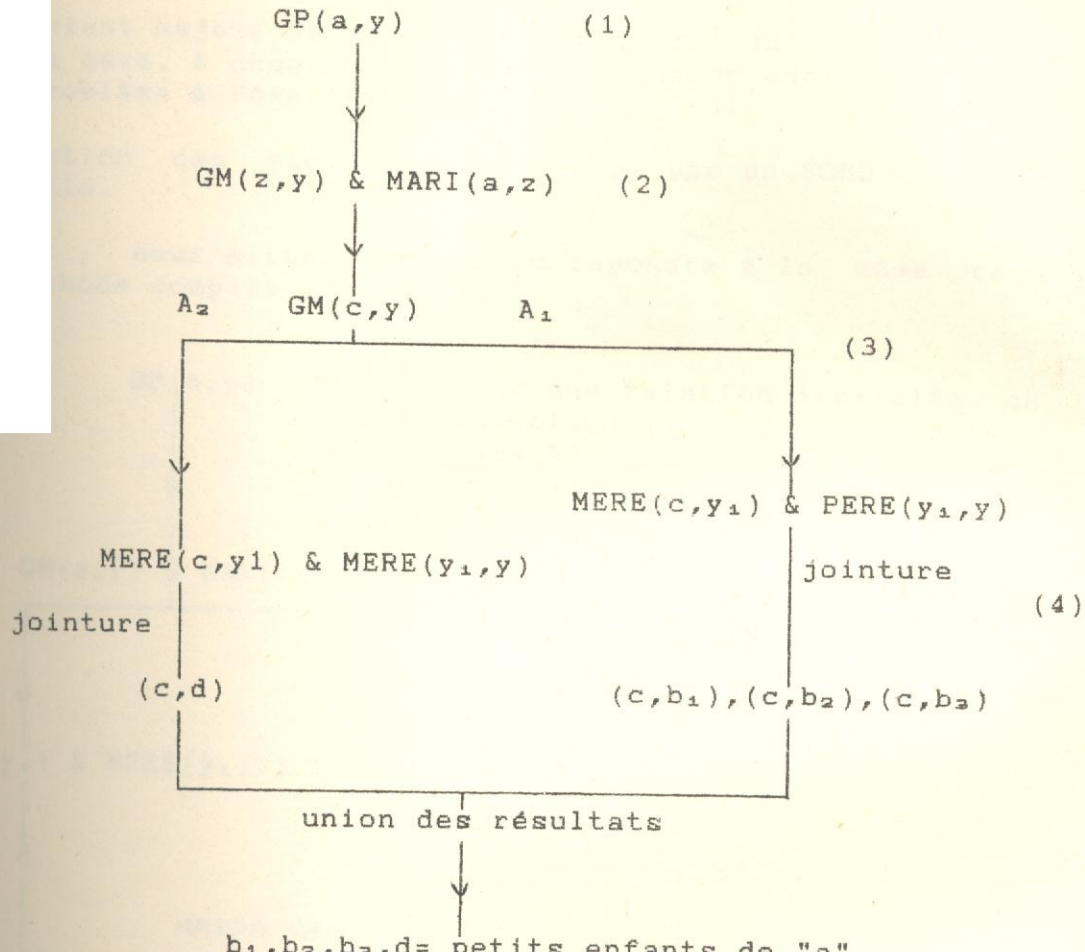


figure 5

Arbre de dérivation dans la méthode interprétée

Dans cet arbre de dérivation ,au niveau :

(1) GP est une relation implicite . En appliquant la règle (A3) et en utilisant le principe de résolution de Robinson , nous aurons la conjonction (2).

(2) A ce stade , la fonction de sélection choisit de résoudre MARI(a,Z) en premier , car elle contient une constante (ce qui diminue la recherche). En accédant à la base de données , on trouve MARI (a,c) avec Z lié à c. Maintenant , le sous-problème à résoudre devient GM(c,Y).

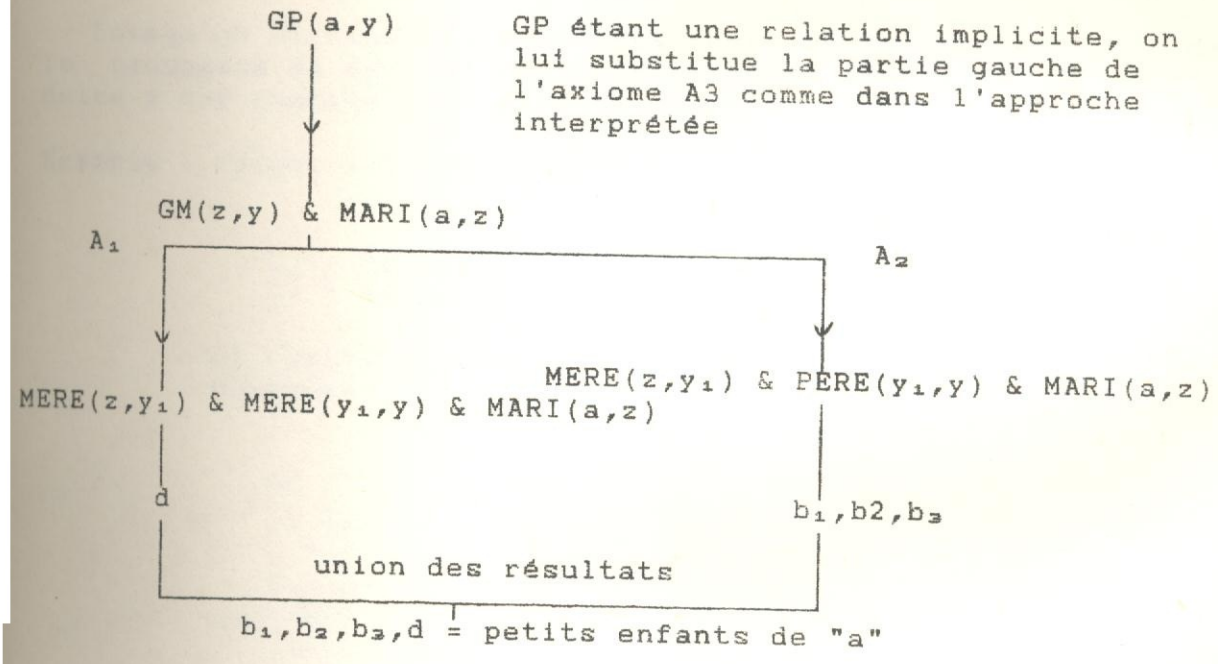
(3) Il y a deux manières de résoudre le problème GM(c,Y).

(4) Les deux relations sont explicites , on effectue une jointure classique.

L'inconvénient majeur de cette méthode , est qu'elle implique un accès à la base, à chaque fois qu'une relation explicite apparait dans le problème à résoudre .

L'optimisation des requêtes (effectuées par un SGBD classique) reste locale.

Maintenant , nous allons essayer de répondre à la même requête par la méthode compilée .(fig.6)



fiureg 6

Arbre de dérivation dans la méthode compilée

Remarque : L'accés à la base de données est retardé dans le processus déductif à la fin de l'évaluation. Ce qui permet une optimisation globale de l'accés à la base de données.

Les approches interprétée et compilée basées sur le mécanisme de chainage arrière ne résolvent pas les problèmes récursifs dans lesquels les chemins de dérivation sont potentiellement infinis

CHAPITRE 4
CYCLES ET RECURSIVITE

4.1.INTRODUCTION

Lorsqu'on utilise les approches compilée ou interprétée, dans le processus de dérivation , les axiomes récursifs peuvent conduire à des chemins de dérivation infinis.

Exemple : soient les deux règles suivantes:

L_1 : PE(x,y) ——> AN(x,y)
L_2 : AN(x,z) & PE(z,y) ——> AN(x,y)

et supposons que la relation PE soit dans l'état suivant:

PE

P	F
a	b
a	a
b	c

Si on veut évaluer AN(x,y) , la méthode interprétée conduit à un chemin de dérivation infini en appliquant récursivement L_2, et la méthode compilée produira un ensemble infini d'expressions de base de la forme

PE(x,y)
PE(x,z) & PE(z,y)
PE(x,z_1) & PE(z_1,z) & PE(z,y)
...
...

Cependant, il est clair que la réponse à la requête consiste en un ensemble de quatre tuples <a,b> , <b,c> <a,a> et <a,c>. Les deux processus auraient pu être stopés tout en préservant la complétude de la réponse .

Il est à noter que dans le cas de la génération , on ne trouve pas le problème du chemin de dérivation infini lorsqu'on utilise des règles récursives , pour illustrer ceci, nous décrivons le processus de génération .

Lorsqu'un nouveau tuple entre dans une relation , toutes les règles (les règles sont de la forme A_1 & A_2 & ... & A_n ——>B) dans lesquelles la relation apparait dans la partie gauche de l'implication , sont marquées .

L'une des règles marquées est sélectionnée , soit:

$$A_1 \& \ldots \& A_n \longrightarrow B$$

Cette règle, sa marque est supprimée et la règle est activée , tous les faits relatifs à B qui sont dérivables (en accord avec la règle) des faits relatifs à A_1 (c.a.d. les tuples dans les extensions des A_1) , sont générés.

Si un ou plusieurs faits générés ne correspondent pas à des tuples dans l'extension de B , les nouveaux tuples sont entrés dans cette extension et toutes les règles où B apparait dans la partie gauche sont marquées . Ensuite , une autre règle marquée est sélectionnée et le processus continue jusqu'à ce qu'il ne reste plus de règles marquées .

Puisqu'il y a un nombre fini de nouveaux faits qui peuvent être générés , les règles sont activées un nombre fini de fois .

4.2. UNE PREMIERE SOLUTION POUR L'ARRET DU PROCESSUS D'INFERENCE.

La solution que nous considérons consiste en une séparation adéquate entre les règles de génération et les régles de dérivation [MINK.81].
Un chemin de dérivation infini peut apparaitre lorsque des cycles existent parmi les règles de dérivation [LEWIS.75].
Un cycle est une séquense de règles de la forme :

1) l_1 ---> l'_2 V c_1

2) l_2 ---> l'_3 V c_2

:

:

n-1) l_{n-1} ---> l'_n V c_{n-1}

n) l_n ---> l'_1 V c_n

Où chaque c_j est une disjonction de littéraux et pour tout i , L_i et L'_i sont des littéraux unifiables .

Un chemin de dérivation infini initialisé avec n'importe quel littéral L_i peut être coupé si l'une des règles dans le cycle devient inutilisable pour une telle dérivation . C'est le cas pour une règle qui est utilisée comme règle de génération , puisque tous les faits qui peuvent être obtenus à partir d'elle sont déja générés .

Ceci conduit à l'idée de couper les cycles en utilisant l'une des règles du cycle comme règle de génération .

Etant donné que le cycle est coupé , le processus de dérivation ne va pas conduire à un chemin de dérivation infini . Cependant , l'utilisation de règles de génération avec des règles de dérivation, soulève quelques problèmes. On doit s'assurer qu'en procédant ainsi, il n'y a pas d'information perdue . Nous illustrons ceci par l'exemple suivant :

Considérons la relation S(x,y) et les deux règles suivantes :

S_1) S(x,z) & S(z,y) & (x<>y) ——> S(x,y)

S_2) S(x,y) ——> S(y,x)

Supposons que S_1 soit utilisée en génération et que S_2 soit utilisée en dérivation.

Supposons aussi qu'initialement, la relation S soit vide .
La figure 7 montre comment la relation S évolue lors des mises à jour

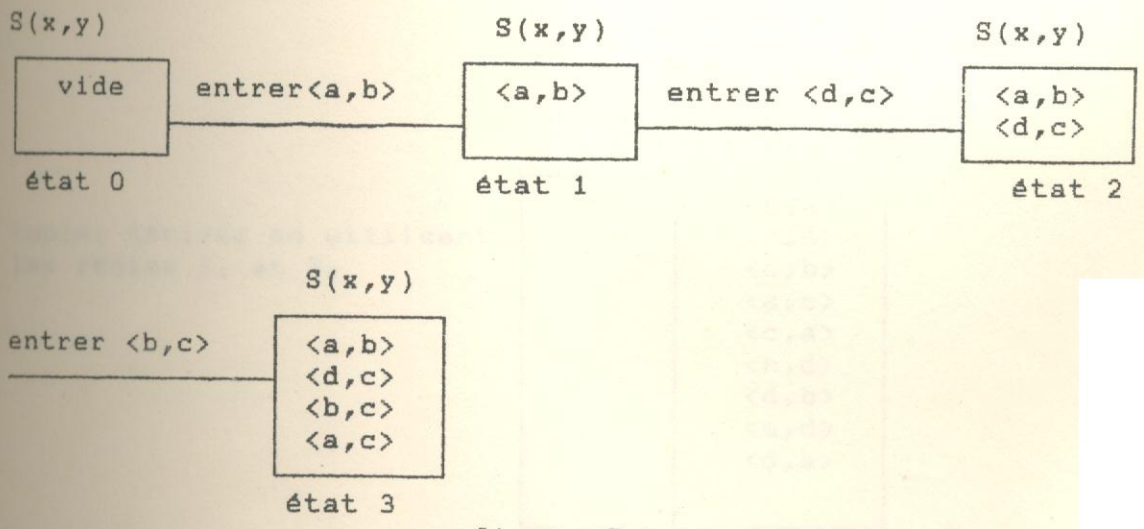


figure 7

Toutes les opérations de mise à jour présentées dans la figure 7 conduisent à l'activation de la règle S_1 . Cependant, La règle S_1 ne génére pas d'informations dans les deux premiers cas, alors que dans le dernier cas , elle génére S(a,c).

Considérons maintenant la B.D. dans l'état 3 , les informations relatives à la relation S, qui sont disponibles dans cet état , sont celles dans l'extension de S et celles dérivables à l'aide de S_2. (figure 8)

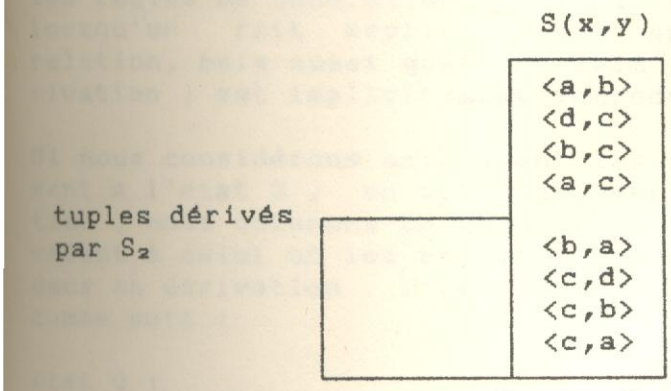


figure 8

Maintenant , considérons le cas où S_1 et S_2 sont utilisées comme règles de dérivation .

Dans ce cas l'extension de S consistera uniquement en trois faits entrés explicitement dans la base de données. Ce sont S(a,b) S(d,c) et S(b,c).

Les informations disponibles seront les trois faits avec les faits dérivés avec les règles S_1 et S_2 soit: (fig.9)

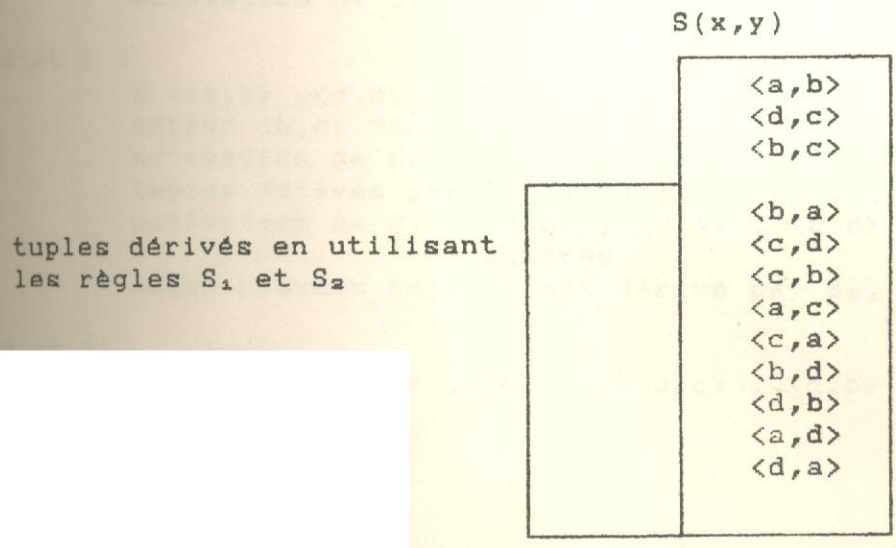


figure 9

Etant donné que l'ensemble des informations disponibles que nous obtenons inclut ceelles obtenues dans le cas précédent , l'exemple semble montrer que l'utilisation d'une régle en génération conduit à une perte d'informations . Mais en réalité , la cause de cette perte est due à une mauvaise activation de la règle de génération .

En effet lorsque les deux types de règles sont utilisés ensemble, les règles de génération doivent être activées non seulement lorsqu'un fait explicite est entré dans l'extension d'une relation, mais aussi quand un fait (dérivé par des règles de dérivation) est implicitement introduit

Si nous considérons maintenant les différents états qui conduisent à l'etat 3 , en activant correctement les règles de génèration , nous obtenons un ensemble d'informations disponibles équivalent à celui où les règles S_1 et S_2 sont utilisées toutes les deux en dérivation . Plus précisement , le processus travaille comme suit :

état 0 :
L'extension de S est vide,
entrer <a,b> dans S,
Activation de S_1 : aucun tuple n'est généré,
tuple dérivé par S_2 : <b,a>,
activation de S_1 aucun tuple n'est généré,

état 1 :
S = <a,b>,
entrer <d,c> dans S,
activation de S_1 : aucun tuple n'est généré,
tuples dérivés par S_2 : <b,a> , <c,d>,
activation de S_1 : aucun tuple n'est généré,

état 2 :
S =<a,b> ,<d,c>,
entrer <b,c> dans S,
activation de S1 : <a,c> généré,
tuples dérivés par S_2 :<b,a> , <c,d> , <c,b> , <c,a>,
activation de S_1 : <d,b> , <d,a> , <b,d> , <a,d> sont générés,
aucun nouveau tuple n'est dérivé par S_2,

état 3:
S={<a,b> , <d,c> , <b,c> , <a,c> , <d,b> , <d,a> , <b,d> <a,d>}

Nous pouvons vérifier que l'ensemble des informations disponibles obtenues de l'extension de S dans l'état 3 et de la régle de dérivation S_2 est exactement le même que celui obtenu de {s(a,b), s(d,c), s(b,c) } où S_1 et S_2 sont utilisées toutes les deux comme règles de dérivation . Donc ,aucune information n'est perdue.

Mais cette solution présente l'inconvénient de conduire à une redondance considérable entre les informations explicites et implicites. En effet , il est suffisant d'avoir dans l'extension de S seulement les tuples :

<a,b> , <b,c> , <d,c> , <a,c> , <b,d> , <a,d>

pour que le même ensemble d'informations soit disponible avec S_2

4.3. DEUXIEME SOLUTION

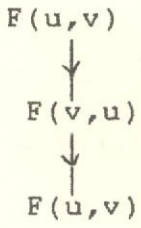
La solution que nous présentons ici est basée sur la comparaison d'expressions générées durant le processus de dérivation [MINK.81]. En fait , une condition est proposée , et quand elle est satisfaite par deux expressions sur le même chemin de dérivation , on autorise la coupure du chemin tout en conservant la complètude de la réponse .

La condition d'arrêt a été suggérée par des techniques utilisées dans la démonstration automatique de théorèmes utilisées pour détecter des sous buts redondants . Dans ce cas , on a à détecter si , dans un chemin de dérivation , un de ces sous buts générés est une instance d'un de ces prédécesseurs, par exemple :

Etant donné le but F(u,v), et la règle de dérivation

L: F(x,y) ——> F(y,x)

Nous aurons le chemin de dérivation suivant :



le deuxième sous but généré est identique à l'un de ces prédécesseurs .

Dans notre cas , puisque le but initial est une requête , le chemin de dérivation devient infructueux lorsque quelques expressions générées caractérisent un ensemble de tuples dans la base de données , qui est inclus dans l'union des ensembles des tuples caractérisées par les expressions prédécesseurs .

Donc nous allons chercher une condition liée à la forme des expressions qui, lorsqu'elle est satisfaite , garantit l'inclusion en question.

Lorsque la condition est l'identité formelle d'une expression, avec l'une des expressions précédentes , l'inclusion est satisfaite. La condition d'arrêt à définir affaiblira la condition d'identité qui est trop forte .

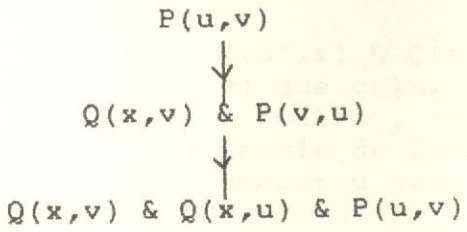
En général , nous manipulons des conjonctions de littéraux plutôt que de simples littéraux comme dans l'exemple précédent. Maintenant , l'ensemble des tuples caractérisés par une conjonction est nécessairement inclus dans l'ensemble caractérisé par l'une de ses sous conjonctions . La condition d'arrêt peut être relaxée en exigeant l'identité d'une sous conjonction d'une expression générée avec une de ses expressions prédécesseurs .

Exemple

Considérons la requête P(u,v) et la règle de dérivation

$$L : Q(x,y) \ \& \ P(y,z) \ \text{---}> \ P(z,y)$$

on obtient le chemin de dérivation suivant :



L'ensemble des deux tuples (u,v) caractérisé par l'expression

$$(1) \quad Q(x,v) \ \& \ Q(x,u) \ \& \ P(u,v)$$

est inclus dans (ou bien égal à) l'ensemble caractérisé par sa sous conjonction P(u,v) qui est identique à une des expressions prédécesseurs . Donc le chemin de dérivation peut être coupé juste avant l'expression (1) tout en préservant la complètude de la réponse .

Si nous voulons relaxer plus la condition d'arrêt , nous devons distinguer entre deux types de variables qui apparaissent dans les littéraux :

1/ Variables de sortie : ce sont des variables dont la valeur est demandée en sortie

2/ Variables non de sortie : ce sont les autres variables

On marquera les variables de sortie avec une étoile

Exemple

Dans la requête A(x*,y*) (ce qui signifie , donner toutes les paires ancêtre-descendant), x et y sont des variables de sortie par contre dans la requête A(x*,y) , (qui signifie , donner tous les ancêtres de y), seul x est une variable de sortie .

La distinction entre les variables de sortie et les autres variables est due au fait que les variables de sortie sont le seul lien entre les différentes expressions générées en cours de dérivation. Les autres variables sont indépendantes d'une expression à une autre . Par conséquent , il est possible de renommer les variables non de sortie indépendemment de leur occurrence dans d'autres expressions (ce qui n'est pas possible pour les variables de sortie)

Exemple

Une expression telle que P(u*,x) & Q(x,v*) caractérise le même ensemble des 2-tuples (u,v) que celui caractérisé par l'expression P(u*,y) & Q(y,v*) . Par contre , l'expression P(u*,v*) , ne caractérise pas le même ensemble de 2-tuples (u,v) que l'expression P(v*,u*) obtenu en renommant u par v et v par u

Nous utiliserons cette possibilité de renommage de variables non de sortie pour identifier des sous-expressions avec des expressions prédécesseurs .
Nous pouvons maitenant utiliser comme condition d'arrêt à une expression , l'identité (à un renommage près des variables non de sortie) d'une des sous expressions avec une expression prédécesseur. Nous allons maintenant caractériser la condition d'arrêt de façon formelle.

4.3.1. Substitition

Une substitution est un ensemble de paires de variables $N=\{ x_1/y_1 , \ldots , x_n/y_n \}$ où les x_i sont appelés anciennes variables et les y_i nouvelles variables

L'application d'une substitution N à une expression E , consiste à remplacer les variables de E qui apparaissent dans N comme anciennes variables par les nouvelles variables correspondantes dans N. L'expression ainsi obtenue est notée E(N)

Exemple

Soit E : R(x,y) & Q(y,z)
N : { x/u , z/w , t/v }

alors E(N) = R(u,y) & Q(y,w).

4.3.2. Substitution sûre

Une substitution est dite sûre pour une expression E si et seulement si

(1) aucune variable ancienne dans N n'est une variable de sortie dans E,

(2) toutes les variables nouvelles dans N , sont différentes, et aucune ne figure dans E .

La notion de substitution sûre pour une expression nous permet de renommer seulement les variables non de sortie (1iére condition) et ne pas introduire de nouveaux liens dans E(N)(2ième condition)

4.3.3. Contenance

Etant donné deux expressions E_1 et E_2 qui sont une conjonction de littéraux , on dira que E_1 est contenue dans E_2 si et seulement si, il existe une substitution N_1 (possible vide),sûre pour E_1 et une substitution N_2 (possible vide) sûre pour E_2 telles que chaque littéral dans $E_1(N_1)$ soit identique à un littéral dans $E_2(N_2)$.

4.3.4. Condition d'arrêt

Une dérivation peut être coupée (tout en préservant la complètude de la réponse) immédiatement avant une expression générée qui contient l'une des expressions prédécesseurs .

Exemple

Soit la requête

$$R(u^{\sim},v^{\sim},w^{\sim})$$

et la règle récursive suivante :

$$R(x,y',z) \;\&\; R(x,y,z') \longrightarrow R(x,y,z)$$

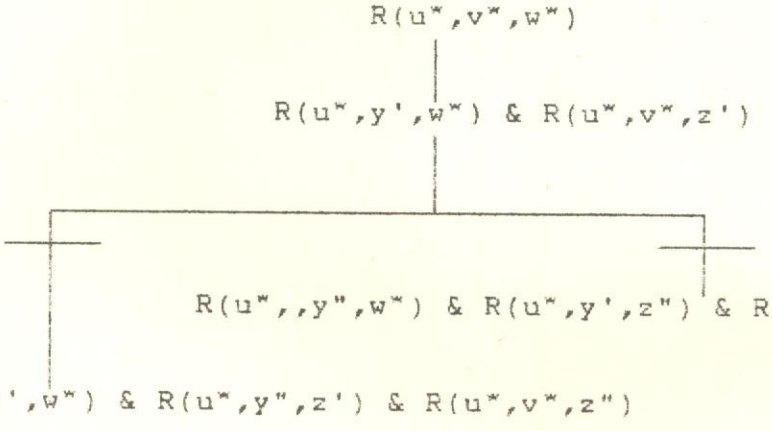


figure 10

L'expression (E_2) ne contient pas (E_1) car toute tentative d'identification implique les variables de sortie .

L'expression (E_3) contient (E_2) car chaque littéral dans

$$E_2(\{y'/t\}) = R(u^{\sim},t,w^{\sim}) \;\&\; R(u^{\sim},v^{\sim},z')$$

apparait dans

$$E_3(\{y"/t\}) = R(u^{\sim},t,w^{\sim}) \;\&\; R(u^{\sim},y',z") \;\&\; R(u^{\sim},v^{\sim},z')$$

et les substitutions $\{y'/t\}$ et $\{y"/t\}$ respectivement à E_2 et E_3 sont sûres. Ce chemin peut être coupé , comme le montre la figure 10. De façon similaire , le second chemin émanant de E_2 peut être coupé puisque (E_3') contient (E_2).

La condition d'arrêt qu'on vient de voir est suffisante mais pas nécessaire c'est à dire qu'elle ne nous permet pas de couper n'importe quel chemin de dérivation potentiellement infini .

La condition d'arrêt est satisfaite ou non indépendemment de l'état de la base de données. Mais il existe des chemins de dérivation dont la partie utile du chemin dépend de l'état de la base de données .

CHAPITRE 5
QUELQUES METHODES PROPOSEES POUR RESOUDRE LE PROBLEME DE LA RECURSIVITE

5.1.EVALUATION OPTIMISEE DE RELATIONS VIRTUELLES PAR PARCOURS DE STRUCTURES ARBORESCENTES .

La méthode proposée par P.BAZEX ET A.HAMEURLAIN [BAZ.88] traite des requêtes manipulant des relations virtuelles qui sont définies au moyen de formules basées sur une union de requêtes du langage QUEL, et cataloguées dans la méta base après avoir été transformées par le SGBD en une structure arborescente d'opérateurs algébriques .

Les algorithmes de traitement des requêtes sont basés sur leur transformation en une structure arborescente , éventuellement complétée par les définitions cataloguées des relations virtuelles .

Chaque définition de relation virtuelle est cataloguée dans la méta base sous forme de structure arborescente dont les noeuds sont les opérateurs algébriques (sélection, projection, jointure et union) et dont les feuilles sont les relations impliquées dans la définition . Ces relations peuvent être des relations réelles ou virtuelles .

La requête de l'utilisateur est transformée par le SGBD sous forme d'arbre : à chaque feuille correspondant à une relation virtuelle , le système rajoute sa définition cataloguée .

Lors de la compilation du schéma de la base , le SGBD crée de manière classique les fichiers nécessaires à la manipulation des tuples relatifs aux relations réelles .

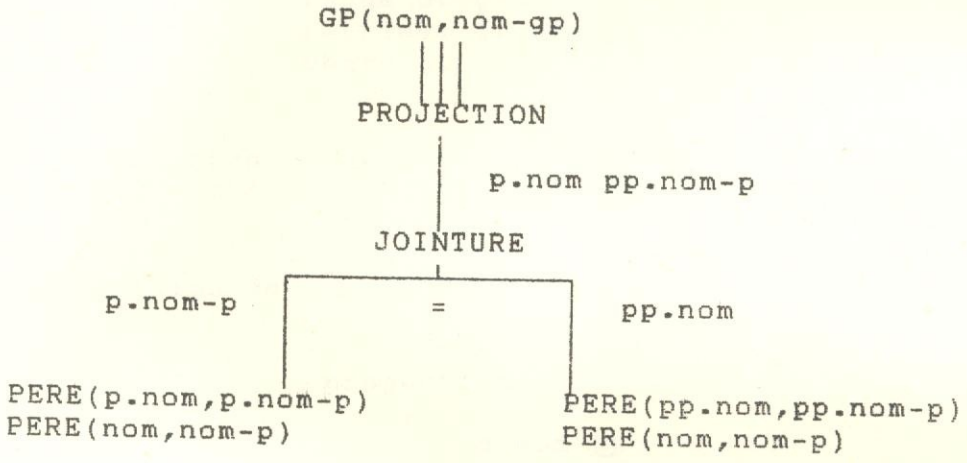
En ce qui concerne les définitions des relations virtuelles, elles sont mises sous forme de structure arborescente qui décrit les opérations élémentaires à effectuer sur les relations et l'ordre dans lequel ces opérations devront être réalisées .

Exemple :

La relation virtuelle GP est définie par :

```
relation : PERE(nom,nom-p)
           GP(nom,nom-gp)={range of p,pp is PERE,PERE
                           retrieve p.nom , pp.nom-p
                           where p.nom-p=pp.nom}
```

L'arbre syntaxique de cette définition est le suivant:



Cet arbre syntaxique , qui décrit les opérations permettant de calculer les tuples de la relation virtuelle GP est catalogué dans la méta base , et se comporte comme un sous programme dont les paramètres formels sont les attributs p.nom et pp.nom-p.

5.1.2. Principe général du traitement d'une requête.

La prise en compte par le SGBD d'une requête se fait selon les techniques classiques, par décomposition préliminaire de la requête en une structure arborescente d'opérateurs algébriques .

L'arbre obtenu est éventuellement restructuré par applications successives de règles de réécriture en vue d'obtenir un arbre dont l'évaluation sera la plus optimisée possible . Ensuite , un premier parcours de l'arbre permet la création des relations intermédiaires . Enfin , l'évaluation de la requête se fait lors d'un deuxième parcours.

Dans le cas où apparait , dans la requête une ou plusieurs relations virtuelles , le SGBD doit , après la décomposition de la requête , rajouter à chaque feuille correspondant à une relation virtuelle , l'arbre catalogué de la définition de cette relation. cet arbre devient un sous arbre de l'arbre de la requête .

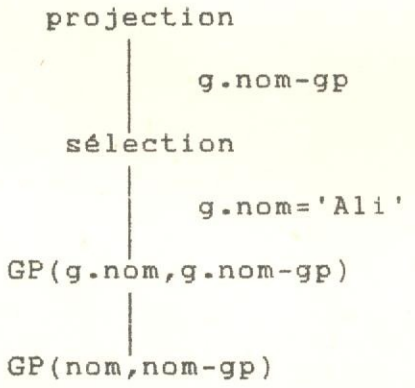
Exemple :

Reprenons l'exemple précédent et soit la requête suivante :

```
range of g:GP
retrieve g.nom-gp
where g.nom ='Ali'
```

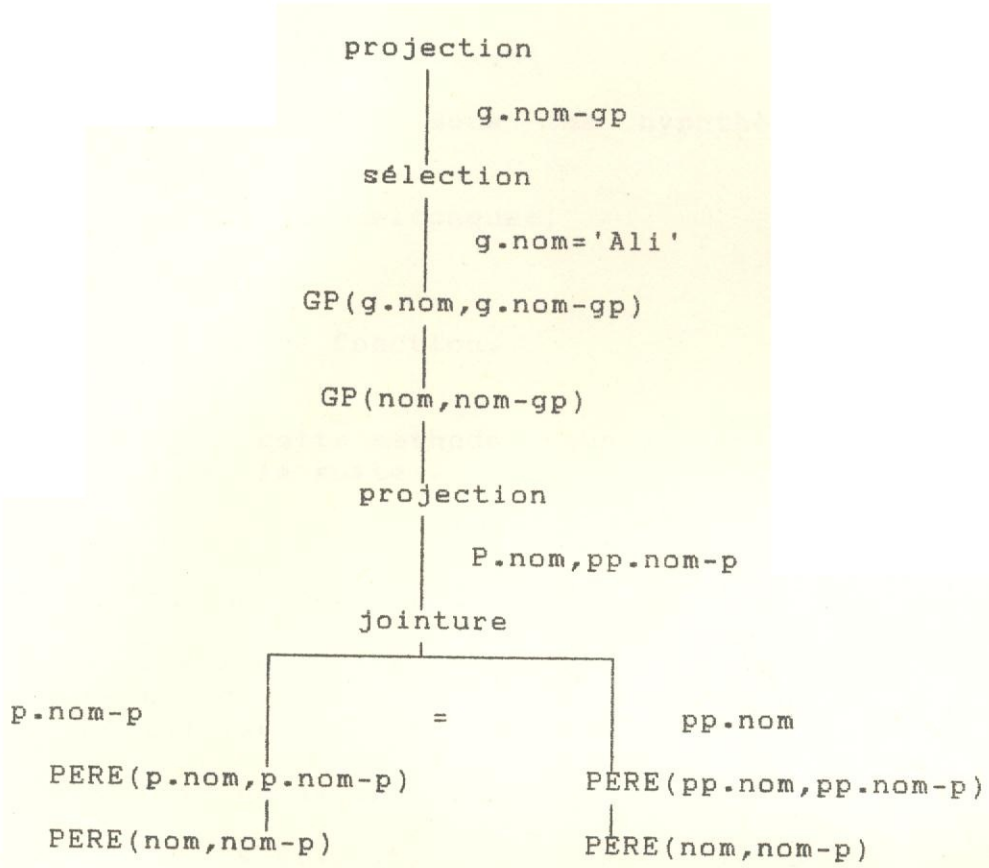
Les différentes étapes de traitement effectuées par le SGBD sur une telle requête sont les suivantes .

E_1: décomposition de la requête



E_2: Intégration des définitions de relations virtuelles

La relation GP est virtuelle , donc le SGBD rajoute à la feuille GP de l'arbre, la définition cataloguée de cette relation.
Ce qui donne l'arbre syntaxique suivant :



5.1.3. Problème des relations définies récursivement.

Lorsque l'arbre de la requête soumise par l'utilisateur a été construit , le SGBD doit le parcourir afin de chercher les feuilles correspondant aux relations virtuelles.

A chacune de ces feuilles, le SGBD rajoute la relation catalo-guée correpondante, qui devient un sous arbre de l'arbre général.

Lorsque la relation est définie selon une formule de récurrence; cette relation apparait au niveau de la racine de ce sous arbre et au niveau de la feuille .

A cette feuille on devrait rajouter une nouvelle fois cette même définition , entrainant du même coup un problème de branche infinie que l'on résoud en réévaluant l'arbre tant qu'à chaque évaluation apparaissent des tuples nouveaux, principalement dans des relations virtuelles

5.2. METHODE DE LOZINSKII.[LOZI.85]

Cette méthode travaille sous les hypothèses reprises dans [BANC.86]

1) règles récursives quelconques,

2) variable simples,

3) pas de symbole de fonction.

Avant de décrire cette méthode, nous donnons deux définitions qu'on utilisera par la suite .

5.2.1. Définition de l'ensemble MIG(X)

MIG(X) est l'ensemble des termes vers lesquels X peut migrer lors du processus de résolution .
L'ensemble MIG(X) est défini comme suit:

1) S'il existe un axiome de la forme suivante :

$Q(...) \ \& \ ... \ \& \ P(...,X,...) \ \& \ ... \ X=Y \ ... \longrightarrow R(...,Y,...)$

alors $Y \in MIG(X)$;

2) Si $Y \in MIG(X)$ et $Z \in MIG(Y)$
alors $Z \in MIG(X)$

5.2.2. Définition des faits relevants.

Soient une requête

$$Q: R(X_1,X_2,...,X_i=a_i,...)$$

et un fait

$$P(Y_i=b_i,...,Y_j:=b_j,...)$$

P est dit relevant de Q si et seulement si P contient Y_j tel que

$$\begin{cases} X_i \in MIG(Y_j) \\ b_j=a_i \end{cases}$$

5.2.3. Représentation des axiomes sous forme d'un graphe et/ou.

Les axiomes sont représentés par un graphe "et/ou", où les noeuds "et" représentent les relations et les noeuds "ou", les axiomes proprement dits.

Dans le cas où une relation est définie de manière récursive, l'occurrence de celle ci apparaissant dans la prémisse d'une clause sera représentée par un noeud principal et celle apparaissant en partie droite, par un noeud secondaire.

Exemple :

soit la relation ANCETRE définie par les axiomes suivants :

$$PERE(U_1,U_2) \;\&\; (U_1=W_1) \;\&\; (U_2=W_2) \longrightarrow ANCETRE(W_1,W_2)$$

$$ANCETRE(X_1,X_2) \;\&\; PERE(Y_1,Y_2) \;\&\; (X_2=Y_1)\ (X_1=Z_1) \;\&\; (Y_2=Z_2) \longrightarrow ANCETRE(Z_1,Z_2)$$

Cette définition conduira au graphe suivant (fig.11):

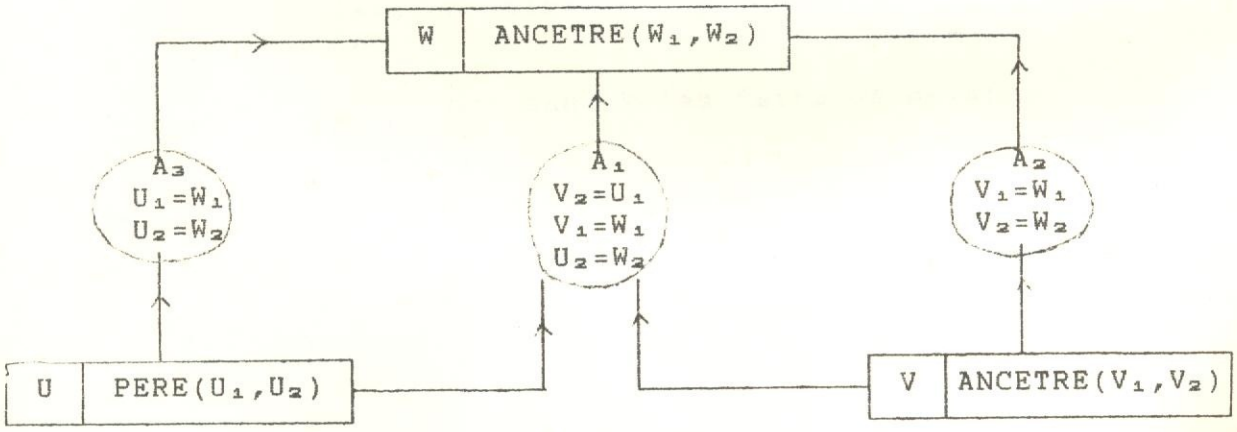


figure 11

5.2.4. Méthode

Cette méthode est une méthode interprétée , elle consiste à déteminer de manière dynamique, à partir de la requête initiale, les sous problèmes engendrés et les faits nécessaires à la résolution de ce sous problème.

Ces derniers seront résolus à l'exécution et leur réunion formera l'ensemble des solutions demandées.

Le problème consiste donc , pour chaque sous requête, à déterminer les faits relevants , puis s'ils existent, d'engager en chainage avant le processus de recherche des informations élémentaires dans la base de données.

Cette apprôche utilise le chainage arrière pour décomposer la requête en sous-questions et pour déterminerles faits relevants, et utilise le chainage avant pour évaluer ces sous-questions sur la base de données, ceci permet de s'affranchir des problèmes de la condition d'arrêt, et autorise l'arrêt anticipé du processus engagé en cas de non existence des faits relevants à la requête posée.

5.2.5. Algorithme

Soit Q la requête considérée et F l'ensemble des faits définis en extension dans la base de données. l'algorithme général est le suivant :

1. Sélectionner les faits relevants FR de la requête Q.

2. Repéter

   .S'il existe un axiome exécutable sur FR alors l'exécuter

   .Sinon rechercher dans F les faits nécessaires à l'exécution

   .Ajouter les faits nouveaux dans FR.

   Jusqu'à ce qu'aucun fait nouveau ne soit produit

Exemple :

Soient les relations initiales

M , P , R

et les relations déduites

N , S

définies de la manière suivante

A_1) $S(S_1,S_2)$ & $P(P_1,P_2)$ & $(S_2=P_1)$ & $(S_1=N_1)$ & $(P_2=N_2)$ $\longrightarrow N(N_1,N_2)$

A_2) $M(M_1,M_2)$ & $N(N_1,N_2)$ & $(M_2=N_1)$ & $(M_1=S_1)$ & $(N_2=S_2)$ $\longrightarrow S(S_1,S_2)$

A_3) $R(R_1,R_2)$ & $(R_1=N_1)$ & $(R_2=N_2)$ $\longrightarrow N(N_1,N_2)$

Soient les extensions suivantes des relations:

M,P et R

M	M_1	M_2
	b	c
	c	e
	f	t

P	P_1	P_2
	d	a
	g	h

R	R_1	R_2
	e	d
	t	g

nous allons appliquer l'algorithme précédent à la requête

Q: S(X,a).

Remarque

les faits relevants ajoutés à chacune des sept itérations sont soulignés .

<u>P(d,a)</u> est le seul fait relevant de Q.

1. Pour exécuter A_1, il faut un fait du type S(X,d), un tel fait s'obtient en résolvant

   Q_2: $S(X_1,d)$

   R(e,d) est relevant de Q_2 car $S_2 \in MIG(r_2)$

2. A_3 peut être exécuté sur R(e,d) et donne <u>N(e,d)</u>

3. Pour exécuter A_2 sur N(e,d), il faut un fait du type $M(X_2,e)$. Résolvons Q_3: $M(X_3,e)$. On a pour solution M(c,e).

4. L'exécution de A_2 génére S(c,d).

5. Celle de A_1 fournit <u>N(c,a)</u>.

6. Pour déclencher A_2, il faut un fait $M(X_3,c)$, ce qui conduit à la sous requête Q_4:$M(X_3,c)$. Q_4 donne pour solution M(b,c)

7. A_2 génére la solution S(b,a).
   Aucun fait nouveau ne peut plus être produit . Fin.

5.3. METHODE DE CHANG.[CHANG.81]

5.3.1. Introduction

Cette méthode est une méthode compilée. Elle travaille sous les contraintes suivantes:[BANC.86]

1) Une règle récursive(mais la méthode se généralise sans problème) régulière (c.a.d. ne comporte au plus qu'une relation virtuelle dans le membre gauche),

2) variable simple,

3) pas de symbole de fonction .

Le programme est obtenu en quatre phases.

1. Construction du graphe de connexion pour la requête et les axiomes ,

2. Obtention de règles de réécriture à partir de ce graphe,

3. Obtention d'une expression régulière en utilisant les règles précédemment trouvées,

4. Obtention d'un programme à partir de l'expression régulière,

5.3.2. Génération du graphe de connexion

A) Chaque axiome de la forme:

$$A_1 \ \& \ \ldots \ \& \ A_n \ \& \ F \longrightarrow B$$

est représenté par le noeud

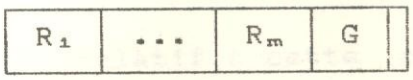
A_1	A_2	...	A_n	F	B

où

A_i=littéral gauche
B =littéral droit
F =formule gauche

Une requête de la forme:

R_1 & ... & R_m & G est représentée par :



Elle constitue aussi un noeud du graphe .

Les A_i et les R_j sont des formules relationnelles . F et G sont des formules sans relations explicites ni déduites.

On renomme les variables de sorte que les clauses (axiomes et questions) n'aient pas de variables en commun.

B) Pour chaque couple (L_1,L_2) de littéraux dans le graphe , L_1 étant un littéral droit et L_2 un littéral gauche , si L_1 et L_2 contiennent une relation déduite et sont unifiables, on construit un arc de L_1 vers L_2 et on attribue une étiquette à cet arc.

Exemple:

Soient les relations explicites suivantes :

EMP (nom=X , chef=Y , dep=Z),

VENTE (dep=U , objet=V),

et la relation déduite :

COMMANDE(superieur=S , nom=T)

définie par:

EMP(Y,X) ——> COMMANDE(X,Y)

EMP(Z,Y) & COMMANDE(X,Y) ——> COMMANDE(X,Z)

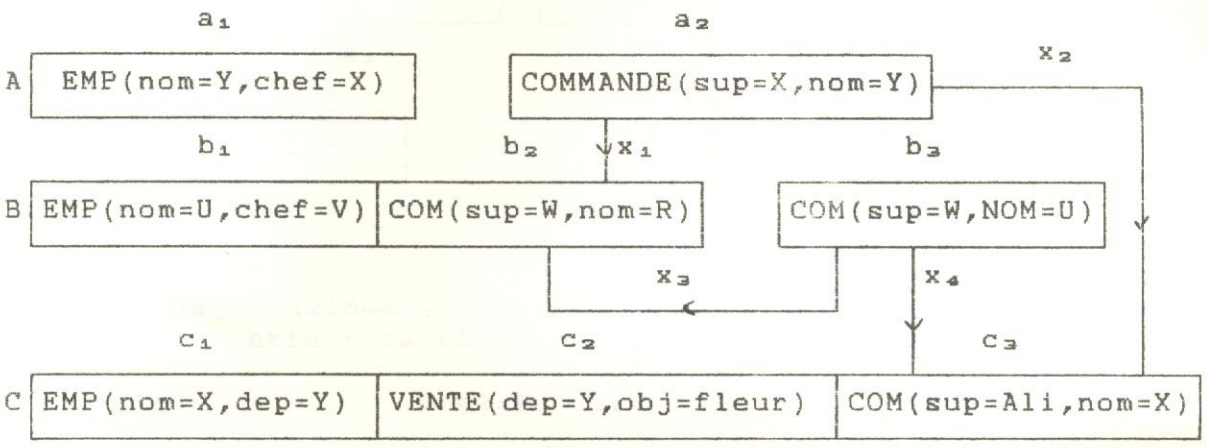
La relation COMMANDE est donc définie par un axiome de type récursif. Soit la requête suivante:

"Trouver toutes les personnes qui sont commandées par Ali et qui vendent des fleurs".

Celle-ci s'exprimera dans la syntaxe du langage DEDUCE2 (élaboré par CHANG) par:

EMP(*nom=X,dep=Y) & VENTE(dep=Y,obj=fleurs) & COM(sup=Ali,nom=X)

Le graphe de connexion relatif à cette requête et aux axiomes sera :



où

$x_1 = b_2 a_2$
$x_2 = c_3 a_2$
$x_3 = b_2 b_3$
$x_4 = c_3 b_3$

5.3.3. Obtention des règles de réécriture

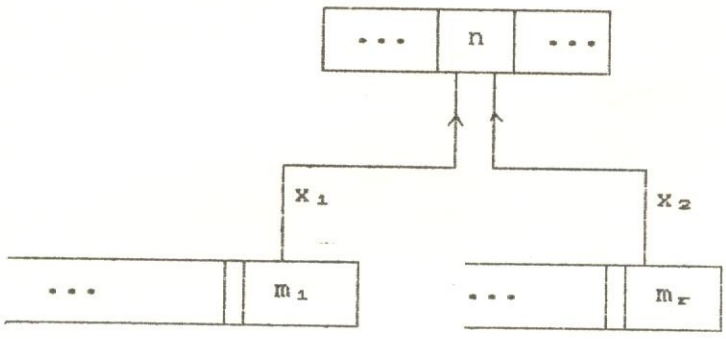
On donne un nom aux clauses (c.a.d aux axiomes et aux requêtes) et on réfère un littéral dans une clause par sa position dans cette clause.

Si C est le nom d'une clause , alors C_n est celui du nième littéral de la clause C (en comptant à partir de la gauche).

Les règles de réécriture sont obtenues à partir du graphe de connexion à l'aide des trois règles suivantes:

1. Pour chaque littéral gauche n, si $m_1, m_2, \ldots, m_r$ sont des littéraux droits pointant vers n, alors on obtient comme règle de réécriture pour n:

$$W(n)=x_1 W(m_1) \cup \ldots \cup x_r W(m_r)$$

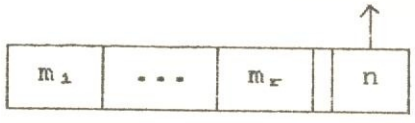


2. Pour chaque axiome , s'il existe un arc partant du littéral droit on obtient la règle de réécriture suivante :

$$W(n)=P_1 \ldots P_r$$

où les p_i sont définis par

$P_i=W(m_i)$ si m_i contient une relation virtuelle
$P_i= m_i$ sinon



3. Pour la requête

$$R_1 \& \ldots \& R_m$$

la règle obtenue est la suivante:

$$T=Q_1 \ldots Q_m$$

où les Q_i sont définis comme suit

$Q_i=W(R_i)$ si R_i contient une relation virtuelle

$Q_i=R_i$ sinon

Pour l'exemple précédent, on obtient les règles de réécriture suivantes

$W(b_2) = x_1W(a_2) \cup x_3W(b_3)$ par application de 1.

$W(c_3) = x_2W(a_2) \cup x_4W(b_3)$ par application de 1.

$W(a_2) = a_1$ par application de 2.

$W(b_3) = b_1W(b_2)$ par application de 2.

$T = c_1c_2W(c_3)$ par application de 3.

5.3.4. Obtention d'une expression régulière

Après avoir trouvé les règles de réécriture dans l'exemple précédent , par substitution , on obtient les règles de réécriture suivantes:

$$W(b_2) = x_1a_1 \cup X_3b_1W(b_2)$$

$$T = c_1c_2x_2a_1 \cup c_1c_2x_4b_1W(b_2)$$

Le problème maintenant est de s'affranchir du terme défini récursivement $W(b_2)$. Pour ce , nous donnons deux définitions et un théorème.

Définition-1

Soit un alphabet $\{a_1, \ldots, a_n\}$. Les expressions régulières sur cet alphabet sont données par ce qui suit:

1. $a_1,\ldots,a_n,k$ (le mot vide) et $\emptyset$ (l'ensemble vide) sont des expressions régulières.

2. Si P et Q sont régulières alors

$$P \cup Q \ , \ PQ, \text{ et } P^*=k \cup P^1 \cup P^2 \cup \ldots$$

sont des expressions régulières

Définition-2

La fonction q des expressions régulières dans {0,1} est définie par la suite des règles suivantes:

1. $q(a_i) = 0$

2. $q(\emptyset) = 0$

3. $q(k) = 1$

4. $q(P \cup Q) = \begin{cases} 0 & \text{si } q(P)=q(Q)=0 \\ 1 & \text{sinon} \end{cases}$

5. $q(PQ) = \begin{cases} 0 & \text{si } q(P)=q(Q)=1 \\ 1 & \text{sinon} \end{cases}$

6. $q(P^*) = 1$

Théorème [CHANG.81]

L'équation

$$D = a \cup bD$$

où a et b sont des expressions régulières et q(a)=0, a pour solution

$$D = b^*a$$

Une fois obtenues les règles de réecriture, on les simplifie, puis , en utilisant le théorème , on obtient des expressions régulières où l'ordre des symboles doit être préservé .

Dans la règle de réécriture de la requête, on remplace les noms des arcs par les couples des littéraux reliés par ces arcs. On réécrit ensuite explicitement les littéraux , en prenant garde de ne pas employer les mêmes variables pour deux copies d'une même clause.

Le théorème appliqué à l'exemple précédent donne:

$$T = c_1c_2x_2a_1 \cup c_1c_2x_4b_1(x_3b_1)^*x_1a_1$$

En remplaçant les x_i par les littéraux concernés on a

$$T = c_1c_2(c_3a_2)a_1 \cup c_1c_2(c_3b_3)b_1\ [(b_2b_3)b_1]^*\ (b_2a_2)a_1$$

5.3.5. Obtention d'un programme

Si l'expression régulière obtenue pour la requête est l'union de n expressions régulières, alors le programme est constitué de n parties.

On va donner les étapes permettant d'obtenir une partie de programme correspondant à une expression régulière qui fait intervenir l'opérateur "*"(c.a.d, qui nécessite une boucle).

A) On unifie les littéraux dans chaque couple de littéraux. En faisant les substitutions et en supprimant tous les couples, on obtient une expression Q, l'opérateur "*" dans Q indique que Q représente une séquence de questions .

B) Ayant obtenu une expression ne contenant que des relations explicites , on la représente par un graphe dans lequel chaque formule relationnelle constitue un noeud que l'on appelle noeud relationnel. Pour chaque couple de noeuds, s'il existe une même variable apparaissant dans chacun d'eux , on construit un arc qui les relie .

Un noeud N sera un noeud d'extrémité si et seulement si N est directement connecté à au plus un noeud N'.

C) S'il existe un noeud relationel extrémité contenant une variable X et ne contenant pas de symbole d'impression "*", on cherche toutes les valeurs possibles de X et on élimine le noeud du graphe.
On répéte cette étape tant qu'il existe un noeud relationnel extrémité.

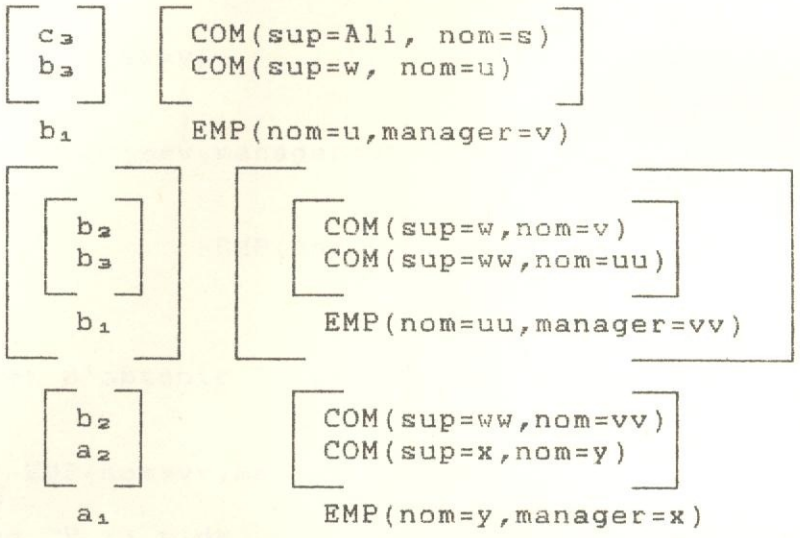
S'il y a un seul noeud relationnel , on cherche tous les tuples satisfaisant la condition représentée dans le graphe.

D) Après l'étape (C) , s'il reste encore des noeuds relationnels et si les noeuds N_1 et N_2 son reliés l'un à l'autre , on fusionne ces deux noeuds en exécutant les jointures correspondantes .
Si un noeud extrémité est généré après la fusion, on repasse à l'étape (C) , sinon on réexécute l'étape (D).

Dans la phase précédente , on a obtenu T qui est l'union de deux expressions . Considérons la deuxième expression . Par renommage des variables, on obtient les expressions suivantes :

c_1 EMP(*nom=s,dep=t)

c_2 VENTE(dep=t, objet=fleur)



En unifiant les littéraux à l'intérieur de chaque paire de parenthèses , on obtient la substitution suivante :

{ Ali/w , s/v , w/ww , v/uu , ww/x , vv/y }

qui est équivalente à

{ Ali/w , s/u , Ali/ww , v/uu , Ali/x , vv/y }

Après application de ces substitutions et suppression des paires de littéraux identiques , on obtient :

EMP(*nom=s,DEP=t)

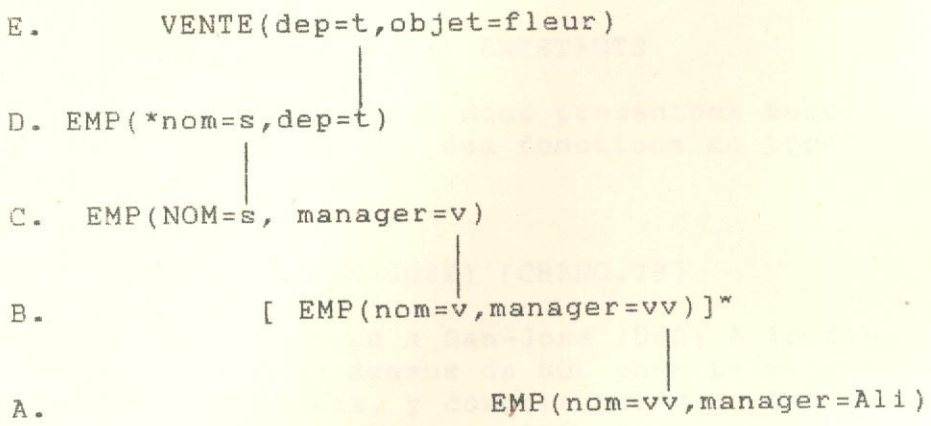
VENTE(dep=t,objet=fleur)

EMP(nom=s,manager=v)

*
EMP(nom=v,manager=vv)

EMP(nom=vv,manager=Ali)

cette suite de requêtes peut être représentée par le graphe suivant:



Ce graphe permet d'obtenir le programme suivant:

```
[a1]   evaluer EMP(nom=vv,manager=Ali)
[a2]   *V:=*VV
[b1]   tant que *V <> vide
       faire
[cde]      pour chaque v ∈ *V
           faire
               evaluer EMP(nom=s,manager=v) &
                       EMP(*nom=s,dep=t)    &
                           VENTE(dep=t,obj=fleur);
           fait;

           imprimer *S;

           pour chaque vv ∈ *VV
           faire
               evaluer EMP(*nom=v,manager=vv);
           fait;

           *VV:=*V;
       fait
```

On obtient un programme efficace répondant à la question de l'utilisateur. Ce programme pourra être optimisé par l'évaluateur du SGBD utilisé

5.4.QUELQUES PROTOTYPES EXISTANTS

Dans cette partie , nous présentons succintement quelques prototypes intégrant des fonctions de type SGBD déductif.

1. DEDUC2 (IBM SAN-JOSE) [CHANG.78]

Il fut prototypé à San-José (USA) à la fin des années 70 comme une interface au dessus de SQL pour permettre la définition de relations déduites, y compris par des règles récursives . C'est l'un des ancêtres intéressants des prototypes de SGBD déductifs. Il traite les règles récursives régulières (c'est à dire que le prédicat récursif n'apparait qu'une seule fois en début de condition)

2. BDGEN (CERT) [NICO.83a-b]

BDGEN fut prototypé au CERT à Toulouse , à la fin des années 70. Ce système permet de définir des règles sous forme de clause de HORN au dessus du SGBD relationnel MRDS sur MULTICS . Les règles sont utilisées en génération lors des mises à jour.

3. PROSQL (IBM YORKTOWN HEIGHTS).

PROSQL rèalise un couplage faible de PROLOG et de SQL [CHAN.86] PROSQL offre à l'utilisateur un interpréteur PROLOG supportant un prédicat unaire spécial SQL.
L'argument du prédicat est toute requête SQL exprimée comme une chaine de caractères .

4. EPSILON (PISE et Université C. Bernard , LYON)

Développé dans le cadre du projet ESPRIT 350, le projet EPSILON réalise un couplage fort statique PROLOG-SGBD[COSCIA.86]

CHAPITRE 6
LE COMPOSANT SYMCOD

Après cette étude, notre choix s'est fixé sur l'intégration dans l'architecture et la génération dans le fonctionnement du composant déductif.

L'intégration a été motivée par l'objectif global dans lequel s'insére le travail , et qui est de réaliser un SGBD déductif "compact" , donc il n'est pas question de concevoir une couche au dessus du SGBD (ce qui est fait dans le couplage).
Quand à la génération , elle a été choisie pour diverses raisons:

1) Même en présence de règles récursives ou de cycles parmi les règles, le chemin de dérivation se termine naturellement. Au contraire aucune solution pratique entiérement satisfaisante n'a été trouvée au problème de terminaison pour les règles de dérivation récursives.

2) Etant donné que chaque état de la base de données est saturé avec les faits déductibles , les chemins de génération sont en général plus courts que les chemins de dérivation correspondants. Ceci d'un coté , d'un autre coté , la vérification de l'intégrité et l'évaluation des requêtes sont faites comme dans les bases de données conventionnelles.

3) Puisque les faits déduits sont enregistrés de façon explicite, il sera nécessaire de disposer de beaucoup plus d'espace de stockage que dans l'apprôche par dérivation . Mais deux remarques peuvent être faites :

   1) Avec les téchnologies actuelles , l'espace de stockage est en developpement perpétuel.

   2) On n'a pas besoin de plus d'espace que dans une base de données conventionnelle équivalente.

Remarque

Etant donné que les requêtes sont en général plus fréquentes que les mises à jour, on perd moins de temps en génération qu'en dérivation.

6.1.LES TROIS ETATS DE L'INFORMATION

Une information dans la base de données peut être sous trois états différents

.Explicite : C'est une information entrée explicitement par l'utilisateur.

.Déduite : C'est une information déduite à partir d'un autre fait.

.Hybride : C'est une information entrée explicitement par l'utilisateur et déduite par ailleurs.

Pour avoir cette information , nous prévoyons dans chaque schéma de relation , un attribut supplémentaire transparent à l'utilisateur qu'on appellera STATUS et dont le domaine est l'ensemble des entiers.

Selon que la valeur du status est paire ou impaire , le fait sera respectivement purement déduit ou a une occurrence explicite (c.a.d. son status est explicite ou hybride). De plus la moitié de la valeur du status (dans le cas paire) et la moitié de (status -1) (dans le cas impaire), correspond au nombre d'occurrences déduites du fait considéré .
Ce nombre est le nombre de façons avec lesquelles le fait peut être déduit en un pas d'autres faits.

Exemple

Considérons une base de données dans laquelle il y a deux règles de génération qui impliquent une relation R.

L_1 P(X,Z) & Q(Z,Y) ——> R(X,Y)

L_2 S(X,Y,Z) ——> R(a,Y)

et supposons que la base de données soit dans l'état suivant

P

a	b
a	d
a'	b
a	g

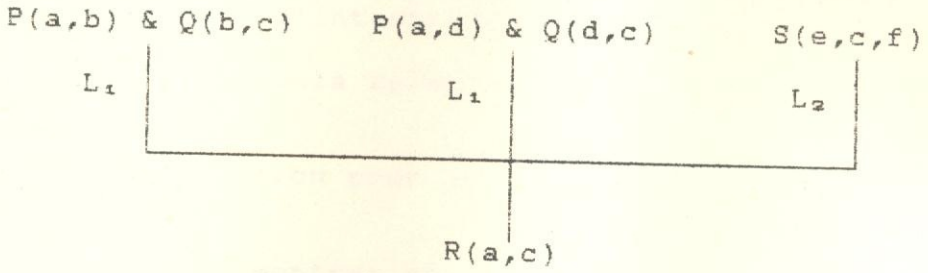
Q

b	c
d	c
d	f

S

e	c	f

Si nous cherchons les antécédants de R(a,c) , on aura l'arbre suivant:



La valeur du status de R(a,c) est égale à 6 et le nombre d'occurrences déduites de R(a,c) est égale à 3. Il est à noter que le nombre d'occurrences déduites d'un fait correspond à des déductions d'un pas. De plus quand nous parlons de nombre d'occurrences déduites , cela n'implique pas que le fait soit representé plusieurs fois , il ne l'est qu'une seule fois.

6.2. FORMALISATION DES MISES A JOUR

Nous allons considérer la base de données comme une interprétation , plus exactement un modèle , d'une théorie du premier ordre.

Cette théorie comporte des symboles de prédicats correspondant aux relations de la base de données et un ensemble de formules bien formées fermées (F.B.F) qui contiennent les axiomes de cette théorie.

Cet ensemble d'axiomes est décomposé en deux sous ensembles

1. Ensemble C : Ensemble de F.B.F considérées comme règles de cohèrence .

2. Ensemble D : Ensemble de F.B.F (sous forme de clauses de HORN) constituant les règles de déduction.

Le problème de décision , à quel sous ensemble appartient une F.B.F a été déja discuté (cf. 1. page 20)

Notons E le domaine des éléments. Nous considérerons les interprétations I_i de cette théorie définie sur ce domaine E et nous noterons I, l'ensemble des interprétations I_i de T et M l'ensemble des modèles M_i de T

(M $\subseteq$ I)

Les valuations affectées aux prédicats vont déterminer l'interprétation (la base de données) et les valeurs de vérité des F.B.F , axiomes fermés de la théorie T.

La mise à jour élémentaire de la base de données correspond à une modification de valuation pour un prédicat P(X), c'est à dire à un changement d'interprétation.

Sur I définissons la relation suivante :

" a même valuation pour la formule bien formée wf" notée Wf

Il est aisé de montrer que c'est une relation d'équivalence. Pour chaque F.B.F wf de la théorie , il existe donc une relation d'équivalence de ce type qui déterminera une partition de I. Nous noterons M_{wf} , l'ensemble des interprétations qui vérifient wf. M_{wf} est la classe d'équivalence de ces interprétations .

Du fait de la convention des informations négatives , toute information est vraie ou fausse . De ce fait , nous avons :

$M_{p(x)}$ = ensemble des interprétations où p(x) est vrai

$M_{\neg p(x)}$ = $I - M_{p(x)}$ ensemble des interprétations où p(x) est faux

6.3. REPRESENTATION D'UNE MODIFICATION ELEMENTAIRE.

Supposons que l'état initial de la base de données soit cohérent et corresponde à un modèle de T donc :

$$I_0 = M_0$$

la modification élémentaire correspond exactement au changement de la valeur de vérité assignée à un prédicat , P(A) par exemple. Ce qui correspond au passage de l'interprétation I_0 à une interprétation I_1.

Si $I_0 \in M_{p(A)}$ alors $I_1 \in I - M_{p(A)}$ (1)

Si $I_0 \in I - M_{p(A)}$ alors $I_1 \in M_{p(A)}$ (2)

Le cas (1) correspond à la suppression de l'information P(A).
Le cas (2) correspond à l'insertion de l' information P(A).

Le passage à une autre interprétation peut mener à deux cas :

a) $I_1 \in M$ et $I_1 = M_1$ est un modèle de T , c'est à dire que l'ensemble des règles est vrai dans I_1.

b) $I_1 \neg\in M$, c'est à dire que I_1 n'est pas un modèle de T.Il faudra donc passer à une autre interprétation qui soit un modèle de T.

6.4. DETERMINATION DE L'ENSEMBLE DES MODELES

Il est possible de déterminer l'ensemble des modèles d'une F.B.F fermée à partir des ensembles des modèles de chaque prédicat que nous appellerons ensemble des modèles élémentaires.

En effet

Si wf est P(A)	alors	$M_{wf} = M_{P(A)}$
Si wf est $\neg$p(A)	alors	$M_{wf} = I - M_{wf}$
Si wf est $w_1 \vee w_2$	alors	$M_{wf} = M_{w1} \cup M_{w2}$
Si wf est $w_1 \& w_2$	alors	$M_{wf} = M_{w1} \cap M_{w2}$
Si wf est $\forall x\, w_1(x)$	alors	$M_{wf} = \bigcap_{e_i \in E} M_{w(e_i)}$
Si wf est $\exists x\, w_1(x)$	alors	$M_{wf} = \bigcup_{e_i} M_{w(e_i)}$

Remarque : $e_i \in E$ et $\cup\{e_i\} = E$

Nous avons déja vu que lorsqu'on fait une mise à jour , on peut passer à une interprétation I_1 qui n'appartient pas à M. C'est à dire que la modification de la valeur de vérité de P(A) a rendu fausse la valeur de vérité d'une loi L axiome de la théorie T.

Exprimons M_L en fonction des modèles élémentaires. Quelque soit la formule fermée L , l'ensemble de ses modèles s'exprime en termes d'union, d'intersection et de complèment à I.

M_L peut s'écrire :

$$M_L = (M_{P(A)} \cap M_R) \cup (M_{\neg P(A)} \& M_R) \text{ [YAZ.86]}$$

Si $I_1 \in M_{P(A)}$ et $I_1 \neg\in M_L$ alors $I_1 \neg\in M_R$ (1)
Si $I_1 \in M_{\neg P(A)}$ et $I_1 \neg\in M_L$ alors $I_1 \neg\in M_R$. (2)

Pour remédier à ce problème et se trouver dans une interprétation $I_1' \in M_L$, il existe deux cas possibles:

1) Revenir à I_0 et dans ce cas , nous sommes sûrs que

soit $I_0 \in M_{F(A)} \cap M_R$,
soit $I_0 \in M_{\neg F(A)} \cap M_R$.

car I_0 appartenait à M_L.

2) Soit changer à nouveau d'interprétation en effectuant les modifications adéquates dans la base de données de telle sorte que $I_1' \in M_L$

Ces deux cas représentent les deux possibilités d'utilisation d'une loi L . Dans le premier cas elle est utilisée comme règle de cohèrence , dans le second, comme règle de déduction .

Les résultats que nous venons de voir et qui sont relatifs à une règle , peuvent se généraliser à un ensemble de règles L_i . Selon ce qui précéde , la base de données doit être un modèle de T , donc vérifier toutes les règles L_i . D'où toute interprétation I représentant la base de données doit appartenir à :

$$M_{L1} \cap M_{L2} \cap \ldots \cap \ldots$$

Tout se passe comme s'il n'y avait qu'une seule règle générale constituée de la conjonction des lois L_i

le rôle du composant déductif élaboré [SYMCOD] est de ramener la base à une interprètation qui soit un modèle pour les lois générales. Ce composant comprend cinq parties

6.5. DESCRIPTION DE SYMCOD

Dans les deux premières parties de SYMCOD, nous nous intéressons à l'insertion et à la suppression d'une information dans la base de données .

L'insertion ou la suppression d'une information fait évoluer la base , c'est à dire , lui change d'état . Donc on passe d'une interprétation à une autre qui peut ne pas être un modèle pour l'ensemble des règles .

SYMCOD va chercher une interprétation qui soit un modèle pour l'ensemble des règles , tout en préservant la mise à jour faite.

6.5.1. Insertion d'une information

Cette insertion peut être explicitement demandée par l'utilisateur ou bien demandée par le composant déductif à la suite d'une déduction.

L'information est d'abord recherchée dans la base de données. Si elle existe déja , alors si l'insertion est implicite , alors la valeur du status de l'information est incrémentée de deux.

Dans le cas où l'insertion est explicite, le status est rendu impair.

Si l'information n'a pas été trouvée dans la base , il s'agit d'une nouvelle information à entrer dans la base , et selon le cas , il s'agit d'une insertion implicite ou explicite, et le status est mis respectivement à 1 ou 2 . Toutes les informations déductibles de cette information sont générées et mises dans une file d'attente, en vue d'être insérées.

6.5.2. Suppression d'une information

La suppression d'une information peut être demandée explicitement par l'utilisateur ou bien demandée par SYMCOD lors d'une déduction .

Dans le premier cas , l'information à supprimer est recherchée dans la base, dans le cas où c'est une information explicite , elle est supprimée et tous les tuples qui ont été déduits à partir de cette information , sont mis dans une file d'attente . Dans le cas où c'est une information hybride, le status est décrémenté d'une unité . Si l'information est purement déduite , elle ne sera supprimée qu'après confirmation de l'utilisateur.

Dans le cas où la suppression est implicite , l'information est recherchée dans la base puis son status est décrémenté de deux unités (le status est dans ce cas nécessairemant supérieur ou égal à deux) .
Dans le cas où le status devient nul (c'est à dire qu'il n'y a plus aucune occurrence) , tout le processus est repris sur cette information.

6.5.3. Problème des boucles persistantes.

Considérons l'exemple suivant :

Soit l'ensemble de règles:

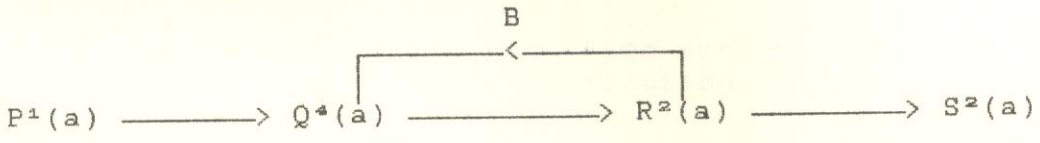
$$P(X) \longrightarrow Q(X)$$

$$R(X) \longrightarrow S(X)$$

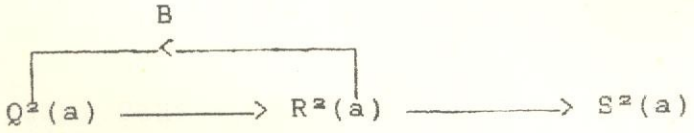
$$Q(X) \longrightarrow R(X)$$

$$R(X) \longrightarrow Q(X)$$

Supposons qu'on insére le fait P(a) dans la base , les faits résultant dans la base (avec leur valeur du status) sont:



Supposons qu'on veuille supprimer P(a) explicitement . Le processus de suppression s'arrêtera à Q(a) conduisant à l'état suivant :



Nous constatons que la suppression de P(a) n'a pas rendu la base à son état initial. Cependant, l'état de la base reste cohérent avec l'ensemble des règles et la valeur du status est correcte.

La boucle B est appelée boucle persistante, elle ne dépend (ni directement ni indirectement) d'aucun fait explicite .La suppression d'une telle boucle peut être obtenue en supprimant R(a).

La troisième et quatrième parties de SYMCOD, concernent l'insertion et la suppression d'une règle.

La base de données doit (pour être cohérente avec les règles) être à tout moment un modèle pour cet ensemble de règles .

L'ajout ou la suppression d'une règle à la base de règles peut entrainer une incohèrence entre la base de données et la base de règles. SYMCOD va changer l'état de la base (passage à une autre interprétation) de telle sorte que la base soit un modèle pour la nouvelle base de règles . Donc il va générer tous les tuples qu'il faut ajouter ou supprimer de la base pour qu'elle soit un modèle.

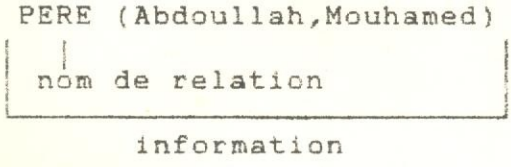
6.5.4. La déduction dans SYMCOD

Le processus de déduction dans SYMCOD passe par plusieurs phases . Nous en décrivons ci dessous les principales.

6.5.4.1. Recherche des règles concernées

On dira qu'une règle est concernée par une information si le nom de la relation dans cette information, figure dans le membre gauche de la règle.

Exemple:



(1)

règle: PERE(X,Y) & PERE(Y,Z) ——> ANCETRE(X,Z) (2)

La règle (2) est concernée par l'information (1)

6.5.4.2. Recherche des instances de chaque règle concernée.

Pour rechercher ces instances , on doit rechercher une substitution pour chaque occurrence du nom de la relation dans le membre gauche de la règle.

Exemple

Dans la règle (2) , nous avons deux occurrences du nom de relation PERE.

Une substitution est un ensemble de couples (a,b) où a représente l'ancienne variable et b , la nouvelle.

Exemple:

Si on s'intéresse à l'information (1) et à la règle (2), la substitution trouvée pour la première occurrence de la relation PERE est:

{ (X,Abdoullah) , (Y,Mouhamed) }

Cette substitution est trouvée dans le but de rendre l'occurrence du nom de relation identique à l'information . On dit qu'on a fait une unification.

Maintenant ayant fait cette unification , il faut propager la substitution sur toutes les variables de la règle, ce qui revient à remplacer toute variable de la règle qui apparait dans le premier élément du couple de la substitution , par le second. Nous obtenons ainsi une occurrence de la règle.

En répétant ce processus pour toutes les occurrences du nom de la relation, nous obtenons un ensemble d'instances de la règle.

Exemple:

Toujours pour l'information (1) et la règle (2) , nous obtenons après l'application du processus précédent , ce qui suit.

1 ière occurrence du nom de relation PERE

PERE(X,Y)

substitution

{ (X,Abdoullah) , (Y,Mouhamed) }

1 iére occurrence de la règle

PERE(Abdoullah,Mouhamed) & PERE(Mouhamed,Z)
——>ANCETRE(Abdoullah,Z) (3)

2 ième occurrence du nom de relation PERE

PERE(Y,Z)

substitution

{ (Y,Abdoullah) , (Z,Mouhamed) }

2 ième occurrence de la règle

PERE(X,Abdoullah) & PERE(Abdoullah,Mouhamed)
——> ANCETRE(X,Mouhamed) (4)

La première occurrence de la règle dit que :

"Si Abdoullah est le pére de Mouhamed alors Abdoullah est un ancêtre de tout fils de Mouhamed "

La seconde occurrence dit que :

"Si Abdoullah est le pére de Mouhamed, alors tout pére de Abdoullah est ancêtre de Mouhamed "

Après avoir trouvé toutes les instances de règles , le processus d'activation de règles est lancé. Ce processus va nous donner un ensemble d'informations déduites à insérer ou à supprimer (implicitement). Ces informations seront mises dans une file d'attente.

6.5.4.3. Activation

L'activation reçoit en entrée un ensemble d'instances de règles. Pour chacune de ces instances , il y a évaluation du membre gauche partiellement instancié (le détail de cette évaluation sera donné plus loin); l'évaluation considére ce membre gauche comme une requête posée au SGBD , la réponse à cette requête est donnée à la sortie de l'évaluation.

Donc à l'issue de l'évaluation , chaque variable aura au moins une valeur , sinon l'activation n'aura pas lieu.

Chaque ensemble de valeurs des variables donnera lieu à une information déduite. Ceci est obtenu en remplaçant les variables du membre droit par leur valeur trouvée dans l'évaluation.

Exemple :

Supposons toujours que l'on insére l'information (1) en présence de la règle (2) . On avait trouvé les instances (3) et (4).

Supposons que l'état de la base soit le suivant

PERE

Mouhamed	Ali
Mouhamed	Omar
Othman	Abdoullah

L'unique variable qui reste dans l'instance (3) est Z.

L'évaluation du membre gauche donnera comme valeur pour Z,

{ Ali , Omar }

Ceci étant obtenu en évaluant

PERE(Mouhamed,Z)

En remplaçant la variable Z dans le membre droit par sa valeur , on obtient deux informations déduites.

ANCETRE(Abdoullah , Ali)

ANCETRE(Abdoullah , Omar)

qui veut dire que Abdoullah est un ancêtre de Ali et Omar .

Si on considére l'instance (4) , on voit que l'unique variable est X . L'évaluation du membre gauche donnera une seule valeur à cette variable.

X=Othman

De la même manière ceci va conduire à l'information déduite :

ANCETRE(Othman , Mouhamed)

Ce qui signifie que Othman est un ancêtre de Mouhamed . Les trois informations déduites sont mises dans une file d'attente pour être inserées.

6.5.4.4. Evaluation du membre gauche partiellement instancié.

Ce module reçoit en entrée , le membre gauche d'une règle partiellement instanciée (c'est à dire qui contient des variables et des constantes) . Il crée une liste dont la tête contient l'ensemble des variables à instancier (c'est à dire pour lesquelles on doit trouver une valeur), puis lance le module de recherche qui va remplir le corps de la liste avec les valeurs de ces variables . Ce même processus est appliqué au prochain nom de relation .

A l'issue de ce qui précéde , nous avons deux listes de valeurs de variables dont il faudra faire la jointure (les détails de cette jointure sont donnés plus loin).

Le résultat de la jointure est une troisième liste qui contient l'ensemble des variables des deux relations précédentes avec leurs valeurs .

Remarque : Les deux listes initiales sont détruites après la jointure.

Ce mécanisme est appliqué jusqu'à ce qu'il n'y ait plus de nom de relation dans le membre gauche .

Le résultat final est évidemment une liste qui contient toutes les variables du membre gauche avec les valeurs de ces variables.

Dans le cas où la règle contient une condition , toutes les valeurs des variables qui ne satisfont pas la condition sont éliminées .

6.5.4.5. Recherche des valeurs des variables

Ce module reçoit en entrée une liste dont la tête contient des variables et dont le corps est vide . Si des constantes figurent dans la tête de la liste , seuls les tuples qui contiennent ces constantes sont retenus . Si aucune constante ne figure dans la tête de la liste, tous les tuples de la relation sont retenus. Dans le cas où des variables se répétent dans la tête de la liste , on doit vérifier que les valeurs correspondantes sont identiques.

Exemple :

Soit le schéma de relation R(X,Y,Z) dont l'extension est la suivante:

R

a	b	c
e	f	g
a	b	h
a	k	a

Si la tête de la liste est la suivante:

a	b	X

le résultat sera :

a	b	c
a	b	h

Si la tête de la liste est la suivante :

W	Y	Z

Le résultat sera

a	b	c
e	f	g
a	b	h
a	k	a

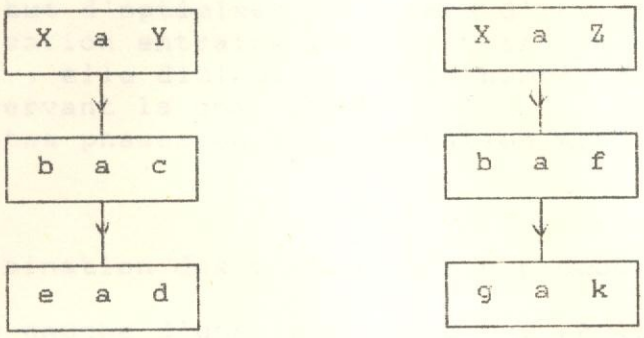
6.5.4.6. La jointure

Ce module reçoit en entrée deux listes dont les têtes contiennent des variables et des constantes et les corps contiennent les valeurs des variables .

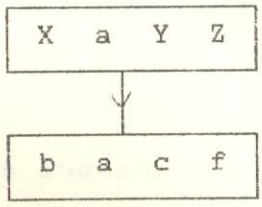
Si les deux têtes contiennent des variables ou des constantes en commun, alors nous avons affaire à une jointure avec une condition (égalité des valeurs des termes en commun), sinon nous aurons un produit cartésien.

Exemple :

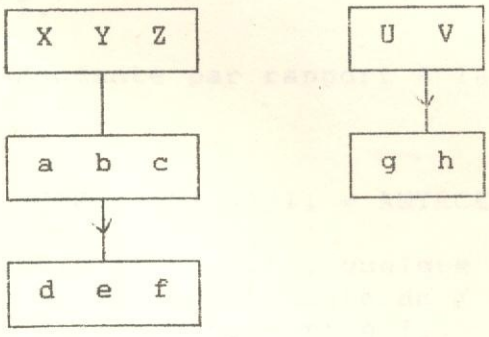
1. Soient les deux listes suivantes



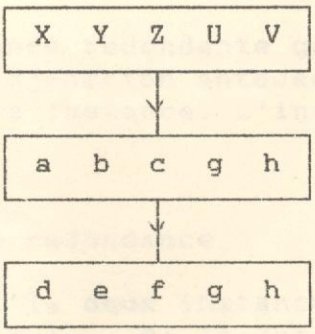
Ce cas correspond à une jointure et le résultat est le suivant :



2. Soient les deux listes suivantes:



Ce cas correspond à un produit cartésien et le résultat est le suivant :



6.5.5. L'optimisation

L'optimisation intervient au niveau des instances d'une règle. Elle a pour but d'optimiser le nombre d'instances de règles . Cette optimisation entraine une optimisation dans l'activation et l'évaluation , elle diminue aussi le nombre de tuples générés tout en préservant la complètude .
Les differentes phases de l'optimisation sont :

6.5.5.1. Elimination des instances non productives

La partie gauche d'une instance d'une règle est formée par une conjonction de littéraux et une conjonction de conditions. Si une condition est complètement instanciée, on peut éliminer toutes les instances dont la condition est évaluée à faux et supprimer les conditions qui sont évaluées à vrai et ce avant tout accès à la base.

6.5.5.2. Elimination des instances redondantes

Soient I_1 et I_2 deux instances d'une règle L et F_1 et F_2 l'ensemble des faits déduits en un pas par I_1 et I_2 .
Soit ANTECEDANT(X)/I la conjonction antécédante de X par rapport à I.

I_1 est dite redondante par rapport à I_2 si

1) $F_1 \subset F_2$

2) $\forall X \in F_1$ ANTECEDANT(X)/I_1 = ANTECEDANT(X)/I_2

La deuxième condition se lit , quelque soit le fait X appartenant à F_1 , la conjonction antécédante de X par rapport à I_1 est la même que celle de X par rapport à I_2.

Une instance redondante génére de la même manière (c'est à dire même conjonction antécédante) , des faits qui sont générés par une autre instance. L'instance redondante peut être éliminée.

Condition de redondance

Soient I_1 et I_2 deux instances d'une règle . I_1 est redondante par rapport à I_2 , si la substitution qui a conduit à I_1 inclut celle qui a conduit à I_2.

Exemple

Soit la règle

$$L : P(X,Y) \ \& \ P(X,b) \ \& \ Q(Y,Z) \longrightarrow R(a,Y)$$

L'insertion du du tuple (a,b) dans la relation P conduit aux deux instances suivantes :

$$I_1 : P(a,b) \ \& \ P(a,b) \ \& \ Q(b,Z) \longrightarrow R(a,b)$$

$$I_2 : P(a,Y) \ \& \ P(a,b) \ \& \ Q(Y,Z) \longrightarrow R(a,Y)$$

I_1 est obtenue de L en appliquant la substitution

$$\{ a/X , b/Y \}$$

I_2 est obtenue de L en appliquant la substitution

$$\{ a/X \}$$

I_1 est donc redondante par rapport à I_2 .

6.5.5.3. Elimination des instances duplicatrices .

Une instance est dite duplicatrice , si l'un des littéraux du membre gauche est identique au membre droit .

Il est clair que tout fait généré par une telle instance , existe déja dans la base. Une telle instance peut donc être supprimée

En plus de cette optimisation , le SGBD intégrant SYMCOD, profite de ses propres optimisations classiques , car SYMCOD n'intervient pas dans l'évaluation des requêtes (il existe un module optimiseur des requêtes du SGBD) .

Conclusion

L'intégration de la déduction dans un SGBD classique est une architecture qui parait attirante pour les bonnes performances qu'elle promet, neanmoins sa mise en oeuvre souléve un problème de taille. En effet l'intégration de la déduction suppose soit la maitrise du code du SGBD, hypothèse qui n'est souvent pas satisfaite, soit la réécriture de tout le SGBD, qui est une tâche assez fastidueuse. C'est donc une architecture coûteuse .

D'autre part l'apprôche par génération permet certe de traitert une large classe de règles et traite de façon efficace la récursivité, mais le nombre de tuples déduits et stockés depasse rapidement le nombre de faits explicites. Ce qui est génant pour les bases de données volumineuses, aussi l'association de l'apprôche par dérivation et de l'apprôche par génération pourait remedier à cette défaillance. La génération pourait être utilisée pour couper les cycles dans les règles et traiter la récursivité. Cette association presente à notre avis (en l'absence d'algorithmes de dérivation efficaces) une bonne base pour un SGBD déductif .

Mis à part ce qui précede, de nombreux problèmes restent à résoudre tels que, la prise en compte des symboles de fonctions dans les règles , le traitement des informations incomplètes ...etc . Mais déja de nombreuses solutions sont proposées et le domaine progresse très rapidement, et l'on peut espérer que cela pourra déboucher sur des systèmes opérationnels à une échéance raisonnable.

BIBLIOGRAPHIE.

[BANC.85a] Bancilhon F.
Naive evaluation of recursivaly defined predicates.
Proc. of Islamadora Workshop on database and artificial intelligence. mars 1985

[BANC.85b] Bancilhon F. Maier D. Savig Y. Ullman J.
Magic set and other strange ways to implement logic programs.
MCC technical report N°DB-121-85 , 1985.

[BANC.86] F.Bancilhon R. Ramakrushan.
An amateur's introduction to recursive query processing strategies.
Proc. of the ACM SIGACT-SIGMOD symp. on princ. of data base systems 1986.

[BAZ.88] P. Bazex A. Hameurlain
Evaluation optimisée de relations virtuelles par parcours de structures arborescentes.
Rapport intern , université Paul Sabatier , laboratoire langage et systèmes informatiques.
1988.

[BOCC.86] J.Bocca, H.Decker, J.M.Nicolas, L.Vieille, M.Wallace
some steps towards a DBMS based KBMS.
IFIP world congress 1986 Elsevier pub. PP.1061-1067

[CHANG.73] Chang C.L. Lee R.C.T.
Symbolic logic and mechanical theorem proving
Academic press, New York. 1973

[CHANG.78] Chang C.L.
Deduce 2: Further investigations of deduction in relational databases. (H.Gallaire and J.Minker EDs.)
Plenum Publishing corp.New York 1978
pages 201 - 236.

[CHANG.81] Chang C.L.
On evaluation of queries containing derived relations.
In :Advances in database theorie
(H.Gallaire ,J.Minker and J.M.nicolas eds)
vol1 , plenum press , New York
1981 pages 235-260.

[CHANG.86] C.L.Chang , A. Walker
PROSQL : A PROLOG programming interface with SQL/DOS, 1 st int. workshop on expert data base systems , Benjamin /Cumming pub. L. kershberg ed. 1986

[CLAR.78] Clark K.L.
Negation as failur in logic and database
H.Gallaire ,J.Minker eds, Plenum, New york, pp. 293-322 1978

[COSCIA.86] P.Coscia P.Franceschi , J. Kouloumdjian
The epsilon knowledge base management systeme: Architecture and data base access optimisation . 1986

[GALL.78] H.Gallaire
Logic and databases.
Plenum Pub.Corp., New York 1978.

[GALL.81] H.Gallaire and J.Minker
Logic and databases
Plenum Press, New York 1981

[GALL.84] H.Gallaire, J.Minker, J.M.Nicolas
Logic and data base :A deductive approche.
Computing surveys, vol. 16, n°2 june 84.

[GARD.85] Gardarin G. De Maindreville
Evaluation of database recursive logic programs as recurent function series.
rapport interne INRIA nov. 1985

[HENS.84] Henscen L.Naqvi S.A.
On compiling queries in recursive first order databases
JACM vol 31 N°1 jan 1984 pages 47-85

[JARK.84] M.Jarke, Y.Vassiliou
Data base and expert systeme: oportunities and architectures for integration
in new applications for data bases
G.Gardarin, E.Gelembe eds. Academic press ,1984.

[KOU.86] J.Kouloumdjian, A.Maakeb
Realisation d'une interface entre PROLOG et bases de données hétérogéne rel.-CODASYL
2éme journée bases de données avancées ,Gien avril 1986 PP.425-433.