Description Logics for Semantic Query Optimization in Object-Oriented Database Systems

DOMENICO BENEVENTANO and SONIA BERGAMASCHI
Università di Modena e Reggio Emilia and CSITE–CNR
and
CLAUDIO SARTORI
Università di Bologna and CSITE–CNR

Semantic query optimization uses semantic knowledge (i.e., integrity constraints) to transform a query into an equivalent one that may be answered more efficiently. This article proposes a general method for semantic query optimization in the framework of Object-Oriented Database Systems. The method is effective for a large class of queries, including conjunctive recursive queries expressed with regular path expressions and is based on three ingredients. The first is a Description Logic, ODL_{RE}, providing a type system capable of expressing: class descriptions, queries, views, integrity constraint rules and inference techniques, such as *incoherence* detection and *subsumption* computation. The second is a semantic expansion function for queries, which incorporates restrictions logically implied by the query and the schema (classes + rules) in one query. The third is an optimal rewriting method of a query with respect to the schema classes that rewrites a query into an equivalent one, by determining more specialized classes to be accessed and by reducing the number of factors. We implemented the method in a tool providing an ODMG-compliant interface that allows a full interaction with OQL queries, wrapping underlying Description Logic representation and techniques to the user.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*object-oriented databases*; *query processing*

General Terms: Algorithms, Management, Languages, Theory

Additional Key Words and Phrases: Semantic query optimization, query rewriting method, integrity constraints rules, semantic expansion of a query, description logics, subsumption

This research was partially funded by the Italian M.U.R.S.T. ex. 40% "INTERDATA" and "D2I" projects.

Authors' addresses: D. Beneventano and S. Bergamaschi, D11, Via Vignolese 905, I-41100 Modena, Italy; email: {domenico.beneventano;sonia.bergamaschi}@unimo.it; C. Sartori. DEIS, Viale Risorgimento 2, I-40136 Bologna, Italy; email: csartori@deis.unibo.it.

1. INTRODUCTION AND MOTIVATION

The purpose of semantic query optimization is that of transforming a query into an *equivalent* one, which may be answered more efficiently. The transformed query is equivalent to the original one if it gives the same answer for every legal database state.

The notion of semantic query optimization for relational databases was introduced in the early 80's by King [1981]; Hammer and Zdonik [1980] independently developed very similar optimization methods. The key idea in King [1981], as well as in Hammer and Zdonik [1980], is that integrity constraints may not only be utilized to enforce consistency of a database, but may also optimize user queries.

Following Hammer and Zdonik [1980] and King [1981], the method for semantic query optimization that we propose uses integrity constraints, declaratively expressed as part of the schema, and is effective for a large class of queries, including conjunctive recursive queries in the framework of Object-Oriented Database Systems. The method includes the different query transformation criteria proposed in the literature [Hammer and Zdonik 1980; King 1981]:

—*Incoherence detection*. If the query is incoherent with respect to the database schema and the integrity constraints, then a null answer can be immediately returned without accessing the database;
—*Factor Removal*. Removal of implied factors (both restriction and join) from the query;
—*Factor Introduction*. Introduction of a factor implied by the query; it may prove useful to produce an alternative evaluation plan in further physical query optimization activities.

The main new achievements are the following:

—it applies to Object, Object-Relational and Relational DBMS;
—it extends the ODMG standard to support declarative integrity constraints and *regular path expressions*;
—a conjunctive query is rewritten in an *optimal form*, by determining more specialized classes to be accessed and by minimizing the number of factors;
—a significant set of recursive (conjunctive) queries is supported.

The method is based on three ingredients. The first is a Description Logics, ODL_{RE} (*Object Description Language with Regular Expressions*) which extends the one developed in Bergamaschi and Nebel [1994] and is capable of expressing class descriptions, queries, views and integrity constraints rules as types. The notion of database schema is thus generalized by introducing *integrity constraint rules* and *views* in the schema. ODL_{RE} provides the core theoretical framework in terms of Description Logics (*DL*) inference techniques, *incoherence* detection and *subsumption* computation, to develop a theory of semantic query optimization. The second one is a *semantic expansion* function [Shenoy and Özsoyoglu 1987, 1989], which incorporates restrictions logically implied by a type and the schema (classes + rules) into the type itself. The third ingredient is a rewriting method

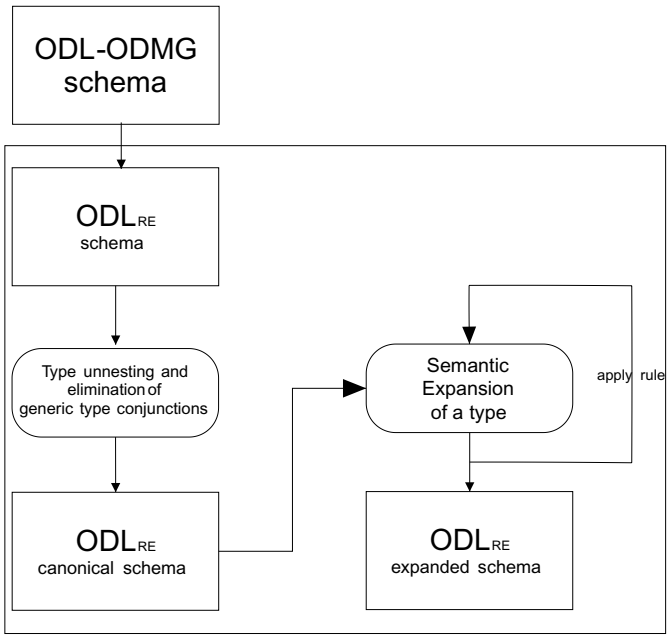


Fig. 1. Schema compilation process.

which produces an equivalent query, by determining more specialized classes to be accessed and by detecting/removing (if necessary) redundant factors.

The chosen strategy for semantic query optimization is the following: We *compile* the database schema (classes+views+integrity constraints rules), thereby creating an *enriched schema*, that is, *expanded schema*; the database schema is provided in ODL-ODMG and the expanded schema is expressed in ODL_{RE}. The compilation process (see Figure 1) is based on the generation of a *semantic expansion* of the schema types. Semantic expansion is based on the iteration of this simple transformation: if a type implies the antecedent of an integrity constraint rule (which is a type) then the consequent of that rule (which is a type too) can be conjuncted. Logical implications between types are evaluated by means of *subsumption computation*; intuitively, subsumption evaluates implicit *is a* relationships between types based on their descriptions (see Brachman and Schmolze [1985] for a general description of subsumption and Bergamaschi and Nebel [1994] for the subsumption algorithm used in this article).

At run time (see Figure 2), we add the query Q to be optimized, expressed in OQL-ODMG, to the compiled schema and reactivate the compilation process for Q, thus obtaining a semantic expansion of Q. If the query is found to be incoherent, a null answer can be immediately returned without accessing the database. If it is coherent, the *expanded query* contains more specialized classes to be accessed and, thus, possibly the *query is moved down* within the classes generalization hierarchy, obtaining an immediate optimization result. Then, the redundant factors (i.e., the factors that can be removed from the query

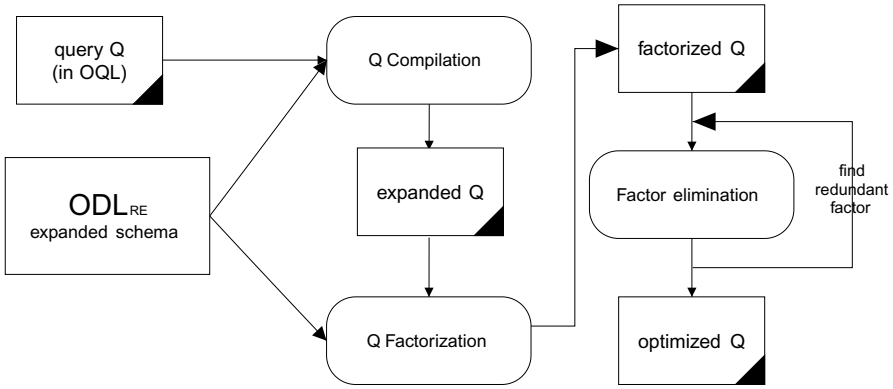


Fig. 2. Query Optimization Process

without affecting the result for any database state) are detected and removed (if necessary) from the query.

The above-mentioned optimization method has been partially implemented[1] in the ODB-QOptimizer tool [Beneventano et al. 1997], available on Internet at the address www.dbgroup.unimo.it/ODB-Tools.html. ODB-QOptimizer supports an ODMG [Cattell 1994] compliant user interface. Schemata and queries are provided in ODL-ODMG and OQL-ODMG languages respectively. The translation of the schema and queries into the Description Logic ODL_{RE} is performed by the system and is completely transparent to the user.[2]

In order to declaratively express integrity constraints and to formulate recursive queries, we extend ODMG as described in the following:

Integrity constraint rules. In an ODL schema, integrity constraints are hidden in methods, that is, in the behavioral part of database schema. Our extension to ODL enables the definition, in a declarative style, of a subset of integrity constraints represented as *if-then rules*. At the schema design level, this choice enables the consistency and redundancy check of the schema as shown in Beneventano et al. [1998]. At the query optimization level, this choice makes further semantic knowledge available to drive the semantic query optimization process. The proposed extension follows the ODMG standard: the antecedent and the consequent of an *if-then rule* are expressed in a restricted OQL syntax (see Section 2.2).

Regular path expressions. OQL path expressions have been extended to *regular path expressions* universally quantified (proposed for query languages in the framework of semistructured data [Abiteboul et al. 1997; Abiteboul and Vianu 1997; Calvanese et al. 1999; Fernandez and Suciu 1998]) in order to express recursive conjunctive queries in a direct and simple way.

[1]The present version supports schema compilation and semantic expansion; it rewrites the semantically expanded query in OQL but does not perform redundant factors elimination.
[2]ODMG is considered in its 93 version. In the following ODL-ODMG and OQL-ODMG will be referenced by ODL and OQL, respectively.

The expressiveness of a query language such as OQL makes it impossible to provide a *DL*, which is able to translate any possible query and to allow decidable reasoning activities (i.e., incoherence and subsumption). In addition, it has been proven that increasing expressiveness of decidable *DL* quickly leads to computational intractability (see, e.g., Donini et al. [1991]). For example, we proved in Beneventano et al. [1998] that for a *DL* allowing the expression of both cyclic views/conjunctive recursive queries and comparisons between path expressions, the subsumption computation becomes undecidable. We thus developed a *DL*, ODL_{RE}, where the reasoning activities are in most real cases tractable and identify the subset of OQL which can be translated into ODL_{RE}.

In other words, following Buchheit et al. [1994], we separate an OQL query into a *clean* part (that can be mapped into an ODL_{RE} type) and a *dirty* part, which exceeds ODL_{RE} expressiveness. The clean part of a query contains comparison predicates between a regular path expression and a complex value (for the syntax see Section 2.2).

Our semantic optimization method will be applied only over the clean part and thus it is intrinsically incomplete with respect to a generic OQL query. Another source of incompleteness is the semantic expansion algorithm introduced in the paper that is sound but not complete. The incompleteness of the method is not a strict restriction, as optimization can be useful even if it does not find the optimum solution. On the other hand, when the costs for optimization computation are not negligible, it is necessary to seek a trade-off between the cost savings due to optimization and the optimization cost. Our experimental results show that our cost to compute a semantic expansion of a query is always negligible with respect to the cost savings (see www.dbgroup.unimo.it/ODB-Tools.html).

Let's refer to the classification proposed by Chakravarthy et al. [1990] as a general framework to introduce our approach: "There are several aspects to semantic query optimization: the type of database under consideration and the type of integrity constraints allowed; the generation of semantically equivalent queries and their correctness; filtering of useless information; the integration of semantic query optimization with conventional query optimization and how to limit the generation process to *promising* candidate queries."

—*Database and integrity constraints under consideration*. The target databases are OODBs supporting complex object data models [Abiteboul and Kanellakis 1989; Lecluse and Richard 1989], Object-Relational databases and Relational databases. We restrict integrity rules to those that can be expressed by logical implications between ODL_{RE} types, thus we can efficiently compute a semantic expansion of a query with our subsumption algorithm [Bergamaschi and Nebel 1994].

—*Generation of semantically equivalent queries and their correctness*. The devised semantic expansion algorithm is correct and, for a given query, gives rise to a class of queries, which are semantically equivalent to it. The correctness of this function is based on the formal semantics of ODL_{RE} types and reasoning activities.

—*Limiting the generation process to "promising" candidate queries and filtering of useless information*. The semantic expansion algorithm, following the usual logic-based approach, does not consider *heuristics* to guide or limit the addition of factors during the query transformation. In this way, the task of choosing the beneficial transformations is delayed until all the possible transformations have been considered: as many factors as possible are generated, the redundancy testing for factors and the incoherence detection can be conducted to a maximum degree [Shenoy and Özsoyoglu 1987, 1989]. The resulting query, that is, expanded query, usually refers to more specialized classes with respect to the original. Moreover, we provide techniques to check and eliminate (if necessary) redundant factors.

—*Integration with conventional query optimization*. The proposed method is general and independent from any specific cost model and storage details and thus it is suitable for an easy integration, on the top of a traditional query optimizer.

Original contributions of this paper with respect to previous proposals in literature (see Section 6 for a more detailed account) are the following:

—We "move down" a query in the class generalization hierarchy, exploiting integrity constraint rules. Thus we go beyond *automatic classification* of a query with respect to views/queries as previously proposed in the *DL* literature [Beck et al. 1989; Beneventano and Bergamaschi 1997; Borgida et al. 1989; Buchheit et al. 1994]. Furthermore, our objective is also different from that of maximal rewriting of conjunctive queries with regard to a set of given views [Beeri et al. 1997; Calvanese et al. 1999]. Considering that our method is based on a *DL* kernel, it also performs automatic classification of a query, but we observe that this technique can be seen as a semantic query optimization only if we assume that views are materialized.

—By means of *semantic expansion*, we address and solve the problem of finding *a more specialized rewriting* of a query in terms of the schema classes. It is an effective query optimization, since many OODBs maintain the extent for a class, that is, the *oids* of all objects in the most specialized class.

—We address and solve the problem of finding *an optimal query rewriting method* minimizing the number of its factors, thus, usually, minimizing the number of classes to be accessed to solve a query. For queries whose semantic expansion does not involve any recursive rule, we provide a very simple method to independently eliminate all redundant factors.

—Our method is applicable to conjunctive recursive queries and in the presence of cyclic schemata; in the case of the restricted form of recursion, that is, linear recursion, we prove that a linear recursive query can be expressed with transitive closure and thus we extend the result of Jagadish et al. [1987] in relational environment.

This work follows the track commenced in Bergamaschi and Sartori [1992] in applying *DLs* to databases and extends previous works of the authors with regard to the proposed *DL* [Bergamaschi and Nebel 1994; Beneventano and

Bergamaschi 1997] and to the semantic query optimization method introduced in Beneventano et al. [1996, 1997] as follows:

—by providing ODL_{RE} which extends the Description Logic introduced in Bergamaschi and Nebel [1994] with integrity constraint rules and regular path expressions;
—by extending the set of queries for which the method is applicable to conjunctive recursive queries including path expressions;
—by providing a correct algorithm for semantic expansion computation;
—by addressing the problem of finding an optimal query rewriting method.

The outline of the article is the following: In this section, we introduce a roadmap to the technical part to help the reader navigate through the complex and different formalizations introduced in the article. Section 2 introduces our approach to semantic query optimization, presenting the extensions introduced in ODMG syntax and then, with reference to a running example used throughout the article, its effectiveness, main achievements and limitations. Section 3 deals with the data model, describing syntax and semantics of ODL_{RE}. Section 4 introduces the formal definitions of: Incoherence, Subsumption, Semantic Expansion of a type and the Semantic Expansion algorithm. Section 5 presents our query rewriting method. Finally, Sections 6 and 7 discuss related works and present our conclusions.

1.1 Roadmap of the Technical Part

The roadmap of the three technical sections of the article (from Section 3 to 5) aims at explaining the meaning and the sequence of formalizations and algorithms that make up our semantic query optimization method. This description is intended especially for the reader who is not interested in going into the various formal methods.

The three technical sections of the article deal, respectively, with the three ingredients mentioned in the beginning. In Section 3, we introduce the ODL_{RE} description logic, illustrate its type constructors and provide its formal semantics. In particular, we describe regular path expressions, which are an essential part of object query languages. Then, we introduce the *rule schema*, which is our specific extension to represent *if-then* rules in an object schema and the distinction of the instances of the schema into *possible* (i.e., in accordance with the type definitions) and *legal* (i.e., possible and in accordance with the rules). This section is dedicated to the reader interested in understanding the semantics of the supported language.

Section 4 provides the definitions for *subsumption* between types, *incoherence* of a type and semantic expansion of a type being the core instrument for our activity. Usually, *DL* algorithms do not explicitly take rules into account, and therefore we eliminate them by transforming the schema with a *semantic expansion* of types into an *expanded schema*. This is an iterative process, based on subsumption, triggered by a rule that is *applicable* to some type and ends when every rule has been incorporated into a type, changing its definition. Section 4.1 implements the above mentioned concepts by providing the

semantic expansion algorithm. In particular, in order to simplify it and to exploit the subsumption and incoherence algorithms already available for ODL Description Logic introduced in Bergamaschi and Nebel [1994], it proves useful to transform the schema into a *canonical form*. A canonical schema is a schema where types are canonical, that is, unnested, conjunctions of types are eliminated (whenever possible) and each name description contains the conjunction of primitive names (without description) and a single canonical type description. The semantic expansion algorithm produces the expanded schema, which is a conservative extension of the original schema. The section is completed with the proofs.

Section 5 describes our optimal query rewriting method, which gives an *equivalent query rewritten on the basis of more specialized classes of the schema*. Considering that a query can be expressed as a virtual ODL_{RE} type, it can be expanded by the semantic expansion algorithm. The expanded query is a "redundant" rewriting of the query in terms of more specific classes and base types. An expanded query, in fact, is the conjunction of many, possibly redundant factors, due both to the semantic expansion process and to the adopted canonical form; furthermore, it includes references to virtual names. Then, to obtain the optimal rewriting, we must remove redundancies and references to virtual names.

The first step is to check the coherence of the expanded query; in the case of incoherence, a null answer can be returned, giving an immediate optimization result. If the query is coherent we rewrite the expanded query as a *conjunction of factors*, expressed in terms of more specialized classes and base types. The factorization is obtained by deriving a proper *finite automaton* from the query, where each factor is a path type whose path is a union-free regular path expression.

The second step is to find out which factors of the query are redundant and thus, if necessary, can be eliminated from the query. The detection of redundant factors is based on very general criteria, based merely on logical properties. To provide a simple method to independently eliminate redundant factors we need to check the absence of recursive rules in the semantic expansion of the query: the check is done by introducing rule graphs to formally define recursive rules in our context and by using subsumption. The elimination of redundant factors generates an *optimized form* of the query that is semantically equivalent to the original one and, hopefully, less expensive, which can be easily translated into an OQL query with path expression extensions.

2. SEMANTIC QUERY OPTIMIZATION: EXPLAINING THE METHOD BY EXAMPLES

In order to illustrate our method we introduce herein the Company Domain example and the ODL extensions; the OQL extensions and the main query optimization achievements.

2.1 The Company Domain and the ODL Extensions

The Company Domain describes (a part of) the organizational structure of a company. Departments have a name and a category (from 1 to 7), employ a set

of employees and have at least one employee with an administrative function. Employees are described by a name, a qualification (from 1 to 10), a salary (from 10 to 100), a level composed of a class and a parameter (from 1 to 5), work in a department, have an employee as head and a set of skills. Managers are employees having a qualification from 8 to 10, a manager as head and direct a department. Core Business Departments (`CB_Department`) are departments having a high category (from 5 to 7) and a manager as administrator. Medium qualification employees (`Mdl_Employee`) are *exactly* those employees having a medium qualification (from 5 to 10). Clerks are *exactly* those employees with a qualification between 7 to 10 who have a clerk as head. A manager having a qualification greater or equal to 9 must direct a `CB_Department`. Departments that employ only employees having a qualification greater or equal to 5 are CB_Departments. Employees with a salary greater or equal to 60 and a head with a qualification greater or equal to 7 are managers. Managers directing departments with a category greater or equal to 4 and a salary greater or equal to 50 have a qualification equal to 10.

In Table I, the Company Domain schema in extended ODL syntax is given. The extensions introduced into ODL to represent integrity constraints are: value ranges, virtual classes (i.e., views) and integrity *if-then* rules (i.e., rule_{i-j} in Table I). A schema may thus contain usual *primitive* classes (`interface`), *virtual* classes (`view`) and complex value types (`relation`). From an extensional point of view, this means that, for example, we have to explicitly fill the interface `Employee`, while every object that is entered as an employee having a qualification from 5 to 10 is automatically entered in the view `Mdl_Employee`. Moreover, from an intensional point of view, views enable *subsumption relations* (i.e., specialization relations implicitly hidden in class descriptions) to be computed. Cyclic views are allowed in the schema, as in the case of `Clerk`. Furthermore, a schema may contain integrity rules (rules `r_1` to `r_4` in Table I) (For the extended ODL syntax, see Appendix A).

2.2 The Target Query Language: OQL Extensions and Limitations

In general, query languages in complex object environment are made of two components: one for selecting objects and one for producing the actual answer by navigating the aggregation hierarchy of the selected objects. We will consider the first component of the language. The clean part of a query (see Table II, column 2) is a generalization of a generic OQL query, where `CleanCondition` restricts/extends OQL conditions.

In Table II, `multipath` is a path including at least a multivalued attribute, `rpe` is a regular path expression, `CleanSet` is a `ClassName` or a `(select X from ClassName X where CleanCondition)`.

OQL has been extended with the capacity of expressing a *regular path expression* (`rpe`). A regular path expression is defined by: *composition*, (represented by dot notation "." and already present in OQL), *Kleene closure*, represented by "*", *union*, represented by "+" (see Hopcroft and Ullman [1979]).

In particular, we include the Kleene closure in our OQL extension, since our focus is on expressing and generalizing the *transitive closure* of an attribute

Table I. The Company Domain Schema in Extended ODL Syntax

```
relation Level ()
{  attribute string class;
   attribute range 1 to 5 parameter;  };

interface Department ()
{  attribute string dname;
   attribute range 1 to 7 category;
   attribute set<Employee> employs;
   attribute Employee administrator;  };

interface Employee ()
{  attribute string name;
   attribute range 1 to 10 qualification;
   attribute range 10 to 100 salary;
   attribute Level level;
   attribute Employee head;
   attribute Department worksin;
   attribute set<string> skill;  };

interface Manager: Employee ()
{  attribute range 8 to 10 qualification;
   attribute Manager head;
   attribute Department directs;  };

interface CB_Department: Department ()
{  attribute range 5 to 7 category;
   attribute Manager administrator;  };

view Mdl_Employee: Employee ()
{  attribute range 5 to 10 qualification;  };

view Clerk: Employee ()
{  attribute range 7 to 10 qualification;
   attribute Clerk head;  };

rule r_1 forall X in Manager: X.qualification >= 9
   then X.directs in CB_Department;

rule r_2 forall X in Department: forall Y in X.employs : Y.qualification >= 5
   then X in CB_Department;

rule r_3 forall X in Employee: X.salary >= 60 and X.head.qualification>=7
   then X in Manager;

rule r_4 forall X in Manager: X.directs.category >= 4 and X.salary>=50
   then X.qualification = 10;
```

in a query formulation, as introduced for OODBs query languages in Bertino et al. [1992]. In Bertino et al. [1992], the transitive closure of an attribute a is expressed by means of the ρ operator, which has the following behavior: given an object o, instance of a class C, and a property a of o, $o.\rho(a)$ defines a set of objects that are either values of the property a of o, or values of property a of

Table II. Clean Query & OQL Condition vs. Clean Condition

	`select *`
	`from ClassName`
	`where CleanCondition`

OQL Condition	**(CC) CleanCondition**
query θ query	path θ value
query <and|or|not> query	CC and CC
query **in** query	path **in** CleanSet
<forall|exists> X in query : query	<forall|exists> X in multipath : CC(X)
query <intersect|forall|exists> query	CleanSet intersect CleanSet
	<forall> X in rpe : CC(X)

Table III. Query Q1 in Extended OQL

```
select   *
from     Employee
where    salary >=80
and      forall X in head.(head)* : X.qualification >= 7
```

some object o', which is a value of property a of object o, etc. Objects in this set are both cyclic and acyclic objects and are implicitly bound to class C, which is the domain of property a. This form of recursion, which corresponds to the notion of linear recursion, makes it possible to express most of the recursive queries encountered in real applications [Bertino et al. 1992; Florescu et al. 1998].

We propose denoting in OQL the transitive closure of an attribute `a` by `(a)*`; as in general a `rpe` identifies a set of objects, it will be used in a quantified expression as follows:

```
forall  X in rpe : CC(X)
```

In this way, we can express conjunctive recursive queries as the following: "select the names of the employees with a salary greater or equal to 80 and having a head, at each level, with a qualification greater or equal to 7" (see Table III).

If we express the closure of a multivalued attribute (or, in general, of a path with a multivalued attribute) we obtain a set of objects at each level and thus the CC(X) condition must be quantified, as shown in the following example:

```
select   *
from     Employee
where    forall X in (worksin.employs)* :
         exists Y in X :Y.qualification >= 7
```
(Q2)

The above Q1 and Q2 queries are *clean* and thus are optimized by our method. What is left out in our optimization method are recursive queries that can be

expressed by means of rpe with existential quantification:

```
exists  X in rpe : CC(X).
```

2.3 Main Query Optimization Contributions

Hereafter, the main semantic query optimization achievements are pointed out and explained by means of some examples related to the Company Domain. Achievements 1–3 and 5 are obtained by exploiting semantic expansion and *DL* inference techniques; achievement 4 is obtained on the expanded query by exploiting our optimal rewriting method. Finally, at point 6, the limitations of the method related to the expressiveness of the supported query language are sketched. All the query transformations performed below (except redundant factors elimination) can be directly obtained by using the ODB-QOptimizer available on Internet at the address www.dbgroup.unimo.it/ODB-Tools.html.

1. *Incoherence Detection*. If the semantic expansion of a query is found to be incoherent, a null answer can be immediately returned without accessing the database.

As a simple example of incoherence detection, let's introduce the query "select the core business departments (CB_Department) with an administrator of qualification 4":

```
select  *
from    CB_Department                              (Q3)
where   administrator.qualification = 4
```

The query is found to be incoherent by ODB-QOptimizer. In fact, the set of employees filling the administrator attribute must belong to the Manager class having a qualification value in the [8–10] range, thus not including the qualification value "4." The result, in terms of query optimization, is that a null answer can be immediately returned without accessing the database.

By means of integrity rules, less intuitive incoherencies may be detected, as in the following example:

```
select  *
from    Department
where   forall X in employs : X.qualification = 6   (Q4)
and     category = 4
```

The above query is found to be incoherent, by the definition of CB_Department and enforcement of rule r_2. In fact, a Department employing only employees with qualification greater than 5 is a CB_Department, but a CB_Department must be of category between 5 and 7.

2. *Query Classification with respect to Classes (not only Views)*. The semantic expansion of a query contains a more specialized class of the class-inheritance hierarchy rooted at the query and more specialized classes rooted at its subqueries. The substitution in the query of the explicitly given

classes with more specialized ones is an effective query optimization, despite any specific cost model, as many OODBMs maintain the extent for a class, that is, the *oids* of all objects in those classes. As an example of automatic query classification (by subsumption computation) with respect to views, let's introduce the query Q5 "select the employees having a qualification greater or equal to 6":

```
select   *
from     Employee                                    (Q5)
where    qualification >= 6
```

By classifying the above query into the classes taxonomy returns the `Mdl_Employee` view as the only immediate subsumer and the `Manager` class as the only immediate subsumee. Using this result, all instances of `Manager` are returned and only the instances of `Mdl_Employee` that are not instances of `Manager` are checked against the query-condition ("qualification greater or equal to 6") and included in the answer-set. If `Mdl_Employee` is a materialized view, an interesting query optimization result is thus obtained.

Automatic query classification has been previously proposed in the *DL* literature [Beck et al. 1989; Borgida et al. 1989; Beneventano and Bergamaschi 1997; Buchheit et al. 1994]; it can be considered a semantic query optimization technique only in the presence of materialized views. In this article, we go beyond this technique since we can "move down" queries with respect to class-inheritance hierarchy by exploiting integrity constraint rules. As an example of classification with respect to primitive classes, let's introduce the query "select the departments employing *only* managers and with an administrator of qualification equal to 10":

```
select   *
from     Department
where    forall X in employs : X in Manager
and      administrator in (select T                  (Q6)
                   from  Employee as T
                   where T.qualification=10)
```

After the application of rule `r_2`, Q6 is transformed as follows:

```
select   *
from     CB_Department
where    forall X in employs : X in Manager
and      administrator in (select T                  (Q6) optimized
                   from  Manager as T
                   where T.qualification=10)
```

This example shows an effective query optimization, resulting from the substitution of the classes mentioned in the query with their specializations. In fact, by replacing `Department` with `CB_Department` and `Employee` with `Manager` we reduce the number of classes to be accessed to process the query, since we eliminate the need to access the `Department` and `Employee` classes.

3. *Index Introduction*. The semantic expansion of a query expands it with factors logically implied by the query and by the schema (classes + integrity constraints rules). Some of these factors may be useful to produce an alternative evaluation plan in physical query optimization. In the relational literature [Shenoy and Özsoyoglu 1987; Siegel et al. 1992], this problem is generally addressed as *index introduction*, that is, the introduction of a factor including an indexed attribute. More generally speaking, in object-oriented database systems, *access support relations* are introduced as a means for optimizing query processing; the idea is that of keeping separate structures to redundantly store those object references that are frequently traversed by path expression in queries [Kemper and Moerkotte 1990].

A simple example of index introduction is given in the following query:

```
select  *
from    Manager
where   directs in CB_Department
and     salary > 60
```

Rule r_4, together with the definition of CB_Department, enables the addition of factor qualification = 10 to the query: if qualification is an indexed attribute, this factor may be useful in physical query optimization.

4. *Query Rewriting Method (Detecting Redundant Factors)*. Semantic expansion is the basis of a further optimization activity since it enables the detection of the *redundant* factors of a query, namely the factors logically implied by the query. In this way, we obtain a further and general query optimization, as, by eliminating factors, we reduce the number of classes, on a class-composition hierarchy rooted at the query, to be accessed. In the relational literature [Shenoy and Özsoyoglu 1987; Siegel et al. 1992], this problem is generally addressed as *constraint removal*, that is, removal of implied restrictions from the query. As we will illustrate hereafter, this activity must be performed on a semantic expansion of a query to be conducted to a maximum degree.

As an example of *redundant* factors, let's consider the following query:

```
select  *
from    Manager
where   qualification = 10
and directs.category >= 4
```
(Q7)

By rule r_1, since the second factor is implied by the first one, it can be eliminated from the query, avoiding the access to the CB_Department class. On the other hand, the following example shows how semantic expansion is useful to detect further redundant factors and thus eliminate expensive factors in terms of access.T

```
select  *
from    Manager
where   salary = 100
and directs.category >= 4
```
(Q8)

Table IV. Query Q1 Optimized (in OQL)

```
select      *
from        Manager
where       salary >=80
```

In this case, the factor `directs.category >= 4` is not redundant with respect to the rest of the query; however, semantic expansion adds the factor `qualification = 10` (by applying rule `r_4`) to the query, thus the query can be transformed after the expansion as follows, avoiding access to the class `CB_Department`:

```
select   *
from     Manager
where    salary = 100
and      qualification = 10
```
(Q8) optimized

5. *Recursive Queries Optimization (Factor Removal).* All the above optimization achievements can be obtained for conjunctive recursive queries too. Let's refer, as an example, to queries including a path expression with Kleene closure. With reference to query Q1, by applying rule `r_3`, the query is transformed as shown in Table IV, where the original target class `Employee` is substituted with its specialization `Manager` and thus the factor `head.(head)*.qualification >= 7` is eliminated from the query (as it is included in `Manager` description). Moreover, classifying query Q1 into the classes taxonomy returns the cyclic view `Clerk` as the only immediate subsumer.

6. *Limitations.* Only the *clean* part of an OQL query is transformed by our method. Let's introduce, as an example, the following query: "select employees having a qualification greater than or equal to 6 and working as administrators":

```
select   *
from     Employee as E
where    qualification >= 6
and      E=worksin.administrator
```
(Q9)

The dirty part of the above query (`E=worksin.administrator`) exploits *self* reference and *equality path* as usual in OODBs query languages: it only selects the employees who play the administrator role in the department where they work. This apparently simple factor is ignored since a *DL* allowing cyclic concepts and comparisons between path expressions leads to undecidability of subsumption computation [Beneventano et al. 1998]. Note that, in this case, even if we are able to perform subsumption computation only over the clean part of a query, we obtain an optimization result, since the above query is subsumed by the `Mdl_Employee` view.

The next two queries show cases where, by limiting the analysis to the clean part, subsumption and incoherencies are not detected.

```
select   *
from     Employee
```

```
where   worksin in CB_Department                          (Q10)
and     qualification=
            worksin.administrator.qualification
```

This query is intuitively subsumed by the view `Mdl_Employee`, whose range of qualification includes the qualification of the administrator of a `CB_Department`. On the other hand, subsumption is triggered by the predicate `qualification= worksin.administrator.qualification`, that is not included in the clean part, and thus ignored by our method. Similarly, the following incoherent query is not detected.

```
select  *
from    Manager                                           (Q11)
where   qualification=worksin.category
```

A further restriction in the expressiveness of clean queries is the existential quantification of `rpe`. For example, consider the existential reformulation of query Q1 "select the names of employees with a salary greater or equal to 80 and having at least a head with a qualification greater or equal to 7":

```
select  *
from    Employee
where   salary >=80
and     exists X in head.(head)* : X.qualification >= 7
```

The factor `exists X in head.(head)* : X.qualification >= 7` is not included in the clean part of the query and is thus ignored by our method. To deal with this kind of query in a Description Logics setting a disjunction operator, not included in our *DL*, would be necessary. As shown in a recent work [Calvanese et al. 1999], the computation of subsumption is decidable in a *DL* including these features. Nevertheless, the integration of these features in a context of semantic query optimization and its implementation in a system are such a complex problem to require a new research effort.

3. THE ODL_{RE} DESCRIPTION LOGICS

ODL_{RE} is an extension of the ODL (*Object Description Language*),[3] introduced in Bergamaschi and Nebel [1994], used in Beneventano and Bergamaschi [1997] and is in the tradition of complex object data models [Abiteboul and Kanellakis 1989; Lecluse and Richard 1989]. ODL_{RE}, as its ancestor ODL, provides a *system of base types*: string, Boolean, integer, real; the type constructors *tuple*, *set*[4] and *class* allow the construction of complex value types and class types. Class types (also briefly called *classes*) denote sets of *objects with an identity and a value*, while *value-types* denote sets of *complex, finitely nested values without object identity*. Additionally, an intersection operator can be used to create intersections of previously introduced types allowing simple and multiple inheritance.

[3]Not to be confused with ODL-ODMG.

[4]ODL_{RE} extends the set type with both existentially and universally quantified set.

Finally, types can be given names. Named types come in two flavours: a named type may be *primitive*, which means that the user must specify the membership of an *element* in the interpretation of the name, or *virtual*, in which case its interpretation is computed.

The main extension to ODL introduced in ODL_{RE} are *regular path expressions* and *integrity constraints*.

Path expressions, which are essentially sequences of attributes, represent the central ingredient of O–O query languages to navigate through the aggregation hierarchies of classes and types of a schema [Bertino et al. 1992; den Bussche and Vossen 1993; Kifer et al. 1992]. Moreover, as introduced in Coburn and Weddel [1991], path expressions are useful to express integrity constraints. *DLs* with regular expressions, which can be seen as a form of fixpoints to model recursive concepts, have been studied by Baader [1991]. Recently, path expressions have been extended to *regular expressions* in the framework of semistructured data, both for query languages schema [Abiteboul et al. 1997; Abiteboul and Vianu 1997; Calvanese et al. 1999; Fernandez and Suciu 1998] and integrity path constraints [Abiteboul and Vianu 1997].

Integrity constraints are represented as *if-then rules*, where the antecedent and consequent are both ODL_{RE} types.

3.1 Values and Objects

We assume the union of the integers, the strings, the booleans, and the reals as the set $\mathcal{D}$ of *base values*. To build *complex values*, we further assume two disjoint countable sets of symbols $\mathbf{A}$ and $\mathcal{O}$, respectively called the *attributes* (denoted by $a, a_1, a_2, \ldots$), and the *object identifiers* (denoted by $o, o_1, o_2, \ldots$), both disjoint from $\mathcal{D}$. The set $\mathcal{V}$ of all *values over* $\mathcal{O}$ is defined as the smallest set containing $\mathcal{D}$ and $\mathcal{O}$, such that, if $v_1, \ldots, v_p$ are values, then the set $\{v_1, \ldots, v_p\}$ is a value, and a partial function $t: \mathbf{A} \rightarrow \{v_1, \ldots, v_p\}$ is a value. Function t is the usual tuple value; the standard notation $[a_1: v_1, \ldots, a_p: v_p]$ will be henceforth used. A *total value function* δ from $\mathcal{O}$ to $\mathcal{V}$ assigns values to object identifiers.

3.2 Paths

Let's consider the finite alphabet $\Gamma = \mathbf{A} \cup \{\triangle, \forall, \exists\}$. The symbol $\triangle$ is introduced to navigate through class types; $\forall$ and $\exists$ are introduced to navigate through set types, by a universal and existential quantification, respectively. A *word* w of Γ is either the symbol ϵ, or a dot-separated sequence of elements $e_1 . e_2 . \cdots . e_n$, where $e_i \in \Gamma$ $(i = 1, \ldots, n)$, $n > 0$. ϵ denotes the unique word of length 0.

Let's recall the definition of regular expression [Hopcroft and Ullman 1979]. The *regular expressions* over Γ and the sets of words, that is, the languages that they denote, are defined recursively as follows:

(1) $\emptyset$ is a regular expression and denotes the empty set.
(2) ϵ is a regular expression and denotes the set $\{\epsilon\}$.
(3) Every symbol e of Γ is a regular expression and denotes the set $\{e\}$.

(4) If h, k are regular expressions denoting the languages H, K, respectively, then
—the *concatenation* $(h.k)$,
—the *union* $(h+k)$,
—the *Kleene closure* $((h)^*)$
are regular expressions denoting the languages: $HK = \{w_1.w_2 \mid w_1 \in H, w_2 \in K\}$, $H \cup K$ and $(H)^* = \bigcup_{n\geq 0} H^n$, where $H^n = \{w_1.w_2 \ldots w_n \mid n \geq 0 \text{ and } w_i \in H \text{ for } 1 \leq i \leq n\}$, respectively.

A *path*, denoted by p, is a *regular path expression*. With $\mathcal{L}(p)$ we indicate the language denoted by p. For instance, $p = \texttt{head}.(\texttt{head})^*$ of query Q1 describes the infinite set of words $\mathcal{L}(p) = \{\texttt{head}, \texttt{head.head}, \texttt{head.head.head}, \ldots\}$.

3.3 Types and Schemata

Suppose that $\mathbf{B}$ is a countable set of base-type designators (denoted by $B, B_1, B_2, \ldots$) that contains $\mathcal{D}$, and let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathbf{D}}$ such that for all $d \in \mathbf{B}$: $\mathcal{I}_{\mathbf{B}}[d] = \{d\}$. Hereafter we will adopt as base-type system: $\mathbf{B} = \{\texttt{Int}, \texttt{String}, \texttt{Bool}, \texttt{Real}, i_1..j_1, i_2..j_2, \ldots, d_1, d_2, \ldots\}$, where the d_k's denote all the elements of $\texttt{Int} \cup \texttt{String} \cup \texttt{Bool} \cup \texttt{Real}$ and the $i_k..j_k$'s denote all possible ranges of integers (i_k can be $-\infty$ to denote the minimum element of Int and j_k can be $+\infty$ to denote the maximum element of Int).

Supposing that $\mathbf{N}$ is a countable set of *type names* (denoted by $N, N_1, N_2, \ldots$), such that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{N}$ are pairwise disjoint. $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ denotes the set of all *finite type descriptions* (denoted by $S, S_1, S_2, \ldots$), also briefly called *types*, over given $\mathbf{A}, \mathbf{B}, \mathbf{N}$, obtained according to the following abstract syntax rule, where $a_i \neq a_j$ for $i \neq j$ and p is a path:

$$\begin{aligned} S \rightarrow\ & \top \mid \bot \mid B \mid N \mid S_1 \sqcap S_2 \\ & \mid \forall\{S\} \mid \exists\{S\} \mid [a_1 : S_1, \ldots, a_k : S_k] \mid \triangle S \\ & \mid (p : S) \end{aligned}$$

$\top$ denotes the *top type*, $\bot$ denotes the *empty type*, $\forall\{\}$ and $[\ ]$ denote the usual type constructors of set and record (tuple), respectively. The $\exists\{S\}$ construct is an existential set specification, where at least one element of the set must be of type S. The construct $\sqcap$ stands for *intersection*, whereas $\triangle$ is an object set forming constructor. The type $(p\colon S)$ is called *path type*.

Given a set of type descriptions $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, a *database schema* over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ is a couple $\Sigma = (\sigma, \mathbf{R})$ where

—σ is a total function $\sigma\colon \mathbf{N} \rightarrow \mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, which associates type names to descriptions. σ is divided into two functions: σ_P, which introduces the description of *primitive type names* ($\mathbf{P}$), whose extensions must be explicitly provided by the user; and σ_V, which introduces the description of *virtual type names* ($\mathbf{V}$), whose extensions can be recursively computed from the extension of the types occurring in their description.

—$\mathbf{R}$, called *rule schema*, is a subset of the cartesian product $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}) \times \mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$. An element $R = (S^a, S^c)$ of $\mathbf{R}$ expresses a rule where S^a is the antecedent and S^c the consequent.

Table V. Part of the Company Domain Schema in ODL_{RE} Syntax

$$
\sigma_P \begin{cases} \sigma_P(\texttt{Level}) = [\texttt{class: String}, \texttt{parameter: 1..5}] \\ \sigma_P(\texttt{Employee}) = \triangle[\texttt{name: String}, \\ \qquad \texttt{qualification: 1..10}, \texttt{salary: 10..100}, \texttt{level: Level}, \\ \qquad \texttt{head: Employee}, \texttt{worksin: Department}, \texttt{skill: } \forall\{\texttt{String}\}] \\ \sigma_P(\texttt{Manager}) = \texttt{Employee} \sqcap \triangle[\texttt{qualification: 8..10}, \texttt{head: Manager}, \\ \qquad \texttt{directs: Department}] \end{cases}
$$

$$
\sigma_V \begin{cases} \sigma_V(\texttt{Mdl\_Employee}) = \texttt{Employee} \sqcap \triangle[\texttt{qualification: 5..10}] \\ \sigma_V(\texttt{Clerk}) = \texttt{Employee} \sqcap \triangle[\texttt{qualification: 7..10}, \texttt{head: Clerk}] \end{cases}
$$

$$
\mathbf{R} \left\{ R_3 = (\texttt{Employee} \sqcap \triangle[\texttt{salary: 60..}\infty] \sqcap (\triangle.\texttt{head}.\ \triangle\ .\texttt{qualification: 7..}\infty), \texttt{Manager}) \right.
$$

Table VI. Query Q1 in ODL_{RE} Syntax

$$
\sigma_V(Q1) = \texttt{Employee} \sqcap \triangle[\texttt{salary: 80..}\infty] \sqcap (\triangle.\texttt{head}.(\triangle.\texttt{head})^*.\ \triangle\ .\texttt{qualification: 7..}\infty)
$$

Part of the schema of the Company Domain example of Table I is translated into the database ODL_{RE} schema specified in Table V. A query has the semantics of a virtual type name and may include references to other virtual classes in its description: query Q1 in ODL_{RE} syntax is presented in Table VI.

Notice how the intersection operator ($\sqcap$) constructs a type that satisfies all the constraints of its operand types. In particular, it can be used to express inheritance in a name description. Formally, N_1 *inherits directly from* N_2, iff $\sigma(N_1) = S_1 \sqcap \cdots \sqcap S_n$, $n > 0$, and $N_2 = S_i$ for some i, $1 \leq i \leq n$.

Notice that the $\triangle$ operator allows the distinction of object types from value types. *Class descriptions* are preceded by $\triangle$ (as in the case of `Employee`), while descriptions without $\triangle$ are related to relation names (as in the case of `Level`). For instance, when we define a new specialized class, its differential attributes must be preceded by $\triangle$, as for `Manager`. Notice also that the $\triangle$ symbol is used both as a path element and as a class constructor. The overloading matches the intuition that $(\triangle\,.\,\texttt{salary: 60..}\infty)$ is semantically equivalent to $\triangle[\texttt{salary} : 60..\infty]$ (see the definition of interpretation function below).

Cyclic type names are permitted, in fact, since a type name may appear in type descriptions, we can have *circular references*, that is, type names which make direct or indirect references to themselves. Moreover, also *cyclic rules* are permitted, as a type name may appear both in the antecedent and in the consequent of a rule.

Formally, cyclic type names are recognized by means of the notion of *dependence*: N_1 *depends on* N_2, written $N_1 \hookrightarrow N_2$, where $N_1, N_2 \in \mathbf{N}$, if:

—N_2 is contained in the expression defining N_1, $\sigma(N_1)$, or
—there exists a rule $R = (S^a, S^c)$ such that N_1 is contained in S^a and N_2 is contained in S^c.

Table VII. Objects of a Company Domain *Possible* Instance

$\mathcal{O}$	=	$\{o_1, o_2, o_3, o_4, o_5, o_6\}$
$\delta(o_1)$	=	[dname: *"Administration"*, category : 5, employs: $\{o_3, o_4, o_5\}$, administrator: o_5]
$\delta(o_2)$	=	[dname: *"Development"*, category: 6, employs: $\{o_6\}$, administrator: o_6]
$\delta(o_3)$	=	[name: *"Robert"*, qualification: 8, salary: 43, head: o_5, worksin: o_1, skill: { *"prg"*}]
$\delta(o_4)$	=	[name: *"Mark"*, qualification: 4, salary: 32, head: o_5, worksin: o_1, skill: { *"wp"*}]
$\delta(o_5)$	=	[name: *"Franz"*, qualification: 8, salary: 43, head: o_5, worksin: o_1, skill: { *"prg"*}]
$\delta(o_6)$	=	[name: *"Andy"*, qualification: 9, salary: 67, head: o_6, worksin: o_2, skill: { *"wp"*, *"prg"*}]

The transitive closure of $\hookrightarrow$ is denoted by $\overset{+}{\hookrightarrow}$. We say that $N \in \mathbf{N}$ is *cyclic*, iff $N \overset{+}{\hookrightarrow} N$.

3.4 Interpretations and Database Instances

In the following, we will write $\mathbf{S}$ instead of $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ when the components are obvious from the context.

Let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathcal{D}}$. For a given object assignment δ, each type expression S is mapped into a set of values (its interpretation). An *interpretation function* is a function $\mathcal{I}$ from $\mathbf{S}$ to $2^{\mathcal{V}}$ satisfying the following equations:

$$
\begin{aligned}
\mathcal{I}[\top] &= \mathcal{V} \\
\mathcal{I}[\bot] &= \emptyset \\
\mathcal{I}[B] &= \mathcal{I}_{\mathbf{B}}[B] \\
\mathcal{I}[\forall\{S\}] &= \{M \mid M \subseteq \mathcal{I}[S]\} \\
\mathcal{I}[\exists\{S\}] &= \{M \mid M \cap \mathcal{I}[S] \neq \emptyset\} \\
\mathcal{I}[[a_1 : S_1, \ldots, a_p : S_p]] &= \{[a_1{:}v_1, \ldots, a_q{:}v_q] \mid p \leq q, v_i \in \mathcal{I}[S_i], 0 \leq i \leq p, \\
&\qquad\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\} \\
\mathcal{I}[S_1 \sqcap S_2] &= \mathcal{I}[S_1] \cap \mathcal{I}[S_2] \\
\mathcal{I}[\triangle S] &= \{o \in \mathcal{O} \mid \delta(o) \in \mathcal{I}[S]\} \\
\mathcal{I}[(p{:}S)] &= \bigcap \mathcal{I}[(w_i{:}S)] \text{ for all words } w_i \in L(p)
\end{aligned}
$$

where $\mathcal{I}[(w_i{:}S)]$ is inductively defined as follows:

$$
\begin{aligned}
\mathcal{I}[(\epsilon{:}S)] &= \mathcal{I}[S] \\
\mathcal{I}[(a\,.w{:}S)] &= \mathcal{I}[[a{:}(w{:}S)]] \\
\mathcal{I}[(\triangle\,.w{:}S)] &= \mathcal{I}[\triangle(w{:}S)] \\
\mathcal{I}[(\forall\,.w{:}S)] &= \mathcal{I}[\forall\{(w{:}S)\}] \\
\mathcal{I}[(\exists\,.w{:}S)] &= \mathcal{I}[\exists\{(w{:}S)\}]
\end{aligned}
$$

Table VII shows some examples of objects, with reference to the type descriptions of Table V.

Table VIII. Examples of Type Interpretations

$\mathcal{I}[(\triangle.\texttt{employs}.\exists.\ \triangle.\texttt{skill}.\forall\texttt{:}\ \textit{"wp"})]$	=	$\{o_1\}$
$\mathcal{I}[(\triangle.\texttt{employs}.\forall.\ \triangle.\texttt{skill}.\exists\texttt{:}\ \textit{"wp"})]$	=	$\{o_2\}$
$\mathcal{I}[(\triangle.\texttt{employs}.\exists.\ \triangle.\texttt{skill}.\exists\texttt{:}\ \textit{"wp"})]$	=	$\{o_1, o_2\}$
$\mathcal{I}[(\triangle.\texttt{head}.\ \triangle.\texttt{qualification}\texttt{:}8)]$	=	$\{o_3, o_4, o_5\}$
$\mathcal{I}[((\triangle.\texttt{head})^*.\ \triangle.\texttt{qualification}\texttt{:}8)]$	=	$\{o_3, o_5\}$

Note that the interpretation of tuples implies an open world semantics for tuple types similar to that adopted by Cardelli [1984]:

$$[\texttt{name}\texttt{:}\ \textit{"Mark"}, \texttt{salary}\texttt{:}32]\ \in\ \mathcal{I}[[\texttt{salary}\texttt{:}\ \texttt{Int}]] = \mathcal{I}[(\texttt{salary}\texttt{:}\ \texttt{Int})]$$

Table VIII shows some examples of path type interpretations, with reference to the objects of Table VII.

A database schema defines a set of possible interpretations that are derived from type interpretations by adding the constraints of name definitions and integrity rules. A name definition relates the interpretation of the name to that of its associated type description, while a rule states an inclusion relation between the S^a and S^c type interpretations.

Definition 1 (*Possible Instance*). Given a database schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, a set of object identifiers $\mathcal{O}$, and a domain δ over $\mathcal{O}$, an interpretation function $\mathcal{I}$ of $\mathbf{S}$ over δ is a *possible instance* of Σ iff the set $\mathcal{O}$ is *finite* and

(1) $\mathcal{I}[P] \subseteq \mathcal{I}[\sigma_P(P)]$, if $P \in \mathbf{P}$.
(2) $\mathcal{I}[V] = \mathcal{I}[\sigma_V(V)]$, if $V \in \mathbf{V}$.
(3) $\mathcal{I}[S^a] \subseteq \mathcal{I}[S^c]$, if $R = (S^a, S^c) \in \mathbf{R}$.

From the above illustrated definition, we can see that the interpretation of a primitive type name *is included* in the interpretation of its description, while the interpretation of a virtual type *is* the interpretation of its description. In other words, the interpretation of a primitive type name must be provided by the user, according to the given description, while the interpretation of a virtual type name is computed from its definition and from the interpretation of primitive type names, thus corresponding to a view in a database context. Furthermore, rules are inclusion statements, with the usual semantics as in Donini et al. [1993] and Calvanese et al. [1998].

3.5 Legal Database Instances

From the definition of Possible Instance of a database schema the rule (N, S) is consequently equivalent to $\sigma_P(N) = S$; the pair of rules (N, S) and (S, N) is equivalent to $\sigma_V(N) = S$. Nevertheless, we introduce σ_P and σ_V for a "deep" semantic motivation: we need to uniquely define the extension of a cyclic virtual class. The interpretation of a cyclic virtual name N is different if N is introduced in the schema with a $\sigma_V(N) = S$ definition, or, by means of the pair of rules (N, S) and (S, N). Let's explain the difference by considering the objects of Table VII and the following assigned interpretation of the classes `Department`

and Employee:

$$\mathcal{I}[\texttt{Department}] = \{\ o_1, o_2\ \}, \quad \mathcal{I}[\texttt{Employee}] = \{\ o_3, o_4, o_5, o_6\ \}$$

The instance of the acyclic virtual class Mdl_Employee can be uniquely computed as $\mathcal{I}[\texttt{Mdl\_Employee}] = \{o_3, o_5, o_6\}$ (o_4 does not satisfy the qualification value range). On the other hand, we may have many possible instances for the cyclic virtual class Clerk: $\mathcal{I}[\texttt{Clerk}] = \{o_3, o_5, o_6\}$, or $\mathcal{I}[\texttt{Clerk}] = \{o_3, o_5\}$, or $\mathcal{I}[\texttt{Clerk}] = \{o_6\}$, or $\mathcal{I}[\texttt{Clerk}] = \{\}$, either introducing Clerk by rules or by a σ_V definition.

In order to express definitions as $\sigma_V(N) = S$ where S is a "function" of N, that is, N appears in S, *we need to adopt a fixed point semantics*, either least fixed point (lfp) or greatest fixed point (gfp). Descriptive semantics is the standard first-order semantics and interprets statements just restricting the set of possible models, with no definitional import; hence, it can be suitably adopted for rules interpretations. Taking our example again, if the class Clerk is introduced by rules, the four interpretations above are all possible.

Between lfp and gfp we select gfp, since it corresponds to the semantics of the transitive closure of an attribute.[5] Adopting gfp-semantics, in the above example, we have:

$$\mathcal{I}[\texttt{Clerk}] = \mathcal{I}[\texttt{Employee} \sqcap ((\triangle.\texttt{head})^*.\ \triangle\ .\texttt{qualification}\colon 7..10)] = \{o_3, o_5, o_6\}$$

Let's introduce the notion of *legal instance* of a schema, which makes it possible to interpret cyclic virtual names defined by σ_V under gfp-semantics and cyclic virtual names defined in $\mathbf{R}$ under descriptive semantics.

Definition 2 (*Legal Instance*). Given a $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, let Ψ be the *set of possible instances with identical $\mathcal{O}$ and δ* such that for all $\mathcal{I}, \mathcal{I}' \in \Psi$: $\mathcal{I}[P] = \mathcal{I}'[P]$ if $P \in \mathbf{P}$. Further, let "$\overset{\Psi}{\sqsubseteq}$" be the relation over Ψ such that for all $\mathcal{I}, \mathcal{I}' \in \Psi$: $\mathcal{I} \overset{\Psi}{\sqsubseteq} \mathcal{I}'$ iff $\mathcal{I}[V] \subseteq \mathcal{I}'[V]$ for all $V \in \mathbf{V}$. Then $(\Psi, \overset{\Psi}{\sqsubseteq})$ forms a partial ordering. We say that $\mathcal{I}$ is a *legal instance* of a database schema Σ iff it is the *greatest* instance of the set Ψ with respect to $\overset{\Psi}{\sqsubseteq}$.

THEOREM 1. *If $\mathcal{I}$ is a possible instance, then a legal instance $\mathcal{I}'$ exists such that $\mathcal{I} \overset{\Psi}{\sqsubseteq} \mathcal{I}'$.*

PROOF. See Bergamaschi and Nebel [1994]. □

4. SUBSUMPTION, INCOHERENCE AND SEMANTIC EXPANSION OF A TYPE

Semantic expansion of a type enables us to incorporate possible restrictions that are not present in the original type but that are *logically implied* by the type and by the schema. Our method for the computation of a semantic expansion of a type is based on two ingredients: iteration of the transformation "if a type implies the antecedent of a rule, then the consequent of that rule can be intersected to the type" [Shenoy and Özsoyoglu 1987, 1989]; evaluation of logical implications by means of *subsumption computation* between such expanded types.

[5] An in-depth discussion of different semantics is given in Beneventano and Bergamaschi [1997].

Definition 3 (*Subsumption*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, the *subsumption relation* with respect to Σ, written $S \sqsubseteq_\Sigma S'$ for each pair of types $S, S' \in \mathbf{S}$, is $S \sqsubseteq_\Sigma S'$ iff $\mathcal{I}[S] \subseteq \mathcal{I}[S']$ for all legal instances $\mathcal{I}$ of Σ.

It immediately follows that $\sqsubseteq_\Sigma$ is a preorder (i.e., transitive and reflexive but antisymmetric) that induces an *equivalence relation* (Σ-*equivalence*) $\simeq_\Sigma$ on types: $S \simeq_\Sigma S'$ iff $S \sqsubseteq_\Sigma S'$ and $S' \sqsubseteq_\Sigma S$. For a type S, $[S]_\Sigma$ denotes the equivalence class with respect to the $\simeq_\Sigma$ relation.

Definition 4 (*Incoherence*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, we say that a type S is *incoherent* with respect to Σ iff $S \simeq_\Sigma \bot$. It is equivalent to say that for all domains the extension of an incoherent type is always empty.

Given a database schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, let $\Sigma_\sigma = (\sigma, \emptyset)$ be the subschema with no rules; the subsumption and equivalence relations with respect to the subschema Σ_σ will be denoted by $\sqsubseteq_\sigma$ and $\simeq_\sigma$, respectively.

Of course, the subsumption (equivalence) relations with respect to the subschema Σ_σ implies the subsumption (equivalence) relations with respect to the schema $\Sigma = (\sigma, \mathbf{R})$, as in Σ there are more formulas/rules. This is expressed in Theorem 2.

THEOREM 2. *Given a schema* $\Sigma = (\sigma, \mathbf{R})$ *over* $\mathbf{S}$*, for all* $S, S' \in \mathbf{S}$ *we have* $S \sqsubseteq_\Sigma S'$ *if* $S \sqsubseteq_\sigma S'$.

Definition 5 (*Applicable Rule*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, we say that a rule $R = (S^a, S^c) \in \mathbf{R}$ is *applicable* to $S \in \mathbf{S}$ with respect to σ if $S \sqsubseteq_\sigma S^a$ and $S \not\sqsubseteq_\sigma S^c$. The set of rules applicable to S with respect to σ will be denoted by $\Lambda_\sigma(S)$.

On the basis of the *applicable rule* notion, we define the notion of *semantic expansion* of a type. Intuitively, a semantic expansion of a type S is a type S' equivalent to S so that no rule is applicable to S' or to any type *used in* S'.

Definition 6 (*Semantic Expansion*). Let $\Sigma = (\sigma, \mathbf{R})$ be a schema over $\mathbf{S}$ and let S be a type in $\mathbf{S}$. A type $S' \in [S]_\Sigma$ is a *semantic expansion* of S with respect to Σ (for short $S' = EXP_\Sigma(S)$) iff $S' \sqsubseteq_\sigma S$ and $\Lambda_\sigma(S'') = \emptyset$, for each $S'' \in \mathbf{S}$ such that a word w and $S' \sqsubseteq_\sigma (w : S'')$ exists.

Intuitively, from rule semantics it follows that, if a rule $R = (S^a, S^c)$ is applicable to S, then $S \sqcap S^c \simeq_\Sigma S$. Thus, if R is applicable to S, we can transform S into the equivalent type $S \sqcap S^c$. This transformation is the core of our algorithm for the computation of a semantic expansion $EXP_\Sigma(S)$ which is an iterative process applying rules to a type and terminating when no more rules are applicable.

Note that $EXP_\Sigma(S)$ is not guaranteed to be the least element, that is, it is possible that a type $S' \simeq_\Sigma S$ exists such that $S' \sqsubseteq_\sigma EXP_\Sigma(S)$. For example, in the following schema

$$\sigma_P \begin{cases} \sigma_P(V_1) = [\mathrm{a}: \mathtt{Int}] \\ \sigma_P(V_2) = [\mathrm{a}: \mathtt{Int}] \end{cases} \qquad \mathbf{R} \begin{cases} R_1 = (V_1 \sqcap [\mathrm{a}: -\infty..0], V_2) \\ R_2 = (V_1 \sqcap [\mathrm{a}: 1..\infty], V_2) \end{cases}$$

a semantic expansion of V_1 is V_1 itself, that is, $EXP_\Sigma(V_1) = V_1$, as no rule is applicable to V_1. But, since $V_1 \sqsubseteq_\Sigma V_2$, the type $S' = V_1 \sqcap V_2$ is consequently equivalent to V_1 and is more specialized than $EXP_\Sigma(V_1) = V_1$ with respect to the subschema with no rules, that is, $S' \sqsubseteq_\sigma EXP_\Sigma(V_1)$.

More generally speaking, it is not guaranteed that a semantic expansion exists, for an arbitrary *DL*. To give an intuition, let's consider the rule $R = (S, (a\!:\!S))$: it is easy to see that S is equivalent to $S \sqcap (a\,.\,a \cdots a\!:\!S)$ for chains of attributes of arbitrary length. In our *DL* such an infinite structure can be described with a finite type description by using a path expression and then, intuitively, $EXP_\Sigma(S) = ((a)^*\!:\!S)$. Furthermore, as we adopt the gfp-semantics, $EXP_\Sigma(S)$ is expressed in an equivalent form, by the cyclic virtual name V, with $\sigma_V(V) = S \sqcap [a\!:\!V]$: this is the form computed by our semantic expansion algorithm (see next section).

4.1 Algorithms

In Bergamaschi and Nebel [1994], a framework including algorithms for detecting incoherence and computing subsumption between types of an ODL schema without rules, adopting a gfp-semantics, has been introduced. In the stated paper, it has been demonstrated that these problems are computationally intractable from a purely theoretical point of view, but intractability does not very often show up in practice. Furthermore, if algorithms on a *simplified schema description* (named *canonical schema*) are provided, both problems can be solved in time polynomial in the size of the canonical schema.

In order to extend this framework to a schema with integrity rules and to compute a semantic expansion of types, we proceed as follows:

(1) transform any path type into an equivalent ODL type and the overall schema into a *canonical schema*; thus we can use the algorithm given in Bergamaschi and Nebel [1994] to compute the subsumption relation $\sqsubseteq_\sigma$ in Σ_σ (see Section 4.1.1);
(2) provide an algorithm for semantic expansion computation (see Section 4.1.2);
(3) compute subsumption relations $\sqsubseteq_\Sigma$ among types by using the semantic expansion of types and the subsumption relation $\sqsubseteq_\sigma$ in Σ_σ (see Section 4.1.2).

4.1.1 *Canonical Schema Generation.* First of all, we show that, by adopting gfp-semantics, any schema can be transformed into a (possibly cyclic) schema without path types. We introduce the definition of *conservative extension* of a schema which will be guaranteed for any schema transformation.

Definition 7 (*Conservative Extension*). A schema $\Sigma_2 = (\sigma_2, \mathbf{R}_2)$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_2)$ is a *conservative extension* of a schema $\Sigma_1 = (\sigma_1, \mathbf{R}_1)$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_1)$ iff $\mathbf{R}_2 = \mathbf{R}_1$, $\mathbf{N}_1 \subseteq \mathbf{N}_2$ and for any legal instance $\mathcal{I}$ of Σ_1 a legal instance $\mathcal{I}'$ of Σ_2 exists and vice-versa such that $\mathcal{I}[N] = \mathcal{I}'[N]$ for all $N \in \mathbf{N}_1$.

Note that this means that for the names in $\mathbf{N}_1$, the incoherencies and subsumptions computed with respect to Σ_1 *are the same as the ones* computed with respect to Σ_2.

Table IX. Query Q1 in ODL_{RE} Without Path Types

$\sigma_V(Q1)$	=	$\texttt{E} \sqcap \triangle[\texttt{s}: 80..\infty] \sqcap \triangle[\texttt{h}: NQ1]$
$\sigma_V(NQ1)$	=	$\triangle[\texttt{q}: 7..\infty] \sqcap \triangle[\texttt{h}: NQ1]$

PROPOSITION 1. *Any schema* Σ_1 *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_1)$ *can be effectively transformed into a schema* Σ_2 *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_2)$ *without path types that is a conservative extension of* Σ_1.

PROOF SKETCH. The transformation rules are the following:

$$(\epsilon: S) \longrightarrow S \tag{1}$$

$$(a: S) \longrightarrow [a: S] \tag{2}$$

$$(\triangle: S) \longrightarrow \triangle S \tag{3}$$

$$(\forall: S) \longrightarrow \forall\{S\} \tag{4}$$

$$(\exists: S) \longrightarrow \exists\{S\} \tag{5}$$

$$(p\,.\,p': S) \longrightarrow (p: (p': S)) \tag{6}$$

$$((p)^*: S) \longrightarrow V, \tag{7}$$

where V is a new virtual type name and $\sigma_V(V) = S \sqcap (p: V)$ is the extension of the schema function to V

$$((p + p'): S) \longrightarrow (p: S) \sqcap (p': S) \tag{8}$$

Note that Σ_2 is distinguished from Σ_1 by the additional virtual type names and the additional values for σ introduced by the application of Rule 7. Transformations (1)–(6) and transformation (8) are obvious. Transformation (7) translates a factor including Kleene closure of a path in a cyclic virtual type name; the equivalence is implied by the gfp-semantics adopted in ODL_{RE} and derives from a result of Baader [1991]. The applicability of Baader's results in our context is proved in Beneventano [2002]. □

As an example, let's consider query Q1. For the sake of simplicity, the examples refer to the schema and queries of Section 2.2, but the names are reduced to the initials.

$$\sigma_V(Q1) = \texttt{E} \sqcap \triangle[\texttt{s}: 80..\infty] \sqcap (\triangle.\texttt{h}.(\triangle.\texttt{h})^*.\ \triangle\ .\texttt{q}: 7..\infty).$$

First, applying rules from (2) to (6), we obtain:

$$\sigma_V(Q1) = \texttt{E} \sqcap \triangle[\texttt{s}: 80..\infty] \sqcap \triangle[\texttt{h}: \underbrace{((\triangle.\texttt{h})^*: \triangle[\texttt{q}: 7..\infty])}_{NQ1}],$$

then, a cyclic virtual type name $NQ1$ is introduced by applying rule (7):

$$\sigma_V(NQ1) = \triangle[\texttt{q}: 7..\infty] \sqcap \triangle[\texttt{h}: NQ1].$$

A *canonical schema* is a schema where types are unnested, conjunctions of types are eliminated, whenever possible[6] and each class description contains

[6]Only type conjunctions involving an existentially quantified term cannot be eliminated.

the conjunction of primitive names (without description) and a canonical type description. Furthermore, see syntax below, virtual type names are introduced. Obviously, in a canonical schema the comparison between types to determine the applicability of a rule is greatly simplified. Formally:

Definition 8 (*Canonical Schema*). A canonical schema is a schema $\Sigma = (\sigma, \mathbf{R})$ where:

—for all $P \in \mathbf{P}$, $\sigma_P(P) = \top$, that is, primitive type names have no descriptions; we will refer to these primitive names as *atomic names*, denoted by $\overline{P}$;
—for all $V \in \mathbf{V}$, $\sigma_V(V) = \overline{P}_1 \sqcap \cdots \sqcap \overline{P}_n \sqcap S$ (with $n \geq 0$), where S is a *canonical* type obtained according to the following abstract syntax rule:

$$S \rightarrow V \mid B \mid \top \mid \bot \mid [a_1 : V_1, \ldots, a_k : V_k] \mid \forall\{V\} \mid \exists\{V_1\} \sqcap \cdots \sqcap \exists\{V_n\} \sqcap \forall\{V\} \mid \triangle V, \quad \text{with } V \in \mathbf{V}$$

where for the type $\exists\{V_1\} \sqcap \cdots \sqcap \exists\{V_n\} \sqcap \forall\{V\}$ we have that $V_i \sqsubseteq_\sigma V, \forall i, 1 \leq i \leq n$.
—for all $R = (S^a, S^c) \in \mathbf{R}$, $S^a, S^c \in \mathbf{V}$, that is, antecedent and consequent of a rule are virtual type names.

The main points of the transformation of a schema σ into a canonical schema are:

(a) Each primitive type name P of the schema σ is transformed into a virtual type name expressed as the conjunction of an atomic name, $\overline{P}$, and its original description, $\sigma(P)$;
(b) Each rule $R = (S^a, S^c) \in \mathbf{R}$, is transformed into a pair of virtual names VR^a and VR^c, with $\sigma_V(VR^a) = S^a$ and $\sigma_V(VR^c) = S^c$.

while (there is a virtual name V such that $\sigma_V(V)$ is not in canonical form) the canonical description of V with respect to σ is obtained by:

1. (*name substitution*). Substituting only the names of the types from which V inherits with their descriptions;
2. (*conjunction elimination*). Applying suitable conjunction rules, the conjunctions of types are eliminated (whenever possible);
3. (*type renaming*). Assigning a new virtual name V' to each new type S obtained at Step (**2**), if a V'' with $\sigma_V(V'') = S$ does not exist.

As an example of canonical schema, we consider the computation of the canonical form of `Employee` and `Manager` (some attributes are omitted and all the names are denoted with the initials):

$$\sigma_P(\mathtt{E}) = \triangle[\mathtt{q}: 1..10, \mathtt{s}: 10..100, \mathtt{h}: \mathtt{E}]$$
$$\sigma_P(\mathtt{M}) = \mathtt{E} \sqcap \triangle[\mathtt{q}: 8..10, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$$

`E` and `M` are primitive class names thus they are transformed applying Step (**a**):

$$\sigma_V(\mathtt{E}) = \overline{\mathtt{E}} \sqcap \triangle[\mathtt{q}: 1..10, \mathtt{s}: 10..100, \mathtt{h}: \mathtt{E}]$$
$$\sigma_V(\mathtt{M}) = \overline{\mathtt{M}} \sqcap \mathtt{E} \sqcap \triangle[\mathtt{q}: 8..10, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$$

Table X. Query Q1 in Canonical Form

$\overline{\sigma}_V(Q1)$	$=$	$\overline{\mathtt{E}} \sqcap \triangle V_5$	$\overline{\sigma}_V(V_5) = [\mathtt{q}: 7..10, \mathtt{s}: 80..100, \mathtt{h}: V_6]$
$\overline{\sigma}_V(V_6)$	$=$	$\overline{\mathtt{E}} \sqcap \triangle V_7$	$\overline{\sigma}_V(V_7) = [\mathtt{q}: 7..10, \mathtt{s}: 10..100, \mathtt{h}: V_6]$

The canonical description of E is obtained simply by introducing a new virtual name V_1 (Step (**3**)):

$$\begin{aligned} \sigma_V(\mathtt{E}) &= \overline{\mathtt{E}} \sqcap \triangle V_1 \\ \sigma_V(V_1) &= [\mathtt{q}: 1..10, \mathtt{s}: 10..100, \mathtt{h}: \mathtt{E}] \end{aligned}$$

As M inherits from E, it is transformed applying Step (**1**):

$$\sigma_V(\mathtt{M}) = \overline{\mathtt{M}} \sqcap \overline{\mathtt{E}} \sqcap \triangle[\mathtt{q}: 1..10, \mathtt{s}: 10..100, \mathtt{h}: \mathtt{E}] \sqcap \triangle[\mathtt{q}: 8..10, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$$

then the conjunction of types are solved (Step (**2**))—for the attribute q, $1..10 \sqcap 8..10 = 8..10$ and for the attribute h, $\mathtt{E} \sqcap \mathtt{M} = \mathtt{M}$ since M inherits from E:

$$\sigma_V(\mathtt{M}) = \overline{\mathtt{M}} \sqcap \overline{\mathtt{E}} \sqcap \triangle[\mathtt{q}: 8..10, \mathtt{s}: 10..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$$

and then a new virtual name V_2 is introduced, obtaining the canonical description of M:

$$\sigma_V(\mathtt{M}) = \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_2 \qquad \sigma_V(V_2) = [\mathtt{q}: 8..10, \mathtt{s}: 10..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$$

It can be easily proved that these transformations preserve type interpretations.

PROPOSITION 2. *Any schema* Σ_1 *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_1)$ *without path types can be transformed into a canonical schema* Σ_2 *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N}_2)$ *which is a conservative extension of* Σ_1.

In the following, a canonical form of a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ will be denoted by $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$, over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{N}})$, with $\overline{\mathbf{N}} = \overline{\mathbf{P}} \cup \overline{\mathbf{V}}$. Note that: each name $N \in \mathbf{N}$ is, in $\overline{\Sigma}$, a virtual name $N \in \overline{\mathbf{V}}$ and to each type S in Σ we assign in $\overline{\Sigma}$ a new virtual name. The canonical form of query Q1 is given in Table X.

4.1.2 *Semantic Expansion Computation.* Thanks to Propositions 1 and 2, a semantic expansion of types of a schema $\Sigma = (\sigma, \mathbf{R})$ can be computed with respect to a canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ of Σ. In the following, we introduce a semantic expansion algorithm that, given a canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ as input, computes a semantic expansion for each name of $\overline{\Sigma}$ on the basis of its set of *applicable rules* (see Definition 5).

In order to formally define the correspondence between the input and output of the semantic expansion algorithm, we introduce the concept of *expanded schema*, that is, a schema where there are no rules applicable to any virtual type names.

Definition 9 (*Expanded Schema*). A canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$, is called *expanded* if $\Lambda_{\overline{\Sigma}}(V) = \emptyset$, $\forall V \in \overline{\mathbf{V}}$.

Algorithm 1 (Semantic Expansion).

Input: A canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$.
Output: An expanded schema $\widetilde{\Sigma} = (\widetilde{\sigma}, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}})$.

set. $i = 0$, $\mathbf{V}^i = \overline{\mathbf{V}}$ and $\sigma^i = \overline{\sigma}$

while. there is a name $V \in \mathbf{V}^i$ and a rule $R = (VR^a, VR^c)$ such that $R \in \Lambda_{\Sigma^i}(V)$, where $\Sigma^i = (\sigma^i, \overline{\mathbf{R}})$

choose V and R
define. the σ^{i+1} over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \mathbf{V}^{i+1})$ in the following way:

(1) assign to $\sigma_V^{i+1}(V)$ the canonical form of $\sigma_V^i(V) \sqcap \sigma_V^i(VR^c)$ obtained in the following way: for any new type S, a new name V' is introduced in σ^{i+1}, with $\sigma_V^{i+1}(V') = S$ if there is no j, $(0 \leq j \leq i)$ and V'' such that $\sigma_V^j(V'') = S$.
(2) $\mathbf{V}^{i+1} = \mathbf{V}^i \cup \{V \mid V$ is a new name introduced in Step (1)$\}$

set. $\widetilde{\sigma} = \sigma^i$ and $\widetilde{\mathbf{V}} = \mathbf{V}^i$.

THEOREM 3. *Given a canonical schema* $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$ *as input, the output of algorithm* 1 *is an expanded schema* $\widetilde{\Sigma} = (\widetilde{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}})$ *which is a conservative extension of* $\overline{\Sigma}$.

PROOF. See Appendix C. □

Intuitively, for each $V \in \overline{\mathbf{V}}$ of the schema $\overline{\Sigma}$, a semantic expansion $EXP_{\overline{\Sigma}}(V)$ is given by $\widetilde{\sigma}_V(V)$ in the expanded schema $\widetilde{\Sigma}$. The formal specification of this correspondence is more complex than the intuitive description given above, since $EXP_{\overline{\Sigma}}(V)$ must be expressed as a type of the schema $\overline{\Sigma}$.

COROLLARY 1. *Given a canonical schema* $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$ *and the extended schema* $\widetilde{\Sigma} = (\widetilde{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}})$ *computed by Algorithm* 1, *we denote with* $\overline{\mathbf{V}}_{exp}$ *a* copy *of the names in* $\overline{\mathbf{V}}$, *that is,* $\overline{\mathbf{V}}_{exp} = \{V_{exp} \mid V \in \overline{\mathbf{V}}\}$ *and we define the* union schema $\overline{\Sigma}_u = (\overline{\sigma}_u, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}}_u)$ *as follows:* $\overline{\mathbf{V}}_u = \overline{\mathbf{V}} \cup \overline{\mathbf{V}}_{exp} \cup \{V \mid V \in \widetilde{\mathbf{V}} \setminus \overline{\mathbf{V}}\}$ *and* $\overline{\sigma}_u(N) = \overline{\sigma}(N)$, *if* $N \in \overline{\mathbf{P}} \cup \overline{\mathbf{V}}$, $\overline{\sigma}_u(N) = \widetilde{\sigma}(N)$, *if* $N \in \overline{\mathbf{V}}_u \setminus \overline{\mathbf{V}}$. *Then* $EXP_{\overline{\Sigma}_u}(V) = V_{exp}$, $\forall V \in \overline{\mathbf{V}}$.

PROOF SKETCH. As $\widetilde{\Sigma}$ is an extended schema then $EXP_{\overline{\Sigma}}(V_{exp}) = V_{exp}$. As $\widetilde{\Sigma}$ is a conservative extension of $\overline{\Sigma}$ (from Theorem 3), then $[V]_{\overline{\Sigma}} = [V_{exp}]_{\overline{\Sigma}}$ and consequently a semantic expansion of V_{exp} is a semantic expansion of V also. □

Moreover, if we consider two expanded schemata $\widetilde{\Sigma}' = (\widetilde{\sigma}', \overline{\mathbf{R}})$ and $\widetilde{\Sigma}'' = (\widetilde{\sigma}'', \overline{\mathbf{R}})$ computed by Algorithm 1 with different sequences of choices, then, for each $V \in \overline{\mathbf{V}}$, $\widetilde{\sigma}'_V(V)$ and $\widetilde{\sigma}''_V(V)$ are equivalent with respect to the subschema without rules. The following corollary specifies this result more formally.

COROLLARY 2. *Given a canonical schema* $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$ *and the expanded schemata* $\widetilde{\Sigma}' = (\widetilde{\sigma}', \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}}')$ *and* $\widetilde{\Sigma}'' = (\widetilde{\sigma}'', \mathbf{R})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}}'')$ *computed by Algorithm* 1, *for any legal instance* $\mathcal{I}'$ *of* $\widetilde{\Sigma}'_\emptyset$ *a legal instance* $\mathcal{I}''$ *of* $\widetilde{\Sigma}''_\emptyset$ *exists such that* $\mathcal{I}'[N] = \mathcal{I}''[N]$ *for all* $N \in \overline{\mathbf{V}}$.

Table XI. Query Q1 Expanded Form

$\widetilde{\sigma}_V(Q1) = \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_{11}$,	$\widetilde{\sigma}_V(V_{11}) = [\mathtt{q}: 8..10, \mathtt{s}: 80..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: V_{12}]$
$\widetilde{\sigma}_V(V_{12}) = \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_{13}$,	$\widetilde{\sigma}_V(V_{13}) = [\mathtt{q}: 8..10, \mathtt{s}: 10..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: V_{12}]$

PROOF SKETCH. Is an immediate consequence of the fact that $\widetilde{\Sigma}'$ and $\widetilde{\Sigma}''$ are two conservative extension of the same schema $\overline{\Sigma}$. □

As an example of semantic expansion computation, let's consider Q1 and its semantic expansion by applying rule R_3; the canonical form of the query is given in Table X. For rule R_3 the canonical form is:

$$\begin{aligned} \overline{\sigma}_V\left(VR_3^a\right) &= \overline{\mathtt{E}} \sqcap \triangle V_8 \quad \overline{\sigma}_V(V_8) = [\mathtt{q}: 1..10, \mathtt{s}: 60..100, \mathtt{h}: V_9] \\ \overline{\sigma}_V(V_9) &= \overline{\mathtt{E}} \sqcap \triangle V_{10} \quad \overline{\sigma}_V(V_{10}) = [\mathtt{q}: 7..10, \mathtt{s}: 60..100, \mathtt{h}: \mathtt{E}] \\ \overline{\sigma}_V\left(VR_3^c\right) &= \mathtt{M} \end{aligned}$$

For $i = 0$, as $R_3 \in \Lambda_{\Sigma^i}(Q1)$ we assign to $\sigma_V^1(Q1)$ the canonical form of $\sigma_V^0((Q1)) \sqcap \sigma_V^0(VR_3^c)$. Since $\sigma_V^0(VR_3^c) = \mathtt{M}$ and $\sigma_V^0(\mathtt{M}) = \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_2$, $\sigma_V^0(V_2) = [\mathtt{q}: 8..10, \mathtt{s}: 10..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: \mathtt{M}]$ we obtain

$$\begin{aligned} \sigma_V^1(Q1) &= \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_{11}, \quad \sigma_V^1(V_{11}) = [\mathtt{q}: 8..10, \mathtt{s}: 80..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: V_{12}] \\ \sigma_V^1(V_{12}) &= \overline{\mathtt{E}} \sqcap \overline{\mathtt{M}} \sqcap \triangle V_{13}, \quad \sigma_V^1(V_{13}) = [\mathtt{q}: 8..10, \mathtt{s}: 10..100, \mathtt{d}: \mathtt{D}, \mathtt{h}: V_{12}] \end{aligned}$$

No more rules are applicable to the names of this schema, then $\widetilde{\sigma} = \sigma^1$ is an expanded schema and then $\widetilde{\sigma}(Q1)$ is a semantic expansion of $Q1$.

The comparison between the query Q1 before (see Table X) and after (see Table XI) semantic expansion computation, shows how we address and solve the problem of finding *a specialized rewriting* of a query in terms of the schema classes and base types. In fact:

—the *target* class of the query, $\overline{\mathtt{E}}$, is specialized in $\overline{\mathtt{E}} \sqcap \overline{\mathtt{M}}$

—the *target* class of the subquery V_6 related to the h attribute, $\overline{\mathtt{E}}$, is specialized in $\overline{\mathtt{E}} \sqcap \overline{\mathtt{M}}$ (see $\widetilde{\sigma}_V(V_{12})$)

—the q attribute: $\mathtt{q}: 7..10$ (in $\overline{\sigma}_V(V_5)$) is specialized in $\mathtt{q}: 8..10$ (see $\widetilde{\sigma}_V(V_{11})$)

Let's summarize the method we developed to compute subsumption in a schema with rules. Starting from a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, we obtain a canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$, which is a conservative extension of Σ, and an expanded schema $\widetilde{\Sigma} = (\widetilde{\sigma}, \overline{\mathbf{R}})$, which is a conservative extension of $\overline{\Sigma}$. The subsumption relations for the names in $\mathbf{N}$ with respect to the three schemata, $\sqsubseteq_\Sigma$, $\sqsubseteq_{\overline{\Sigma}}$ and $\sqsubseteq_{\widetilde{\Sigma}}$, are the same. On the basis of Theorem 2, we provide a sound method to obtain the subsumption relation $\sqsubseteq_{\widetilde{\Sigma}}$ by computing the relation $\sqsubseteq_{\widetilde{\sigma}}$ with the subsumption algorithm introduced in Bergamaschi and Nebel [1994]. Note that we compute $\sqsubseteq_{\widetilde{\sigma}}$ since it is easy to prove that $\sqsubseteq_{\widetilde{\sigma}}$ imply $\sqsubseteq_{\overline{\sigma}}$ but not vice-versa.

5. OPTIMAL REWRITING OF A QUERY

The previous sections showed how to produce a specialized rewriting of a query where the schema constraints are included in the query and the query is rewritten with respect to more specialized classes and base types; here, we

will show a method to find out redundant/eliminable components of the specialized rewritten query in order to obtain an equivalent query, hopefully with a lower execution cost.

According to our type system a query is a virtual name, thus we can compute semantic expansion. First of all, we check the coherence of the expanded query $\widetilde{\sigma}(Q)$. If $\widetilde{\sigma}(Q)$ is incoherent, we get an optimization result since a null answer can be immediately returned without accessing the database. If $\widetilde{\sigma}(Q)$ is coherent, we apply our rewriting method.

The first step is to rewrite the query as a *conjunction of factors*, where a factor is a path type $(p\!:\!S)$, where p is a *union-free* regular path expression and S is a *primitive class* or a *base type*. For this factorization, we exploit the results of Baader [1991, 1996] and transform the factorization problem into the problem of finding the languages accepted by a finite automaton (see Theorem 4). Factorization of the query is computed with respect to the expanded schema in order to exploit the specialization of factors with respect to schema classes and base types. In particular, the specialization of a factor with a base type produces a more selective predicate that may be useful at execution time, either for indexed access or for the reduction of intermediate results.

The second step is to find out which factors of the query are redundant and thus eliminable from the query. To provide a simple method to independently eliminate redundant factors we need to check the absence of recursive rules in the semantic expansion of the query: the check is done by introducing rule graphs to formally define recursive rules in our context and using subsumption. The elimination of redundant factors generates an *optimized form* of the query, which is semantically equivalent to the original one. Our elimination criteria are based only on logical properties, and are independent from any specific cost model and storage details.

5.1 Factorization

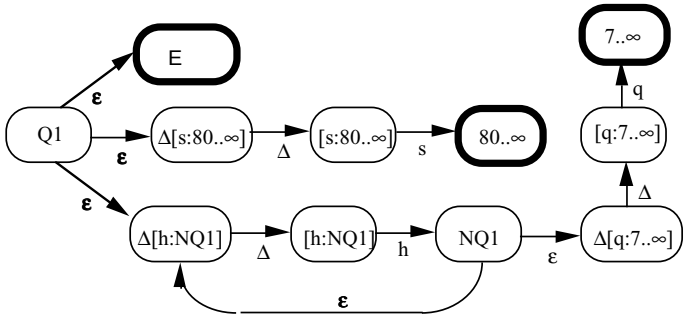
As a query is a virtual name, we generally formalize the method with respect to a virtual name V.

Definition 10 (*Factor*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$, a *factor* of $\mathbf{S}$ is a path type $(p\!:\!S)$ where $S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}$ and p is a *union-free* regular path expression. A factor will be denoted by f.

Definition 11 (*Factorization*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ and a virtual type name V, a finite set of factors $F = \{f_1, f_2, \ldots, f_n\}$ of $\mathbf{S}$ is a *factorization* of V with respect to Σ iff $\sqcap F \simeq_\Sigma V$.

We consider only *union-free* regular path expressions in a factor in order to simplify the elimination of redundant factors. For example, a type $[a_1\!:\![a_2\!:\!\texttt{Int}], a_3\!:\![a_2\!:\!\texttt{Int}]]$ can be expressed in two equivalent forms, say $(a_1.a_2\!:\!\texttt{Int}) \sqcap (a_3.a_2\!:\!\texttt{Int})$ or $((a_1 + a_3).a_2\!:\!\texttt{Int})$; obviously the first form is more suitable for the task of checking/eliminating redundant factors.

The factorization of a type can be computed, as shown in the following, by applying the results of Baader [1991, 1996]. Given a virtual name V of a schema



$$\sigma_V(Q1) = \mathtt{E} \sqcap \triangle[\mathtt{s}:80..\infty] \sqcap \triangle[\mathtt{h}:NQ1]$$
$$\sigma_V(NQ1) = \triangle[\mathtt{q}:7..\infty] \sqcap \triangle[\mathtt{h}:NQ1]$$

$$\mathcal{L}(Q1, \mathtt{E}) = \{\epsilon\}$$
$$\mathcal{L}(Q1, 80..\infty) = \{\triangle.\mathtt{s}\}$$
$$\mathcal{L}(Q1, 7..\infty) = \{(\triangle.\mathtt{h})^*.\ \triangle.\mathtt{q}\}$$

Fig. 3. Query Q1 Automata and languages.

$\Sigma = (\sigma, \mathbf{R})$, for each type $S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}$ of $\sigma_V(V)$, we define a finite automaton having V as initial state and S as final state, denoted as $\mathcal{A}(V, S)$. $\mathcal{A}(V, S)$ defines a regular language, denoted by $\mathcal{L}(V, S)$, namely the set of all words which are labels of paths from V to S. The formalization of the automaton is presented in Appendix B; here we introduce an example by considering $Q1$ of Table IX: Figure 3 shows $\sigma_V(Q1)$, the three automata associated to Q1, $\mathcal{A}(Q1, \mathtt{E})$, $\mathcal{A}(Q1, 80..\infty)$ and $\mathcal{A}(Q1, 7..\infty)$ (the final states are denoted with thick ovals) and the related accepted languages.

The following proposition is a straightforward application to our context of Proposition 19 of Baader [1996].

PROPOSITION 3. *Given a schema $\Sigma = (\sigma, \mathbf{R})$, let V be a virtual name and let $\mathcal{I}$ be a legal instance of the subschema with no rules, $\Sigma_\sigma = (\sigma, \emptyset)$. For any value $v \in \mathcal{V}$, $v \in \mathcal{I}[V]$ iff for all $S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}$ and for all words $w \in \mathcal{L}(V, S)$, we have that $v \in \mathcal{I}[(w\colon S)]$.*

PROOF. See Beneventano [2002]. □

The proposition can intuitively be understood as follows: the language $\mathcal{L}(V, S)$ stands for the possibly infinite number of constraints of the form $(w\colon S)$, which the schema imposes on V. The more constraints are imposed, the smaller $\mathcal{I}[V]$ is.

THEOREM 4. *Given a schema $\Sigma = (\sigma, \mathbf{R})$, for any virtual name V we have that the set*

$$F_\Sigma^V = \{(p_1\colon S), (p_2\colon S), \ldots, (p_n\colon S) \mid S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}, \mathcal{L}(V, S) \neq \emptyset,$$
$$p_1 + p_2 + \cdots + p_n \text{ is a regular expression describing } \mathcal{L}(V, S)\}$$

is a factorization of V with respect to Σ.

PROOF SKETCH. Regular languages can be *described* [Hopcroft and Ullman 1979] by regular expressions. Then, as an immediate consequence of Proposition 3 and of the definition of path type interpretation (see Section 3.4), any virtual name V can be expressed, with respect to the subschema with no

Table XII. Query Q1 Factorized Form

$Q1 \simeq_\Sigma (\epsilon\colon \mathrm{E}) \sqcap (\triangle.\mathrm{s}\colon 80..\infty) \sqcap ((\triangle.\mathrm{h})^*.\ \triangle.\mathrm{q}\colon 7..\infty)$

rules, by a conjunction of path types:

$$V \simeq_\sigma \sqcap\{(p\colon S) \mid S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}, \mathcal{L}(V, S) \neq \emptyset, \\ p \text{ is a regular expression describing } \mathcal{L}(V, S)\} \qquad (9)$$

A regular expression p can always be translated into a $p_1 + p_2 + \cdots + p_n$ where n is finite and p_i is a *union-free* regular expression, by applying the following equivalence rules repeatedly, until either $+$ is eliminated or it compares at the outermost level:

(1) $(p_1 + p_2 + \cdots + p_n).p_0 = (p_1.p_0) + (p_2.p_0) + \cdots + (p_n.p_0)$
(2) $p_0.(p_1 + p_2 + \cdots + p_n) = (p_0.p_1) + (p_0.p_2) + \cdots + (p_0.p_n)$
(3) $(p_1 + p_2 + \cdots + p_n)^* = ((p_1)^*.(p_2)^* \cdots (p_n)^*)^*$

It follows immediately that any path type $(p\colon S)$ of (9) can be expressed as a conjunction of factors, that is, $(p\colon S) \simeq_\sigma (p_1\colon S) \sqcap (p_2\colon S) \sqcap \cdots \sqcap (p_n\colon S)$, and then

$$F = \{(p_1\colon S), (p_2\colon S), \ldots, (p_n\colon S) \mid S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}, \mathcal{L}(V, S) \neq \emptyset, \\ p_1 + p_2 + \cdots + p_n \text{ is a regular expression describing } \mathcal{L}(V, S)\}$$

is a factorization of V, as $V \simeq_\sigma \sqcap F$ and then $V \simeq_\Sigma \sqcap F$. □

Thus, in the following, we consider a virtual name V (the query) and its factorization with respect to Σ, F^V_Σ as defined in Theorem 4. Such factorization is unique up to regular path expression equivalence. Table XII shows the factorization for Q1, obtained by the languages of Figure 3.

5.2 Specialization and Eliminability of Factors

We introduce the *specialization* property of a factor.

Definition 12 (*Specialization*). Given a schema $\Sigma = (\sigma, \mathbf{R})$, a virtual name V and its factorization F^V_Σ, a factor $(p\colon S) \in F^V_\Sigma$ admits a *specialization* if another factorization exists, say F′, of V which *contains* a factor $(p'\colon S')$ with $(p'\colon S') \sqsubseteq_\sigma (p\colon S)$.

LEMMA 1. *Given a schema $\Sigma = (\sigma, \mathbf{R})$, a virtual name V and its factorization F^V_Σ, let F' of V such that $\sqcap F'$ is a semantic expansion of V. Then, for any factor $(p\colon S) \in F^V_\Sigma$, a factor $(p'\colon S')$ in F' with $(p'\colon S') \sqsubseteq_\sigma (p\colon S)$ exists.*

Let's introduce the definitions of *redundancy* of a factor with respect to a set of factors and of *eliminability* of a factor from a query.

Definition 13 (*Redundancy*). Given a schema $\Sigma = (\sigma, \mathbf{R})$ and a set of factors F of $\mathbf{S}$, a factor $f \in F$ is *redundant* with respect to F iff it is implied by the remaining factors, that is, iff $\sqcap F \setminus \{f\} \sqsubseteq_\Sigma f$.

Intuitively, given a factorization F of V, a factor f of F is *eliminable* from V if another factorization exists, say F′, of V that *does not contain* f. The formal definition of eliminability is quite complex due to paths: if a factorization F' of V *does not contain* $(p\colon S)$, then it is possible, with F', to solve query V without accessing S *through* p. Thus, first $(p\colon S')$ with $S \sqsubseteq_\sigma S'$ must not be included in F'; second, as p denotes a set of words, F' must not include any path $(w\colon S)$ with $w \in \mathcal{L}(p)$.

Definition 14 (*Eliminability*). Given a schema $\Sigma = (\sigma, \mathbf{R})$, a virtual name V and its factorization F^V_Σ, a factor $(p\colon S) \in F^V_\Sigma$ is *eliminable* from V iff a factorization F' of V exists with respect to Σ such that, for all $(p'\colon S') \in F'$, if $\mathcal{L}(p) \cap \mathcal{L}(p') \neq \emptyset$, then $S \not\sqsubseteq_\sigma S'$.

The following lemma states that redundancy is a sufficient condition for eliminability.

LEMMA 2. *Given a schema $\Sigma = (\sigma, \mathbf{R})$, a virtual name V and its factorization F^V_Σ, then a factor $f \in F^V_\Sigma$ is eliminable from V if the factor f is redundant with respect to F^V_Σ.*

It is easy to prove that opposite does not apply, that is, eliminability does not imply redundancy, as shown in the example of query Q8 where the factor `directs.category >= 4` is eliminable but not redundant with respect to F^{Q8}_Σ. On the other hand, as introduced in query Q8 and explained in the following, the factor `directs.category >= 4` is redundant with respect to a factorization, which is a semantic expansion of $Q8$, that is, the redundancy check on a semantic expansion can detect a greater number of eliminable factors.

To conclude, it is convenient to refer to a factorization that is a semantic expansion for both specialization and eliminability of the factors of a query. In order to compute this factorization, we introduce the *P-reduction* algorithm, having the factorization $F^V_{\overline{\Sigma}}$ of V with respect to the expanded schema $\overline{\Sigma}$ as input. $\overline{\Sigma}$ is a canonical schema and, therefore, $F^V_{\overline{\Sigma}}$ contains the factors of the descriptions of primitive type names (i.e., a set of factors redundant with respect to the primitive type names). Algorithm *P-reduction*, and thus the motivation for its name, eliminates these redundant factors.

Algorithm 2 (P-reduction).

Input. A schema $\Sigma = (\sigma, \mathbf{R})$, an expanded schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$, which is a conservative extension of Σ, a virtual name $V \in \overline{\mathbf{V}}$ and its factorization $F^V_{\overline{\Sigma}}$.

Output. $\Phi(V)$, a factorization of V with respect to the schema Σ.

(1) for each $(p\colon \overline{P}) \in F^V_{\overline{\Sigma}}$ we consider the factor $(w^0_p\colon P)$, where w^0_p is the shortest word denoted by p; if $(w^0_p\colon P) \sqsubseteq_\sigma (p\colon P)$ then we replace $(p\colon \overline{P})$ with $(w^0_p\colon P)$, otherwise, we replace $(p\colon \overline{P})$ with $(p\colon P)$.
Let w^0_p be the shortest word in $\mathcal{L}(p)$:

$$K_1(V) = \{(p'\colon P) \mid (p\colon \overline{P}) \in F^V_{\overline{\Sigma}}, \overline{P} \in \overline{\mathbf{P}}, \\ p' = w^0_p \text{ if } \left(w^0_p\colon P\right) \sqsubseteq_\sigma (p\colon P),\ p' = p \text{ otherwise}\} \\ \cup \left\{(p\colon S) \mid (p\colon S) \in F^V_{\overline{\Sigma}}, S \notin \overline{\mathbf{P}}\right\}$$

Table XIII. Query Q1 after P-Reduction Algorithm

$Q1 \simeq_\Sigma (\epsilon\text{:M}) \sqcap (\triangle\text{s:}80..100)$

Table XIV. Comparison Between Factorizations F_Σ^V and $\Phi(V)$

V	F_Σ^V	$\Phi(V)$
Q1	$\{(\epsilon\text{:E}), (\text{s:}80..\infty), ((\text{h})^*.\text{q:}7..\infty)\}$	$\{(\epsilon\text{:M}), (\text{s:}80..100)\}$
Q8	$\{(\epsilon\text{:M}), (\text{s:}100..100), (\text{d.c:}4..\infty)\}$	$\{(\epsilon\text{:M}), (\text{s:}100..100), (\text{d:CBD}), (\text{q:}10..10)\}$

(2) for each $(p\text{:}P) \in K_1(V)$ we eliminate from $K_1(V)$ the factors $(p'\text{:}S')$ such that $(p\text{:}P) \sqsubseteq_\sigma (p'\text{:}S')$:

$$\Phi(V) = K_1(V) \setminus \{(p'\text{:}S') \mid \exists (p\text{:}P) \in K_1(V), (p\text{:}P) \sqsubseteq_\sigma (p'\text{:}S')\}$$

To give a simple example, let's consider a simplified version of the semantic expansion of $Q1$ (we do not consider the attribute d:D of V_{13}) introduced in Table XI:

$$\overline{\sigma}(Q1) = \overline{\text{E}} \sqcap \overline{\text{M}} \sqcap \triangle V_{11}, \quad \overline{\sigma}(V_{11}) = [\text{q:}8..10, \text{s:}80..100, \text{h:}V_{12}]$$
$$\overline{\sigma}(V_{12}) = \overline{\text{E}} \sqcap \overline{\text{M}} \sqcap \triangle V_{13}, \quad \overline{\sigma}(V_{13}) = [\text{q:}8..10, \text{s:}10..100, \text{h:}V_{12}]$$

we obtain:

$$F_{\overline{\Sigma}}^{Q1} = \{((\triangle.\text{h})^*\text{:}\overline{\text{E}}), ((\triangle.\text{h})^*\text{:}\overline{\text{M}}), (\triangle.\text{s:}80..100), ((\triangle.\text{h})^*.\triangle.\text{q:}8..10), (\triangle.\text{h}.(\triangle.\text{h})^*.\triangle.\text{s:}10..100)\}$$

(1) For the factor $((\triangle.\text{h})^*\text{:}\overline{\text{E}})$: the shortest word in $\mathcal{L}((\triangle.\text{h})^*)$ is ϵ and $(\epsilon\text{:E}) \sqsubseteq_\sigma ((\triangle.\text{h})^*\text{:E})$.
For the factor $((\triangle.\text{h})^*\text{:}\overline{\text{M}})$: the shortest word in $\mathcal{L}((\triangle.\text{h})^*)$ is ϵ and $(\epsilon\text{:M}) \sqsubseteq_\sigma ((\triangle.\text{h})^*\text{:M})$.
Then:

$$K_1(Q1) = \{(\epsilon\text{:E}), (\epsilon\text{:M}), (\triangle.\text{s:}80..100), ((\triangle.\text{h})^*.\triangle.\text{q:}8..10), (\triangle.\text{h}.(\triangle.\text{h})^*.\triangle.\text{s:}10..100)\}$$

(2) Since $(\epsilon\text{:E}) \sqsubseteq_\sigma (\triangle.\text{h}.(\triangle.\text{h})^*.\triangle.\text{s:}10..100)$, $(\epsilon\text{:M}) \sqsubseteq_\sigma ((\triangle.\text{h})^*.\triangle.\text{q:}8..10)$ and $(\epsilon\text{:M}) \sqsubseteq_\sigma (\epsilon\text{:E})$
we obtain $\Phi(Q1) = \{(\epsilon\text{:M}), (\triangle.\text{s:}80..100)\}$ (Q1 after P-reduction Algorithm is shown in Table XIII).

THEOREM 5. *Given a schema* $\Sigma = (\sigma, \mathbf{R})$ *and an expanded schema* $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$*, which is a conservative extension of* Σ*, the set* $\Phi(V)$ *computed by Algorithm* 2 *is a factorization of* V *with respect to* Σ *and* $\sqcap\{f \mid f \in \Phi(V)\}$ *is a semantic expansion of* V.

PROOF. See Appendix C. □

In order to point out the properties of $\Phi(V)$ let's compare it with the original factorization F_Σ^V; in Table XIV we consider these factorizations for Q1 and Q8 ($\triangle.a$ is denoted by a).

In general, a factor $(p\colon S) \in F^V_\Sigma$ is specialized/eliminated in $\Phi(V)$:

Specialization. There is a factor $(p'\colon S')$ in $\Phi(V)$ with $(p'\colon S') \sqsubseteq_\sigma (p\colon S)$.

For example, for Q1, the factors $(\epsilon\colon \mathtt{E})$ and $(\mathtt{s}\colon 80..\infty)$ are specialized in $(\epsilon\colon \mathtt{M})$ and $(\mathtt{s}\colon 80..100)$ respectively.

Note that a particular case of specialization is *simplification*, which occurs when a factor is specialized in $\Phi(V)$ with respect to the schema classes. For example, for Q8, the factor $(\mathtt{d.c}\colon 4..\infty)$ has been simplified in $(\mathtt{d}\colon \mathtt{CBD})$.

Elimination. A factor $(p\colon S)$ is eliminated (and thus it has no correspondence in $\Phi(V)$).

For example, for Q1, the factor $((\triangle.\mathtt{h})^*.\triangle.\mathtt{q}\colon 7..\infty)$ has been eliminated, since it is contained in the description of the primitive class `M`.

The final optimization step is redundancy check. For example, for Q1, $\Phi(\mathrm{Q1})$ is the result of our optimal rewriting method as it does not contain redundant factors.

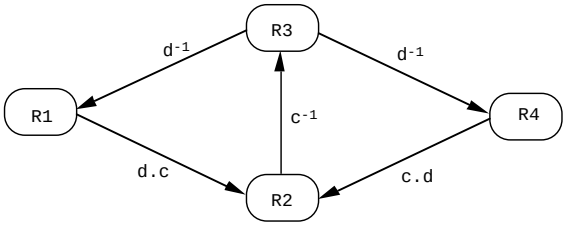
For Q8, the factors $(\mathtt{d}\colon \mathtt{CBD})$ and $(\mathtt{q}\colon 10..10)$ are redundant with respect to $\Phi(Q8)$, but cannot be eliminated together since $(\epsilon\colon \mathtt{M}) \sqcap (\mathtt{s}\colon 100..100) \not\sqsubseteq_\Sigma (\mathtt{q}\colon 10..10) \sqcap (\mathtt{d}\colon \mathtt{CBD})$. This example shows the problem of the elimination of a set of factors, since it may be the case that either f_1 or f_2 can be eliminated but they cannot be eliminated together. Intuitively speaking, this happens when *recursive rules* occur, that is, when the consequent of a rule R_1 is subsumed by the antecedent of a rule R_2 and vice-versa. On the contrary, when the semantic expansion of a name does not involve any recursive rule, *the conjunction of redundant factors is redundant* too; in this case, it is possible to detect all redundant factors by computing a linear number of subsumption relationships. In order to formalize this case, let's introduce the definition of *rule graph*.

Definition 15 (*Rule Graph*). Given a schema $\Sigma = (\sigma, \mathbf{R})$, the *rule graph* $G_\Sigma(\mathbf{R}, \mathbf{E})$ is a labeled directed graph that has $R \in \mathbf{R}$ as nodes and the edges $E \in \mathbf{E}$, with $E = (R_i, w_{i,j}, R_j)$ defined as follows: the label $w_{i,j}$ is a word of the alphabet $\{e, e^{-1} \mid e \in \mathbf{A} \cup \{\triangle, \forall, \exists\}\}$, $R_i = (S^a_i, S^c_i)$, $R_j = (S^a_j, S^c_j)$, and there is $(p\colon S)$ in $\Phi(S^c_i)$ and $(p'\colon S')$ in $\Phi(S^a_j)$ such that one of the following conditions holds:

(1) $(p\colon S) \sqsubseteq_\sigma (p'\colon S')$; in this case $w_{i,j} = \epsilon$.
(2) $(w.p\colon S) \sqsubseteq_\sigma (p'\colon S')$ for a word $w = e_1.e_2 \cdots e_{n-1}.e_n$; in this case $w_{i,j} = e_n.e_{n-1} \cdots e_2.e_1$.
(3) $(p\colon S) \sqsubseteq_\sigma (w.p'\colon S')$ for a word $w = e_1.e_2 \cdots e_{n-1}.e_n$; in this case $w_{i,j} = e_1^{-1}.e_2^{-1} \cdots e_{n-1}^{-1}.e_n^{-1}$.

An example of rule graph is given in Figure 4; in particular, we have:
$(\mathtt{e}\colon \mathtt{B}) \in \Phi(S^c_4)$, $(\mathtt{d.c.e}\colon \mathtt{B}) \in \Phi(S^a_2)$, with $(w.\mathtt{e}\colon \mathtt{B}) \sqsubseteq_\sigma (\mathtt{d.c.e}\colon \mathtt{B})$, where $w = \mathtt{d.c}$, then there is an edge from R_4 to R_2 with label `c.d` (case (2) of the definition); $(\mathtt{d.a}\colon \mathtt{B}) \in \Phi(S^c_3)$, $(\mathtt{a}\colon \mathtt{B}) \in \Phi(S^a_4)$, with $(\mathtt{d.a}\colon \mathtt{B}) \sqsubseteq_\sigma (w.\mathtt{a}\colon \mathtt{B})$, where $w = \mathtt{d}$, then there is an edge from R_3 to R_4 with label $\mathtt{d}^{-1}$ (case (3) of the definition).

Definition 16 (*Recursive Rule*). Given a schema $\Sigma = (\sigma, \mathbf{R})$, a rule R is *recursive* if the graph $G_\Sigma(\mathbf{R}, \mathbf{E})$ contains at least a cyclic path from R to R with



$R_1 = ((\texttt{a:B}), (\texttt{b:B}))$
$R_2 = ((\texttt{c.d.b:B}) \sqcap (\texttt{d.c.e:B}), (\texttt{c.h:B}))$
$R_3 = ((\texttt{h:B}), (\texttt{d.a:B}))$
$R_4 = ((\texttt{a:B}), (\texttt{e:B}))$

Fig. 4. Example of Rule Graph.

label ϵ or of the form $e_1.e_2 \cdots e_{n-1}.e_n.e_n^{-1}.e_{n-1}^{-1} \cdots e_2^{-1}.e_1^{-1}$. $\mathbf{R}_R$ denotes the set of recursive rules in $\mathbf{R}$.

As a simple example of recursive rule, let's consider $R_1 = ((\texttt{a}: 10..30))$, $((\texttt{b}: 5..15))$ and $R_2 = (\texttt{b}: 5..25), (\texttt{a}: 10..20)$; of course, both R_1 and R_2 are recursive with label ϵ. A more complex example is shown in Figure 4, where R_1 is the unique recursive rule since there is a cyclic path with label $\texttt{d.c.c}^{-1}\texttt{.d}^{-1}$ from R_1 to R_1.

THEOREM 6. *Given a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, a name V and its factorization $\Phi(V)$, let $f_1, \ldots, f_n \in \Phi(V)$ be redundant factors with respect to $\Phi(V)$. If there is no recursive rule $R = (S^a, S^c)$ and path p such that $V \sqsubseteq_\Sigma (p: S^a)$, then $\Phi(V) \setminus \{f_1, \ldots, f_n\}$ is a factorization of V.*

PROOF. See Beneventano [2002]. □

Note that, the set $\mathbf{R}_R$ can be computed on the schema $\Sigma = (\sigma, \mathbf{R})$ at once, as the schema is available. Having the set $\mathbf{R}_R$ it is possible to easily check the condition of Theorem 6. In fact, for a given virtual name V, by using Algorithm 1, we can find all the rules $R = (S^a, S^c)$ such that there is a path p with $V \sqsubseteq_\Sigma (p: S^a)$ and verify if at least one of these rules belongs to $\mathbf{R}_R$.

Let's show the method based on Theorem 6 by an example. Having the rules:

$$R_1 = ((\texttt{a}: 10..30), (\texttt{b}: 5..15))$$
$$R_2 = ((\texttt{b}: 5..25), (\texttt{a}: 10..20))$$
$$R_3 = ((\texttt{c}: 10..30), (\texttt{a}: 10..40) \sqcap (\texttt{d}: 5..35)).$$

Of course, both R_1 and R_2 are recursive with label ϵ. Let's consider the queries Q_a, Q_b, Q_c and their $\Phi(Q)$:

(1) $\Phi(Q_a) = \{(\texttt{a}: 10..40), (\texttt{b}: 5..45), (\texttt{c}: 10..20), (\texttt{d}: 5..35)\}$
In this case, the redundant factors are $(\texttt{a}: 10..40)$ and $(\texttt{d}: 5..35)$; since

there is no recursive rule R and a path p such that $Q_1 \sqsubseteq_\Sigma (p{:}S^a)$, $\Phi(Q_1)\backslash\{(\mathtt{a}{:}10..40), (\mathtt{d}{:}5..35)\}$ is a factorization of Q_a.

(2) $\Phi(Q_b) = \{(\mathtt{a}{:}10..20), (\mathtt{b}{:}5..15), (\mathtt{c}{:}10..40), (\mathtt{d}{:}5..45)\}$
In this case, the redundant factors are $(\mathtt{a}{:}10..20)$ and $(\mathtt{b}{:}5..15)$; since the recursive rules R_1 and the path ϵ exist such that $Q_b \sqsubseteq_\Sigma (\epsilon{:}S_1^a)$, Theorem 6 is not applicable and we must explicitly check the redundancy of $(\mathtt{a}{:}10..20) \sqcap (\mathtt{b}{:}5..15)$; since $(\mathtt{c}{:}10..40) \sqcap (\mathtt{d}{:}5..45) \not\sqsubseteq_\Sigma (\mathtt{a}{:}10..20) \sqcap (\mathtt{b}{:}5..15)$ the factors $(\mathtt{a}{:}10..20)$ and $(\mathtt{b}{:}5..15)$ cannot be eliminated together.

(3) $\Phi(Q_c) = \{(\mathtt{a}{:}10..20), (\mathtt{b}{:}5..15), (\mathtt{c}{:}10..20), (\mathtt{d}{:}5..35)\}$
In this case, the redundant factors are $(\mathtt{a}{:}10..20)$, $(\mathtt{b}{:}5..15)$ and $(\mathtt{d}{:}5..35)$; since the recursive rules R_1 and the path ϵ exist such that $Q_c \sqsubseteq_\Sigma (\epsilon{:}S_1^a)$, Theorem 6 is not applicable and we must explicitly check the redundancy of $(\mathtt{a}{:}10..20) \sqcap (\mathtt{b}{:}5..15) \sqcap (\mathtt{d}{:}5..35)$; since $(\mathtt{c}{:}10..40) \sqsubseteq_\Sigma (\mathtt{a}{:}10..20) \sqcap (\mathtt{b}{:}5..15) \sqcap (\mathtt{d}{:}5..35)$ the factors can be eliminated together and thus $(\mathtt{c}{:}10..20)$ is a factorization of Q_c.

We conclude by applying these results at the queries of running example; first of all, the recursive rules are r_1, r_2, r_3 and r_4. In the case of query $Q8$, as $Q8 \sqsubseteq_\Sigma (\epsilon{:}S_1^a)$, we must explicitly check the redundancy of $(\mathtt{q}{:}10..10) \sqcap (\triangle.\mathtt{d}{:}\mathtt{CBD})$; since $(\epsilon{:}\mathtt{M}) \sqcap (\triangle.\mathtt{s}{:}100..100) \not\sqsubseteq_\Sigma (\triangle.\mathtt{q}{:}10..10) \sqcap (\triangle.\mathtt{d}{:}\mathtt{CBD})$, the factors cannot be eliminated together.

5.3 Linear Cyclic Types

Our model allows the expression of recursive queries, that is, cyclic virtual names. Theorem 4 is applicable to cyclic virtual names and states that any cyclic virtual name under gfp-semantics can be rewritten as a factorization, and thus, necessarily, with an *acyclic* description. Comparable results have been obtained by other authors. In Baader [1991], it is shown that cyclic definitions under gfp-semantics can be replaced by role definitions involving union, composition, and transitive closure of roles. Bertino et al. [1992] introduced a restricted form of recursion for OODB query languages, which corresponds to the notion of linear recursion for logic queries; this is expressed by means of a transitive closure operator. Also in relational environment, Jagadish et al. [1987] prove that the class of the linear recursive queries can be expressed with transitive closure.

This result applies in our context too, that is, every linear recursive query can be factorized in terms of transitive closure.

Formally speaking, given a canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$, a cyclic virtual name V is *linear* if $\overline{\sigma}_V(V)$ includes exactly one occurrence of a cyclic virtual name V_1 such that $V_1 \overset{+}{\hookrightarrow}_V V$, where $\overset{+}{\hookrightarrow}_V$ is the relation $\overset{+}{\hookrightarrow}$ with respect to the set of virtual type names. For example, in the following schema

$$\begin{aligned}\overline{\sigma}_V(V_1) &= [\mathtt{a}{:}V_1, \mathtt{b}{:}V_2]\\ \overline{\sigma}_V(V_2) &= S \sqcap [\mathtt{c}{:}V_1]\end{aligned}$$

the cyclic virtual name V_1 is not linear since $\overline{\sigma}_V(V_1)$ includes V_1 and V_2, with $V_1 \overset{+}{\hookrightarrow}_V V_1$ and $V_2 \overset{+}{\hookrightarrow}_V V_1$. The following schema includes only linear

cyclic names:

$$\begin{aligned}\overline{\sigma}_V(V_0) &= [\mathtt{a}: V_1, \mathtt{b}: V_2]\\ \overline{\sigma}_V(V_1) &= S \sqcap [\mathtt{c}: V_1]\\ \overline{\sigma}_V(V_2) &= S \sqcap [\mathtt{c}: V_2]\end{aligned}$$

since V_0 is not cyclic and V_1 and V_2 are cyclic but depend only on themselves.

PROPOSITION 4. *Given a canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$, and a virtual name V, if V is either acyclic or linear, then V can be factorized in terms of transitive closure, that is in every factor $(p: S) \in F^V_\Sigma$ (defined in Theorem 4) p is obtained with the following syntax rules*

$$\begin{aligned}p' &\rightarrow e \mid e\dot{p}'\\ p &\rightarrow p' \mid (p')^*.\end{aligned}$$

PROOF. See Appendix C. □

6. RELATED WORK

Many research efforts have been devoted to semantic query optimization in the area of relational and deductive databases [Hammer and Zdonik 1980; King 1981; Shenoy and Özsoyoglu 1987, 1989; Jarke et al. 1984; Siegel et al. 1992; Chakravarthy et al. 1990] and in the area of OODBs [Hacid and Rigotti 1995; Yoon et al. 1995; Yoon and Kerschberg 1993; Grant et al. 1997; Pang et al. 1991; Sheu et al. 1989; Jeusfeld and Staudt 1993].

OODBs provide a richer type (class) system with respect to relational databases and a subclass of integrity constraints can be directly represented in the database schema. By exploiting these rich semantics, semantic query optimization can be performed to a great extent. For this reason, we focus on the related works in the area of OODBs, even if some basic techniques have been introduced in the earlier works on relational and deductive databases.

The most followed approach for semantic query optimization in OODBs is *logic-based*. Queries, rules, and integrity constraints allow a uniform representation in a logical formalism and deduction is used to produce equivalent query forms. In Grant et al. [1997], Yoon and Kerschberg [1993], and Yoon et al. [1995], the semantic query optimization techniques developed for deductive databases are extended to object-oriented databases. In particular, in Grant et al. [1997] the ODMG93 standard is used: the ODL object schema and the OQL query are translated into a DATALOG representation and semantic query optimization is performed in this representation and subsequently an OQL query is obtained. In Yoon and Kerschberg [1993], a set of deductive rules based on an object-oriented database schema or semantic knowledge about the domain of the database is generated and the *residue* technique [Chakravarthy et al. 1990] is used to optimize a query. In Yoon et al. [1995], the schema and integrity constraints are represented in the form of nonrecursive Horn clauses.

A *logic-based* approach is also followed in Jeusfeld and Staudt [1993], where a *deductive object base* is defined as a special case of a deductive database with integrity constraints. Object-oriented abstraction principles like object identity,

attribute typing and specialization become axioms of a first-order database theory. By exploiting the axioms, some optimizations can be achieved, such as elimination of class membership predicates and view maintenance optimization.

Another logic-based approach is presented in Sheu et al. [1989]. It introduces a technique for predicate reordering in conjunctive recursive queries. Optimization is obtained by deferring the evaluation of the most expensive predicates. Reduction of the search space is also obtained by applying a sound set of rewriting rules. Their object data model includes deduction laws and integrity constraints expressed in a first-order language. This work is in some way complementary to ours, since it performs predicate reordering (which we do not consider, being dependent from specific cost models), but does not deal with hierarchy-based optimizations, which is one of our main achievements.

Generally speaking, with respect to the works above, our approach is different, as we extend the idea of semantic query expansion to OODBs (originally developed for relational databases) and we adopt a restricted first-order logic, namely a *DL*, as reasoning environment. The choice of adopting a restricted logic-based language to describe the schema, the integrity constraints and the queries is motivated by decidability issues. In fact, it permits the use of a decidable subsumption relation that helps to discover equivalences and simplifications that cannot be computed in the full logic-based framework, where this subsumption relation is undecidable in general [Shmueli 1993]. Furthermore, we deal with a less expressive query language, but we devise an optimal rewriting algorithm that rewrites a query determining more specialized classes to be accessed and minimizing the number of factors, enabling us to optimize a significant set of recursive queries, that is, conjunctive queries.

DL has already been used to solve a particular aspect of semantic query optimization: *automatic classification* of a query with respect to views/queries [Beck et al. 1989; Borgida et al. 1989; Beneventano and Bergamaschi 1997; Buchheit et al. 1994]. From a theoretical point of view, a query and a view both coincide with the semantics of defined concepts (i.e., virtual type names) and the problem is led back to subsumption computation. Since our method is based on a *DL* kernel, it performs the automatic classification of a query too (see Beneventano and Bergamaschi [1997]), but we observe that automatic classification is only useful if we assume that views are materialized. Our approach goes further, since our model allows the representation of integrity constraints, which we exploit to "move down" queries with respect to the database classes hierarchy. In Hacid and Rigotti [1995], an attempt to combine resolution-based reasoning [Yoon and Kerschberg 1993] and classification-based (i.e., *DL*-based) reasoning in a common framework to perform complementary semantic query optimizations is presented, but the investigation is only at a very preliminary stage.

Some other works in OODBs do not follow a logic-based approach. One of the most general works introducing semantic transformation rules in the context of query processing in OODBs is Straube and Özsu [1990]. In this work, a query processing methodology, completed with an object calculus and a closed object algebra, is presented. Query processing issues such as query safety and

object calculus into object algebra translation are discussed in detail. The paper contains a discussion of equivalence-preserving transformation rules for object algebra expressions: *algebraic rules* create equivalent expressions based upon pattern matching and textual substitution; *semantic rules* are similar, but they are additionally dependent on the semantics of class definitions and inheritance lattice of a database schema. The generation of the canonical form of a query proposed in our article includes this kind of algebraic and semantic transformations and goes further, taking also integrity rules in the query transformation into account.

Pang et al. [1991] present a method that performs semantic expansion of a query and three semantic query optimization transformations (restriction elimination, index and restriction introduction, class elimination). The class of considered queries is a proper subset of our clean query class and the task of choosing the beneficial transformations, as in our approach, is delayed until all the possible transformations have been considered. The algorithm presented in Pang et al. [1991] classifies query predicates, on the basis of the available integrity constraints, as imperative, optional and redundant, where the optional predicates could be removed or otherwise, depending on their profitability in terms of possible cost savings due to searchable indexes. In our work, the redundant predicates are directly eliminated in the factorization of the expanded query that is the starting point of the optimal rewriting of a query. Furthermore, the optional predicates (which we call redundant) are found in the optimal rewriting. Note that, in Pang et al. [1991], the removal of optional predicates does not consider the presence of recursive rules that do not allow, as we proved, the independent elimination of optional predicates.

Several papers have dealt with the problem of *conjunctive query containment*, one of the aspects of semantic query optimization,[7] useful in many contexts, including information integration, view maintenance, and data warehousing.

Kolaitis and Vardi [1998] map the query containment problem with respect to an acyclic schema to a constraint satisfaction problem and solve it for generic conjunctive queries. Chan [1992] has investigated the containment and minimization problem for a class of conjunctive queries in an object-oriented setting but he considered some minimal schema information such as subclass relationship and disjointness of classes. Levy and Suciu [1997] propose a query language for complex objects including conjunctive queries and aggregation operators and provide an algorithm to decide containment for queries. In comparison with these papers, we deal with a less expressive query language, but we are able to support (and thus to optimize) a significant set of conjunctive recursive queries including path expressions. Moreover, in all the above papers, the only considered integrity constraints are: attribute types and inheritance relations between classes, whereas we also consider *if-then* integrity rules.

Florescu et al. [1998] give a semantic and syntactic notion for conjunctive query containment, and provide decidability and complexity results. Their

[7]In particular, the already cited *DL*-based works on automatic classification of queries solve the query containment problem as a subsumption computation problem.

query language, StructQL$_O$, allows regular path expressions with path variables to be represented. A very interesting fragment of StructQL$_O$ that captures a class of frequently used recursive queries such as ``a.*.b'' where a, **b** are constant labels and * matches any sequence of any label, is introduced and the NP complete complexity of containment is proved. We share with the authors the choice to limit the kind of recursive queries considered.

Calvanese et al. [1998] study the problem of checking whether a query q is contained in a query q' with respect to the constraints specified in a schema S, where q and q' are nonrecursive Datalog programs whose atoms are complex expressions. Constraints are specified in the form of inclusions between complex expressions, built by using intersection and difference of n-relations, special forms of quantification, regular expressions over binary relations and cardinality constraints. Their DL, $\mathcal{DLR}_{reg}$, is more expressive than ours, allowing negation and cardinality constraints, but their objective is to provide decidability and complexity results rather than algorithms. Moreover, a significant semantic difference with our work is in the definition of cyclic queries; as we explained in Section 3.5, their way to define a query with a couple of inclusion statements is not suitable to uniquely determine the answer of a cyclic query.

Several papers have dealt with the problem of maximal rewriting of conjunctive queries with respect to a set of given views [Beeri et al. 1997; Calvanese et al. 1999], which is relevant in many contexts, including query optimization in information integration [Levy et al. 1996; Ullman 1997] and data warehousing [Zhuge et al. 1995]. The objective of maximal rewriting is opposite to that of optimal rewriting of a query as proposed in this paper. In fact, the maximal rewriting gives a set of views that are equivalent or maximally contained in a given query Q, whereas optimal rewriting gives an equivalent query rewritten on the basis of more specialized classes of the schema. Therefore, the goal of maximal rewriting is to obtain (usually) *a subset of the query answer set* in terms of the available views while, ours is to obtain *exactly the query answer set*, possibly with reduced execution costs. Optimal rewriting could be extended to take materialized views in the semantic query optimization process into account.

In Baader et al. [2000], the following rewriting problem is considered: "given a terminology $\mathcal{T}$ (i.e., a set of concept definitions) and a concept description C that does not contain concept names defined in $\mathcal{T}$, can this description be rewritten into a "related better" description E by using (some of) the names defined in $\mathcal{T}$?" The authors first introduce a general framework for the rewriting problem in Description Logics and then concentrate on the minimal rewriting problem, where "better" means shorter and "related" means equivalent. Our optimal query rewriting method gives an equivalent query rewriting on the basis of *more specialized primitive* classes of the schema. The method could be reformulated with respect to the general framework for the rewriting introduced in Baader et al. [2000] taking the differences between the *DLs* proposed into account: they are more expressive on one hand, allowing negation, disjunction and cardinality constraints, less expressive on the other hand as we consider cyclic definition.

7. CONCLUSIONS AND FUTURE WORK

Semantic query optimization has never caught up in the commercial world with reference to relational databases for many reasons. The most prominent one is the fact that semantic query optimization was in many cases designed for deductive databases [Chakravarthy et al. 1990; Hammer and Zdonik 1980; Jarke et al. 1984] and due to this association, semantic query optimization might not seem useful for relational database technology. Secondly, at the time when semantic query optimization techniques were being developed, the relative CPU and I/O speeds were not as dramatically different as they are now. Finally, it has been usually assumed that many integrity constraints have to be defined for a given database if semantic query optimization is to be useful there; otherwise, only few queries could be optimized semantically, but many of the integrity constraints considered in early days of semantic query optimization are not expressible in most commercial database systems even today.

We believe that, with the more expressive object-relational and object data models and the declarative expression of integrity constraints, semantic query optimization techniques could provide an effective enhancement to the traditional query optimization. Thus, in this article, we have presented a general semantic query optimization method in this framework. The method is effective for conjunctive recursive queries including path expressions and is implemented in a tool providing an ODMG compliant interface, which permits full interaction with OQL queries, adopting *DL* techniques and wrapping underlying *DL* representation and techniques to the user.

We are currently extending our work in several directions. A first direction concerns the investigation of the extensions of the method for semistructured environments. In fact, in the context of semantic optimization, a major achievement of our work is the possibility of expressing queries and integrity constraints with regular path expressions. These aspects are even more important when semistructured data sources are considered. Our method needs a schema (classes and integrity constraints) that is missing in semistructured environments, but many authors have acknowledged that querying semistructured data can greatly profit by some imposed or discovered schema [Buneman 1997; Buneman et al. 1997; Goldman and Widom 1997; Nestorov et al. 1997, 1998] and Abiteboul and Vianu [1997] introduce path constraints in semistructured data. The models proposed in semistructured environments subsume our data model as they provide the union operator, which is fundamental in this context. From a theoretical point of view our method can be extended in this direction as, recently, Calvanese et al. [1999] proved that subsumption in a *DLs* including these features is decidable. Nevertheless, the integration of new modelling features in a context of semantic query optimization, its formalization and implementation are complex problems requiring a new research effort.

A second direction is the extension of semantic query optimization techniques in the environment of integration of information from distributed heterogeneous sources: a first step in this direction is in Beneventano et al. [2001].

Further directions are: the extension of the optimal rewriting method that we introduced taking not only classes into account but also materialized views (this extension could be useful in the environments of information integration and data warehousing); the capability of expressing other kinds of integrity constraints, for example primary keys; the integration of our method with physical query optimization: the optimal form could be compared with other semantically equivalent forms, to select the best one on the basis of a specific physical access cost model.

APPENDIXES

A. EXTENDED ODL SYNTAX

This section shows the extensions to the ODL-ODMG syntax that we propose. The reader is referred to Cattell [1994, Sect. 3.5], for the original ODL syntax. The modified production rules are ⟨definition⟩ and ⟨base_type_spec⟩ (⟨ODL_definition⟩ and ⟨ODL_base_type_spec⟩ denote the original definitions).

⟨definition⟩ ::= ⟨ODL_definition⟩
| ⟨view_dcl⟩ **;**
| ⟨rule_dcl⟩ **;**

⟨view_dcl⟩ ::= **view** ⟨identifier⟩
[⟨inheritance_spec⟩]
[⟨type_property_list⟩]

⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_pre⟩ **then** ⟨rule_post⟩
⟨rule_pre⟩ ::= **forall** ⟨identifier⟩ **in** ⟨identifier⟩ **:** ⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨rule_body_list⟩ ::= (⟨rule_body_list⟩)
| ⟨rule_body⟩
| ⟨rule_body_list⟩ **and** ⟨rule_body⟩
| ⟨rule_body_list⟩ **and** (⟨rule_body_list⟩)
⟨rule_body⟩ ::= ⟨dotted_name⟩ ⟨rule_const_op⟩ ⟨literal_value⟩
| ⟨dotted_name⟩ **in** ⟨dotted_name⟩
| **forall** ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
| **exists** ⟨identifier⟩ **in** ⟨dotted_name⟩ **:** ⟨rule_body_list⟩
⟨rule_const_op⟩ ::= $= \mid \geq \mid \leq \mid > \mid <$
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**⟨dotted_name⟩

⟨base_type_spec⟩ ::= ⟨ODL_base_type_spec⟩ | ⟨range_type⟩
⟨range_type⟩ ::= **range** ⟨range_specifier⟩
⟨range_specifier⟩ ::= ⟨const_exp⟩ .. ⟨const_exp⟩
| ⟨const_exp⟩ **,** $+$**inf**
| $-$**inf ,** ⟨const_exp⟩

B. MAPPING SCHEMATA TO FINITE AUTOMATA

Let Γ be a finite *alphabet*. A *Nondeterministic Finite Automaton* $\mathcal{A} = (\mathcal{Q}, \Gamma, \delta, q_0, \mathcal{F})$ consists of a finite set of *states* $\mathcal{Q}$, a finite *alphabet* Γ, and a finite set of *transitions* $\delta \subseteq \mathcal{Q} \times (\Gamma \cup \{\epsilon\}) \times \mathcal{Q}$, an *initial state* q_0 and a set of *final states* $\mathcal{F}$ [Hopcroft and Ullman 1979].

Given a virtual name V of a schema $\Sigma = (\sigma, \mathbf{R})$ over $\mathbf{S}$, we associate to V an automaton $\mathcal{A} = (\mathcal{Q}_V, \Gamma_V, \delta_V, q_0, \mathcal{F})$, with the following meaning:

—$\mathcal{Q}_V = \{V\} \cup \{S \in \mathbf{S} \mid S \text{ is used in } \sigma_V(V)\}$ is the set of nodes;
—$\Gamma_V = \{e \in \mathbf{A} \cup \{\triangle, \forall, \exists\} \mid e \text{ is used in } \sigma_V(V)\}$ is the set of transition labels;
—$q_0 = V$, is the initial state;
—$\mathcal{F} = \{S\}$, where $S \in \mathbf{B} \cup \mathbf{P} \cup \{\top, \bot\}$ is the final state

and the transitions between states δ_V are defined as follows:

$$\begin{aligned}
\langle V, \epsilon, S\rangle \in \delta_V \quad & \text{iff } \sigma_V(V) = S \\
\langle S \sqcap S', \epsilon, S\rangle\langle S \sqcap S', \epsilon, S'\rangle \in \delta_V \quad & \text{iff } S \sqcap S' \in \mathcal{Q}_V \\
\langle \forall\{S\}, \forall, S\rangle \in \delta_V \quad & \text{iff } \forall\{S\} \\
\langle \exists\{S\}, \exists, S\rangle \in \delta_V \quad & \text{iff } \exists\{S\} \in \mathcal{Q}_V \\
\langle [\ldots, a_i : S_i, \ldots], a_i, S_i\rangle \in \delta_V \quad & \text{iff } [\ldots, a_i : S_i, \ldots] \in \mathcal{Q}_V \\
\langle (\triangle S), \triangle, S\rangle \in \delta_V \quad & \text{iff } (\triangle S) \in \mathcal{Q}_V
\end{aligned}$$

In particular, the primitive type names are final states of the automaton with no outgoing transition.

In the following, $\mathcal{A}(V, S)$ will indicate the automaton associated to the name V with S as final state. $\mathcal{A}(V, S)$ defines a regular language, denoted by $\mathcal{L}(V, S)$, namely the set of all words which are labels of paths from N to S.

C. PROOFS

The following are the proofs of Theorems 3 and 5 and Proposition 4.

THEOREM 3. *Given a canonical schema* $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$ *as input, the output of Algorithm* 1 *is an expanded schema* $\widetilde{\Sigma} = (\widetilde{\sigma}, \overline{\mathbf{R}})$ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \widetilde{\mathbf{V}})$ *which is a conservative extension of* $\overline{\Sigma}$.

PROOF SKETCH. The termination of Algorithm 1 is guaranteed by the following facts:

—a rule cannot be applied more than once to the same name; in fact, when a rule turns out to be applicable to a name, the consequent of the rule is conjuncted with the name description: this makes the rule no more applicable to the name;
—the rule set is finite;
—Step (1) of Algorithm 1 generates a finite set of names. In fact, given the input canonical schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}})$, we can consider a canonical

schema $\overline{\Sigma}_1 = (\overline{\sigma}_1, \overline{\mathbf{R}})$ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \overline{\mathbf{P}} \cup \overline{\mathbf{V}}_1)$ such that, for each $\mathbf{X} \subseteq 2^{\overline{\mathbf{V}}}$, there is $V \in \overline{\mathbf{V}}_1$ with $\overline{\sigma}_V(V)$ equal to $\sqcap_{V_i \in \mathbf{X}^{\overline{\sigma}_V}}(V_i)$. Note that the $\overline{\mathbf{V}}_1$ set is finite.

The names generated by Step (1) are contained in $\overline{\mathbf{V}}_1$ In addition, it is not possible to generate two names for the same conjunction, in fact, at Step (i), for any new type S, a new name V' is introduced in $\overline{\sigma}^{i+1}$, with $\overline{\sigma}^{i+1}(V') = S$ if there is no j, $(0 \leq j \leq i)$ and V'' such that $\overline{\sigma}^j(V'') = S$. Then the Step (1) of Algorithm 1 generates a finite set of names.

As the algorithm stops when $\Lambda_{\overline{\Sigma}}(V) = \emptyset$, $\forall V \in \overline{\mathbf{V}}$, the computed schema is an expanded schema. The proof that the computed schema is a conservative extension of $\overline{\Sigma}$ proceeds by induction: we assume that the claim applies for Σ^i and prove it applies for Σ^{i+1}, $i \geq 0$.

If $R = (VR^a, VR^c) \in \Lambda_{\Sigma^i}(V)$ then $V \sqsubseteq_{\sigma^i} VR^a$ and thus, from Theorem 2, it follows that $V \sqsubseteq_{\Sigma^i} VR^a$; moreover, from rule semantics we have that $VR^a \sqsubseteq_{\Sigma^i} VR^c$ then $V \sqsubseteq_{\Sigma^i} VR^c$ and thus $V \simeq_{\Sigma^i} V \sqcap VR^c$. From virtual name semantics, it follows immediately that $\sigma^i(V) \simeq_{\Sigma^i} \sigma^i(V) \sqcap VR^c$. Then, from Proposition 2, it follows that the schema Σ^{i+1} obtained from Σ^i by assigning to $\sigma^{i+1}(V)$ the canonical form of $\sigma^i(V) \sqcap VR^c$ is a conservative extension of Σ^i. Note that we assign a new virtual name V' to each new type S if a V'' with $\sigma^i_V(V'') = S$ does not exist (Step (3), page 26) . On the other hand, by applying the induction hypothesis, each schema Σ^j, with $0 \leq j \leq i$, is a conservative extension of $\overline{\Sigma}$, for this reason, if we assign a new virtual name V' to each new type S if there is no j, $(0 \leq j \leq i)$ and V'' such that $\sigma^j_V(V'') = S$, then we can conclude that the schema Σ^{i+1} is still a conservative extension of $\overline{\Sigma}$. □

THEOREM 5. *Given a schema $\Sigma = (\sigma, \mathbf{R})$ and an expanded schema $\overline{\Sigma} = (\overline{\sigma}, \overline{\mathbf{R}})$ that is a conservative extension of Σ, the set $\Phi(V)$ computed by Algorithm 2 is a factorization of V with respect to Σ and $\sqcap\{f \mid f \in \Phi(V)\}$ is a semantic expansion of V.*

PROOF SKETCH. As $\overline{\Sigma}$ is a conservative extension of Σ, it follows that, given the factorization $F^V_{\overline{\Sigma}}$ of V with respect to $\overline{\Sigma}$, the set $K'_1(V) = \{(p\colon P) \mid (p\colon \hat{P}) \in F^V_{\overline{\Sigma}}, \hat{P} \in \overline{\mathbf{P}}\} \cup \{(p\colon S) \mid (p\colon S) \in F^V_{\overline{\Sigma}}, S \notin \overline{\mathbf{P}}\}$ is a factorization of V with respect to Σ and $\sqcap\{f \mid f \in K'_1(V)\}$ is a semantic expansion of V.

Let w_p be a word in $\mathcal{L}(p)$; of course, $(p\colon P) \sqsubseteq_\sigma (w_p\colon P)$, then $(w_p\colon P) \sqsubseteq_\sigma (p\colon P)$ implies that $(w_p\colon P) \simeq_\sigma (p\colon P)$. Then, if $(w_p\colon P) \sqsubseteq_\sigma (p\colon P)$, in the set $K'_1(V)$, we can replace $(p\colon P)$ by $(w_p\colon P)$, obtaining again a factorization of V with respect to Σ. Thus, the set $K_1(V)$ (where we consider the shortest word w^0_p in $\mathcal{L}(p)$) is a factorization of V with respect to Σ. Furthermore, $\sqcap\{f \mid f \in K'_1(V)\} \simeq_\sigma \sqcap\{f \mid f \in K_1(V)\}$ and thus $\sqcap\{f \mid f \in K_1(V)\}$ is a semantic expansion of V. Since the set $\Phi(V)$ is obtained from $K_1(V)$ by eliminating redundant factors with respect to the description of primitive type name, it follows that:

(1) $\Phi(V)$ is a factorization of V with respect to Σ.
(2) $\sqcap\{f \mid f \in \Phi(V)\} \simeq_\sigma \sqcap\{f \mid f \in K_1(V)\}$ and thus $\sqcap\{f \mid f \in \Phi(V)\}$ is a semantic expansion of V. □

PROPOSITION 4. *Given a schema* $\Sigma = (\sigma, \mathbf{R})$*, and a virtual name* V*, if* V *is either acyclic or linear, then* V *can be factorized in terms of transitive closure, that is in every factor* $(p{:}S) \in F_{\Sigma}^{V}$ *(defined in theorem* 4*)* p *is obtained with the following syntax rules*

$$p' \rightarrow e \mid e\dot{p}'$$
$$p \rightarrow p' \mid (p')^*.$$

PROOF. Each automaton $\mathcal{A}(V, S)$ can be represented as an array of labels $l_{ij}, i, j = 1, \ldots, n$, where i, j indicate the states of the automaton and $l_{i,j}$ is the label of the edge from state j to state i. The array for the automaton of a *linear cyclic type* can be sorted to be *lower triangular*, with the initial state, associated to the type V itself, in the first row and the final state associated to the last row, where the elements of the main diagonal are union-free. In fact, the initial state allows a single incoming edge at the most, when V is recursive on itself, the final state can have incoming edges from any of the other states; in addition, from the hypothesis of linear cyclic, if there is a path from state i to state j, then there cannot be any path from state j to state i, for $i \neq j$, and there is one path at the most from a node to itself. For instance, the automaton associated to type V_0 (see Section 5.3) is described by the following array:

	V_0	V_1	V_2	S
V_0	$-$	$-$	$-$	$-$
V_1	a	c	$-$	$-$
V_2	b	$-$	c	
S	$-$	ϵ	ϵ	$-$

According to Perrin [1990], a *rational expression* defining the language accepted by an automaton can be computed solving the following system of equations:

$$X_i = \sum_{1 \leq j \leq n} X_j l_{ij} + \epsilon_j,$$

where $\epsilon_i = \epsilon$ for $i = 1$, 0 otherwise. To solve this system, one uses substitutions in accordance with the following rule: *the set* $ZY^{\star}$ *is the unique solution of the equation* $X = XY + Z$. The regular expression is given by the solution for X_n. From the shape of the array of labels it turns out that the solution can be built by solving the X_i for increasing i and, when the ith equation is reduced to the form above, Y is a *union-free literal* (i.e., without X_j inside) and the variables X_j contained in Z are such that $j < i$. For this reason, the solution will have the form $\sum w_k$, where w_k is a union-free word. Coming back to the example above, the system of equations is the following:

$$\begin{aligned} V_0 &= \epsilon \\ V_1 &= V_0\mathtt{a} + V_1\mathtt{c} \\ V_2 &= V_0\mathtt{b} + V_2\mathtt{c} \\ S &= V_1 + V_2 \end{aligned}$$

The system above can be solved for V_0, V_1, V_2, S in sequence, obtaining the following solutions:

$$V_0 = \epsilon, \quad V_1 = \mathtt{ac}^\star, \quad V_2 = \mathtt{bc}^\star, \quad S = \mathtt{ac}^\star + \mathtt{bc}^\star$$

and, from the solution for S, the factorization of V_0 is the following:

$$V_0 = [(\mathtt{ac}^\star + \mathtt{bc}^\star) : S\,] \qquad \square$$

ACKNOWLEDGMENTS

We are very grateful to the anonymous referees for their helpful comments.

We are very grateful to Maurizio Vincini who was in charge of ODB-QOptimizer project during his PhD studies in Information and Communication Technology at the University of Modena and Reggio Emilia and to the undergraduate students: Alberto Corni, Ilario Benetti, Paolo Apparuti who contributed to the project.

REFERENCES

ABITEBOUL, S. AND KANELLAKIS, P. 1989. Object identity as a query language primitive. In *SIGMOD*. ACM, New York, 159–173.

ABITEBOUL, S., QUASS, D., MCHUGH, J., WIDOM, J., AND WIENER, J. 1997. The lorel query language for semistructured data. *J. Dig. Lib. 1*, 1, 68–88.

ABITEBOUL, S. AND VIANU, V. 1997. Regular path queries with constraint. In *Proceedings of the 16th Annual ACM SIGACT-SIGMOD SIGART Principles of Database Systems* (*PODS '99*) (Tucson, Az., May 12–14) ACM, New York.

BAADER, F. 1991. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* (Sydney, Australia). Morgan-Kaufmann, San Francisco, Calif.

BAADER, F. 1996. Using automata theory for characterizing the semantics of terminological cycles. *Ann. Math. Artif. Intel. 18*, 2–4, 175–219.

BAADER, F., KÜSTERS, R., AND MOLITOR, R. 2000. Rewriting concepts using terminologies. In *Proceedings of the 7th International Conference on Knowledge Representation and Reasoning* (*KR2000*), A. Cohn, F. Giunchiglia, and B. Selman, Eds. Morgan-Kaufmann, San Francisco, Calif., 297–308.

BECK, H. W., GALA, S. K., AND NAVATHE, S. B. 1989. Classification as a query processing technique in the CANDIDE data model. In *Proceedings of the 5th International Conference on Data Engineering*. IEEE Computer Society, Los Angeles, Calif., 572–581.

BEERI, C., LEVY, A. Y., AND ROUSSET, M. 1997. Rewriting queries using views in description logics. In *Proceedings of the Sixteenth ACM SIG-SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS '97*) (Tucson, Az., May 12–14). ACM, New York, 99–108.

BENEVENTANO, D. 2002. Description logics for semantic query optimization in object-oriented database systems. Tech. rep., Dipartimento di Ingegneria dell'Informazione, Via Vignolese 905—Modena. http://www.dbgroup.unimo.it/TechnicalReport/TechReport2002-1.pdf.

BENEVENTANO, D. AND BERGAMASCHI, S. 1997. Incoherence and subsumption for recursive views and queries in object-oriented data models. *Data Knowl. Eng. 21*, 3 (Feb.), 217–252.

BENEVENTANO, D., BERGAMASCHI, S., LODI, S., AND SARTORI, C. 1998. Consistency checking in complex object database schemata with integrity constraints. *IEEE Trans. Knowl. Data Eng. 10*, 576–598.

BENEVENTANO, D., BERGAMASCHI, S., AND MANDREOLI, F. 2001. Extensional Knowledge for semantic query optimization in a mediator based system. In *Proceedings of the International Workshop on Foundations of Models for Information Integration* (*FMII-2001*) (Viterbo, Italy, Sept.). Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, Germany.

BENEVENTANO, D., BERGAMASCHI, S., AND SARTORI, C. 1996. Semantic query optimization by subsumption in OODB. In *Flexible Query Answering Systems*, H. Christiansen, H. L. Larsen, and

T. Andreasen, Eds. Datalogiske Skrifter—ISSN 0109-9799, vol. 62. Roskilde University, Roskilde, Denmark.

BENEVENTANO, D., BERGAMASCHI, S., SARTORI, C., AND VINCINI, M. 1997. ODB-QOPTIMIZER: A tool for semantic query optimization in oodb. In *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society, Los Angeles, Calif. http://sparc20.dsi.unimo.it.

BERGAMASCHI, S. AND NEBEL, B. 1994. Acquisition and validation of complex object database schemata supporting multiple inheritance. *J. Appl. Intel. 4*, 185–203.

BERGAMASCHI, S. AND SARTORI, C. 1992. On taxonomic reasoning in conceptual design. *ACM Trans. Datab. Syst. 17*, 3 (Sept.), 385–422.

BERTINO, E., NEGRI, M., PELAGATTI, G., AND SBATTELLA, L. 1992. Object-oriented query languages: The notion and the issues. *IEEE Trans. Knowl. Data Eng. 4*, 3 (June), 223–236.

BORGIDA, A., BRACHMAN, R. J., MCGUINNESS, D. L., AND RESNICK, L. A. 1989. CLASSIC: A structural data model for objects. In *SIGMOD* (Portland, Ore.). ACM, New York, 58–67.

BRACHMAN, R. J. AND SCHMOLZE, J. G. 1985. An overview of the KL-ONE knowledge representation system. *Cognit. Sci. 9*, 2, 171–216.

BUCHHEIT, M., JEUSFELD, M. A., NUTT, W., AND STAUDT, M. 1994. Subsumption between queries to object-oriented database. In *Extending Database Technology*. Springer, Heidelberg, Germany, 348–353.

BUNEMAN, P. 1997. Semistructured data. In *Proceedings of the 16th Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Tucson, Az., May 12–14). ACM, New York, 117–121.

BUNEMAN, P., DAVIDSON, S., FERNANDEZ, M., AND SUCIU, D. 1997. Adding structure to unstructured data. In *Database Theory—ICDT'97, 6th International Conference* (Delphi, Greece), F. N. Afrati and P. Kolaitis, Eds. Lecture Notes in Computer Science, vol. 1186. Springer-Verlag, Heidelberg, Germany, 336–350.

CALVANESE, D., GIACOMO, G. D., AND LENZERINI, M. 1998. On the decidability of query containment under constraints. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Seattle, Wash., June 1–8). ACM New York, 149–158.

CALVANESE, D., GIACOMO, G. D., LENZERINI, M., AND VARDI, M. Y. 1999. Rewriting of regular expressions and regular path queries. In *Proceedings of the 18th Annual ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Philadelphia, Pa., May 31–June 2). ACM New York, 194–204.

CARDELLI, L. 1984. A semantics of multiple inheritance. In *Semantics of Data Types*. Lecture Notes in Computer Science, vol. 173. Springer-Verlag, Heidelberg, Germany, 51–67.

CATTELL, R. G. G. 1994. *The Object Database Standard: ODMG93*. Morgan-Kaufmann, San Mateo, CA.

CHAKRAVARTHY, U. S., GRANT, J., AND MINKER, J. 1990. Logic-based approach to semantic query optimization. *ACM Trans. Datab. Syst. 15*, 2 (June), 162–207.

CHAN, E. P. F. 1992. Containment and minimization of positive conjunctive queries in OODB's. In *Proceedings of the ACM Symposium on Principles of Database Systems*. ACM, New York, 202–211.

COBURN, N. AND WEDDEL, G. E. 1991. Path constraints for graph-based data models: Towards a unified theory of typing constraints, equations and functional dependencies. In *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases*. Springer-Verlag, Heidelberg, Germany, 312–331.

DEN BUSSCHE, J. V. AND VOSSEN, G. 1993. An extension of path expressions to simplify navigation in object-oriented queries. In *Proceedings of the 3rd International Conference on Deductive and Object-Oriented Databases* (Phoenix, Az., Dec.). Lecture Notes in Computer Science, vol. 760. Sringer-Verlag, New York, 267–282.

DONINI, F. M., LENZERINI, M., NARDI, D., AND NUTT, W. 1991. The complexity of concept languages. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning* (KR '91). J. Allen, R. Fikes, and E. Sandewall, Eds. Morgan-Kaufmann, Cambridge, Mass, 151–162.

DONINI, F. M., SCHAERF, A., AND BUCHHEIT, M. 1993. Decidable reasoning in terminological knowledge representation systems. In *Proceedings of the 13th International Joint Conference on Artificial Inteligence*. Morgan-Kaufmann, San Francisco, Calif.

FERNANDEZ, M. F. AND SUCIU, D. 1998. Optimizing regular path expressions using graph schemas. In *Proceedings of the 14th International Conference on Data Engineering* (Orlando, Fla., Feb. 23–27). IEEE Computer Society, Los Angeles, Calif., 14–23.

FLORESCU, D., LEVY, A. Y., AND SUCIU, D. 1998. Query containment for conjunctive queries with regular expressions. In *Proceedings of the 17th Annual ACM SIGACT-SIGMOD-SIGART Principles of Database Systems* (*PODS '98*) (Seattle, Wash., June 1–3). ACM, New York, pp. 139–148.

GOLDMAN, R. AND WIDOM, J. 1997. Dataguides: Enabling query formulation and optimization in semistructured data. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases* (Athens, Greece, Aug. 25–29). M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, Eds. Morgan-Kaufmann, San Francisco, Calif., 436–445.

GRANT, J., GRYZ, J., MINKER, J., AND RASCHID, L. 1997. Semantic query optimization for object databases. In *Proceedings of the 13th International Conference on Data Engineering (ICDE'97)*. IEEE, Washington - Brussels - Tokyo, 444–454.

HACID, M. AND RIGOTTI, C. 1995. Combining resolution and classification for semantic query optimization in dood. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*. Springer-Verlag, Heidelberg, Germany, 447–466.

HAMMER, M. M. AND ZDONIK, S. B. 1980. Knowledge based query processing. In *Proceedings of the 6th International Conference on Very Large Databases*. IEEE Computer Society, Los Angeles, Calif., 137–147.

HOPCROFT, J. E. AND ULLMAN, J. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, Mass.

JAGADISH, H. V., AGRAWAL, R., AND NESS, L. 1987. A study of transitive closure as a recursion mechanism. In *Proceedings of the 1987 Annual Conference for ACM SIGMOD* (San Francisco, Calif., May 27–29). ACM New York, pp. 331–344.

JARKE, M., CLIFFORD, J., AND VASSILIOU, Y. 1984. An optimizing prolog front-end to a relational query system. In *SIGMOD'84, Proceedings of Annual Meeting* (Boston, Mass., June 18–21). B. Yormark, Ed. ACM, New York, 296–306.

JEUSFELD, M. AND STAUDT, M. 1993. Query optimization in deductive object bases. In *Query Processing for Advanced Database System*, Freytag, Maier, and Vossen, Eds. Morgan-Kaufmann Publishers, Inc., San Francisco, Calif.

KEMPER, A. AND MOERKOTTE, G. 1990. Access support in object bases. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data* (Atlantic City, N.J., May 23–25). H. Garcia-Molina and H. V. Jagadish, Eds. ACM Press, New York, 510–517.

KIFER, M., KIM, W., AND SAGIV, Y. 1992. Querying object-oriented databases. In *Proceedings of the 1992 Annual Conference for ACM SIGMOD (SIGMOD '92)*. ACM, New York, 393–402.

KING, J. J. 1981. QUIST: a system for semantic query optimization in relational databases. In *Proceedings of the 7th International Conference on Very Large Databases*. IEEE Press, Los Angeles, Calif., 510–517.

KOLAITIS, P. G. AND VARDI, M. Y. 1998. Conjunctive-query containment and constraint satisfaction. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Seattle, Wash., June 1–3). ACM, New York, pp. 205–213.

LECLUSE, C. AND RICHARD, P. 1989. Modelling complex structures in object-oriented databases. In *Proceedings of the Symposium on Principles of Database Systems* (Philadelphia, Pa). ACM, New York, 362–369.

LEVY, A. Y., RAJARAMAN, A., AND ORDILLE, J. J. 1996. Querying heterogeneous information sources using source descriptions. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases* (Mumbai (Bombay), India, Sept. 3–6). T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Morgan-Kaufmann, San Francisco, Calif., 251–262.

LEVY, A. Y. AND SUCIU, D. 1997. Deciding containment for queries with complex objects (extended abstract). In *PODS '97. Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Tucson, Az., May 12–14). ACM, New York, 20–31.

NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. 1997. Inferring structure in semistructured data. *SIGMOD Record (ACM Special Interest Group on Management of Data) 26*, 4, 39–43.

NESTOROV, S., ABITEBOUL, S., AND MOTWANI, R. 1998. Extracting schema from semistructured data. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data* (Seattle, Wash., June 2–4). L. M. Haas and A. Tiwary, Eds. ACM, New York, 295–306.

PANG, H. H., LU, H., AND OOI, B. C. 1991. An efficient semantic query optimization algorithm. In *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society, Los Angeles, Calif., 326–335.

PERRIN, D. 1990. Finite automata. In *Formal Models and Semantics*, J. van Leeuwen, Ed. Elsevier, Amsterdam, Holland, 14–19.

SHENOY, S. T. AND ÖZSOYOGLU, Z. M. 1987. A system for semantic query optimization. In *Proceedings of the 1987 Annual Conference for ACM SIGMOD* (San Francisco, Calif., May 27–29). U. Dayal and I. L. Traiger, Eds. ACM, New York, pp. 181–195.

SHENOY, S. T. AND ÖZSOYOGLU, Z. M. 1989. Design and implementation of a semantic query optimizer. *IEEE Trans. Knowl. Data. Eng. 1*, 3, 344–361.

SHEU, P. C., KASHYAP, R. L., AND YOO, S. 1989. Query optimization in object-oriented knowledge bases. *Data Knowl. Eng. 3*, 285–302.

SHMUELI, O. 1993. Equivalence of datalog queries is undecidable. *J. Logic Prog. 15*, 3 (Feb.), 231–241.

SIEGEL, M., SCIORE, E., AND SALVETER, S. 1992. A method for automatic rule derivation to support semantic query optimization. *ACM Trans. Datab. Syst. 17*, 4 (Dec.), 563–600.

STRAUBE, D. D. AND ÖZSU, T. 1990. Queries and query processing in object-oriented database systems. *ACM Trans. Inf. Syst. 8*, 4 (Oct.), 387–430.

SUN, W. AND CLEMENT, T. Y. 1994. Semantic query optimization for tree and chain queries. *IEEE Trans. Knowl. Data Eng. 6*, 1 (Feb.), 136–151.

ULLMAN, J. D. 1997. Information integration using logical views. In *Preceedings of Database Theory—ICDT '97, 6th International Conference* (Delphi, Greece, Jan. 8–10). F. N. Afrati and P. Kolaitis, Eds. Lecture Notes in Computer Science, vol. 1186. Springer-Verlag, Heidelberg, Germany, 19–40.

YOON, J. P. AND KERSCHBERG, L. 1993. Semantic query optimization in deductive object-oriented databases. In *Proceedings of the 3rd International Conference on Deductive and Object-Oriented Databases (DOOD'93)* (Phoenix, Az., Dec. 6–8). S. Ceri, K. Tanaka, and S. Tsur, Eds. Lecture Notes in Computer Science, vol. 760. Springer-Verlag, New York, pp. 169–182.

YOON, S.-C., SONG, I.-Y., AND PARK, E. K. 1995. Semantic query processing in object-oriented databases using deductive approach. In *CIKM '95, Proceedings of the 1995 International Conference on Information and Knowledge Management* (Baltimore, Md., Nov. 28–Dec. 2). ACM, New York, 150–157.

ZHUGE, Y., GARCIA-MOLINA, H., HAMMER, J., AND WIDOM, J. 1995. View maintenance in a warehousing environment. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (San Jose, Calif., May 22–25). M. J. Carey and D. A. Schneider, Eds. ACM, New York, 316–327.

Received February 2000; revised December 2001; accepted August 2002