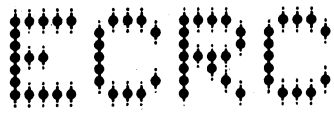
in: *Compte-rendus des 4èmes Journées*
Bases de Données Avancées
17-20 mai 1988, Bénodet, France



Internal Report KB-48
December 87

Préserver l'Intégrité d'une Base de Données Déductive : une Méthode et son Implémentation

François Bry
Hendrik Decker

European Computer-Industry
Research Centre GmbH
Arabellastr. 17
D-8000 Muenchen 81
West Germany

in: *Compte-rendus des 4èmes Journées Bases de Données Avancées 17-20 mai 1988, Bénodet, France*

PRESERVER L'INTEGRITE D'UNE BASE DE DONNEES DEDUCTIVE : UNE METHODE ET SON IMPLEMENTATION

François Bry et Hendrik Decker
ECRC, Arabellastr. 17, D - 8000 München 81

Résumé

Si une base de données est astreinte à satisfaire des contraintes d'intégrité, cette propriété doit être vérifiée à chaque modification de faits ou de règles de déduction. Des méthodes de maintien de l'intégrité ont été proposées, qui simplifient l'évaluation des contraintes d'intégrité dans une base de données mise à jour. Ces méthodes tirent partie de l'intégrité de la base de données initiale. Cet article décrit les principes et l'implémentation d'une telle méthode. Une approche nouvelle est proposée, consistant à s'appuyer autant que possible sur l'évaluateur de requêtes du SGBD. Cette approche étend d'autres propositions en considérant des transactions de mise à jour générales et des règles de déduction récursives. Elle permet une implémentation très simple, reposant sur Prolog et sur une technique de "programmation par méta-règles". Cette technique consiste à utiliser le langage de règles de déduction comme langage de programmation.

1. Introduction

Une base de données est dite "intègre" lorsque ses faits - explicitement stockés aussi bien que dérivables - satisfont certaines propriétés, les contraintes d'intégrité. L'intégrité d'une base de données peut être compromise à chaque modification de faits ou de règles de déduction. [Nicolas 79] a montré, pour des bases de données relationnelles, comment tirer partie de ce que la base de données courante est intègre pour établir à moindre coût que la base de données mise à jour l'est aussi. La méthode consiste à reconnaître celles des instances des contraintes d'intégrité dont les évaluations peuvent être affectées par une mise à jour. Elle nécessite que les contraintes d'intégrité soient "indépendantes du domaine", une propriété généralement admise pour tout type de requêtes posées à une base de données.

Etendre cette méthode de maintien de l'intégrité aux bases de données déductives est, d'un point de vue théorique, relativement simple. De même que les faits explicitement stockés d'une base de données déductive définissent avec les règles de déductions des faits induits, les mises à jour explicites définissent des mises à jour induites. Il est nécessaire et suffisant de considérer toutes les mises à jour, explicites aussi bien qu'induites, pour étendre la méthode aux bases de données déductives. Plusieurs telles extensions ont été proposées, dont les principes ont été initialement présentés dans [Decker 86] et [Lloyd-Topor 86] (ces deux méthodes ne diffèrent que dans la détermination des mises à jour induites). Elles dépendent toutes de l'évaluateur de requêtes considéré. Pour [Decker 86] et [Lloyd-Topor 86], cet évaluateur est Prolog. Pour [Sadri-Kowalski 86], c'est une méthode de résolution linéaire. [Martens-Bruynooghe 87] considère la méthode du graphe règles/but de Ullman. Aucune ne prend en compte des règles de déduction récursives ou des mises à jour générales.

L'approche décrite dans cet article, partiellement présentée dans [Bry et al. 88], est au contraire indépendante de l'évaluateur de requêtes. Si les mêmes évaluateurs sont considérés, elle étend et améliore les méthodes précedemment citées. Si l'évaluateur considéré traite correctement les règles de déduction récursives, il en est de même de la méthode de maintien de l'intégrité. Cette approche permet très facilement de considérer des mises à jour multiples (transactions) spécifiées par des formules. De plus, la base de données mise à jour est simulée, afin qu'aucune modification ne soit réalisée avant que la transaction ait été validée. Finalement, nous proposons une implémentation très simple, qui repose sur Prolog et sur une technique de "programmation par méta-règles".

La programmation par méta-règles consiste à utiliser le langage de règles de déduction du SGBD pour

implémenter certaines procédures du système. C'est une technique classique en programmation logique, qui est apparue au tout début de l'utilisation de Prolog, avec la définition de procédures telles que "setof", "once", etc... Son utilisation en Prolog a été ensuite étudiée plus systématiquement dans [Bowen-Kowalski 82]. La programmation par méta-règles ne nécessite en principe aucune des particularités de Prolog - prédicats non logiques, schéma d'exécution, etc... Elle est possible avec tout système à base de règles de déduction, en particulier avec un SGBD déductif, quel que soit le principe d'évaluation, génératif ou déductif, orienté tuple ou orienté ensemble, etc... Nous souhaitons montrer par cet article que la programmation par méta-règles permet d'implémenter très simplement certaines procédures d'un SGBD déductif. Nous pensons qu'elle peut être utilisée en bases de données à d'autres fins que celles décrites ici.

L'article est organisé de la manière suivante. La première section est cette introduction. La section 2 donne des définitions et précise les notations. Les principes de la méthode de maintien de l'intégrité sont introduits en section 3. Son implémentation est décrite en section 4. La section 5 est une conclusion.

2. Définitions

Une base de données déductive B est formée de trois ensembles finis : un ensemble F de faits, un ensemble R de règles de déduction et un ensemble de contraintes d'intégrité. Si R est vide, B est une base de données relationnelle. Un fait est un atome complètement instancié.

Une règle de déduction est une expression H ← C dont la tête H est un littéral positif et le corps C un littéral ou une conjonction "L_1 et ... et L_n" de litéraux. Nous supposons que les règles sont sans fonction et qu'elles vérifient la propriété du champ restreint, i.e. chaque variable de la tête H ou d'un littéral négatif du corps C de la règle figure dans un littéral positif de C. L'ensemble des faits dérivables est l'ensemble des tuples de l'interprétation standard de F∪R, définie dans [Apt et al. 87]. Afin que cette interprétation soit définie de manière unique, on suppose que R est stratifié [Apt et al. 87]. Les négations sont interprétées par l'échec. L'ensemble des faits dérivables est indépendant de l'ordre dans lequel règles de déduction et faits sont spécifiés.

Les contraintes d'intégrité sont des formules fermées, sans fonction et à quantifications restreintes. Cette propriété signifie que les sous-formules quantifiées sont de la forme "$\forall x_1...x_n$ $(A_1 \wedge ... \wedge A_m => G)$" ou "$\exists x_1...x_n$ $(A_1 \wedge ... \wedge A_m \wedge G)$" où les A_i sont des atomes tels que chaque variable x_j figure dans au moins l'un d'entre eux, et où G est une formule. La conjonction $(A_1 \wedge ... \wedge A_m)$ est le champ des variables x_1, ..., x_n. De telles formules seront représentées en Prolog par "pourtout([X_1, ..., X_n],A_1 et ... et A_m => G)" et "existe([X_1, ..., X_n],A_1 et ... et A_m et G)" (nous supposons que les connecteurs logiques sont déclarés comme des foncteurs Prolog). Notons qu'une requête en calcul relationnel de tuples correspond à une formule à quantifications restreintes. Pour une large classe de formules indépendantes du domaine, il est possible de générer des équivalents logiques à quantifications restreintes [Decker 88].

La requête d'insertion (resp., de suppression) d'une règle de déduction H ← C est de la forme "insérer(H←C)" (resp., "supprimer(H←C)"). Une mise à jour de tuples s'exprime par "maj(L, Q)", où L est un littéral et Q une formule dont la fermeture existentielle est à quantifications restreintes. Un littéral positif représente un insertion, un littéral négatif une suppression. La formule Q est la qualification de la mise à jour. On suppose que les variables du littéral L figurent (comme variables libres) dans Q. L'évaluation de Q sur la base de données courante retourne un ensemble d'instanciations σ_1, ..., σ_n pour ces variables. Les litéraux complètement instanciés $L\sigma_1$, ..., $L\sigma_n$ sont les mises à jour explicitement requises. L'insertion (resp., la suppression) d'un atome complètement instancié A s'exprime "maj(F, vrai)" (resp., "maj(¬F, vrai)"). Un remplacement de tuples s'exprime par une suppression et une insertion de même qualification.

Une transaction T est un ensemble de requêtes de mise à jour de tuples ou de règles. Si B est une base de données et T une transaction, on notera T(B) la base de données mise à jour. T(B) est définie de la manière suivante :

- Soit M(B, T) l'ensemble des mises à jour de tuples explicitement requises par T sur B. Si A est un atome de

M(B, T) et si A n'est pas explicite dans B, A est ajouté à B. Si $\neg A \in M(B, T)$ et si A est explicite dans B, A est supprimé de B.

- Si T contient une requête "insérer(H←C)", la règle H ← C est ajoutée à B.
- Si T contient une requête "supprimer(H←C)" et si cette règle est dans B, elle est retirée de B.

Cette sémantique déclarative permet qu'une transaction exprime des requêtes contradictoires. Par définition, une transaction T est contradictoire s'il existe un atome A dérivable dans T(B) ou élément de M(B, T) tel que $\neg A \in M(B, T)$ - i.e. T requiert que A soit faux dans T(B). Les transactions contradictoires doivent bien sûr être refusées.

Un littéral positif A (resp., un littéral négatif $\neg A$) est une mise à jour potentielle explicitement définie par une transaction T si T contient une requête "insérer(A←C)" ou "maj(A, Q)" (resp., "supprimer(A←C)" ou "maj($\neg$A, Q)").

Nous supposons que toutes les formules - contraintes d'intégrité et qualifications - sont rectifiées, i.e. les variables ont été renommées en sorte que le même symbole de variable n'apparaîsse pas dans deux quantifications. On dira qu'un atome A figure positivement (resp., figure négativement) dans une formule F si A figure dans le champ d'aucune négation ou d'un nombre pair de négations (resp., dans le champ d'un nombre impair de négations), les prémisses d'implications étant considérés comme des négations implicites.

3. Principes de la méthode

3.1. Indépendance du domaine et maintien de l'intégrité

L'évaluation de certaines formules dépend de l'ensemble - le domaine - des constantes figurant dans la base de données. C'est le cas par exemple des requêtes "$\forall x\ p(x)$" et "$\exists x \neg p(x)$". D'autres formules peuvent être évaluées sans consulter explicitement le domaine. Le concept de formule indépendante du domaine ou "définie" [Kuhns 67] définit formellement ces formules.

Définition 1 :
Une formule F est indépendante du domaine si pour toute base de données B, la valuation de F dans B ne dépend que des valuations dans B des atomes figurant dans F.

Afin de permettre des évaluations de requêtes efficaces, on suppose généralement que les requêtes posées à une base de données sont indépendantes du domaine. Les formules à quantifications restreintes - en particulier celles exprimées en calcul relationnel de tuples - sont indépendantes du domaine. L'indépendance du domaine des contraintes d'intégrité est une propriété essentielle pour une vérification efficace de l'intégrité. Ainsi que l'établit la proposition 1 ci-dessous, elle permet de reconnaître par un critère syntaxique assez simple ne demandant aucune évaluation de requête certaines contraintes d'intégrité non affectées par une transaction.

Définition 2 :
Une formule F est invariante pour une transaction T relativement à la base de données B si les valuations de F sur B et sur la base de données mise à jour T(B) sont identiques.

Notons $\Delta(B, T)$ l'ensemble des mises à jours requises ou induites par une transaction T sur la base de données B. $\Delta(B, T)$ est l'ensemble des litéraux complètement instanciés vrais dans T(B) et faux dans B.

Proposition 1 :
Soient B une base de données et F une formule indépendante du domaine.
F est invariante pour T relativement à B si aucun littéral positif (resp., négatif) de $\Delta(B, T)$ n'est unifiable avec un atome figurant négativement (resp., positivement) dans F.

[**Preuve** : Si aucun littéral négatif (resp., positif) de $\Delta(B, T)$ n'est unifiable avec un atome figurant positivement (resp., négativement) dans F, alors les valeurs prises par les atomes figurant dans F sont identiques dans B et T(B). F étant indépendante du domaine, elle est évaluée identiquement dans ces deux bases de données.]

On vérifie aisément que ce résultat ne s'étend pas aux contraintes d'intégrité qui ne sont pas indépendantes du domaine. D'après la proposition 1, il est possible d'établir l'intégrité d'une base de données mise à jour T(B) sans évaluer toutes les contraintes d'intégrité dans T(B), pourvu (1) que la base de données initiale B soit intègre, (2) que T ne soit pas contradictoire et (3) que les contraintes d'intégrité soient indépendantes du domaine. Il suffit de garantir que les contraintes non invariantes pour T sont satisfaites dans T(B).

3.2. Générer des contraintes dynamiques

Si les contraintes d'intégrité sont à quantifications restreintes, elles sont indépendantes du domaine et la proposition 1 s'applique. Pour ce type de contraintes, il est de plus suffisant d'évaluer des instances simplifiées des contraintes non invariantes pour T pour établir que ces contraintes sont satisfaites. Ces instances simplifiées sont définies ci-dessous. Sans que ce soit systématiquement rappelé, les contraintes d'intégrité considérées dans le reste de cet article seront toujours supposées à quantifications restreintes. Afin de simplifier la définition 3 ci-dessous, sans perdre en généralité, nous supposons de plus qu'elles sont normalisées de la manière suivante :

- Les équivalences et implications sont traduites à l'aide des connecteurs $\wedge$, $\vee$ et $\neg$.
- Les négations n'apparaissent que devant des atomes (forme normale négative).
- Le champ des quantificateurs est réduit autant que possible (champ minimum).
- Les quantificateurs universels sont distribués sur les conjonctions.

Définition 3 :

Soit CI une contrainte d'intégrité. Pour chaque littéral L de CI et chaque mise à jour potentielle M unifiable avec le complément de L, il existe une instance simplifiée IS de CI relative à M. IS est définie à partir d'une substitution τ que l'on obtient de la manière suivante :

1. déterminer un unificateur le plus général σ du complément de L et de M substituant des termes de M aux variables de L et (éventuellement) des constantes aux variables de M.
2. τ est la restriction de σ aux variables de M et aux variables universelles de CI qui ne sont pas quantifiées dans le champ d'une variable existentielle.

L'instance simplifiée IS s'obtient à partir de $CI\tau$ en :

1. éliminant les quantifications portant sur des constantes ou des variables figurant dans $M\tau$,
2. remplaçant chaque littéral N par "faux" si le complément de N est identique à M,
3. appliquant les lois d'absorbtion pour éliminer les occurrances de "faux".

On dit que la substitution τ définit l'instance simplifiée IS.

Considérons une contrainte d'intégrité CI_1 : $\forall x\ \neg p(x) \vee q(x)$ et une mise à jour potentielle p(v). La forme simplifiée associée est q(v) : en présence d'une insertion p(v), il est suffisant d'évaluer q(v) pour garantir que CI_1 reste satisfaite dans la base de données mise à jour. Soit une contrainte d'intégrité CI_2 : $\forall xy\ \neg p(x, y) \vee [\exists z\ q(x, z) \wedge \neg s(y, z, a)]$. La forme simplifiée de CI_2 associée à une mise à jour $\neg q(c_1, c_2)$ est $\forall y\ \neg p(c_1, y) \vee [\exists z\ q(c_1, z) \wedge \neg s(y, z, a)]$. Seule la variable x est instanciée. z ne l'est pas, car c_2 n'est pas nécessairement la seule valeur possible de z pour $x = c_1$ et y quelconque. De manière semblable, l'insertion potentielle s(u, v, w) conduit à l'instance simplifiée $\forall x\ \neg p(x, u) \vee [\exists z\ q(x, z) \wedge \neg s(u, z, a)]$ et à la substitution $w \leftarrow a$: seules les instances de s(u, v, w) vérifiant w = a imposent l'évaluation de la forme simplifiée précédente. Plusieurs exemples sont commentés dans [Nicolas 79], où cette technique fut introduite initialement.

Désignons par δ un opérateur tel que pour tout littéral L, $\delta(L, T)$ est vrai si et seulement si $L \in \Delta(B, T)$. De manière semblable, représentons par $\nu(F, T)$ l'évaluation d'une formule F dans la base de données mise à jour T(B). Il est montré au paragraphe 4.4 comment programmer les opérateurs δ et ν en méta-règles.

Définition 4 :
Soient CI une contrainte d'intégrité, L le complément d'un littéral figurant dans CI et IS une instance simplifiée de CI relative à L définie par une substitution τ.
La fermeture universelle de l'expression $\delta(L\tau, T) \Rightarrow \nu(IS, T)$ est une contrainte dynamique associée à la mise à jour potentielle L.

La vérification de l'intégrité d'une base de données mise à jour peut se faire à l'aide des contraintes dynamiques.

Proposition 2 :
Soit B une base de données intègre.
T(B) est intègre si T n'est pas contradictoire et si les contraintes dynamiques associées aux contraintes d'intégrité sont satisfaites.

[**Preuve** : (esquissée) Supposons T non contradictoire. Si les contraintes dynamiques sont satisfaites, alors toutes les contraintes d'intégrité contenant des litéraux unifiables avec des compléments de litéraux de $\Delta(B, T)$ sont satisfaites dans T(B). Puisque les contraintes sont indépendantes du domaine, cela suffit pour garantir qu'elles sont satisfaites dans T(B).]

Ce résultat fut donné initialement dans [Nicolas 79] pour des contraintes à champs restreints. A condition qu'une définition appropriée des instances simplifiées soit considérée, il s'étend à d'autres classes de contraintes d'intégrité indépendantes du domaine. Les contraintes "évaluables" [Demolombe 82] sont considérées dans [Bry 87].

Les contraintes dynamiques peuvent être calculées à partir des seules contraintes d'intégrité. Elles ne dépendent donc pas des faits et peuvent être déterminées à l'avance. Etant donnée une transaction T, il est possible de calculer sans consulter les faits un ensemble E(B, T) de mises à jour potentielles tel que la satisfaction des contraintes dynamiques associées aux éléments de E(B, T) et la non contradiction de T suffisent à garantir l'intégrité de T(B). La construction de cet ensemble est décrite en 4.2.

La non-contradiction d'une transaction T peut être établie en évaluant les expressions "non ($Q_1\sigma$ et $Q_2\sigma$)" (resp., "non (nu($C\sigma$) et $Q_2\sigma$)") pour toute paire de mises à jour "maj(A_1, Q_1)" (resp., "insère($A_1 \leftarrow C$)") et "maj($\neg A_2$, Q_2)" de T telle que A_1 et A_2 admettent un unificateur le plus général σ. Comme les contraintes dynamiques, ces expressions peuvent être pré-déterminées sans accès aux faits.

3.3. mises à jour potentielles induites par une transaction

La proposition suivante fournit une caractérisation d'un sur-ensemble de $\Delta(B, T)$.

Proposition 3 :
Soit A (resp., $\neg A$) un littéral positif (resp., négatif) de $\Delta(B, T)$. Une des propriétés suivantes est vérifiée :

1. A (resp., $\neg A$) est une mise à jour explicitement requise par T.
2. Il existe une règle de déduction $H \leftarrow C$ dans B, non supprimée par T, et une substitution σ telles que $H\sigma = A$, $C\sigma$ est vraie dans T(B) et $C\sigma$ contient un littéral L de $\Delta(B, T)$ (resp., un littéral L dont le complément appartient à $\Delta(B, T)$).
3. T demande l'insertion (resp., la suppression) d'une règle de déduction $H \leftarrow C$ et il existe une substitution σ telle que $H\sigma = A$ et $C\sigma$ est vraie dans T(B) (resp., dans B).

[**Preuve** : Si un littéral positif A (resp., négatif $\neg A$) de $\Delta(B, T)$ n'est pas explicitement stocké dans T(B) (resp., dans B), c'est nécessairement un fait dérivable dans T(B) (resp., dans B). Par définition de $\Delta(B, T)$, A

(resp., $\neg$A) est faux dans B et vrai dans T(B). Il existe donc une preuve de A (resp., $\neg$(A) dans T(B) qui n'est pas valide dans B. L'énoncé de la proposition formalise cette remarque.]

La proposition 3 fournit une caractérisation des mises à jour induites par une transaction T. De manière semblable, il est possible de caractériser les mises à jour potentielles induites par T.

Proposition 4 :
Soient T une transaction, A un atome (pouvant contenir des variables) et B une base de données intègre.
A (resp., $\neg$A) est une mise à jour potentielle induite par T si une des deux propriétés suivantes est vérifiée :

1. A (resp., $\neg$A) est une mise à jour potentielle explicitement définie par T.
2. Il existe une règle de déduction H $\leftarrow$ C et une substitution σ telle que Hσ = A et Cσ contient une variante (resp., une variante du complément) d'une mise à jour potentielle induite par T.

T(B) est intègre si et seulement si T est non contradictoire et si les contraintes dynamiques associées aux mises à jour potentielles induites par T sont satisfaites.

[**Preuve** : d'après les propositions 2 et 3. Le troisième cas de la proposition 3 est implicite dans le point 1 ci-dessus : une mise à jour de règle définit en effet une mise à jour potentielle de tuple.]

En s'appuyant sur ce résultat, il est possible de vérifier l'intégrité d'une base de données mise à jour T(B) en deux phases. Durant la première phase, les mises à jour potentielles induites par T sont déterminées. Les contraintes dynamiques associées sont générées. (Elles peuvent avoir été pré-déterminées ; elles sont en ce cas simplement rassemblées.) Ces opérations ne nécessitent aucun accès aux faits. Nous montrons dans la section suivante qu'elles peuvent être très élégamment programmées en Prolog. Dans une seconde phase, les contraintes dynamiques ainsi que les conditions de contradiction de la transaction sont évaluées. Nous montrons dans la section suivante comment spécifier en méta-règles les opérateurs δ et ν.

3.4. Comparaison avec d'autres méthodes

Les méthodes décrites dans [Decker 86, Sadri-Kowalski 86, Martens-Bruynooghe 87] ne vérifient pas l'intégrité d'une base de données mise à jour en deux phases. Elles entrelacent au contraire le calcul - en génération - des mises à jour (non potentielles) explicitement requises et induites et l'évaluation des formes simplifiées des contraintes d'intégrité. (La proposition [Ling 87] est semblable dans son principe à [Decker 86] mais incomplète en présence de règles de déduction : les mises à jour induites ne sont pas prises en compte.)

Outre leur dépendence à l'égard de la méthode d'évaluation de requêtes, ces approches souffrent de deux défauts. D'une part, toutes les mises à jour induites sont calculées, même celles qui n'ont aucun effet sur des contraintes d'intégrité. Cela est évité si les mises à jour potentielles sont au préalable calculées. Ce fut initialement proposé dans [Lloyd-Topor 86].

D'autre part, évaluer les instances simplifiées de contraintes d'intégrité indépendemment les unes des autres peut conduire à de coûteuses redondances. Transmettre un ensemble de contraintes dynamiques à l'évaluateur de requête permet au contraire de bénéficier de techniques d'optimisation susceptibles de reconnaître toutes ou certaines redondances.

Bien que ne présentant pas le premier de ces défauts, la méthode proposée dans [Lloyd-Topor 86] peut être elle aussi extrêmement redondante. En effet, au lieu de contraintes dynamiques de la forme "δ(L)=>ν(IS)" comme nous le proposons, cette méthode évalue des expressions "ν(L)=>ν(IS)", selon nos notations. En général, une expression "ν(L)" admet beaucoup plus de solutions que "δ(L)". Cette méthode conduit donc à évaluer beaucoup plus d'instances de contraintes d'intégrité que nécessaire.

Une particularité de l'approche décrite ici est de proposer une implémentation très simple pour les opérateurs δ et ν, i.e. pour simuler la base de données mise à jour (cf. 4.4).

4. Implémentation

4.1. Calculer les instances simplifiées

La génération des instances simplifiées est simple à programmer en Prolog. Supposons que les contraintes d'intégrité soient accessibles par un prédicat "contrainte(Id, CI, V)", où Id est un identificateur unique de la contrainte CI et V la liste des variables universelles de CI qui ne sont pas quantifiées dans le champ d'une variable existentielle. Pour des raisons d'efficacité, nous supposons qu'une relation "contient(L, Id)" indiquant les litéraux L figurant dans la contrainte d'identificateur Id ait été pré-déterminée. Un appel au prédicat suivant fournit, par retours arrières successifs, toutes les instances simplifiées IS de contraintes relatives à une mise à jour potentielle M :

```
instance_simplifiee(M, IS) :-
    contient(M, Id),
    complement(M, CM),
    contrainte(Id, CI1, V1),
    substitue(CI1, CM),
    contrainte(Id, CI2, V2), V2 = V1,
    simplifie(M, CI2, IS).

complement(non A, A) :- !.
complement(A, non A).

substitue(pourtout(V,F1=>F2), CM) :-
    !, (complement(CM, M), substitue(F1, M) ; substitue(F2, CM)).
substitue(existe(V,F), CM) :-
    !, substitue(F, CM).
substitue(F1 et F2, CM) :-
    !, (substitue(F1, CM) ; substitue(F2, CM)).
substitue(F1 ou F2, CM) :-
    !, (substitue(F1, CM) ; substitue(F2, CM)).
substitue(L, L).
```

L'instantiation partielle (cf. définition 3) est obtenue de la manière suivante : l'appel de "contrainte(Id, CI1, V1), substitue(CI1, CM)" retourne l'instance $CI1\sigma$. L'appel suivant "contrainte(Id, CI2, V2)" fournit une variante CI2 de CI1. L'affectation "V2 = V1" réalise l'instantiation désirée $CI2\tau$. Le prédicat "simplifie" se programme facilement à partir de la définition 3.

4.2. Déterminer les mises à jour potentielles induites par une transaction

Supposons que pour chaque règle de déduction $H \leftarrow C$ de B qui n'est pas supprimée par T et pour chaque littéral L figurant dans C un fait Prolog "induit(L, H)" ait été pré-determiné. Supposons que les mises à jour potentielles explicitement définies dans T soient retournées par retours arrières successifs sur le prédicat "maj_potentielle_explicite(M, T)". Les mises à jour potentielles L induites par une transaction T sont définies par les clauses suivantes :

```
maj_potentielle_induite(L, T) :-
    maj_potentielle_explicite(L, T).
maj_potentielle_induite(non L, T) :-
    !, induit(L1, L), maj_potentielle_induite(non L1, T).
maj_potentielle_induite(L, T) :-
    induit(L1, L), maj_potentielle_induite(L1, T).
```

L'ensemble E des mises à jour potentielles L induites par T est obtenu en appelant :

```
setof(L, maj_potentielle_induite(L, T), E)
```

Si certaines règles de déduction sont récursives, une infinité de mises à jour potentielles identiques aux noms des variables près peut être générée. Ce phénomème est aisément évité en testant si une mise à jour potentielle nouvellement générée est une instance d'une mise à jour précedemment produite. Ce test est également souhaitable lorsque les règles ne sont pas récursives, afin d'éviter des redondances dans E. La définition d'une version de "setof" réalisant le test d'instance est classique et relativement aisée : nous ne la donnons donc pas ici.

Soit T une transaction et E l'ensemble des mises à jour potentielles induites par T. L'ensemble F des contraintes dynamiques CD associées à T est obtenu par appel à :

```
setof(CD, contrainte_dynamique(CD, E, T), F)
```

où la procédure "contrainte_dynamique" est définie par :

```
contrainte_dynamique(pourtout(V, delta(M, T)=>nu(IS, T))), E, T) :-
    member(M, E), instance_simplifiee(M, IS).
```

Appeler "member(M, E)" retourne successivement, par retours arrières, les différents éléments M de E. La procédure "contrainte_dynamique" ne détermine pas la liste de variables V. Il est en effet montré au paragraphe suivant qu'elle n'est pas nécessaire pour l'évaluation des contraintes dynamiques.

4.3. Evaluer les formules quantifiées

Alors que les programmes précedemment définis reposent sur Prolog, nous utilisons dans ce paragraphe et le suivant le langage de règles de l'évaluateur de requêtes du SGBD considéré. Les solutions que nous proposons sont indépendantes - à la syntaxe près - de l'évaluateur de requêtes.

L'évaluation des formules à quantifications restreintes peut être spécifiée par les règles de déduction suivantes :

existe(V, F1 et F2) ← F1 et F2.
pourtout(V, F1=>F2) ← non existe(V, F1 et non F2).

Nous notons que les listes de variables V sur lesquelles les quantifications portent ne sont pas utilisées. La seconde définition suppose que des négations puissent figurer dans le corps d'une règle de déduction et soient évaluées par l'échec. Les évaluateurs de vues non récursives autorisent en général de telles définitions et traitent les négations de cette manière. De nombreux évaluateurs de règles récursives font de même, par exemple celui défini dans [Vieille 87]. L'extension des procédures d'évaluation de relations récursives à de telles définitions est activement étudiée en ce moment.

Ces définitions, en particulier la seconde, reposent sur la quantification restreinte. Les formules à quantifications restreintes sont intéressantes précisemment parce que leur évaluation peut être très simplement spécifiée par les règles qui précèdent.

Une procédure pré-définie pour l'évaluation des formules existentielles serait plus efficace que la définition par règle donnée ci-dessus. En effet, pour évaluer l'expression "existe(V, F1 et F2)", il n'est pas nécessaire de calculer toutes les solutions de "F1 et F2". Une telle procédure pré-définie est semblable au coupe-choix (!) de Prolog.

4.4. Spécifier les méta-prédicats "nu" et "delta" par des règles de déduction

Supposons que les mises à jour de tuples M explicitement définies par une transaction T soient retournées par appel à une relation auxilliaire de la base de données : "maj_explicite(M, T)". (Il est raisonnable de supposer que cette relation ait été pré-déterminée. En théorie, elle pourrait être définie par des règles de déduction.) Supposons que les règles de déduction R de la base de données considérée B soient stockées comme des tuples "regle(Id, R)", où Id est un identificateur de la règle R. Supposons que les insertions (resp., suppressions) de règles R requises par la transaction T soient stockées comme des faits "inserer(R, T)" (resp., "supprimer(Id, T)", où Id est l'identificateur de R). Supposons enfin qu'un appel à "litteral(L)" réussisse si et seulement si L représente, au moment de l'appel, un littéral. Le méta-prédicat ν simulant la base de données mise à jour T(B) peut être défini par :

nu(vrai, T).
nu(F1 et F2, T) ← nu(F1, T) et nu(F2, T).
nu(F1 ou F2, T) ← nu(F1, T).
nu(F1 ou F2, T) ← nu(F2, T).
nu(L, T) ← maj_explicite(L, T).
nu(L, T) ← litteral(L) et L et complement(L, CL) et non maj_explicite(CL, T).
nu(non A, T) ← non nu(A, T).
nu(A, T) ← regle(Id, A←C) et non supprimer(Id, T) et nu(C, T).
nu(A, T) ← inserer(A←C, T) et nu(C, T).

complement(A, non A) ← A≠non B.
complement(non A, A).

Nous supposons que le prédicat ≠ réalise un test et qu'il échoue lorsque l'un de ses arguments n'est pas instancié. Bien que le méta-prédicat "nu" apparaîsse dans le corps de règles le définissant, il faut remarquer que sa définition n'est proprement récursive que si certaines relations de la base de données sont elles-mêmes récursives. Si aucune relation n'est récursive, l'évaluation d'un terme "nu" à l'aide d'un évaluateur de vues conventionnel s'arrête toujours. Un évaluateur apte à traiter les règles récursives est toujours à même d'évaluer correctement un terme "nu". Ces remarques s'appliquent aussi au méta-prédicat "delta" défini ci-dessous.

Le méta-prédicat δ pourrait être défini par la règle de déduction "delta(L, T) ← nu(L, T) et non L". La définition suivante, inspirée de la caractérisation de Δ(B, T) donnée en proposition 3, est en général plus efficace. Nous supposons que pour toute règle de déduction A ← C d'identificateur Id de B (resp., pour toute règle A←C dont l'insertion est requise par T) et pour tout littéral L figurant dans la conjonction C, une expression "regle_compilee(Id, A, L, R)" (resp., "regle_compilee(A, L, R)") ait été pré-déterminée, où R représente C privé de L ("vrai" si C est réduit au seul littéral L).

delta(vrai, T).
delta(L, T) ← maj_explicite(L, T) et non L.
delta(A, T) ← regle_compilee(Id, A, L, R) et non supprimer(Id, T)
et delta(L, T) et nu(R, T) et non A.
delta(A, T) ← regle_compilee(A, L, R) et delta(L, T) et nu(R, T) et non A.
delta(non A, T) ← regle_compilee(Id, A, L, R) et non supprimer(Id, T)
et complement(L, CL) et delta(CL, T) et R et nu(non A, T).
delta(non A, T) ← regle(Id, A←C) et supprimer(Id, T) et C et nu(non A, T).

La sous-requête "non L" de la seconde règle sert à éliminer les litéraux L vrais dans la base de données T(B) qui étaient déjà satisfaits dans la base de données initiale B. Il est en effet possible qu'un littéral L satisfaisant "maj_explicite(L, T)" soit également satisfait dans la base de données initiale. Les dernières sous-requêtes des règles suivantes jouent un rôle semblable.

Les méta-règles définissant "nu" et "delta" fournissent une implémentation très simple d'un simulateur de la base de données mise à jour. Une simulation fondée sur un évaluateur dédié serait-elle nécessairement beaucoup plus efficace? Bien que la décomposition par "nu" ou "delta" de la requête à évaluer dans la base de donnée simulée induise un surcoût incontestable, il est permit d'en douter. Premièrement, ce surcoût est toujours faible comparé aux coûts d'accès à la mémoire secondaire. Une technique de mise à plat (unfolding) ou évaluation partielle [Sestof-Sørndergaard 87] permet de le supprimer complêtement. D'autre part, les optimisations qui ont été proposées pour l'évaluation d'instances simplifiées de contraintes d'intégrité sont également possibles pour l'évaluation de requêtes générales. Finalement, la complexité apparente de nombreux exemples de contraintes d'intégrité n'est pas supérieure à celle de requêtes syntaxiquement simples impliquant de nombreuses règles de déduction. Quoiqu'il en soit, des recherches devront être consacrées à reconnaître l'efficacité effective de l'implémentation que nous proposons.

5. Conclusion

Une méthode de maintien de l'intégrité indépendante de l'évaluateur de requête a été décrite et justifiée. Si les mêmes évaluateurs sont considérés, cette méthode améliore et étend d'autres propositions, définies pour des évaluateurs particuliers. Elle permet en particulier de prendre en compte des règles de déduction récursives, sous réserve que l'évaluateur considéré soit à même de les évaluer correctement.

Une implémentation de la méthode a été décrite. Cette implémentation repose d'une part sur Prolog - mais ne nécessite pas de couplage entre Prolog et le SGBD - et d'autre part sur une technique de "programmation par méta-règles". Cette technique consiste à utiliser le langage de règles de déduction du SGBD comme langage de programmation. Elle est possible avec tout SGBD déductif.

Les programmations en Prolog et en méta-règles nous semble être applicables à d'autres sous-systèmes d'un

SGBD déductif ou relationnel. Prolog permet souvent des implémentations extrêmement courtes et néanmoins efficaces. D'une manière générale, Prolog nous parait tout particulièrement approprié à la manipulation formelle de requêtes à des fins d'optimisation.

La programmation par méta-règle est une technique relativement classique en programmation logique que nous proposons d'utiliser en bases de données. C'est une direction pour de futures recherches que de reconnaître les diverses possibilités d'application de la programmation par méta-règles en bases de données. Il serait souhaitable de comparer l'efficacité de procédures programmées en méta-règles avec celle de procédures implémentées de manières conventionnelles.

6. Remerciements

Ce travail a grandement bénéficié des suggestions de Rainer Manthey et des conseils de Jean-Marie Nicolas, que nous sommes heureux de remercier.

7. Références

[Apt et al. 87] Apt, K.R., Blair, H. and Walker, A.
Towards a theory of declarative knowledge.
In *Proc. Workshop on Deductive Databases and Logic Programming.* Aug., 1987.

[Bowen-Kowalski 82]
Bowen, A.K. and Kowalski, R.A.
Amalgamating language and meta-language.
Logic Programming.
Academic Press, London, 1982, pages 153-172.

[Bry 87] Bry, F.
Maintaining integrity of deductive databases.
Internal Report IR-KB-45, ECRC, July, 1987.

[Bry et al. 88] Bry, F., Decker, H. and Manthey, R.
A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases.
In *Proc. 1st Int. Conf. Extending Data Base Technology (EDBT).* March, 1988.

[Decker 86] Decker, H.
Integrity enforcement on deductive databases.
In *Proc. 1st Int. Conf. on Expert Database Systems.* Apr., 1986.

[Decker 88] Decker, H.
The range form of databases and queries, or: How to avoid floundering.
Internal Report IR-KB-26, ECRC, 1988.

[Demolombe 82] Demolombe, R.
Syntactical characterization of a subset of domain independent formulas.
Technical Report, ONERA-CERT, Toulouse, France, 1982.

[Kuhns 67] Kuhns, J.L.
Answering questions by computers - A logical study.
Rand Memo RM 5428 PR, Rand Corp., Santa Monica, Calif., 1967.

[Ling 87] Ling, T.
Integrity constraint checking in deductive databases using the Prolog not-predicate.
Data & Knowledge Engineering 2, 1987.

[Lloyd-Topor 86] Lloyd, J.W. and Topor, R.W.
Integrity checking in stratified databases.
Technical Report 86/5, Univ. of Melbourne, May, 1986.

[Martens-Bruynooghe 87]
Martens, B. and Bruynooghe M.
Integrity constraint checking in deductive databases using a rule/goal graph.
Technical Report, Dpt. Computer Science, Katholieke Universiteit Leuven, 1987.

[Nicolas 79] Nicolas, J.-M.
Logic for improving integrity checking in relational databases.
Technical Report, ONERA-CERT, Toulouse, France, Feb., 1979.
Also in Acta Informatica 18, 3, Dec. 1982.

[Sadri-Kowalski 86]
Sadri, F. and Kowalski, R.A.
A theorem-proving approach to database integrity.
In *Proc. Workshop on Foundations of Deductive Databases and Logic Programming.* Aug., 1986.

[Sestof-Søndergaard 87]
Sestoft, P. and Søndergaard, H.
A bibliography on partial evaluation.
SIGPLAN Notices 23(2), 1987.

[Vieille 87] Vieille, L.
Recursive query processing: The power of logic.
Technical Report TR-KB-17, ECRC, Oct., 1987.