Combining Resolution and Classification for Semantic Query Optimization in DOOD *

Mohand-Saïd Hacid and Christophe Rigotti

Laboratoire d'Ingénierie des Systèmes d'Information
INSA Lyon, Bâtiment 501
F-69621 Villeurbanne Cedex
{mohand,crig}@lisiecrin.insa-lyon.fr

Abstract. This paper proposes a framework for semantic query optimization in deductive object-oriented databases. The intentional database is described by means of clauses and a more restricted language is used for the integrity constraints. We apply a specific resolution and a classification mechanism to rewrite a query into a less expensive yet equivalent one. The main contribution of this paper is to show how resolution and classification can be used together within a common framework to perform complementary semantic query optimizations in deductive object-oriented databases.

1 Introduction

In object-oriented database management systems, query optimization is performed by transforming algebraic expressions and by taking into account the physical representation of data [19][8]. Semantic query optimization takes advantage of the semantic knowledge (e.g., integrity constraints) about the content of databases to reformulate a query into a less expensive yet equivalent query.

A number of techniques have been followed to perform semantic query optimization in classical deductive databases (see [7]). Recently, this problem has been addressed in the context of deductive object-oriented databases. Two different approaches have been proposed: the first, described in [20], is an extension of [7] for deductive object-oriented databases. It uses a specific resolution to reason about clauses. The main advantage of this approach is that these clauses allow to take into account general descriptions of the intensional database. The second approach [2][3][5] uses a restricted language to describe the intensional database and the integrity constraints. This permits the use of a classification mechanism based on a decidable subsumption relation. This decidable relation helps to discover equivalences and simplifications that cannot be computed in the first framework, where this subsumption relation is undecidable in general [18].

* In Proc. of DOOD'95 (Deductive and Object-Oriented Databases), LNCS 1013.

To our knowledge, no formal framework has been proposed to combine these two promising approaches. This paper is a contribution in this direction. We present a reformulation scheme that uses a resolution method and a classification-based rewriting in complementary ways. We propose three types of reformulation:

Resolution: we reduce some atoms referring to the intensional database to atoms referring to the extensional database, and we remove parts of the query that cannot have any answers with respect to the integrity constraints.
Factorization and classification [14]: we simplify the query by eliminating some redundant selection conditions. In addition, we reformulate the query so that it takes advantage of some materialized views.
Propagation: we make explicit a part of the knowledge which is implicitly available in the query. In this way, we have more information about objects to be retrieved. This allows the search space of objects which will be considered during the query evaluation process to be reduced.

These reformulations can be applied in stages or iteratively. They can also be integrated within a query evaluation process. These aspects are not developed in this paper.

We use a specific language which will be described in Section 4. However, the reformulation scheme can be applied to other languages for deductive object-oriented databases.

The kind of databases we consider is outlined in the next Section. In Section 3 we give an example of a semantic query optimization in our framework. Section 4 formally defines the language used in this paper. The different query reformulations are detailed in Section 5. Then, we conclude in Section 6.

2 Context

In this section, by means of an example, we delimit the kind of database we consider. We also give an informal introduction to the knowledge representation language we use.

A deductive object-oriented database is split into three parts: $\mathcal{SCH}$, $\mathcal{EDB}$ and $\mathcal{IDB}$.
$\mathcal{SCH}$ is the database schema. It describes the classes used in the extensional database, the integrity constraints, and the materialized views. It will be seen as a set of constraints to be satisfied.
$\mathcal{EDB}$ corresponds to the extensional database.
$\mathcal{IDB}$ is a set of clauses that defines the intensional part of the database.

Basically, three forms of atoms are used in the descriptions: $X < Y$ (X inherits from Y), $X : Y$ (X is an instance of Y), $X.R \rightarrow Y$ (for X, one of the values of the attribute R is Y).

A clause has the form :

$\mathsf{A} :- \mathsf{A}_{idb_1}$ & ... & A_{idb_n} // A_{edb_1} & ... & A_{edb_m}.

In the body of the clause, we make a syntactic difference between a conjunction of atoms A_{idb_1} & ... & A_{idb_n}, that refers to the intensional database, and a conjunction of atoms A_{edb_1} & ... & A_{edb_m}, that refers to the extensional database.

We shall now comment on the following example, which will be used throughout this paper. It describes the possible structure of a laboratory in a university.

$\mathcal{SCH}$:

(s1) person < anything

(s2) permanent_staff_member < person

(s3) teacher < person

(s4) student < and(person, all(works_in_project, project), mono(works_in_project))

(s5) professor < and(permanent_staff_member, teacher)

(s6) lecturer < and(permanent_staff_member, teacher, not(professor))

(s7) project < and(all(managed_by, permanent_staff_member), exist(managed_by, permanent_staff_member))

(s8) theme < and(all(managed_by, professor), exist(managed_by, permanent_staff_member), mono(managed_by))

(s9) assistant := and(student, teacher)

$\mathcal{IDB}$:

(r1) E:satisfied :- Z:important // Z.managed_by → E & Z:theme.

(r2) E:satisfied :- X.supervised_by → E // X:teacher.

(r3) very_satisfied < satisfied.

(r4) X.supervised_by → E :- E:entitled // X.works_in_project → Z & Z.managed_by → E & X:student.

(r5) E:very_satisfied :- ...

(r6) E:entitled :- ...

(r7) Z:important :- ...

$\mathcal{EDB}$:

(e1) deductive_databases:theme

(e2) deductive_databases.managed_by → sam

(e3) sam:professor

The schema $\mathcal{SCH}$ asserts that person is a subclass of the universal class anything *(s1)* and that permanent_staff_member, teacher and student are subclasses of person *(s2)(s3)(s4)*. *(s4)* also tells us that for a student, the attribute works_in_project is of type project (constructor all) and it has at most one value (constructor mono).

A professor is a permanent_staff_member and a teacher *(s5)*. A lecturer is also a permanent_staff_ member and a teacher, but not a professor *(s6)*. A research project is managed by at least one (constructor exist) permanent_staff_member and only by permanent_staff_members *(s7)*. A theme is managed by one and only one permanent_staff_member, who must be a professor *(s8)*.

The description *(s9)* defines the materialized view assistant as the intersection of the classes student and teacher.

In the intensional database, the instances of the class satisfied are all the managers of the important themes *(r1)* and all the supervisors of the teachers *(r2)*. The Rule *(r4)* defines the derived attribute supervised_by. It states that a student's supervisor is an entitled manager of a project in which the student is involved. *(r3)* is a clause with an empty body that describes an inheritance link between two classes (very_satisfied and satisfied) of the intensional database.

The derivation rules of classes very_satisfied, entitled and important *(r5) (r6) (r7)* are left blank since they will not be used in this paper.

Finally, the extensional database tells us that deductive_databases is a research theme *(e1)* which is managed by sam *(e2)* who is a professor *(e3)*. Note that $\mathcal{EDB}$ will be in fact of no use during semantic query optimization.

3 Reformulation Principle

In our approach, the main query reformulations are performed using a resolution and a classification mechanism. Before we develop the technical details, it is important that we give the reader an informal description of the query reformulations.

A query has the same form as a clause body. Let Q be the following query:

E:satisfied // E:lecturer

which retrieves all satisfied lecturers. Figure 1 shows some possible reformulations of Q.

The resolution uses the clauses of $\mathcal{IDB}$ and the schema $\mathcal{SCH}$ to rewrite Q into a disjunction of queries (Q_1 *or* ... *or* Q_n), which is semantically equivalent[2] to Q.

By reformulating Q with clause *(r1)* we should obtain the query Q_1. But when we look at the part of Q_1 that refers to the extensional database:

E:lecturer & Z:theme & Z.managed_by $\rightarrow$ E

[2] Informally, two queries are semantically equivalent (with respect to $\mathcal{IDB}$ and $\mathcal{SCH}$) if they have the same answers for all databases constructed from $\mathcal{IDB}$ and $\mathcal{SCH}$ [7]

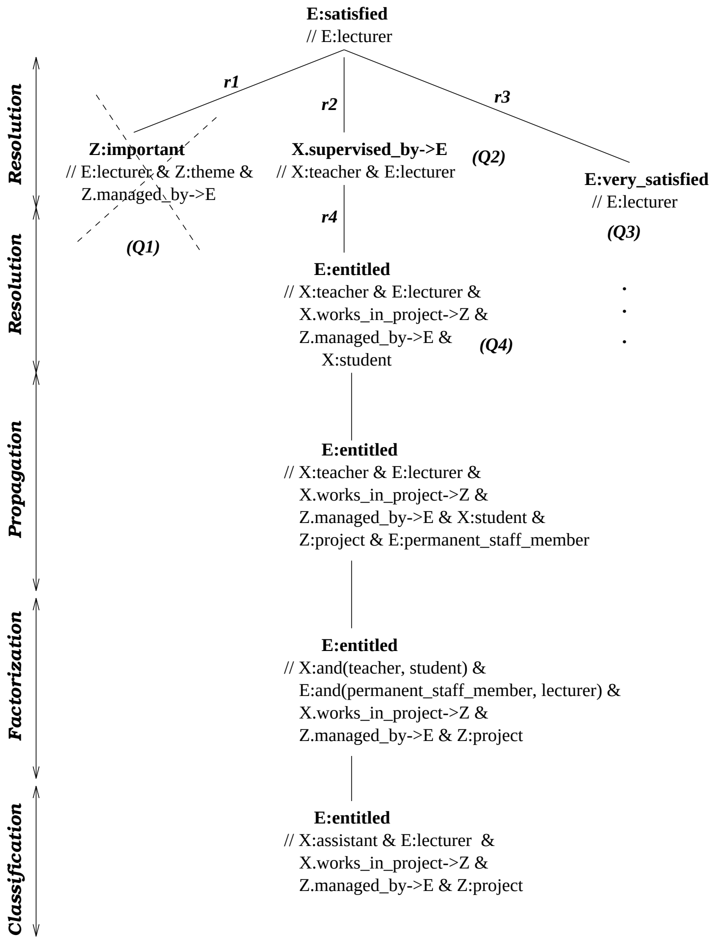

Fig. 1. Reformulation of a query.

and since the schema imposes that all themes are managed by professors *(s8)* and that a professor is not a lecturer *(s6)*, Q_1 will always have an empty set of answers. A satisfiability test (with respect to the schema) applied to this part of the query during the resolution step allows us to detect such contradictions, and then to eliminate Q_1.

The reformulation of the query Q with the clause *(r2)* generates the query Q_2. From Q we obtain Q_3 using clause *(r3)* and a specific resolution rule that handles inheritance links.

No other direct resolution steps can be performed to reformulate Q with a clause of $\mathcal{IDB}$. So Q is equivalent to the disjunction (Q_2 *or* Q_3).

Another resolution-based reformulation step can be performed. For example, the reformulation of the query Q_2 with the clause *(r4)* produces Q_4:

$$\begin{aligned}\mathsf{E{:}entitled}\ \ /\!/\ \mathsf{X{:}teacher}\ \ \&\ \mathsf{X{:}student}\ \ \&\ \mathsf{E{:}lecturer}\ \ \& \\ \mathsf{X.works\_in\_project} \rightarrow \mathsf{Z}\ \ \&\ \mathsf{Z.managed\_by} \rightarrow \mathsf{E}\end{aligned}$$

The propagation-based reformulation is used on Q_4 to add selection information to the part that refers to the $\mathcal{E}DB$, and, thus, helps to reduce the search space of objects. For example, when looking at the schema, we see that a student is involved in a project *(s4)*, which is managed by permanent_staff_members *(s7)*. Thus, using the atoms X.works_in_project $\rightarrow$ Z and Z.managed_by $\rightarrow$ E in the query, we can add Z:project and E:permanent_staff_member.

Finally, the example illustrates the classification-based reformulation. This reformulation is performed in two steps: factorization and classification-based rewriting. The key idea behind the factorization is to put together, for each variable, instantiation links in which the variable appears. For example, from X:teacher and X:student, we obtain X:and(teacher, student).

The classification consists in finding the simplest and most specific description of the objects to be retrieved. The classification of the description and(permanent_staff_member, lecturer) allows us to deduce that E:and(permanent_staff_member, lecturer) can be simplified by rewriting it as E:lecturer. The classification of and(teacher, student) leads to a replacement of X:and(teacher, student) by X:assistant, and, thus, takes advantage of the materialized view assistant.

The ordering of the reformulations used in this example is not determined by our reformulation scheme. This ordering is out of the scope of this paper.

4 Language

We use a knowledge representation language designed to allow the combination of two complementary kinds of reasoning: abduction and deduction of objects structural properties [16]. The attentive reader will notice that the syntax and the semantics are slightly richer than those used in the examples of this paper. This is a language of clauses with constraints in the sense of constraint logic programming [12]. Its semantics is drawn from F-logic [13], terminological logics [15] and the generalized framework for constraint logic programming proposed by Höhfeld and Smolka [11].

First, we present its syntax, its declarative semantics, and a resolution method. Then, we use the language to describe databases and queries. Informally, in a logical framework for deductive databases, one can see a model as a set $\mathcal{M}$ of true atoms. In this paper, we see such a model $\mathcal{M}$ as two sets $\mathcal{E}$ and $\mathcal{B}$, with $\mathcal{E} \subseteq \mathcal{M} \subseteq \mathcal{B}$.

$\mathcal{B}$ is a set of possibly true atoms with respect to the database $\mathcal{S}CH + \mathcal{E}DB + \mathcal{I}DB$ (hence $\mathcal{M} \subseteq \mathcal{B}$) and $\mathcal{E}$ is a set of necessarily true atoms with respect to the intensional database $\mathcal{I}DB$ (hence $\mathcal{E} \subseteq \mathcal{M}$).

This will be used in two ways, when reasoning to reformulate a query Q. Firstly, since we do not want to access the $\mathcal{E}DB$, we will use $\mathcal{B}$ as an hypothetical denotation of the database. And secondly, since we allow to access the $\mathcal{I}DB$, we will used $\mathcal{E}$, as a necessary denotation of the $\mathcal{I}DB$ such that the query Q will have answers.

4.1 Syntax

Let $\mathcal{K}$ be a decidable set of symbols of constants and $\mathcal{V}$ be a decidable, infinite set of symbols of variables. *Terms* are of two kinds: *object names* (names of classes, of instances or of attributes) and *concepts*.

Terms are obtained according to the following abstract syntax:

$$\begin{aligned} X &\xrightarrow{def} v \mid k \\ C &\xrightarrow{def} X \mid \mathit{anything} \mid \mathit{nothing} \mid \mathit{and}(C_1, \ldots, C_n) \mid \\ &\qquad \mathit{all}(R, C) \mid \mathit{mono}(R) \mid \mathit{exist}(R, C) \mid \mathit{not}(X) \end{aligned}$$

where v denotes a variable, k denotes a constant, X and R denote object names, whereas C and C_i (for $i = 1, \ldots, n$) denote concepts.

Atoms are also of two kinds: *atomic links* and *atomic constraints*. The syntax for forming atoms is the following (X, Y and R denote object names, C denotes a concept, Al is an atomic link and Ac is an atomic constraint):

$$\begin{aligned} Al &\xrightarrow{def} X < Y \mid X : Y \mid X.R \rightarrow Y \\ Ac &\xrightarrow{def} X < C \mid X : C \mid X.R \rightarrow Y \mid X := C \mid X = Y \end{aligned}$$

Sometimes, we will put atoms in brackets in order to facilitate reading.

Goals and clauses are built according to the following syntax:

$$\begin{aligned} F &\xrightarrow{def} \epsilon \mid Al \mid Al_1 \ \& \ \ldots \ \& \ Al_n \\ H &\xrightarrow{def} \epsilon \mid Ac \mid Ac_1 \ \& \ \ldots \ \& \ Ac_m \\ \mathit{Goal} &\xrightarrow{def} F \ /\!/ \ H \\ \mathit{Clause} &\xrightarrow{def} Al :\!- F \ /\!/ \ H \end{aligned}$$

where Al and Al_i (for $i = 1, \ldots, n$) denote atomic links, Ac and Ac_i (for $i = 1, \ldots, m$) denote atomic constraints, F denotes a conjunction of atomic links, H denotes a conjunction of atomic constraints and ϵ the empty conjunction of constraints.

We use the conventional designations of clause *body* and *head*. A fact is a clause whose body is $\epsilon \ /\!\!/ \ \epsilon$. It is abbreviated by writing only the head of the clause. Finally, variable symbols start with an upper case letter and constant symbols start with a lower case letter.

A *language* $\mathcal{L}$ is the set of goals and clauses constructed from the two sets of symbols $\mathcal{K}$ and $\mathcal{V}$.

4.2 Declarative Semantics

Let $\mathcal{L}$ be a language constructed from the set $\mathcal{V}$ of variables and the set $\mathcal{K}$ of constants.

We use two levels of interpretation: *underlying interpretations* for conjunctions of atoms and *interpretations* for goals and clauses.

An underlying interpretation $\mathcal{J}$ of $\mathcal{L}$ is of the form:

$$\mathcal{J} = \langle D, id, member, inherit, attribute\rangle, \text{ with}$$

- D a non-empty set called the domain of $\mathcal{J}$.
- id a total injection $id : \mathcal{K} \to D$. It assigns an element of the domain to each constant (name of a class, of an instance or of an attribute).
- $member \subseteq D \times D$, a relation linking instances to their classes.
- $inherit \subseteq D \times D$, a strict ordering relation linking classes to their superclasses.
- $attribute \subseteq D \times D \times D$, a relation where every occurrence links an instance, an attribute name and a value for this attribute.
- $member$ and $inherit$ satisfying:
  $\forall d_1, d_2, d_3 \in D, \langle d_1, d_2\rangle \in member \wedge \langle d_2, d_3\rangle \in inherit \Rightarrow \langle d_1, d_3\rangle \in member$
  (i.e., instances of a class are also instances of its superclasses.)

Let $\mathcal{J}$ be an underlying interpretation of $\mathcal{L}$. A *$\mathcal{J}$-assignment* α is a total function $\alpha : \mathcal{V} \to D$, which assigns elements of the domain of $\mathcal{J}$ to the variables of the language. We write $ASS^{\mathcal{J}}$ for the set of all $\mathcal{J}$-assignments. All the $\mathcal{J}$-assignments $\alpha \in ASS^{\mathcal{J}}$ are extended to $\mathcal{V} \cup \mathcal{K}$ by id: $\forall k \in \mathcal{K}, \alpha(k) := id(k)$.

For an underlying interpretation $\mathcal{J} = \langle D, id, member, inherit, attribute\rangle$ and an assignment α, we define a function $ext_{\mathcal{J}\alpha}$ which assigns a set of elements of the domain to every concept. This function is defined by the following equations:

$$\begin{aligned}
ext_{\mathcal{J}\alpha}(X) &:= \{d \in D \mid \langle d, \alpha(X)\rangle \in member\} \\
ext_{\mathcal{J}\alpha}(anything) &:= D \\
ext_{\mathcal{J}\alpha}(nothing) &:= \emptyset \\
ext_{\mathcal{J}\alpha}(and(C_1, \ldots, C_n)) &:= ext_{\mathcal{J}\alpha}(C_1) \cap \ldots \cap ext_{\mathcal{J}\alpha}(C_n) \\
ext_{\mathcal{J}\alpha}(all(R, C)) &:= \{d_1 \in D \mid \forall d_2 \in D, \langle d_1, \alpha(R), d_2\rangle \in attribute \Rightarrow \\
&\qquad d_2 \in ext_{\mathcal{J}\alpha}(C)\}
\end{aligned}$$

$$
\begin{aligned}
ext_{\mathcal{J}\alpha}(mono(R)) &:= \{d_1 \in D \mid \forall d_2, d_3 \in D, \langle d_1, \alpha(R), d_2\rangle \in attribute \wedge \\
&\qquad\qquad\qquad\qquad \langle d_1, \alpha(R), d_3\rangle \in attribute \Rightarrow d_2 =_D d_3\} \\
ext_{\mathcal{J}\alpha}(exist(R, C)) &:= \{d_1 \in D \mid \exists d_2 \in D, \langle d_1, \alpha(R), d_2\rangle \in attribute \wedge \\
&\qquad\qquad\qquad\qquad d_2 \in ext_{\mathcal{J}\alpha}(C)\} \\
ext_{\mathcal{J}\alpha}(not(X)) &:= \{d \in D \mid d \notin ext_{\mathcal{J}\alpha}(X)\}
\end{aligned}
$$

where X and R are object names, C and C_i (for $i = 1, \ldots, n$) are concepts. $ext_{\mathcal{J}\alpha}$ corresponds to the interpretation of concepts in terminological logics [15].

The function $ext_{\mathcal{J}\alpha}$ allows us to define the notion of *$\mathcal{J}$-solutions* [11] of an atom and of a conjunction of atoms. The set of $\mathcal{J}$-solutions of an atom A will be written $[A]^{\mathcal{J}}$.

For a given underlying interpretation $\mathcal{J}$, the sets of $\mathcal{J}$-solutions of the different forms of atoms are:

$$
\begin{aligned}
[X : C]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid \alpha(X) \in ext_{\mathcal{J}\alpha}(C)\} \\
[X.R \to Y]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid \langle \alpha(X), \alpha(R), \alpha(Y)\rangle \in attribute\} \\
[X < C]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid \langle \alpha(X), \alpha(C)\rangle \in inherit\} \text{ if } C \text{ is an object name} \\
[X < C]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid ext_{\mathcal{J}\alpha}(X) \subseteq ext_{\mathcal{J}\alpha}(C)\} \text{ if } C \text{ is not an object name} \\
[X := C]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid ext_{\mathcal{J}\alpha}(X) = ext_{\mathcal{J}\alpha}(C)\} \\
[X = Y]^{\mathcal{J}} &:= \{\alpha \in ASS^{\mathcal{J}} \mid \alpha(X) =_D \alpha(Y)\} \\
[\epsilon]^{\mathcal{J}} &:= ASS^{\mathcal{J}}
\end{aligned}
$$

The notion of set of $\mathcal{J}$-solutions is extended to conjunctions of atoms:

$$[A_1 \ \& \ \ldots \ \& \ A_n]^{\mathcal{J}} = [A_1]^{\mathcal{J}} \cap \ldots \cap [A_n]^{\mathcal{J}}.$$

The underlying interpretations of a given language $\mathcal{L}$ are partially ordered by $\leq$ as follows:
let $\mathcal{J}_1 = \langle D_1, id_1, member_1, inherit_1, attribute_1\rangle$ and
$\mathcal{J}_2 = \langle D_2, id_2, member_2, inherit_2, attribute_2\rangle$, then $\mathcal{J}_1 \leq \mathcal{J}_2$ iff $D_1 = D_2, id_1 = id_2, member_1 \subseteq member_2, inherit_1 \subseteq inherit_2$ and $attribute_1 \subseteq attribute_2$.

An *interpretation* $\mathcal{I}$ of $\mathcal{L}$ is of the form $\mathcal{I} = \langle \mathcal{E}, \mathcal{B}\rangle$ where $\mathcal{E}$ and $\mathcal{B}$ are underlying interpretations of $\mathcal{L}$ such that: $\mathcal{E} \leq \mathcal{B}$. The underlying interpretation $\mathcal{B}$ is called the base of $\mathcal{I}$. The set of *$\mathcal{I}$-assignments* is $ASS^{\mathcal{I}} = ASS^{\mathcal{B}} = ASS^{\mathcal{E}}$.

We define a partial ordering relation on interpretations: let $\mathcal{I}_1 = \langle \mathcal{E}_1, \mathcal{B}_1\rangle$ and $\mathcal{I}_2 = \langle \mathcal{E}_2, \mathcal{B}_2\rangle$ be two interpretations, then $\mathcal{I}_1 \leq \mathcal{I}_2$ iff $\mathcal{E}_1 \leq \mathcal{E}_2$ and $\mathcal{B}_1 = \mathcal{B}_2$.

We now extend the notion of solution to goals and clauses. Let $\mathcal{I} = \langle \mathcal{E}, \mathcal{B}\rangle$ be an interpretation, we define the following sets of *$\mathcal{I}$-solutions*:

$$
\begin{aligned}
[F \ /\!/ \ H]^{\mathcal{I}} &:= [F]^{\mathcal{E}} \cap [H]^{\mathcal{B}} \\
[A \text{ :- } F \ /\!/ \ H]^{\mathcal{I}} &:= (ASS^{\mathcal{I}} - [F \ /\!/ \ H \ \& \ A]^{\mathcal{I}}) \cup [A]^{\mathcal{E}}
\end{aligned}
$$

A more classical point of view for the second definition would have been:

$$[A :- F \mathbin{/\!/} H]^{\mathcal{I}} := (ASS^{\mathcal{I}} - [F \mathbin{/\!/} H]^{\mathcal{I}}) \cup [A]^{\mathcal{E}}$$

which requires that if $\alpha \in [A :- F \mathbin{/\!/} H]^{\mathcal{I}}$ is a solution of $F \mathbin{/\!/} H$, then α must be a solution of A.

In our definition, the requirement is weaker: for $\alpha \in [A :- F \mathbin{/\!/} H]^{\mathcal{I}}$, α must be a solution of A only if α is a solution of $F \mathbin{/\!/} H \;\&\; A$ (i.e., the body is satisfied and the head seen as a constraint is also satisfied).

A conjunction of atomic constraints H is *satisfiable* iff there exists at least one underlying interpretation $\mathcal{J}$ such that $[H]^{\mathcal{J}} \neq \emptyset$.

An interpretation $\mathcal{I}$ is a *model* of a set of clauses S iff $\forall C \in S, [C]^{\mathcal{I}} = ASS^{\mathcal{I}}$.

The notion of logical consequence can be defined as follows:
let S be a set of clauses, H be a conjunction of atomic constraints and Q be a goal,

Q is a *logical consequence of* S *under hypothesis* H (written $S \models H \Rightarrow Q$) iff for every model $\mathcal{I} = \langle \mathcal{E}, \mathcal{B} \rangle$ of S, we have $\forall \alpha \in ASS^{\mathcal{I}}, \alpha \in [H]^{\mathcal{B}} \Rightarrow \alpha \in [Q]^{\mathcal{I}}$.

We can now define the notion of answer. Let S be a set of clauses, then an *answer* for a goal Q is a conjunction of atomic constraints H, such that H is satisfiable and $S \models H \Rightarrow Q$.

Informally, an answer is often seen as a substitution on the variables of the goal. Here, an answer can be seen as a formula that constrains these variables. For example, when a classical answer is the substitution $\{X/a, Y/Z\}$, we have an answer of the form $X = a \;\&\; Y = Z$. In this aspect, our definitions of answer and of logical consequence are in the spirit of [6] and are also related[3] to constraint logic programming [12].

4.3 Resolution Method

We now formalize our resolution method by defining a relation of goal reduction [11].

Preliminary Notions

Let F be a conjunction of atoms, a goal or a clause. We write $var(F)$ the set of variables in F.

Two atoms A and A' are said to be *compatible* iff :

[3] A classical constraint logic programming scheme uses a fixed interpretation for the atomic constraints, and, thus, will not be appropriate here.

(1) $A = (X : Y)$ and $A' = (X' : Y')$. *In this case we write* $(A \overset{arg}{=} A')$ *for the conjunction of constraints:* $(X = X')$ & $(Y = Y')$.

or (2) $A = (X < Y)$ and $A' = (X' < Y')$. *We also write in this case* $(A \overset{arg}{=} A')$ *for the conjunction of constraints:* $(X = X')$ & $(Y = Y')$.

or (3) $A = (X.R \to Y)$ and $A' = (X'.R' \to Y')$. *In this last case we write* $(A \overset{arg}{=} A')$ *for the conjunction of constraints:* $(X = X')$ & $(R = R')$ & $(Y = Y')$.

A clause C obtained by a renaming of the variables of a clause C' is called a *variant* of C'.

Resolution

For a given language $\mathcal{L}$, let S be a set of clauses, and V be a finite set of variables.

The (S, V)-*reduction* is a binary relation $\xrightarrow{r}_{S,V}$ on the set of goals. It is defined by the following rules, of the form $Q \xrightarrow{r}_{S,V} Q'$ (Where C denotes a variant of a clause of S such that $(V \cup var(Q)) \cap var(C) = \emptyset$):

Direct Reduction

$A \mathbin{\&} G \mathbin{/\!/} H_1 \xrightarrow{r}_{S,V} F \mathbin{\&} G \mathbin{/\!/} H_3$
with $C = (A' \mathrel{:-} F \mathbin{/\!/} H_2)$ and
$H_3 = (H_1 \mathbin{\&} H_2 \mathbin{\&} A' \mathbin{\&} (A \overset{arg}{=} A'))$,
if A and A' are compatible
and H_3 is satisfiable.

Reduction-by-inclusion

$X : Y \mathbin{\&} G \mathbin{/\!/} H_1 \xrightarrow{r}_{S,V} X : Z \mathbin{\&} F \mathbin{\&} G \mathbin{/\!/} H_3$
with $C = (Z < Y' \mathrel{:-} F \mathbin{/\!/} H_2)$ and
$H_3 = (H_1 \mathbin{\&} H_2 \mathbin{\&} Z < Y' \mathbin{\&} Y = Y')$,
if H_3 is satisfiable.

Reduction-by-inheritance

$X < Y \mathbin{\&} G \mathbin{/\!/} H_1 \xrightarrow{r}_{S,V} X < Z \mathbin{\&} F \mathbin{\&} G \mathbin{/\!/} H_3$
with $C = (Z < Y' \mathrel{:-} F \mathbin{/\!/} H_2)$ and
$H_3 = (H_1 \mathbin{\&} H_2 \mathbin{\&} Z < Y' \mathbin{\&} Y = Y')$,
if H_3 is satisfiable.

In the reduction rules, the head A' of clause C is pushed in the conjunction of constraints H_3. This is because A' must be satisfiable w.r.t. the other constraints.

The reader interested in satisfiability tests for conjunctions of atomic constraints may refer to the works done in the field of terminological knowledge bases (e.g., [4]) and to [17] for an application to object-oriented databases.

Properties

The proofs of the two following theorems can be found in [16]. The completeness has been shown using a technique due to Höhfeld and Smolka [11].

Let $\xrightarrow{r}^*_{S,V}$ be the reflexive and transitive closure of $\xrightarrow{r}_{S,V}$.

Theorem 4.1 (Soundness) *If $Q \xrightarrow{r}^*_{S,V} Q'$, then for every model $\mathcal{I}$ of S, $[Q']^{\mathcal{I}} \subseteq [Q]^{\mathcal{I}}$.*

Let $\mathcal{I}$ be an interpretation, α be an assignment and V be a finite set of variables. Then $\alpha|_V$ is the restriction of α to $V \cup \mathcal{K}$.

Let Q be a goal, then $[Q]^{\mathcal{I}}_V := \{\alpha|_V \mid \alpha \in [Q]^{\mathcal{I}}\}$.

Theorem 4.2 (Completeness) *Let $\mathcal{I}$ be a minimal model of S (wrt $\leq$), Q be a goal and $\alpha \in [Q]^{\mathcal{I}}$; then there exists a conjunction of atomic constraints H such that $Q \xrightarrow{r}^*_{S,V} \epsilon \mathbin{/\!/} H$ and $\alpha|_V \in [\epsilon \mathbin{/\!/} H]^{\mathcal{I}}_V$.*

4.4 Database and Query

In our framework, a *database* is a triple $\langle \mathcal{S}CH, \mathcal{E}DB, \mathcal{I}DB \rangle$ where $\mathcal{S}CH$ (the schema) is a conjunction of variable-free atomic constraints, $\mathcal{E}DB$ (the extensional database) is a conjunction of variable-free atomic links and $\mathcal{I}DB$ (the intensional database) is a set of clauses. A *query* is a goal.

An interpretation $\mathcal{I} = \langle \mathcal{E}, \mathcal{B} \rangle$ is a *model of a database* iff $[\mathcal{E}DB]^{\mathcal{B}} = [\mathcal{S}CH]^{\mathcal{B}} = ASS^{\mathcal{B}}$ and $\mathcal{I}$ is a model of $\mathcal{I}DB$.

Informally, the conditions $[\mathcal{E}DB]^{\mathcal{B}} = ASS^{\mathcal{B}}$ and $[\mathcal{S}CH]^{\mathcal{B}} = ASS^{\mathcal{B}}$ ensure that both $\mathcal{E}DB$ and the schema are true in the model.

Let $\mathcal{D}B$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and H be a conjunction of atomic constraints.

Q is a *logical consequence of $\mathcal{D}B$ under hypothesis H on V* (written $\mathcal{D}B \models H \Rightarrow_V Q$) iff for every model $\mathcal{I} = \langle \mathcal{E}, \mathcal{B} \rangle$ of $\mathcal{D}B$, $\forall \alpha \in ASS^{\mathcal{I}}, \alpha \in [H]^{\mathcal{B}} \Rightarrow \alpha|_V \in [Q]^{\mathcal{I}}_V$.

This extends to set of queries in the following way:

Let $\mathcal{R}$ be a set of queries. $\mathcal{R}$ is a *logical consequence of* $\mathcal{DB}$ *under hypothesis* H *on* V (written $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$) iff for every model $\mathcal{I} = \langle \mathcal{E}, \mathcal{B} \rangle$ of $\mathcal{DB}$, $\forall \alpha \in ASS^{\mathcal{I}}, \alpha \in [H]^{\mathcal{B}} \Rightarrow (\exists Q' \in \mathcal{R}$ such that $\alpha|_V \in [Q']^{\mathcal{I}}_V)$.

Informally, this interprets a set of queries as a disjunction of queries.

Finally, H is a V-*answer* to Q (resp. $\mathcal{R}$) iff H is satisfiable and $\mathcal{DB} \models H \Rightarrow_V Q$ (resp. $\mathcal{DB} \models H \Rightarrow_V \mathcal{R}$).

This definition is more general than the classical notion of answer. Here an answer is a hypothesis H such that if H is satisfied then the query is also satisfied.

5 Reformulation

This section presents the various reformulations outlined in Section 3. Proofs of the theorems can be found in [10].

5.1 Resolution-Based Reformulation

In this kind of reformulation, we perform resolution steps involving the query and the $\mathcal{IDB}$, by application of the reduction rules. Then, we obtain a new query in which the disjunctive parts that cannot have any answers with respect to the integrity constraints of the schema have been removed.

Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, Q be a query and V be a finite set of variables. Then, a set $\mathcal{R}$ of queries obtained by (V, r)-*reformulation* of Q is a set that satisfies the following four conditions:

1. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the direct reduction rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.
2. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the reduction-by-inclusion rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.
3. $\forall C \in \mathcal{IDB}$,
   let $\mathcal{R}_C := \{Q' \mid Q \xrightarrow{r}_{\mathcal{IDB},V} Q'$ using a variant of C and the reduction-by-inheritance rule$\}$,
   if $\mathcal{R}_C \neq \emptyset$, then $\mathcal{R}_C \cap \mathcal{R} \neq \emptyset$.
4. $\mathcal{R}$ is minimal with respect to set inclusion.

Informally, $\mathcal{R}$ is the set of all queries that can be obtained from Q by a reduction step, with the following restriction: when a reduction can be performed with different variants of a clause, only one of the variants is used. It should be noted that for a given query Q, there exists several reformulated queries, depending on the different variants one can use in a reduction step.

In order to discard the queries that cannot have any answer with respect to the integrity constraints, we incorporate the schema into the reformulation process. For a given query $Q = F \,/\!/\, H$, this is achieved if we perform the (V, r)-reformulation of $F \,/\!/\, H \;\&\; \mathcal{S}CH$. Informally, this leads to allow only reductions by which $\mathcal{S}CH$ remains satisfiable.

Theorem 5.1 (Soundness of Reformulation by Resolution)
Let $\mathcal{D}B = \langle \mathcal{S}CH, \mathcal{E}DB, \mathcal{I}DB \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \;\&\; \mathcal{S}CH)$. If H is a V-answer to $\mathcal{R}$ wrt $\mathcal{D}B$, then H is a V-answer to Q wrt $\mathcal{D}B$.

Theorem 5.2 (Completeness of Reformulation by Resolution)
Let $\mathcal{D}B = \langle \mathcal{S}CH, \mathcal{E}DB, \mathcal{I}DB \rangle$ be a database, Q be a query, V be a finite set of variables (e.g., the variables of interest in Q) and $\mathcal{R}$ obtained by (V, r)-reformulation of $(Q \;\&\; \mathcal{S}CH)$. For every V-answer H to Q wrt $\mathcal{D}B$, H is also a V-answer to $\mathcal{R}$ wrt $\mathcal{D}B$.

The equality constraints generated by the reduction rules, like in constraint logic programming language (e.g., Prolog III [9]), play a role similar to unification in classical resolutions. In order to facilitate reading, in what follows, the queries have been simplified with respect to these constraints.

Moreover, the presence of $\mathcal{S}CH$ and the form of the reduction rules lead to obtain other additional atomic constraints that will not be given. However, these atomic constraints do not represent accesses to the extensional part of the database and can be easily filtered out.

Consider the example of Section 3. Let $V = \{E\}$ be the set of variables of interest in Q. We use $(\mathcal{I}DB, V)$-reduction to reformulate this query.

A direct reduction of $(Q \;\&\; \mathcal{S}CH)$ using a variant of *(r2)* gives Q_2. A reduction-by-inclusion of $(Q \;\&\; \mathcal{S}CH)$ using a variant of *(r3)* gives Q_3. The other reduction rules do not apply to Q using a variant of *(r2)* or *(r3)*.

A reduction of Q using a variant of *(r1)* is not possible, since it would lead to a set of atomic constraints which is not satisfiable.

Since no other reduction is possible using other clauses of $\mathcal{I}DB$, $\mathcal{R} = \{Q_2, Q_3\}$ is obtained by (V, r)-reformulation of Q. By soundness and completeness of this reformulation, $\mathcal{R}$ is semantically equivalent to Q.

5.2 Propagation-Based Reformulation

Let $Q = F \,/\!/\, H$ be a query. In order to simplify the notation, we view $H = Ac_1 \;\&\; \ldots \;\&\; Ac_n$ as a set of atomic constraints, that is $H = \{Ac_1, \ldots, Ac_n\}$.

To make the presentation easier, we will also work with an expanded version of the schema, obtained by replacing all atomic constraints of the form $C < and(D_1, \ldots, D_m)$ by the conjunction of the atomic constraints $C < D_1, \ldots, C < D_m$

The purpose of this reformulation is to complete H by introducing additional selection information about objects to be retrieved. To achieve this, we add constraints to H by means of so-called propagation rules. Each application of a rule will lead to a new set of constraints.

$H[X/Y]$ denotes the set of constraints obtained from H by substituting each occurrence of the variable X by Y.

Let X, R and Y be object names, C and D be concepts, V_1 and V_2 be two distinct variables and k be a constant. The propagation rules are:

- Variable simplification:
  (p1) $H \longrightarrow (H - \{V_1 = V_2\})[V_1/V_2] \cup \{V_1 = V_2\}$ if $(V_1 = V_2) \in H$.
  (p2) $H \longrightarrow (H - \{k = V_1\})[V_1/k] \cup \{V_1 = k\}$ if $(k = V_1) \in H$.
  (p3) $H \longrightarrow (H - \{V_1 = k\})[V_1/k] \cup \{V_1 = k\}$ if $(V_1 = k) \in H$.
- Propagation of class properties at the instance level:
  (p4) $H \longrightarrow \{X : D\} \cup H$
  if $(X : C) \in H$ and the schema contains $C < D$.
- Propagation of the attribute properties:
  (p5) $H \longrightarrow \{Y : C\} \cup H$
  if $(X.R \rightarrow Y) \in H$ and $(X : all(R, C)) \in H$.

These propagation rules are based on those employed in satisfiability tests for terminological knowledge bases [4]. This set of propagation rules is not exhaustive. Since this paper does not focus on propagation-based reformulation, only a restricted set of rules is given.

Theorem 5.3 (Correctness of Propagation-Based Reformulation) *Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \mathbin{/\!/} H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a finite sequence of applications of the propagation rules; then for every model $\mathcal{I}$ of $\mathcal{DB}$, $[F \mathbin{/\!/} H_1]^{\mathcal{I}} = [F \mathbin{/\!/} H_2]^{\mathcal{I}}$.*

Consider the example of Section 3. The reformulation by propagation is applied to the part of Q_4 which refers to the extensional database:

X:teacher & E:lecturer & X.works_in_project → Z &
Z.managed_by → E & X:student

The class student inherits from the concept all(works_in_project, project) *(s4)* and the query contains the atomic constraint X:student. Thus, by the application of the rule *(p4)* we obtain the new atomic constraint X:all(works_in_project, project).

Rule *(p5)* applies to X.works_in_project → Z and X:all(works_in_project, project). It generates Z:project.

The rule *(p4)* uses all(managed_by,permanent_staff_member) inherited by the class project *(s7)* and Z:project to add Z:all(managed_by,permanent_staff_member).

Finally, rule *(p5)* can be used to obtain E:permanent_staff_member from Z.managed_by $\rightarrow$ E and Z:all(managed_by,permanent_staff_member).

The rules *(p1) (p2) (p3)* are required to handle the equality constraints generated by resolution. As this kind of constraint has already been simplified (to facilitate reading), these rules are not used in the example. However, their application is straightforward.

5.3 Factorization and Classification-Based Reformulation

Let $Q = F \not\!\!/\!\!/ H$ be a query. Like in the Section 5.2, in order to simplify the notation, we view $H = Ac_1 \ \& \ \ldots \ \& \ Ac_n$ as a set of atomic constraints, that is $H = \{Ac_1, \ldots, Ac_n\}$.

H may contain redundant parts, and conjunctions of atomic constraints whose evaluation can be replaced by accesses to materialized views. So, it is useful to retain only the most specific constraints, and to detect parts of the query corresponding to the evaluation of materialized views. To do this, we rewrite H by means of a technique based on concept classification.

First we put together constraints related to the same object. This factorization process is performed by means of the following factorization rule (used iteratively until it can no longer be applied):

$$\begin{array}{l} H \longrightarrow \{X : and(C1, C2)\} \cup (H - \{X : C1, X : C2\}) \\ \qquad \text{if } (X : C1) \in H \text{ and } (X : C2) \in H \text{ and } (X : and(C1, C2)) \notin H. \end{array}$$

Theorem 5.4 (Termination of the Factorization Process)
Any sequence of application of the factorization rule is finite.

Theorem 5.5 (Correctness of Factorization)
Let $Q = F \not\!\!/\!\!/ H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by the factorization process; then for every model $\mathcal{I}$ of any database, $[F \not\!\!/\!\!/ H_1]^{\mathcal{I}} = [F \not\!\!/\!\!/ H_2]^{\mathcal{I}}$.

Before describing the classification used, we need to define a partial ordering on concepts, called subsumption relation. Let C_1 and C_2 be two concepts: C_1 subsumes C_2 iff for each underlying interpretation $\mathcal{J}$ such that $[\mathcal{SCH}]^{\mathcal{J}} = ASS^{\mathcal{J}}$ (i.e., the schema is satisfied by $\mathcal{J}$) and for each $\mathcal{J}$-assignment α we have $ext_{\mathcal{J}\alpha}(C_2) \subseteq ext_{\mathcal{J}\alpha}(C_1)$.

C_1 is called subsumer of C_2, and C_2 is said to be subsumed by C_1.

Let Ω be the set of all concepts which are not of the form $and(D_1, \ldots, D_m)$.

The classification of a concept C consists in finding in Ω its least subsumers and its greatest subsumees.

The reader interested in decidability and complexity of concept subsumption, or in concept classification may refer to [15].

Let H' be obtained from H by the factorization process. The core of the classification-based reformulation is performed as follows: for every atoms of the form $X : C$ in H', first classify C and then replace it in $X : C$ by the conjunction of its least subsumers.

For example, with respect to a schema containing only one atomic constraint $C_1 < C_2$, for the concept $and(C_1, C_2, C_3)$ we obtain the two most-specific subsumers C_1 and C_3 in Ω. Thus, the concept $and(C_1, C_2, C_3)$ is simplified and replaced by $and(C_1, C_3)$.

In the case of another schema containing only the definition of a materialized view $C_4 := and(C_1, C_2)$, the classification of $and(C_1, C_2, C_3)$ gives the two most-specific subsumers C_4 and C_3 in Ω. Then, $and(C_1, C_2, C_3)$ is replaced by $and(C_4, C_3)$, which takes advantage of the materialized view.

Theorem 5.6 (Correctness of Classification-Based Reformulation) *Let $\mathcal{DB} = \langle \mathcal{SCH}, \mathcal{EDB}, \mathcal{IDB} \rangle$ be a database, $Q = F \,/\!/\, H_1$ be a query, H_2 be a set of atomic constraints obtained from H_1 by a classification-based reformulation; then for every model $\mathcal{I}$ of $\mathcal{DB}$, $[F \,/\!/\, H_1]^{\mathcal{I}} = [F \,/\!/\, H_2]^{\mathcal{I}}$.*

Consider the example of Section 3. The factorization and classification-based reformulation is applied to the following query part which refers to the extensional database:

X:teacher & E:lecturer & X.works_in_project → Z & Z.managed_by → E
& X:student & Z:project & E:permanent_staff_member

The result of the factorization process is:

X:and(teacher, student) & E:and(permanent_staff_member, lecturer) &
X.works_in_project → Z & Z.managed_by → E & Z:project

Using the atomic constraint assistant := and(student, teacher) contained in the schema *(s9)*, the classification of the concept and(teacher, student) gives a single least subsumer: the concept assistant. Likewise, because of *(s6)* lecturer < and(permanent_staff_member, teacher, not(professor)) the classification of and(permanent_staff_member, lecturer) gives lecturer as the only least subsumer.

Then, X:and(teacher, student) and E:and(permanent_staff_member, lecturer) are replaced respectively by X:assistant and E:lecturer. So, we have eliminated a redundant constraint and taken into account a materialized view to simplify the query.

5.4 Reformulation Heuristics

A query can often be reformulated in several ways, and it is important to choose the most interesting reformulation. Chakravarthy et al. [7] have suggested three solutions: (1) generate several equivalent queries and let the evaluation plan generator (of the database) select the optimal one. (2) use heuristics based on an evaluation cost model to guide the query reformulation. (3) combine (1) and (2) by tightly integrating the plan generation and the query reformulation.

These aspects are out of the scope of this paper. However, we mention a preferential ordering for the different kinds of reformulations.

The transformations based on resolution and those based on classification are complementary. The former helps to simplify the part of the query that refers to the intensional database, while the latter removes redundant atoms that refer to the extensionnal database. Since a resolution step can add new atoms that refer to the extensional database, it is preferable to perform a classification-based reformulation after a reformulation by resolution.

The propagation rules simplify equality constraints (e.g., those generated by resolution) and also adds atoms to the query. This can be useful to perform a more accurate classification. Thus a propagation-based reformulations should be done before a classification process and after a resolution-based reformulation.

6 Conclusion

Two techniques have already been proposed for semantic query optimization in deductive object oriented databases:

- The first uses a highly expressive language for database description and a specific resolution for query reformulation. This technique is an extension of that of Chakravarthy et al. [7]. It has been developed by Yoon and Kerschberg [20] to exploit the inheritance links of an object-oriented database and the clauses describing its intensional part.
- The second technique is based on a restricted language to describe the schema and the deduction rules. This turns a particular subsumption relation to be decidable and then provides a classification process. This computation leads to simplifications of the query that cannot be obtained in the first approach, where this subsumption relation is undecidable in general [18]. This technique has been explored by Beck et al. [2], Borgida et al. [3] and Buchheit et al. [5].

Our proposal is a reconciliation of these two approaches. Indeed, we use clauses to describe the intensional database and more restricted formulae to specify the database schema. Thus, on one hand, we can describe derived classes and derived attributes with enough generality. And, on the other hand, we can take into account conventional integrity constraints of object-oriented databases (e.g., types of attributes, single-valuations of attributes, mandatory valuations, disjunctions of classes), and a restricted form of materialized views to perform simplifications by means of a decidable subsumption relation.

Important problems not covered by the proposal are the handling of negation in the clause bodies and the ordering of the reformulations.

The main contribution of this paper is to show that resolution-based reasoning and classification-based reasoning can be used together in a common framework to perform complementary semantic query optimizations.

To simplify the presentation we have used only the core of an integrity constraint language. However, it can be easily extended without any change to the framework, if we take care to keep the subsumption relation decidable. The interested reader may find such extensions in [4] for cardinality constraints and in [1] for numerical constraints. Another possibility is to allow also clauses describing integrity constraints. In this case, the classical technique of Chakravarthy et al.[7] can be adapted, but this kind of integrity constraints will not be used in the classification-based reformulation.

An important further issue is the integration of the reformulation scheme proposed in this paper within a query evaluation process.

References

1. F. Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence*, pages 452–457, Sydney, Australia, August 1991.
2. H. Beck, S. Gala, and S. Navathe. Classification as a query processing technique in the CANDIDE semantic data model. In *Proc. of the Fifth International Conference on Data Engineering*, pages 572–581, Los Angeles, California, USA, February 1989.
3. A. Borgida, R. Brachman, D. McGuinness, and L. Resnick. CLASSIC : a structural data model for objects. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 58–67, Portland, Oregon, June 1989.
4. M. Buchheit, F. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, pages 704–709, Chambery, France, August 1993.
5. M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, January 1994.
6. H.-J. Bürckert and W. Nutt. On abduction and answer generation through constrained resolution. Technical Report RR-92-51, German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany, October 1992.
7. U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. on Database Systems*, 15(2):162–207, June 1990.
8. S. Cluet and C. Delobel. A general framework for the optimization of object-oriented queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 383–392, San Diego, California, USA, June 1992.
9. A. Colmerauer. An introduction to prolog III. *Communications of the ACM*, 33(7):69–90, July 1990.

10. M. S. Hacid and C. Rigotti. Combining resolution and classification for semantic query optimization in DOOD. Technical Report RR-95-02, LISI, Lyon, 1995.
11. M. Höhfeld and G. Smolka. Definite relations over constraint languages. LILOG Report 53, IBM Deutschland, Stuttgart, Germany, October 1988.
12. J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proc. of the 14th ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 1987.
13. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.
14. R. Mac Gregor. The evolving technology of classification-based knowledge representation systems. In J. Sowa, editor, *Principles of Semantic Networks*, pages 385–400. Morgan Kaufmann, San Mateo, California, 1991.
15. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* Number 422 in LNAI. Springer-Verlag, Berlin, 1990.
16. C. Rigotti. *Abduction and Deduction of Object Structural Properties.* Doctoral Dissertation (Forthcoming, in French), INSA Lyon, 1995.
17. C. Rigotti, M. S. Hacid, and J.-F. Boulicaut. Une approche multi-paradigmes pour tester la cohérence d'applications BDOO. In N. Bidoit, editor, *Actes des 10èmes journées Bases de Données Avancées, BDA'94*, pages 55–74, Clermont-Ferrand, France, September 1994.
18. O. Shmueli. Equivalence of DATALOG queries is undecidable. *Journal of Logic Programming*, 15(3):231–241, February 1993.
19. D. D. Straube and M. T. Özsu. Queries and query processing in object-oriented database systems. *ACM Transactions on Information Systems*, 8(4):387–430, October 1990.
20. J. P. Yoon and L. Kerschberg. Semantic query optimization in deductive object-oriented databases. In *Proc. of the 3th Int. Conf. on Deductive and Object-Oriented Databases*, number 760 in LNCS, pages 169–182, Phoenix, Arizona, USA, December 1993. Springer-Verlag.