Un modèle formel des P-types Application à l'organisation des données

Mathieu ROGER
TIMC-IMAG Faculté de Médecine de Grenoble
38706 La Tronche Cedex – France
Email:
Mathieu.Roger@imag.fr
Fax: (33) 4 76 51 86 67
Tel.: (33) 4 76 63 71 39

Résumé *Le modèle de données des p-types a d'abord été développé pour répondre aux besoins des bases de données. Certaines de ses caractéristiques sont encore inhabituelles pour un DBMS et, par certains aspects, elles le rendent plus proche des systèmes de logique de description (DL) que d'un SGBD classique. Les vues ne sont pas seulement utilisées comme un raccourci pour les requêtes mais jouent un rôle central dans le système. Elles sont définies d'une manière hiérarchique avec des contraintes sur le type des rôles (ou attributs) comme dans les DLs, et le classement (ou identification) d'instance (view recognition) est un mécanisme de base de l'implantation des p-types par le système Osiris. Bien que le modèle ait été initialement décrit dans le paradigme des Types Abstraits Algébriques, ses similarités avec les DLs nous ont conduits à l'exprimer dans ce nouveau paradigme.*
mots-clé *: Vue persistante, Logique de description*

1 Introduction

Le modèle de données des P-types a été défini au début des années 80 [SAL 84] avec pour objectif de centrer le système sur l'utilisateur final. Ceci a été réalisé en donnant aux vues un statut à part entière. Dans le monde des bases de données, les vues sont synonymes de raccourcis pour des requêtes SQL. Elles ne sont ni persistantes ni "updatables", ne permettant donc pas de satisfaire les recommandations ANSI-SPARC [AXS 75], où les vues constituent le niveau externe d'une architecture à trois niveaux et doivent constituer la seule interface avec la base de données.
Ce papier s'intéresse à la mise à jour des vues persistantes dans un système de bases de données. Icic seules les vues dites « object-preserving » seront prises en comptes. Pour se faire nous proposons de regrouper les objets qui exhibent les même propriétés vis-à-vis des vues au sein des entités appelées Eq-Classes. Nous montrerons qu'une indexation des objets à travers les Eq-Classes permet le classement des objets

incomplets et la mises à jour a travers les vues (conformément à [AXS 75]). Nous montrerons aussi qu'une vue est un ensemble d'Eq-Classes consistantes. Ainsi, le concept d'Eq-Classe joue un rôle majeur tant pour l'indexation des objets que pour le calcul de la subsomption.
La deuxième partie décrit la syntaxe du langage des p-types ; on définit ensuite sa sémantique. La quatrième partie montre comment on peut alors indexer les objets d'une base de données à travers la partition ainsi réalisée.

2 Syntaxe du langage de description

Dans cette section, nous donnons une définition de la syntaxe du langage de description de concepts du système OSIRIS, qui implante le modèle des p-types. D'un point de vue opérationnel, ce langage sert à spécifier un schéma de base de données. Considérons :

- Un ensemble de types ***T*** (notés A,B,T) qui contient en particulier les types prédéfinis INT, REAL, CHAR, STRING, qu'on appellera Predefined,
- Un ensemble de vues ***V*** (notés U,V) qui contient les vues prédéfinies {]v_1,v_2[, [v_1,v_2[,]v_1,v_2] , [v_1,v_2], {a_1,…,a_n} | v_1 et v_2 sont tous deux éléments d'un des types INT, CHAR ou REAL et a_i sont tous des éléments d'un des types INT,REAL, CHAR ou STRING} qu'on appelera PredefinedViews (remarquez que cet ensemble est infini, en pratique, seul un nombre fini de vues prédéfinies seront utilisées par le designer de la base)
- Un ensemble de rôles ***R*** (noté R), chaque rôle pourra être considéré soit comme une relation binaire, soit comme la réunion d'un attribut avec son attribut inverse.

Ce sont des ensembles de noms disjoints. Les vues prédéfinies correspondent à la partition du domaine d'un attribut [SS 89], elles servent de briques élémentaires à la construction de la partition de l'espace des objets (voir définition 11). Comme dans [BDN *et al.* 98], on utilise deux langages distincts: un pour décrire les types (considérés comme des concepts primitifs), et un autre pour les vues (considérées comme des concepts définis).

Définition 1. Etant donné *T*,*V* et *R* nous définissons de manière inductive le *type concept language* (noté TL):

- R.A, où R$\in$ *R* et A$\in$ *T*
- R.A $\cap$ (= 1 R), où R $\in$ *R* et A $\in$ *T*
- C $\cap$ D, où C et D $\in$ TL

Remarque. L'objet de ce langage est de dire, pour un type donné, quels sont ses attributs, leurs types et leurs cardinalité (un ou plusieurs).

Exemple. Etant donné *T*={INT, STRING, PERSON, COURSE}, *R*={partner, age, namePERSON, nameCOURSE, follow, teach, teacher}, *V*={VIEWPERSON, VIEWCOURSE, STUDENT, TEACHER, TEACHINGASSISTANT}, voici des *type concepts*:

1. $\forall$partners.PERSON $\cap$ $\forall$ age.INT $\cap$ (= 1 age)
2. $\forall$teacher.PERSON $\cap$ (= 1 teacher) $\cap$ $\forall$ nameCOURSE.STRING $\cap$ (= 1 nameCOURSE)

Définition 2. Etant donné *T*,*V* et *R R* nous définissons de manière inductive le *view concept language* (noté VL):
1. V, où V $\in$ *V*
2. U $\cap$ V, où U et V $\in$ VL
3. $\neg$U, où U $\in$ VL
4. Undefined(R), où R $\in$ *R*
5. $\forall$ R.V, $\forall$ R. $\neg$V où V $\in$ *V* et R $\in$ *R*
6. $\exists$ R.V, $\exists$ R. $\neg$V où V $\in$ *V* et R $\in$ *R*

Les cas 4 à 6 sont appelés contraintes élémentaires. Elles sont de la forme attribut $\in$ domaine.

Remarque. Le but de ce langage est de décrire les sous-ensembles des types à l'aide de contraintes.

Exemple. Etant donné *T*={INT, STRING, PERSON, COURSE}, *R*={partner, age, namePERSON, nameCOURSE, follow, teach, teacher}, *V*={VIEWPERSON, VIEWCOURSE, STUDENT, TEACHER, TEACHINGASSISTANT}, voici des *view concepts*:
1. $\neg$Undefined(namePERSON) $\cap$ $\forall$ age. [0, 140]
2. VIEWPERSON $\cap$ $\forall$ age. [18, 100 [

Définition 3. Etant donné *T*,*V* et *R* un **type scheme** est un ensemble S d'axiomes qui peuvent être:
1. A $\subseteq$ C, où A $\in$ ***T****-Predefined* et C $\in$ TL
2. A $\subseteq$ $\neg$B, où A $\in$ ***T*** et B$\in$ ***T***
3. R $\subseteq$ A$\times$B, où R$\in$ ***R***, A $\in$ ***T****-Predefined* et B $\in$ ***T***
4. V = A $\cap$ U, où V $\in$ ***V***, A $\in$ ***T*** et U $\in$ VL

Tels que:
- ***Unicité de la définition d'un type:*** pour tout A $\in$ ***T****-Predefined*, il existe un unique axiome A $\subseteq$ C dans S. De tels types sont appelés *p-types*.
- ***Un rôle appartient à un seul type et tous les types sont définis:*** pour tout R il y a exactement un axiome R $\subseteq$ A$\times$B et A $\subseteq$ C tel que C utilise R.
- ***Les types sont disjoints:*** pour tout A,B dans ***T***, il y a un axiome A $\subseteq$ $\neg$B.
- ***Unicité de la définition d'une vue:*** pour tout V$\in$ *V* il y a exactement un axiome V = U.
- ***Les vues forment une hiérarchie:*** si on considère la relation binaire d'inclusion de vue "V utilise U dans sa définition, V = U$\cap$ …", le graphe dirigé, formé par les vues comme nœuds et cette relation, est acyclique et chacune de ses composante connexe possède une unique racine appelée *minimal view*.
- ***Une vue est sous-ensemble d'un unique type:*** pour chaque vue V il existe

exactement un p-type A tel que minimal_view(V) = A ∩ ….

- ***Il n'y a pas de cycle dans les contraintes de domains:*** si on considère la relation binaire d entre p-types telle que A d B si pour une certaine vue V de A la définition de V utilise une vue de B, alors le graphe formé des p-types comme nœuds et de d est acyclique.

Remarque. De manière informelle ces axiomes ont les significations suivantes:

1. Les types sont donnés par un ensemble de rôles
2. Les types sont disjoints
3. Les rôles ont un domaine et un codomaine
4. Les vues sont des définitions

Exemple. Etant donné *T*={INT, STRING, PERSON, COURSE}, *R*={partner, age, namePERSON, nameCOURSE, follow, teach, teacher}, *V*={VIEWPERSON, VIEWCOURSE, STUDENT, TEACHER, TEACHINGASSISTANT} nous pouvons définir le *type scheme* S avec les contraintes suivantes:

- PERSON ⊆ ∀partners. PERSON ∩ ∀ age. INT ∩ (= 1 age) ∩ ∀follow. COURSE ∩ ∀teach. COURSE ∩ ∀ namePERSON. STRING ∩ (= 1 namePERSON)
- COURSE ⊆ ∀teacher. PERSON ∩ (= 1 teacher) ∩ ∀ nameCOURSE. STRING∩ (= 1 nameCOURSE)
- VIEWPERSON = PERSON ∩ ¬Undefined(age) ∩ ¬Undefined(partner) ∩ ¬Undefined(namePERSON) ∩ ∀ age. [0, 100]
- STUDENT = VIEWPERSON ∩ ¬Undefined (follow) ∩ ∀ age. [0, 50[
- TEACHER = VIEWPERSON ∩ ¬Undefined (teach) ∩ ∀ age. [18, 70[
- TEACHINGASSISTANT = STUDENT ∩ TEACHER
- MAJOR= VIEWPERSON∩∀age.[18, 100[
- VIEWCOURSE=COURSE∩¬Undefined(teacher)∩¬ ndefined(nameCOURSE)

Remarque. Il s'agit d'une notation de type logiques terminologiques, ce qui est très utile pour la définition formelle du système. Pour l'utilisation courante, pour construire des schémas de bases de données, nous préférons une notation plus proche des systèmes à objets classiques, qui est celle du système Osiris.

3 Sémantique

La définition d'une interprétation suit une approche classique en logiques de description, qui est elle-même héritée de la logique classique. Le langage considéré est relativement simple, il s'agit d'un sous-ensemble de ALC. L'originalité de ce travail tient dans le traitement des valeurs indéfinies et inconnues.

La principale différence entre indéfini et inconnu est qu'une valeur inconnue à un certain moment "finira" par être connue plus tard, alors que si un attribut (ou rôle) a une valeur indéfinie, on ne peut pas lui assigner de valeur de son domaine "normal" de valeurs. Par exemple l'attribut *follow* est indéfini pour une personne qui n'est pas un étudiant, de la même manière que la fonction $\sqrt{x}$ est indéfinie pour $x < 0$.

Définition. Une interprétation I = (ΔI,.I) pour un type scheme S est donnée par un ensemble Δ^I et une fonction d'interprétation $^{.I}$ telle que les concepts et les rôles initiaux sont interprétés de la manière suivante:

- A^I est un sous-ensemble de Δ^I pour tout A dans ***T***
- V^I est un sous-ensemble de Δ^I pour tout V dans ***V***
- R^I est un triplet <Relation, Undefined, Unknown> avec Relation $\subseteq \Delta^I \times \Delta^I$, Undefined $\subseteq \Delta^I$, Unknown $\subseteq \Delta^I$ et $\{x \in \Delta^I \mid \exists\, y \in \Delta^I$ et $(x,y) \in$ Relation$\} \subseteq \Delta^I$ - (Undefined $\cup$ Unknown)

Et tel que les concepts complexes sont interprétés suivant les règles suivantes:

- $(\neg B)^I = \{x \in \Delta^I \mid x \notin B^I\}$, où $B \in$ ***T***
- $(\forall\, R.A)^I = \{x \in \Delta^I \mid \forall y \in \Delta^I: (x,y) \in Rel^I \Rightarrow y \in A^I\}$, où $A \in$ ***T***
- $(C \cap D)^I = C^I \ \cap D^I$, où C et D sont des view concepts, ou bien C et D sont dans TL
- $(\ (= 1\ R)\ )^I = \{x \in \Delta^I \mid \#\{y \in \Delta^I: (x,y) \in \text{Relation}\} = 1\ \}$, où $R \in$ ***R*** et R^I = <Relation, Undefined, Unknown>
- $(\text{Undefined}(R))^I = \{x \in \Delta^I \mid x \in \text{Undefined}\}$, où R^I = <Relation, Undefined, Unknown>
- V^I sont interprétés comme eux-mêmes lorsque $V \in$ PredefinedViews
- $(\forall\, R.V)^I = \{x \in \Delta^I \mid \forall y \in \Delta^I: (x,y) \in Rel^I \Rightarrow y \in V^I\}$, où $V \in$ ***V***
- $(\exists\, R.V)^I = \{x \in \Delta^I \mid \exists y \in \Delta^I\ (x,y) \in Rel^I$ et $y \in V^I\}$, où $V \in$ ***V***

Remarque. L'ensemble Undefined dans l'interprétation d'un rôle représente les objets pour lesquels le rôle n'a pas de valeur significative. C'est le cas par exemple de la fonction 1/x qui est indéfinie pour x=0. Par ailleurs, et comme dans les logiques de description, nous supposons que les noms des objets sont uniques (unique name assumption), i.e. deux éléments distincts de Δ^I représentent des objets distincts; dans le contexte d'une base de données ces noms sont des oids.

Définition 5. Etant donné I et J deux interprétations d'un type scheme S, on dit que J est une interprétation **plus spécifique** que I (noté $J \subseteq I$) ssi:

- $\Delta^I \subseteq \Delta^J$
- pour tout A dans *T*, $A^I \subseteq A^J$.
- pour tout V dans *V*, $V^I \subseteq V^J$.
- pour tout R dans *R*, R^I = <Relation$_1$, Undefined$_1$, Unknown$_1$>, R^J = <Relation$_2$, Undefined$_2$, Unknown$_2$>, Relation$_1 \subseteq$ Relation$_2$, Undefined$_1 \subseteq$ Undefined$_2$ et Unknown$_2 \subseteq$ Unknown$_1$.

Autrement dit J, une interprétation plus spécifique que I, contient moins d'informations inconnues et peut être plus d'objets que I.

Définition. Une interprétation I est dite **finie** si Δ^I est de cardinal fini.

Définition. Une interprétation I est dite **complète** si pour tout $R \in R$, tel que R^I =

<Relation, Undefined, Unknown>, on a Unknown = ∅.

Remarque. Intuitivement, une interprétation est complète si toutes les valeurs d'attributs pour tous les objets sont connus.

Définition 8. Soit un *type scheme* S, et une interprétation finie I, on dit que:

- I satisfait $A \subseteq C$ ssi $A^I \subseteq C^I$
- I satisfait $A \subseteq \neg B$ ssi $A^I \cap B^I = \emptyset$

I satisfait $R \subseteq A \times B$ ssi Relation $\subseteq A^I \times B^I$, Undefined $\subseteq A^I$ et Unknown $\subseteq A^I$, où R^I = < Relation, Undefined, Unknown>

- I satisfait $V = A \cap U$ ssi $V^I = A^I \cap U^I$

Définition. On dit qu'une interprétation finie satisfait un *type scheme* S (ou bien que I est une interprétation valide de S) ssi:
I satisfait tous les axiomes de S
Pour chaque p-type A , avec V comme vue minimale, on a $A^I = V^I$.
Il existe une interprétation complète J telle que J satisfait S et $J \subseteq I$

Remarque. La précédente définition nécessite quelques explications. Premièrement, on ne considère que des interprétations finies parce que œ sont celles qui traduisent le mieux notre intuition: en pratique, l'ensemble des objets stockés dans une BD est fini. La contrainte 1 est classique et dit qu'une interprétation valide doit vérifier toutes les définitions de concepts. La contrainte 2 dit qu'une vue minimale est en fait un concept primitif. Enfin, la contrainte 3 traite des valeurs inconnues en disant que dans une interprétation valide, si une valeur est inconnue, alors il sera possible de mettre une "vraie" valeur dans le futur.

Définition. On dit qu'une vue V_1 **subsume** une vue V_2 , ssi $V_2^I \subseteq V_1^I$ pour toute interprétation valide complète I.

Remarque. Tout comme en Logique de Description, la déduction ne s'intéresse pas aux valeurs inconnues. Ces valeurs n'apparaissent que dans la construction d'une interprétation particulière. Mais avant d'entrer les objets dans la BD, et ainsi construire une interprétation, la seule chose que l'on peut faire est de déduire des faits qui sont vrais pour toute les interprétations.

4 Application aux bases de données

Le but de cette section est de montrer comment on peut organiser le stockage des objets en fonction des définitions de vues. Cette méthode repose sur une partition de l'espace des objets et est utile au calcul de la subsomption. La solution proposée est proche de la notion de *Compound Concepts* dans [CAL 96].

4.1 Espace Virtuel

Définition. Etant donné un rôle R d'un p-type A, on appelle constraints(R) l'ensemble

des contraintes élémentaires (voir Définition 2) qui utilisent R ; un sous-domaine de stabilité pour R est ou bien une partiede constraints(R) ou bien le sous-domaine de stabilité sds-undefined$_R$.
L'interprétation d'un sous-domaine de stabilité s est respectivement $s^I = A^I \cap (\cap_{c \in s} c^I) - (\cup_{c \in s} c^I)$ et sds-undefined$_R{}^I$ = Undefined, où R^I = <Relation, Undefined, Unknown>.

Définition. Un sous-domaine de stabilité s est dit **consistant** si pour une certaine interprétation valide I on a $s^I \neq \varnothing$.

Exemple. Les sous-domaines stables consistants pour le p-type PERSON et le rôle age sont:

- Sds11={∀age.[0,50], ∀age.[0,70], ∀age.[0,100]} : ceux dont l'age est entre 0 et (inclus)
- Sds12={∀age.[0,70], ∀age. [0,100]} : ceux dont l'age est entre 50 (exclu) et 70 (inclus)
- Sds13={∀age.[0,100]} : ceux dont l'age est entre 70 (exclu) et 100 (inclus)
- Sds14={} : ceux dont l'age n'est pas entre 0 (inclus) et 100 (inclus)

Remarque. Ces ensembles sont appelés sous-domaines de stabilité pour une raison précise. Imaginons qu'on a un objet o qui soit dans l'interprétation de sds12, si sont age est modifié et que o reste dans l'interprétation de sds12, alors o vérifié toujours exactement les mêmes propriétés élémentaires, et donc appartient exactement aux mêmes vues qu'avant la modification. En résumé, en cas de modifications, si on reste dans le même sous-domaine de stabilité, alors on a encore les mêmes propriétés d'appartenance aux vues.

Propriété. Etant donnée une interprétation I, les interprétations des sous-domaines stable pour un rôle R d'un p-type A forment une partition de A^I.
Preuve. Soit un objet $o \in A^I$. Pour une contrainte C de constraint(R), soit o appartient à C^I, soit o appartient à $\neg C^I$. Ceci est vrai pour toutes les contraintes.

Définition. Etant donné un p-type A dont les rôles sont $R_1, \ldots, R_n$, une eq-classe pour A est $(s_1, \ldots, s_n)$ où chaque s_i est un sous-domaine de stabilité pour R avec comme interprétation $(s_1, \ldots, s_n)^I = \cap s_i^I$.
Propriété. Les eq-classes sont les classes d'équivalence pour la relation d'équivalence : vérifier exactement les mêmes contraintes élémentaires.

Les deux définitions suivantes ainsi que la propriété ont à voir avec le stockage d'une interprétation spécifique..

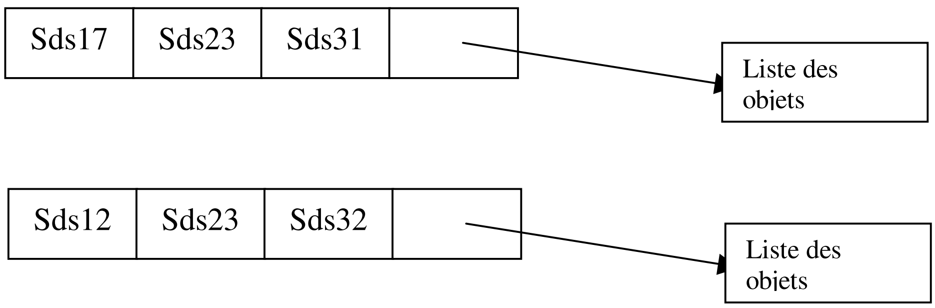
Propriété. Etant donnée une interprétation I, et un objet o tel que pour tout $R \in \mathcal{R}$, avec R^I = <Relation, Undefined, Unknown>, on a $o \notin$ Unknown (i.e. o complètement connu), alors il existe une unique eq-classe eq, appelée EquivalentClass(o,I) telle que $o \in eq^I$.

Définition. Etant donnée une interprétation I, et un objet o, on défini Possible(o,I) comme l'ensemble des eq-classe associées à o, EquivalentClass(o,J), pour toute interprétations J plus spécifique que I.

Propriété. Si o est complètement connu alors Possible(o,I) = {EquivalentClass(o,I)}.

Dans une implantation de bases de données, il est ainsi possible de ranger les objets dans des ensembles d'eq-classe. Par construction, touts les objets qui sont dans une eq-classe ont exactement les mêmes propriétés quant à l'appartenance aux vues. Ainsi, on peut attacher ces propriétés aux eq-classes, factorisant les informations communes aux objets qui y sont stockés. De manière pratique, et dans le cas de valeurs inconnues, on considère un sur-ensemble de Possible(o,I), pour cela tout se passe comme si l'attribut inconnu faisait partie du sous-domaine de stabilité constitué par l'union de tous les sous-domaines stables pour ce rôle. Ainsi, on peut ranger les objets dans des classes d'équivalences, i.e. la donnée d'un sous-domaine de stabilité pour chaque attribut (figure 1). L'implantation actuelle d'Osiris est faite de cette manière. Toutefois, dans la version actuelle, les domaines de contraintes sont limités aux intervalles entiers ou réels ([18, 25], [0.5, 8.3]) et aux ensembles énumérés ({bleu, rouge, vert}). Dans ce travail nous généralisons cette approche aux domaines définis par l'arrivée dans une vue quelconque.

Fig. 1. Deux eq-classes, avec leurs listes d'objets.



Lors de la modification d'un objet, il faut calculer l'eq-classe à laquelle il appartient désormais, si elle existe déjà, alors il n'est pas besoin de déterminer à quelles vues il appartient puisque ce travail est déjà fait.

4.2 Propriétés de l'espace virtuel

Définition. Etant donné un sous-domaine de stabilité s, on défini virtual(s) comme l'ensemble des eq-classes eq telles que eq = (…,s,…).

Définition. Etant donné une contrainte élémentaire C, on défini virtual(C) comme l'union des virtual(s) où s est un sous-domaine de stabilité et s contient C.

Propriété. Etant donné un sous-domaine de stabilité s, pour chaque interprétation I on a $s^I = \cup_{eq \in virtual(s)} eq^I$.

Propriété. Etant donné une contrainte élémentaire C, pour chaque interprétation I on a $(A \cap C)^I = \cup_{eq \in virtual(C)} eq^I$.

Propriété. Les sous-domaines stables ont des interprétations disjointes. Les eq-classes ont des interprétations disjointes.

Lemme. Etant donné un p-type A, et virtual(A) l'ensemble des eq-classes pour A, on a $A^I = \cup_{eq \in virtual(A)} eq^I$ pour chaque interprétation valide complète I.

Preuve: On a $\cup_{eq \in E} eq^I \subseteq A^I$ par définition. Maintenant démontrons l'inclusion inverse. Soit I une interprétation valide complète et $o \in A^I$, R est un role de A vers B. Il y a deux cas. D'abord R est indéfini pour o alors $o \in \text{sds-undefined}_R{}^I$. Ensuite, les valeurs pour o selon R sont connues et non indéfinies, pour chaque contrainte élémentaire C sur R, ou bien o satisfait C, ou bien o satisfait $\neg C$. On conclut que o appartient à l'interprétation d'un sous-domaine de stabilité pour R consistant, et donc que o appartient à l'interprétation d'une eq-classe.

Définition 15. Etant donné un p-type A, on appelle consistent(A) l'ensemble des eq-classes pour A telle que pour une interprétation valide I on a $eq^I \neq \emptyset$.

Lemme. Etant donné une vue V, il existe un ensemble d'eq-classes appellé virtual(V), tel que, pour toute interprétation valide complète I, on a $V^I = \cup_{eq \in virtual(V)} eq^I$.

Preuve : On étend la preuve donnée en [SAL 84] pour toutes les contraintes.

On a $V = A \cap U$. Démontrons ce lemme par induction sur la taille de U.

1. On a U = V. Par hypothèse, la propriété est vraie.

2. On a $U = U1 \cap U2$, par hypothèse d'induction sur U1 et U2 il existe $R1 = \{eq_1, \ldots, eq_n\}$ et $R2 = \{eq_1, \ldots, eq_n\}$. On définit R comme étant $R1 \cap R2$. Pour une interprétation valide complète I, on a $U^I = U1^I \cap U2^I = \cup_{eq \in R1} eq^I \cap \cup_{eq \in R2} eq^I = \cup_{eq \in R} eq^I$ parce que les eq-classes ont des interprétations disjointes.

3. On a $U = \neg U1$, soit $R = virtual(A) - virtual(U1) = \{eq_1, \ldots, eq_n\}$. Soit I une interprétation valide complète, soit o un élément de $(A \cap U)^I$. Par lemme 1, il existe $eq \in virtual(A)$ tel que $o \in eq^I$. Comme $o \notin U1^I$, $eq \notin virtual(U1)$, donc $eq \in R$. Ainsi on a $V^I \subseteq \cup_{eq \in R} eq^I$. Réciproquement, soit o élément de eq^I, avec eq dans R. Alors $o \in A^I$, par lemme 1. Supposons que $o \in U1^I$, alors, parce que les eq-classes ont des interprétations disjointes et $U1^I = \cup_{eq \in virtual(U1)} eq^I$, $eq \in virtual(U1)$, ce qui est en contradiction avec l'hypothèse. Donc $o \notin U1^I$, et ainsi $o \in V^I$.

4. On a U = Undefined(R). Soit S = virtual(sds-undefined$_R$). Soit I une interprétation valide complète, soit o un élément de $(A \cap U)^I$. Par la Propriété 1, il existe eq $\in$ virtual(sds-undefined$_R$) tel que o $\in$ eq^I. Donc $V^I \subseteq \cup_{eq \in S}$ eq^I. Réciproquement, soit o un élément de eq^I, avec eq dans S. Alors o $\in$ A^I, Par lemme 1. Par Définition 10, o $\in$ Undefined, où R^I = <Relation, Undefined, Unknown>. Donc o $\in$ $(A \cap U)^I$.

5. On a U = $\forall$ R.[a,b[. Soit S = virtual($\forall$ R.[a,b[), où $\forall$ R.[a,b[est considéré comme une contrainte C1. Soir I une interprétation valide complète, soit o un élément de $(A \cap U)^I$. Par rapport à chaque contrainte C sur R, et parce que I est complète, o appartient ou bien à C^I ou bien à $\neg C^I$. Donc il existe un sous-domaine de stabilité s tel que o appartient à s^I et C1 $\in$ s. Alors il est facile de voir que o $\in$ $\cup_{eq \in S}$ eq^I. Réciproquement, soit eq $\in$ S, et o $\in$ eq^I. Par Propriété 2, o $\in$ $(A \cap U)^I$.

Lemme. Etant donné une vue V du p-type A, on a $V^I = \cup_{eq \in \text{virtual(V)} \cap \text{consistent(A)}}$ eq^I.

Propriété. Dans un p-type A, une vue V1 subsume une vue V2, ssi virtual(V1) $\cap$ consistent(A) $\subseteq$ virtual(V2) $\cap$ consistent(A).

Preuve. Supposons que V1 subsume V2, alors pour une interprétation valide complète I, on a $V2^I = \cup_{eq \in \text{virtual(V1)}}$ eq^I $\subseteq$ $V2^I = \cup_{eq \in \text{virtual(V2)}}$ eq^I. Parce que les classes d'équivalence ont des interprétations disjointes, , on a virtual(V_1) $\subseteq$ virtual(V_2). Maintenant, supposons que virtual(V1) $\cap$ consistent(A) $\subseteq$ virtual(V2) $\cap$ consistent(A). Soit I une interprétation valide complète, alors on a $V2^I = \cup_{eq \in \text{virtual(V1)} \cap \text{consistent(A)}}$ eq^I $\subseteq$ $\cup_{eq \in \text{virtual(V2)} \cap \text{consistent(A)}}$ eq^I = $V2^I$.

5 Conclusion

Ce travail met l'accent sur un formalisme précis, hérité de la logique formelle. Il prolonge le travail présenté en [ROG 99]. En particulier, nous y présentons une façon de découper l'univers utile au rangement des objets dans une base de données. Cette méthode se confond avec des méthodes utilisées pour le classement de concepts, elle montre ainsi tout son intérêt central. On peut ainsi reprendre les méthodes employées dans les logiques terminologiques pour déterminer les sous-domaines de stabilités et les eq-classes consistants. Même si le nombre de classes d'équivalence est exponentiel en fonction du nombre de contraintes sur les attributs, il suffit en pratique de représenter celles qui sont non-vides à un moment donné. Ainsi, on est certain que le nombre d'eq-classes représentées est inférieur ou égal au nombre d'objets entrés dans la base. C'est le chois qui est fait dans le système Osiris.
Dans le futur, nous comptons utiliser plus avant le rapprochement commencé entre ce modèle et les modèles de DLs, et ainsi utiliser les résultats disponibles dans ce domaine. Nous pensons, notamment, au calcul de la subsomption en présence de contraintes de cardinalités qu'on peut trouver en [CAL 96]. Une autre voie d'étude consiste à ajouter un niveau de concepts au-dessus des p-types, ainsi on pourrait considérer des propriétés communes aux p-types.

Références

[AXS 75]: ANSI/ X3/ SPARC Study group on database management systems. Interim report, ACM SiGMOD Bulletin 7, N2, 1975.

[BBL *et al.* 93]: Beneventano, Bergamaschi, Lodi, Sartori, Using subsumption in semantic query optimisation, IJCAI Workshop on object based representation systems, Août 1993.

[BJS *et al.* 94]: M. Buchheit, M. Jeusfeld, M. Staud, W. Nutt, Subsumption between Queries to Object-Oriented Databases, EDBT, 1994.

[BDN *et al.* 98]: Buchheit, Domini, Nutt, Schaerf, A refined architecture for terminological systems : terminology = schema + views, Artificial Intelligence, Vol 99, 1998.

[CAL 96] Diego Calvanese, Finite Model Reasoning in Description Logics, in Proceedings of Knowledge Representation 1996

[DLN *et al.* 96]: Domini, Lenzerini, Nardi, Schaerf, Reasoning in description logics, Principles of Knowledge Representation, pp. 191-236, CSLI Publications, 1996.

[ROG 99]: M. Roger, Une sémantique de type logique terminologique pour un système de gestion des données et des connaissances, Journées ISI/IC : Ingénierie des Systèmes d'Information et Ingénierie des Connaissances, Paris, 17-18 Nov. 1999.

[SAL 84]: A. Sales, Types abstraits et bases de données, Thèse, Université scientifique et médicale de Grenoble,1984.

[SS 89]: A. Simonet, M. Simonet, Les classes d'équivalence induites par des dépendances inter-attributs: fondements pour une représentation des objets partagés dans les bases de connaissances - R.R. 769-I laboratoire Artemis - Grenoble, France, 1989.