An Approach to Subsumption in a DL with Implication

Mathieu Latourrette, Michel Simonet
Laboratoire TIMC-IMAG,
Faculté de Medecine de Grenoble
38706 La Tronche Cedex - France
e-mail : Mathieu.Latourrette@imag.fr, Michel.Simonet@imag.fr

Abstract

Designing a DL system poses the problem of the trade-off between power of expression and complexity. In spite of their rich expressive capacity, the **or** and **not** constructors are usually avoided because of the intractability of the resulting subsomption algorithm. So is implication. However, implication may constitute an interesting compromise between expressive power and tractability. We present an approach to the treatment of implication in the Osiris system, for mono-valued attributes with enumerated domains. It is based on the partitioning of each attribute's domain by the predicates on this attribute, and it has been implemented in a prototype version of the Osiris system.

1 Introduction

The designer of a DL system has to choose the constructors which he will implement and he is faced by the dilemma of the well-known trade-off between the power of expression with which he wants to provide the users, and the complexity of the resulting subsumption algorithm. Most DL systems avoid disjunction and negation because they render subsumption intractable. So is implication. However, it is used in some systems (e.g., BACK, LOOM) for instance classification although limited to its deductive aspect : if the antecedent is true, then the consequent follows. Implication is not used in concept definition because if it were it would imply that it is taken into account by the subsumption algorithm. We present an approach to subsumption which is based on the partitioning of the object space resulting from the partitioning of each attribute's domain by the predicates on this attribute. This partitioning is used in the Osiris system for instance classification with mono-valued attributes [Simonet et al., 94]. Here we show that it can also be used as a basis for concept classification with the same kind of attributes.

The Osiris system was designed as a view-based DBMS conceived within the Abstract Data Type paradigm, independently from the object and terminological currents [Simonet, 84]. It happens to be object-like, and more similar to a DL system than to current OO DBMS. However, one important difference with DL systems is that concepts are grouped into disjoint families of instances named P-types [1]. A P-type is defined as hierarchy of views (similar to concepts) with a root called the minimal view of the P-type. Although they constitute different abstract data types, it will be convenient to assimilate a P-type with its minimal view in the context of this paper and for the purpose of explaining the way implication is dealt with. In Osiris, instance classification has been designed prior to view (concept) classification for which our interest dates from an encounter with the DL community at the KRDB'95 workshop in Bielefeld. Instance classification relies on a partitioning of the object space, which we call the Classification space, and which is also used for object indexing and query optimization [Simonet et al., 96]. A C++ implementation of Osiris is under way, and it ensures the persistency of objects and views through an object manager or a relational DBMS. A Prolog prototype has also been realised to implement the classification of views, which is presented in this paper. We shall first present the Classification Space and instance classification, then our approach to subsumption based on the Classification Space. We end by studying query optimization based on this partitioning of the object space and the resulting indexing of the objects (instances) by its elements.

2 The Osiris Classification Space

2.1 The P-type Data Model

The P-type data model was defined in [Simonet, 84] with the objective of answering database needs. Its main concern was the sharing of objects by several kinds of users seeing them through one or several views. A P-type is organized in a hierarchy of classes, where classes model database views. An object belong to one and only one P-type, and to several views A view is defined by the views it specialises (none for the minimal view), its own attributes and/or assertions (logical properties). A very simple example of P-type is :

[1] P stands for the French *partage which means sharing*

view PERSON
attributes
Name : STRING,
Age : [0..150],
Sex : {M, F},
MilitaryService : {no, done, ongoing, deferred}
assertions
Sex : F → MilitaryService : no
Age < 18 → MilitaryService : no
Age ≥ 18 and Sex : M →
MilitaryService : {done,ongoing,deferred}

view MINOR : PERSON
attributes
Age : [0..17]

view SENIOR : PERSON
attributes
Age : [65..150]

The logical properties which define the views use Domain Predicates, e.g., predicates of the form :

Attribute ∈ *Domain*,

where the considered domains are of enumerated types. The assertions are Horn clauses where the literals are Domain Predicates. The minimal view of a P-type can be considered as a primitive concept while the other views of the P-type are defined concepts. Osiris can be considered as a subset of a DL with the constructors :

all R C, **some** R, **and**, **domain**, **range**, and →.

The above P-type definition is equivalent to the following terminological definitions written in BACK :

Name := domain (PERSON) and range(STRING)
type feature

Age := domain (PERSON) and range([0..150])
type feature

Sex := domain(PERSON) and range({M, F})
type feature

MS := domain(PERSON) and range({no, done,...})
type feature

PERSON :< some(Name) and some(Age)
and some(Sex) and some(MS) and
some(Age, [0..17]) →
all(MS, no) and
some(Sex, F) → all(MS, no) and
some(Age, [18..150]) and some(Sex, M) →
all(MS, {done, ongoing, deferred})

MINOR := PERSON and all(Age, [0..17])

SENIOR := PERSON and all(Age, [65..150])

Another important difference between P-types and DLs is that P-types represent disjoint sets of objects. Therefore it is not necessary to indicate in the definition of the P-type PERSON that it is neither a VEHICLE nor a MUSHROOM nor a DISEASE, which constitute distinct and disjoint P-types. This way of partitioning the modelled universe into disjoint subsets may seem awkward to DL people. In fact, this partitioning of the universe into disjoint P-types is not necessary at the modelling level but is useful (maybe necessary) to provide an efficient support for persistency. P-types are the implementation units for persistency, as can be seen in the relational implementation of Osiris [Quast et al., 95].

The relationships between two P-types are of two kinds : composition and inheritance. Composition allows the type of an attribute to be another P-type (or a view of a P-type, e.g., Possesses : VEHICLE), and inheritance expresses that a P-type inherits some properties of another P-type, which is not presented here.

2.2 The Classification Space

The **Classification Space** of a P-type is defined as the quotient space of the object space of the P-type relative to the equivalence relation : *to have the same truth values for all the Domain Predicates occurring in the assertions of the views of the P-type.*

For a given predicate, e.g., Age ≥ 18, this relation leads to a partitioning of its attribute domain into two elements : its extension ([18..150]), and its complement (]0..18[). The product of all the partitions [Stanat, 77] generated by all the Domain Predicates on a given attribute, e.g., Age, defines a partition of the domain of this attribute, and its elements are called Stable Sub-Domains (SSD). In effect, if the value of an attribute changes while remaining in the same SSD, all the predicates on this attribute keep the same truth value : *true* or *false*. The SSDs of the attribute Age are :
d11=[0..17], d12=[18..64], d13=[65..150].
Similarly, the attribute Sex is partitioned into :
d21={F}, d22={M},
and the attribute MilitaryService into :
d31={done, ongoing, deferred}, d32={no}.

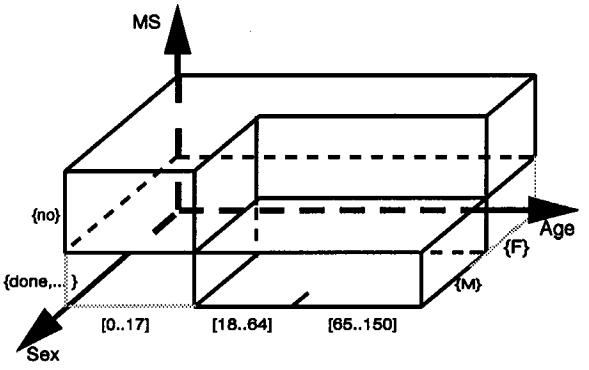

Figure 1: Classification Space of a P-type : PERSON

Prolonging this partitioning to the object space gives the Classification Space, whose elements are called Eq-

classes. The Classification Space can be represented by hypercubes, each representing an Eq-class (Fig. 1).

Because of its size, which is exponential to the number of attributes, the Classification Space is never represented in its totality, but it underlies the classification processes.

Classifying an instance consists in determining its views. In Osiris, it suffices to determine its Eq-class because each eq-class can be associated with the views of the P-type whose elements belong to it. Associating an eq-class with its views is not done once for all at compile time, but is performed for a given Eq-class the first time an instance which belongs to it is entered in the base. Only the Eq-classes containing at least an object are effectively created and they are used to index the objects of the base. This partitioning of the objects presents many similarities with that obtained by the horizontal fragmentation algorithm of [Ozsu et al., 91].

3 Subsumption

A view can be characterized as a set of eq-classes. Let Eq_i be the set of eq-classes *belonging* to the view V_i. Eq_i is not built explicitly. We have chosen to represent a view by two components :

- a **filter** F which limits the domain validity of the view. For a view V_i it is a superset of Eq_i, which is built from the definition domains of the attributes of V_i.
- a set C (for Complement) of tuples of attribute domains corresponding to the areas which are falsified by at least one assertion.

This choice of representing the complement of the valid part of the Classification Space instead of the valid part itself follows the study of [Bassolet et al., 96]. Both components F and C are Cartesian products of attribute domains. The minimal view PERSON is represented by:

Fp = [0..17] x {M,F} x {no, done, ongoing, deferred}
Cp = { [0..150] x {F} x {done, ongoing, deferred},
[0..17] x {M,F} x {done, ongoing, deferred},
[18..150] x {M} x {no}

The view MINOR is represented by :

Fm= [0..17] x {M,F} x {no, done, ongoing, deferred}
Cm= {[0..17] x {M,F} x {done, ongoing, deferred}}

and the view SENIOR by :

Fs = [65..150] x {M,F} x {no, done, ongoing, deferred}
Cs = {[65..150] x {F} x {done, ongoing, deferred},
[65..150] x {M} x {no}}

For a given view, each element of C is obtained as the intersection of F with the *complement* of an assertion. For example, in the minimal view PERSON, the first tuple of Cp represents the area falsifying the assertion

Sex : F → MilitaryService : no.

An instance is represented in the same manner. For example, the instance defined as :

John :: PERSON and all(MS, no) and all(Sex, M)

is represented by :

Fj = [0..150] x {M} x {no}

and the assertions Cp of the minimal view.
Restricting Cp to the space Fj gives :

Cj = {[18..150] x {M} x {no}}

The Eq-class space associated with the instance John is defined as Fj/Cj, which represents the symmetric difference between the sets Fj and Cj. This gives :

{ [0..17] x {M} x {no} },

which implies that John is less than 18.

The **subsumption** relationship $V1 \leq V2$ expresses that the view V1 is subsumed by the view V2, i.e., $V1 \subset V2$, or $Eq1 \subset Eq2$. This is equivalent to :

$$((C2 \cap F1) \cup (F1/F2)) \subset C1$$

Verifying this formula is equivalent to checking that a set of cubes is included in another set of cubes, which is a NP-complete problem. However, when views are defined only by domain restrictions (no assertions), the process is polynomial, because there is only one cube per set. The general method to check that $E1 \subset E2$ consists in verifying for each element of E1 that it is included in E2. This approach can be extended to domain predicates of the form *Attribute* ∈ *View*, e.g., Child : MINOR. However, we have not taken into account this form in the antecedent of an implication because this would lead to the complexity problems of the **not** operator. The presented method has been implemented in a Prolog prototype of Osiris. This prototype supports the propagation of object modification, which is not the case of the ongoing C++ implementation whose aim is more database-oriented.

4 Query Optimization

The partitioning of the object space into Eq-classes is the basis of the implementation of the Osiris system. Although the Classification Space is never represented in its totality, it underlies the whole Osiris implementation. The actual objects of the base are indexed by a structure containing their Eq-class and the corresponding views. This leads to an optimization of query evaluation based on the semantic properties of the views.

Osiris queries are of the form *(Context | condition)* where the *Context* part is expressed by a propositional formula Φ(Vi) on the views of a P-type. The *Condition* part expresses a logical condition on the attributes of the P-type and has the same form as the general Osiris assertions.

A query is rewritten in terms of SSDs and the corresponding elementary predicates. It may be labeled, thus defining a dynamic view (as opposed to the 'static' views which constitute the P-type). However, contrary to static views, dynamic views are not materialized and

the objects contained in a dynamic view have to be re-evaluated every time the view is invoked.

Indexing by Eq-classes enables the system to determine which Eq-classes are valid, invalid, and potential as regards the query. For example, the query

Q1: (PERSON | Age $\leq$ 40)

divides the Classification Space into three sets of Eq-classes :

Eq_v = (d11, *,*) : the persons aged under 18 satisfy the query.

Eq_i = (d13, *,*) : the persons aged over 65 do not satisfy the query.

Eq_p = (d12, *,*) : the persons whose age has to be checked.

The objects indexed by invalid Eq-classes (Eq_i) are automatically rejected and those indexed by valid Eq-classes (Eq_v) are part of the answer. Only the objects indexed by potential Eq-classes (Eq_p) have to be checked individually. Now consider the query :

Q2 : (person | Age > 40 and Income > 10 000).

The sets of objects to consider are represented in Fig. 2, reduced to the two dimensions Age (d1i) and Income (d4j). A couple of values V, I, or P is associated to each Eq-class, indicating its validity relative to the literals P1: Age > 40 and P2: Income > 10000 repectively.

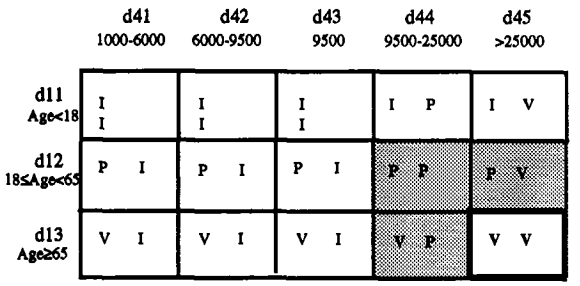

Figure 2: Analysis of the Classification Space of the query Q2

The Eq-class (d13,d45) is valid, and so are the objects which are indexed by it. The Eq-classes which are invalid for at least one predicate are excluded from the answer, and the objects of those which contain a V and a P must be checked individually. However, the test may be refined; for example, it is sufficient to check (d13, d44) for P2 and (d12, d45) for P1.

A dynamic view, defined by a query, can be made static, i.e., integrated to the definition of the P-type. This is a particular case of schema modification, which may lead to refine the SSD partitioning for certain attributes occurring in the Condition of the query. For example, Q1 would refine d12 into [18,40] and [41,64]. Only the objects belonging to the indexing structures containing a modified Eq-class have to be classified again and assigned to their new Eq-class. The benefit is that the view which has been made static is materialized, i.e., automatically maintained, and the access to their elements is straightforward through the indexing structure.

In [Buchheit et al., 94], the authors give the conditions necessary to perform semantic query optimization, based on class, view and query classification relative to subsumption. They propose a 'maximal' set of operators which keeps subsumption polynomial. However, in the object model which they consider, which is the 'standard' object model for databases, the views must be persistent and automatically maintained, i.e., the sets of objects which they designate must be modified when the database is updated. However, view materialization still remains an open problem in databases, even relational.

The 'object-preserving' condition defined by [Scholl et al., 91] is also implicit, because views are subsets of objects instanciated in classes and therefore do not create new objects.

Semantic query optimization consists in keeping the result of queries as persistent views and then classifying any new query relative to these views. The most specialized view represents the most restricted set of objects within which the query can be evaluated without loss of information. This approach shows the interest of reasoning on an intensional definition of views (or concepts), which is the basis of DL. The same approach underlies the view approaches of [Scholl et al., 91] and [Rundensteiner, 92], although it is less developed.

In Osiris, a query is evaluated within a given P-type. It is not classified explicitly relative to the other (static) views of the P-type. However, as it is rewritten in terms of the SSDs of the attributes which it contains, it would be possible to classify it. We think this would not be useful because we have a more refined indexing of objects through the Eq-classes.

5 Conclusion

The P-type data model was conceived as a database model, where the grouping of classes into disjoint families is natural. It is also indicated in [Buchheit et al., 94] as a condition which is necessary to perform query optimization in object databases by using subsumption. Such grouping can be considered as a way to provide a DL with a higher level of structuring, while it is almost necessary in a database perspective. However, it leads to some limitations, which will be overcome in Osiris by introducing the inheritance of properties in the Abstract Data Type sense.

Concept classification was not an initial aim of the Osiris system, where instance classification remains the main objective [Bassolet et al., 96] , and where query optimization is obtained through the indexing of objects by subsets of the Classification Space. However, concept (view) classification provides a higher level of static integrity checking and therefore will be introduced in the ongoing system written in C++, following the approach presented in this paper.

6 Bibliography

[Bassolet et al., 96]: C.-G. Bassolet, A. Simonet, M. Simonet, *Probabilistic Classification in Osiris, a View-based OO DBMS and KBMS*, DEXA96, 7th International DEXA Workshop on Database and Expert Systems Applications, Zurich, Sept. 1996.

[Buchheit et al., 94]: M. Buchheit, M. A. Jeusfeld, W. Nutt, M. Staudt, *Subsumption between queries to object-oriented database*, Information Systems 19(1):33-54, 1994.

[Ozsu et al., 91]: M. T. Ozsu, P. Valduriez, *Distributed Database Design : Fragmentation (Chap. 5.3)*, in Principles of Distributed Database Design, Prentice Hall, 1991

[Quast et al., 95]: M. Quast, A. Simonet, M. Simonet, *A Relational Implementation of a View-based Object System*, OOIS'95 (Object-Oriented Information Systems), J. Murphy and B. Stone eds, Dublin, pp. 111-117, Dec. 1995.

[Rundensteiner, 92]: E. Rundensteiner, *MultiView: A Methodology for Supporting Multiple Views in Object-Oriented Databases*, Proc. of the 18th VLDB Conference, Vancouver, Canada, 1992.

[Simonet, 84]: A. Sales-Simonet, *Types Abstraits et Bases de Données: formalisation du concept de partage et analyse statique de contraintes d'intégrité* - Thèse Docteur Ingénieur, Université Scientifique et Médicale de Grenoble, France, Avril 1984.

[Simonet et al., 94]: A. Simonet, M. Simonet, *Objects with Views and Constraints : from Databases to Knowledge bases*, OOIS'94 (Object-Oriented Information Systems), London Dec. 1994, Springer Verlag, pp 182-197.

[Simonet et al., 96]: A. Simonet, M. Simonet, *Classement d'instance et Evaluation des Requêtes en Osiris*, in BDA'96 : Bases de Données Avancées, Cassis, France, Août 1996.

[Scholl et al., 91]: M. H. Scholl, C. Laasch, M. Tresch, *Updatable Views in Object-Oriented Databases*, Proc. 2nd DOOD conf., pp 187-198, Dec. 1991

[Stanat et al., 77]: D. Stanat, D. McAllister, *Discrete Mathematics in Computer Science*, Prentice Hall, 1977.