Subsumption between Queries to Object-Oriented Databases

Martin Buchheit, Manfred A. Jeusfeld
Werner Nutt, Martin Staudt

RR-93-44
Research Repor

Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-93-44

Subsumption between Queries to Object-Oriented Databases

Martin Buchheit, Manfred A. Jeusfeld
Werner Nutt, Martin Staudt

November 1993

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341

Subsumption between Queries to Object-Oriented Databases

Martin Buchheit, Manfred A. Jeusfeld
Werner Nutt, Martin Staudt

DFKI-RR-93-44

A short version of this report appeared in the Proceedings of the 4th International Conference on Extending Database Technology, EDBT'94. A long version appeared in a special issue of INFORMATION SYSTEMS 1994.

This work has been supported by a grant from The Federal Ministry for Research and Technology (FKZ ITWM-9201).

ISSN 0946-008X

Subsumption between Queries to Object-Oriented Databases

Martin Buchheit[†] Manfred A. Jeusfeld[‡] Werner Nutt[†]
Martin Staudt[‡]

[†]German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{buchheit,nutt}@dfki.uni-sb.de,

[‡]RWTH Aachen, Informatik V, Ahornstr. 55, D-52056 Aachen, Germany,
{jeusfeld,staudt}@informatik.rwth-aachen.de

Abstract

Most work on query optimization in relational and object-oriented databases has concentrated on tuning algebraic expressions and the physical access to the database contents. The attention to semantic query optimization, however, has been restricted due to its inherent complexity. We take a second look at semantic query optimization in object-oriented databases and find that reasoning techniques for concept languages developed in Artificial Intelligence apply to this problem because concept languages have been tailored for efficiency and their semantics is compatible with class and query definitions in object-oriented databases. We propose a query optimizer that recognizes subset relationships between a query and a view (a simpler query whose answer is stored) in polynomial time.

Contents

1 Introduction

From an abstract viewpoint, databases organize information into sets of objects. In object-oriented databases (OODB's), these sets are called *classes* and the elements are constrained by some—not too complicated—type expression.

Similar expressions for describing classes of objects—so-called *concept descriptions* or simply *concepts*—have been investigated in Artificial Intelligence (AI), where they occur in knowledge representation languages of the KL-ONE family [WS92]. This research has come up with techniques to determine satisfiability and subsumption of concepts and has assessed the complexity of these inferences for a variety of languages (see *e.g.*, [DLNN91a]).

We believe that this similarity between work on databases and AI offers a potential of cross-fertilization: database research can profit from reasoning techniques for concepts, and knowledge representation research can learn about properties of practically applied set descriptions like class schemas and queries in OODB's.

The Problem. Query optimization is largely reasoning on intensional representations, especially queries and schema information. Therefore, this area appears as a natural choice for proving the hypothesis of cross-fertilization. We assume that object-oriented queries return sets of objects. The problem we want to solve in this paper is the *subsumption problem* for queries:

> *Given the schema of an OODB and two queries, decide whether in every possible state of the database the answer set of the first query is contained in the answer set of the second.*

If the answers to the second query are stored subsumption can be exploited to speed up evaluation of the first query by filtering the stored objects, instead of computing the answers from scratch. Such a situation is likely to occur in an environment where views are materialized and maintained to be up-to-date. For this reason we will assume that the second query in a subsumption problem defines a view. By abuse of terminology, we will simply refer to it as a *view*.

OODB's offer increased opportunities to reuse queries: their schema is usually richer, *i.e.*, more detailed, than the schema of relational databases. In particular, attribute values are constrained by types or classes. Classes may have subclasses with additional properties and constraints. We argue that this information should be utilized for query optimization.

In order to exploit subsumption of queries by views in a real system, many views might have to be checked for a given query, and checks will be performed for every incoming query. Hence, efficiency is an issue and subsumption checks should run in polynomial time.

The Schema and Query Language. Regrettably, there exists neither a standardized object-oriented data model nor a standardized object-oriented query language. We therefore present a language that features just the properties which are common to most object-oriented data models *and* are relevant to our purpose.

- *Membership of objects in classes:* Classes group a finite set of objects (their instances). In most systems, membership of an object in a class is constrained by the type of the class. Here, we assume that the condition for membership can be expressed in first order logic.
- *Subclass relationships:* Classes are organized in a subclass hierarchy. Any instance of a class is also an instance of the superclasses.
- *Attribute declarations:* Objects may have attributes. Attributes are set-valued. The domain and range of an attribute are restricted by classes. For a subclass of its domain, an attribute may be restricted to take values in a subclass of its range.
- *Number restrictions:* Attributes can be specified as functional, *i.e.*, as having at most one value, or as necessary, *i.e.*, as having at least one value. The last possibility is very important for OODB's because it prevents method execution from errors when accessing such attributes.

Our Approach to Solving the Problem. The expressiveness of query and schema languages in OODB's makes a general solution of our problem impossible. Thus, it has to be relaxed to a simpler problem. We identify portions of the schema and of queries which can be mapped to a concept language where subsumption can be decided efficiently.

Concept languages[1] bear strong similarities to languages for defining schemas and queries of OODB's. Concepts are intensional descriptions of sets of objects built from primitive concepts and attributes. Complex concepts can be constructed from existing ones by set-theoretic operations like intersection, union, and complement, and by imposing restrictions on the fillers of attributes. Attributes can be formed, *e.g.*, as inverses, chains, or intersections of other attributes. Concept languages express fragments of first-order logic: concepts can be viewed as certain logical formulas that are built using unary and binary predicates and contain one free variable (to be bound by the instances of the concept). Subsumption of concepts has been studied for a variety of languages and the borderline between variants where reasoning is tractable and where not is by now well understood (see *e.g.*, [DLNN91b]).

Three points make the ideas of our approach more precise:

[1]Concept languages are also known as *Terminological Logics* or *Description Logics.*

- *Incorporation of structural schema information:* Object-oriented schemas are comparatively large. We identify a so-called *structural part* where inheritance hierarchies, attribute typing and restrictions on attributes are specified. This information can be mapped to a concept language and is explicitly employed for checking subsumption between queries.

- *Structural and non-structural parts of queries:* A query, too, is separated into a *structural* part that can be represented within the concept language, and a *non-structural* part that goes beyond concept language expressiveness. Features like subclassing, path expressions, and coreferences between paths fall into the structural part.

- *Views have only structural parts:* In this paper, views are queries whose answers are materialized. The answers to a view can be used for optimizing queries subsumed by the view. In order to keep our approach sound, views must be captured completely by a concept. Therefore, we forbid non-structural parts for views.

We have designed our concept language so that it covers the core of a general schema and query language and at the same time allows for a polynomial time subsumption checker. Since we ignore information that cannot be expressed in the concept language, not all valid subsumptions are detected: we sacrifice completeness for efficiency. However, we expect the "hit rate" to be high enough for justifying the effort because we take the main schema information into account and the structural fragment of the query language is strong enough to express interesting queries and views.

The following section defines a generic syntax and semantics for object-oriented schemas and queries. Section 3 introduces a concept language for the structural parts of schemas and queries. A subsumption calculus together with a proof for it completeness and soundness is elaborated in Section 4. Section 5 discusses related work and Section 6 concludes.

2 Object-Oriented Databases and Queries

In this section, we introduce a simple frame-like database language $\mathcal{DL}$ which on the one hand provides a generic data model for OODB's and on the other hand has a simple first-order semantics. The language incorporates the three basic abstraction principles: *classification*, *generalization*, and *aggregation*. Subsection 2.1 describes the language used for defining the database schema. In Subsection 2.2 we show how queries can naturally be represented in this framework as special classes.

```
Class Patient isA Person with    Class Person with                Attribute skilled_in with
  attribute                        attribute, necessary, single     domain: Person
    takes: Drug                      name: String                   range: Topic
    consults: Doctor               end Person                       inverse: specialist
  attribute, necessary                                            end skilled_in
    suffers: Disease               Class Doctor with
  constraint:                        attribute
    not (this in Doctor)               skilled_in: Disease
end Patient                        end Doctor
```

Figure 1: A part of the schema of a medical database

2.1 Defining an OODB Schema

A $\mathcal{DL}$ schema consists of a set of attribute and class declarations. For example, Figure 1 demonstrates part of the schema of a medical database containing patients who are persons suffering from diseases, taking drugs, and consulting doctors.[2]

Attributes are typed binary relations with specific classes as domains and ranges. It is allowed to define synonyms for the inverse of an attribute. Thus skilled_in is a relation between persons and general topics whose inverse is called specialist. Synonyms for attributes are not allowed to occur in other declarations of a schema, but are useful for formulating queries. *Classes* group objects, which are restricted by specific conditions. These conditions are necessary, but not sufficient for class membership, *i.e.*, an object is not automatically recognized as a member if it satisfies the restrictions of a class. There are three kinds of restrictions.

Classes may be specializations, respectively *generalizations*, of other classes. This is indicated by isA-statements in class declarations. In our example, Patient is a subclass of Person. As a consequence, in any legal state of the database, every patient must be a person. There is a most general class Object containing any object of the database.

Class declarations state typical properties of their members (*aggregation*), which are expressed through attributes. For members of the class declared, the values of an attribute may be restricted to a subclass of the attribute's range. Thus, in any legal state of our example database, only members of class Disease are admissible as values for the skilled_in attribute of a Doctor. Besides this typing condition, attributes of a class may be declared as mandatory (necessary) or functional (single). In our example, a patient always must suffer from at least one disease, while a person must have exactly one name.

General integrity constraints that class members have to obey can be stated in a

[2]A complete schema must contain a declaration for every class and attribute. A completion of our example would therefore contain additional declarations for the classes Drug, Disease, String, and Topic, as well as for the attributes consults, name, suffers, and takes.

$$\forall x.\ \mathsf{Patient}(x) \Rightarrow \mathsf{Person}(x)$$
$$\forall x,y.\ \mathsf{Patient}(x) \wedge \mathsf{takes}(x,y) \Rightarrow \mathsf{Drug}(y)$$
$$\forall x,y.\ \mathsf{Patient}(x) \wedge \mathsf{consults}(x,y) \Rightarrow \mathsf{Doctor}(y)$$
$$\forall x,y.\ \mathsf{Patient}(x) \wedge \mathsf{suffers}(x,y) \Rightarrow \mathsf{Disease}(y)$$
$$\forall x.\ \mathsf{Patient}(x) \Rightarrow \exists y\, \mathsf{suffers}(x,y)$$
$$\forall x.\ \mathsf{Patient}(x) \Rightarrow \neg\, \mathsf{Doctor}(x)$$
$$\forall x,y.\ \mathsf{skilled\_in}(x,y) \Rightarrow \mathsf{Person}(x) \wedge \mathsf{Topic}(y)$$
$$\forall x,y.\ \mathsf{skilled\_in}(x,y) \Leftrightarrow \mathsf{specialist}(y,x)$$

Figure 2: Translating declarations into logic

constraint clause. Constraints are formulated in a first-order many sorted language where quantifiers are restricted to range over classes. The only atoms allowed in this language are (x in C), denoting membership of x in class C, and (x a y), assigning y as value for attribute a to object x. The variable this is implicitly universally quantified and ranges over the class that is currently declared. Thus, the example constraint on the class Patient forbids a person to be both a patient and a doctor. A database designer is free to associate a constraint to any class he finds suitable. In our example, he could have attached equally well a constraint not (this in Patient) to the class Doctor.

In the following, we will refer to the subclass and attribute part of a class declaration as the *structural* and to the constraint part as the *non-structural* part. The latter is not taken into account by the abstraction in the Section 3.

We will not go into the details of how to specify a *state* of a $\mathcal{DL}$ database. This can be done *e.g.*, by using similar frame-like constructs relating objects to classes by instance-relationships (*classification*) and to each other by assigning values to attributes defined for these classes.

The semantics of the language is given by a mapping from attribute and class declarations to first-order formulas, where class names appear as unary and attribute names as binary predicates. Facts about database objects are mapped to sets of ground atoms built from these predicates. We do not specify the mapping formally, but as an example translate in Figure 2 the declarations of the class Patient and the attribute skilled_in.

We assume that every state of the database gives rise to exactly one model of these formulas. This might be achieved in several ways: either all facts are explicitly stated, or some schema formulas are employed as deductive rules, by which additional facts are derived (see *e.g.*, [SNJ93]). The important point is that every state defines a unique structure that satisfies the schema.

2.2 Query Classes

Querying a database means retrieving stored objects that satisfy certain restrictions or qualifications and hence are interesting for a user. In relational databases, queries are constructed by algebra expressions involving relations from the database, and answers again are relations, *i.e.*, sets of tuples. The correspondence between database and answer format has obvious advantages. In OODB's classes are used to represent sets of objects and thus it is natural to use them also for describing query results.

Object-oriented data models disagree as to whether new objects can be created as answers to queries or not (see *e.g.*, [AK89]). In the object model presented here we restrict answer objects to existing objects[3] that are deduced as instances of so-called *query classes*. In contrast to the classes constituting the database schema the membership conditions in the declarations of query classes are necessary *and* sufficient. Thus, they are completely defined by the declaration, and objects can be recognized as instances although they have not explicitly been entered as such.

An example of a query class is given in Figure 3. Just as schema classes, query classes may be specializations of other classes, especially query classes, and answer objects must be common instances of all superclasses. Hence, every schema class can be turned into a query class.

In order to express more specific conditions on answer objects, so-called *derived* objects can be specified in the derived clause through labeled paths. A *labeled path* is a labeled chain of attributes[4] with value restrictions and has the form

$$\mathsf{l}_j\text{: } (\mathsf{a}_1\text{: } \mathsf{C}_1).(\mathsf{a}_2\text{: } \mathsf{C}_2).\ldots.(\mathsf{a}_n\text{: } \mathsf{C}_n),$$

where l_j is the label, the a_i's are attributes, and each C_i is a class D or a singleton set $\{\mathsf{i}\}$. An example of a labeled path would be l_2: (suffers: Object).(specialist: Doctor). Labels stand for derived objects. A restricted attribute (a: D) ((a: $\{\mathsf{i}\}$)) relates all objects x, y in the database such that y is an instance of D ($y = \mathsf{i}$) and an a-value of x. If an attribute a is only restricted by the universal class Object we write a as a shorthand for (a: Object). The chain of restricted attributes in a labeled path can be conceived as a new attribute obtained by composing the components of the chain. Intuitively, for a given object, a chain denotes the set of objects that can be reached following it. The label of a path can be viewed as a variable ranging over this set, and for an object to be an instance of the query class this variable has to be bound to some element of the set. Summarizing, our labeled paths generalize the common notion of paths (see *e.g.*, [KKS92]) in that they allow one to filter the values of an attribute after each step. Moreover, one can access the values at the end of the path through a label.

[3]For simplicity we ignore additional output attributes of answer objects in this paper.

[4]Attribute synonyms defined in the schema are allowed in paths, too.

```
QueryClass QueryPatient isA Male, Patient with                    (1)
   derived
      l_1: (consults: Female)                                     (2)
      l_2: suffers.(specialist: Doctor)                           (3)
   where
      l_1 = l_2                                                   (4)
   constraint:
      forall d/Drug not (this takes d) or (d = Aspirin)           (5)
end QueryPatient
```

Figure 3: A query

The **where** clause contains equalities $l_j = l_k$ between labels,[5] which the derived objects have to satisfy.

Finally, query classes contain a **constraint** clause (again called *non-structural part*) where additional conditions for class membership are specified by a logical formula similar to the one in schema class constraints. In the formula the labels l_j may appear again. The variable **this** refers to the answer object itself. Labels that occur neither in the **where** nor the **constraint** clause may be omitted in the **derived** part.

The syntax of query classes has been designed in such a way that the structural part is strong enough to formulate interesting queries while offering only constructs that can be mapped immediately to our concept language (see Section 3). An alternative approach to separating queries into two parts would be to offer a homogeneous language and to automatically extract from a query the portions to be handled by a subsumption checker. However, this seems to be so difficult a task that we prefer a hybrid syntax for queries.

In the example, QueryPatient retrieves all patients from the database who consult a female who is a doctor and a specialist in the disease from which the patient is suffering. In addition, the patients do not take any drug except Aspirin.

Again, the semantics of query classes is given by a translation into a predicate logic formula. The formula conjoins the membership predicates for the superclasses, subformulas gained from the labeled paths, equalities, and a straightforward rewriting of the constraint. The query QueryPatient yields the formula given in Figure 4. Each conjunct corresponds to the clause with the same number within the definition of QueryPatient. In a framework that combines deductive databases and object-orientedness, the translated query class can be readily executed (see [SNJ93]).

As can be seen from the example, a query class whose **constraint** part is empty is logically equivalent to a conjunction of atoms where certain variables are existentially

[5]In order to keep the presentation of our algorithm in Section 4 as simple as possible we will consider only the case that a label occurs no more than once in the where clause. Dropping this restriction would still allow for a polynomial algorithm.

$$
\begin{array}{ll}
\mathsf{QueryPatient}(t) \iff \mathsf{Male}(t) \land \mathsf{Patient}(t) \land & (1)\\
\quad \exists l_1, l_2. & \\
\qquad \mathsf{consults}(t, l_1) \land \mathsf{Female}(l_1) \land & (2)\\
\qquad \exists x. \, (\mathsf{suffers}(t, x) \land \mathsf{specialist}(x, l_2) \land \mathsf{Doctor}(l_2)) \land & (3)\\
\qquad l_1 \doteq l_2 \land & (4)\\
\qquad \forall d. \, (\mathsf{Drug}(d) \Rightarrow \neg \mathsf{takes}(t, d) \lor d \doteq \mathsf{Aspirin}) & (5)
\end{array}
$$

Figure 4: Translating the query into logic

quantified. In the context of relational and deductive databases such queries are known as *conjunctive queries* (see [Ull89]).

Based on the entire conceptual schema of a database, users and application programs usually work on subschemas that constitute their *external view* on the database. A common approach (*e.g.*, in SQL) is to use for the definition of such views a sublanguage of the query language. In the same vein, we will assume that views on $\mathcal{DL}$ databases are only defined through structural queries, *i.e.*, queries whose constraint part is empty.

If views are used frequently and the computation of their extension is expensive they can be *materialized*. Materialization means that membership of objects in a view, although derivable from the database by means of the view definition (and hence redundant), is explicitly stored. On the one hand, direct access to materialized views is as fast as to any other class defined in the schema. On the other hand, since views are just special queries with stored answers, the detection that a view subsumes a query, allows one to profit from this fast access by restricting the search space for query evaluation just to the stored instances of the view.

Let us extend our example by a second query class, defined in Figure 5. The class ViewPatient is a view defining another subset of patients in the database, namely those whose name is stored and that consult a doctor who is a specialist for one of their diseases. At first glance it is not obvious whether the ViewPatient subsumes QueryPatient. However, if one takes into account the schema information that (1) every person and hence every patient has a name, that (2) patients suffer from diseases, and (3) the attributes skilled_in and specialist are inverses of each other, moreover, if one joins the paths as required by the labels, one realizes that every instance of QueryPatient is also an instance of ViewPatient.

3 From Queries to Concepts

In this section we introduce two languages for describing schemas and queries, $\mathcal{SL}$ and $\mathcal{QL}$, respectively, that are abstractions of $\mathcal{DL}$. While the frame-based syntax of $\mathcal{DL}$ is user-oriented and similar to that of languages in existing OODB systems the syntax of the abstract languages is well suited for the design, verification and

```
QueryClass ViewPatient isA Patient with
   derived
         (name: String)
     l_1: (consults: Doctor).(skilled_in: Disease)
     l_2: (suffers: Disease)
   where
     l_1 = l_2
end ViewPatient
```

Figure 5: A view

complexity analysis of algorithms. It is inspired by KL-ONE-like concept languages and employs a variable-free notation which is semantically equivalent to certain logical formulas. The lack of variables keeps these languages close to the structural part of the concrete language while the explicit use of quantifiers resolves ambiguities present in $\mathcal{DL}$. We will show how to represent in the abstract languages the structural parts of class and attribute declarations by a set of *schema axioms* and query classes as *concepts*. Then we reformulate the key problem of the paper for the new languages.

3.1 Preliminaries

The elementary building blocks $\mathcal{SL}$ and $\mathcal{QL}$ are primitive concepts (ranged over by the letter A) and primitive attributes (ranged over by P). Intuitively, concepts describe sets and thus correspond to unary predicates while attributes describe relations and thus correspond to binary predicates. We assume also that an alphabet of *constants* (ranged over by a, b, c) is given. Different constant symbols are interpreted as distinct objects (Unique Name Assumption).

In the *schema language* $\mathcal{SL}$ attributes must be primitive. Concepts (ranged over by C, D, E) in $\mathcal{SL}$ are formed according to the following syntax rule:

$$\begin{array}{rcll} C, D, E & \longrightarrow & A \mid & \text{(primitive concept)} \\ & & \forall P.\, A \mid & \text{(typing of attribute)} \\ & & \exists P \mid & \text{(necessary attribute)} \\ & & (\leq 1\; P) & \text{(single-valued attribute).} \end{array}$$

Schema axioms come in the two forms

$$A \sqsubseteq D, \qquad P \sqsubseteq A_1 \times A_2,$$

where A, A_1, A_2 are primitive concepts, D is an arbitrary $\mathcal{SL}$ concept, and P a primitive attribute. The first axiom states that all instances of A are instances of D. So D gives necessary conditions for membership in A. The second axiom states that the attribute P has domain A_1 and range A_2. An $\mathcal{SL}$ *schema* Σ consists of a set of schema axioms. As we will see below, by means of a schema we can represent attribute declarations and the structural part of class declarations.

In the *query language* $\mathcal{QL}$, attributes (ranged over by R) can be primitive attributes P or inverses P^{-1} of primitive attributes. Furthermore, there are *attribute restrictions*, written $(R\colon C)$, where R is an attribute and C is a $\mathcal{QL}$ concept. Intuitively, $(R\colon C)$ restricts the pairs related by R to those whose second component satisfies C. *Paths* (ranged over by p, q) are chains $(R_1\colon C_1)\cdots(R_n\colon C_n)$ of attribute restrictions and stand for the composition of the restricted attributes. The empty path is denoted as ϵ. In $\mathcal{QL}$, concepts are formed according to the rule:

$$
\begin{array}{lcll}
C, D, E & \longrightarrow & A \mid & \text{(primitive concept)} \\
 & & \top \mid & \text{(universal concept)} \\
 & & \{a\} \mid & \text{(singleton set)} \\
 & & C \sqcap D \mid & \text{(intersection)} \\
 & & \exists p \mid & \text{(existential quantification over path)} \\
 & & \exists p \doteq q & \text{(existential agreement of paths).}
\end{array}
$$

The intersection of concepts denotes the intersection of sets, the existential quantification over a path denotes those objects from which some object can be reached along the path, and the existential agreement of paths denotes those objects that have a common filler for the two paths. Observe that concepts and paths can be arbitrarily nested through attribute restrictions.

In Table 1 we present the semantics of attributes and concepts in two steps. As in the previous section, we translate concepts, attributes, and paths into first order formulas (column 2). Then we give a semantics that treats concepts as set descriptions (column 3).

For the transformational semantics we map primitive concepts A and primitive attributes P to atoms $A(\gamma)$ and $P(\alpha, \beta)$. Then column 2 contains for each complex concept C, attribute restriction Q and path p appearing in column 1 a corresponding formula $F_C(\gamma)$, $F_Q(\alpha, \beta)$ and $F_p(\alpha, \beta)$ that has one or two free variables, respectively.

Given a fixed interpretation, each such formula denotes a binary or unary relation over the domain. Thus we can immediately formulate the semantics of attributes and concepts in terms of relations and sets without the detour through predicate logic notation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *extension function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$, every constant to an element of $\Delta^{\mathcal{I}}$, and every attribute to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. In accordance with the Unique Name Assumption we assume that distinct constants have distinct images. Given the denotation of primitive attributes and concepts, complex ones are interpreted according to the third column of Table 1 ("$\sharp\cdot$" denotes the cardinality of a set). It is easy to verify that column 3 gives the sets of pairs and objects for which the formulas in column 2 hold.

Construct	FOL Semantics	Set Semantics
$\forall P.\, A$	$\forall x.\, P(\gamma, x) \Rightarrow A(x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \forall d_2.\, (d_1, d_2) \in P^{\mathcal{I}} \Rightarrow\ d_2 \in A^{\mathcal{I}}\}$
$\exists P$	$\exists x.\, P(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in P^{\mathcal{I}}\}$
$(\leq 1\, P)$	$\forall x, y.\, P(\gamma, x) \wedge P(\gamma, y) \Rightarrow x \doteq y$	$\left\{d_1 \in \Delta^{\mathcal{I}} \;\middle|\; \sharp\{d_2 \mid (d_1, d_2) \in P^{\mathcal{I}}\} \leq 1 \right\}$
P^{-1}	$P(\beta, \alpha)$	$\{(d_2, d_1) \mid (d_1, d_2) \in P^{\mathcal{I}}\}$
$(R{:}C)$	$F_R(\alpha, \beta) \wedge F_C(\beta)$	$\{(d_1, d_2) \mid (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}$
ϵ	$\alpha \doteq \beta$	$\{(d_1, d_1) \mid d_1 \in \Delta^{\mathcal{I}}\}$
Qp	$\exists z.\, F_Q(\alpha, z) \wedge F_p(z, \beta)$	$\{(d_1, d_3) \mid \exists d_2.\, (d_1, d_2) \in Q^{\mathcal{I}} \wedge (d_2, d_3) \in p^{\mathcal{I}}\}$
$\top$	true	$\Delta^{\mathcal{I}}$
$\{a\}$	$\gamma \doteq a$	$\{a^{\mathcal{I}}\}$
$C \sqcap D$	$F_C(\gamma) \wedge F_D(\gamma)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists p$	$\exists x.\, F_p(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in p^{\mathcal{I}}\}$
$\exists p \doteq q$	$\exists x.\, F_p(\gamma, x) \wedge F_q(\gamma, x)$	$\{d_1 \in \Delta^{\mathcal{I}} \mid \exists d_2.\, (d_1, d_2) \in p^{\mathcal{I}} \wedge (d_1, d_2) \in q^{\mathcal{I}}\}$

Table 1: Transformational and set semantics of $\mathcal{SL}$ and $\mathcal{QL}$.

We say that two concepts C, D are equivalent if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$, *i.e.*, equivalent concepts always describe the same sets.

We say that an interpretation $\mathcal{I}$ *satisfies* the axiom $A \sqsubseteq D$ if $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and the axiom $P \sqsubseteq A_1 \times A_2$ if $P^{\mathcal{I}} \subseteq A_1^{\mathcal{I}} \times A_2^{\mathcal{I}}$. If Σ is an $\mathcal{SL}$ schema, an interpretation $\mathcal{I}$ that satisfies all axioms in Σ is called a Σ-*interpretation.* A concept C is Σ-*satisfiable* if there is a Σ-interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. We say that C is Σ-*subsumed* by D (written $C \sqsubseteq_{\Sigma} D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every Σ-interpretation $\mathcal{I}$.

3.2 The Concrete versus the Abstract

Next, we show by an example how to represent the structural part of a $\mathcal{DL}$ schema by a set of schema axioms and the structural part of a query class by a $\mathcal{QL}$ concept. Figure 6 gives the translation of our medical database schema from Figure 1 into schema axioms.

We demonstrate the transformation of query classes into $\mathcal{QL}$ concepts by translating as an example the query classes QueryPatient and ViewPatient from Subsection 2.2 into concepts C_Q and D_V. The inverses of attributes have to be made explicit. For instance, instead of using the attribute specialist one has to take skilled_in^{-1}.

$\mathsf{Patient} \sqsubseteq \mathsf{Person}$	$\mathsf{Person} \sqsubseteq \forall \mathsf{name}.\,\mathsf{String}$	$\mathsf{skilled\_in} \sqsubseteq \mathsf{Person} \times \mathsf{Topic}$
$\mathsf{Patient} \sqsubseteq \forall \mathsf{takes}.\,\mathsf{Drug}$	$\mathsf{Person} \sqsubseteq \exists \mathsf{name}$	
$\mathsf{Patient} \sqsubseteq \forall \mathsf{consults}.\,\mathsf{Doctor}$	$\mathsf{Person} \sqsubseteq (\leq 1\ \mathsf{name})$	
$\mathsf{Patient} \sqsubseteq \forall \mathsf{suffers}.\,\mathsf{Disease}$		
$\mathsf{Patient} \sqsubseteq \exists \mathsf{suffers}$	$\mathsf{Doctor} \sqsubseteq \forall \mathsf{skilled\_in}.\,\mathsf{Disease}$	

Figure 6: Schema axioms of the medical database

$$
\begin{aligned}
C_Q &= \mathsf{Male} \sqcap \mathsf{Patient} \sqcap \\
&\quad \exists(\mathsf{consults}\colon \mathsf{Female}) \doteq (\mathsf{suffers}\colon \top)(\mathsf{skilled\_in}^{-1}\colon \mathsf{Doctor}) \\
D_V &= \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String}) \sqcap \\
&\quad \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease}) \doteq (\mathsf{suffers}\colon \mathsf{Disease}).
\end{aligned}
$$

Now we are able to reformulate our key problem in the new framework. Recall the question:

> *Given a $\mathcal{DL}$ schema $\mathcal{S}$, a query Q and a view V, are the instances of Q contained in the view V for every database state obeying the schema $\mathcal{S}$?*

Let Σ denote the translation of $\mathcal{S}$, C that of Q and D that of V. Since we forget about the non-structural parts of $\mathcal{S}$, the restrictions for Σ-interpretations are weaker than those for database states. Therefore a database state always corresponds to a Σ-interpretation in a natural way. Since we forget about the non-structural parts of Q, the answer set of Q is a subset of the denotation of C. So the instances of Q are surely contained in V if C is Σ-subsumed by D (recall that a view has only a structural part and thus is entirely captured by the concept).

Proposition 3.1 *Let $\mathcal{S}$, Q, V be a schema, a query class, and a view in $\mathcal{DL}$, and let Σ, C, D be their translations into an $\mathcal{SL}$ schema and $\mathcal{QL}$ concepts. If*

$$C \sqsubseteq_\Sigma D$$

then in every state of $\mathcal{S}$, the answer set of Q is a subset of the answer set of V.

The "only if" direction does not hold, since we forget about non-structural parts. So Σ-subsumption gives us a sufficient but not a necessary condition for subsumption of queries. In the next section we will present a calculus for detecting Σ-subsumption, which provides a procedure that runs in time polynomial in the size of schema, query and view.

A subsumption checking component based on this procedure can be embedded into a query optimizer for $\mathcal{DL}$ database systems: instead of just employing conventional compilation techniques for generating query evaluation plans from query classes, a subsumption checker tests whether an incoming query is subsumed by one of the views currently materialized in the database. For this purpose the structural parts of query classes and the view definitions are translated into $\mathcal{QL}$ expressions, the schema declarations into $\mathcal{SL}$ expressions, and the procedure is run on this input. The system modifies the query evaluation plans by adding access operations to the stored extensions of subsuming views, thus restricting the search space. We plan to implement such a subsumption checker within the deductive object base manager *ConceptBase* [JS93, SNJ93] which offers a schema and query language very similar to $\mathcal{DL}$.

4 A Calculus for Deciding Subsumption

The basic idea for deciding subsumption between a query concept C and a view concept D is as follows. We take an object o and transform C into a prototypical interpretation where o is an instance of C. We do so by generating objects, entering them into concepts, and relating them through attributes. Then we evaluate D over this interpretation. If o belongs to the answers of D then C is subsumed by D. If not, we have an interpretation where an object is in C but not in D and therefore C is not subsumed by D.

This approach is similar to the technique used for deciding containment of conjunctive queries (see [Ull89]). But the problem is more complicated in our case because we have to take into account the schema axioms. In particular, axioms of the form $A \sqsubseteq \exists P$ lead to complications, since they can enforce the generation of new objects. To see this suppose that our schema contains the axioms $A \sqsubseteq \exists P$ and $A \sqsubseteq \forall P.\, A$, and that we have on object o in A. Then o must have a filler for the attribute P, say o', which is again in A. Thus, building up a prototypical interpretation one might generate an infinite number of objects if no special care is taken. To guarantee that the interpretation is of polynomial size and that D can be evaluated in polynomial time, D is used to provide guidance for the construction of the interpretation.

Any concept of the form $\exists p \doteq q$ is equivalent to a concept of the form $\exists p' \doteq \epsilon$, since paths can be inverted using inverses of attributes. In the sequel we assume that no concept has subconcepts of the form $\exists p \doteq q$ where $q \neq \epsilon$, since this simplifies the calculus.

D1: $F.G \to \{s{:}\,C,\ s{:}\,D\} \cup F.G$
    if $s{:}\,C \sqcap D$ is in F

D2: $F.G \to \{sRt\} \cup F.G$
    if $tR^{-1}s$ is in F

D3: $F.G \to (F.G)[y/a]$
    if $y{:}\,\{a\}$ is in F

D4: $F.G \to \{spy\} \cup F.G$
    if $s{:}\,\exists p$ is in F, and there is no t with spt in F, and y is a fresh variable

D5: $F.G \to \{sps\} \cup F.G$
    if $s{:}\,\exists p \doteq \epsilon$ is in F

D6: $F.G \to \{sRy, y{:}\,C, ypt\} \cup F.G$
    if $s(R{:}\,C)pt$ is in F, and there is no t' such that sRt', $t{:}'\,C$, $t'pt$ are all in F, and $p \neq \epsilon$, and y is fresh variable

D7: $F.G \to \{sRt, t{:}\,C\} \cup F.G$
    if $s(R{:}\,C)t$ is in F

Figure 7: The decomposition rules

4.1 The Rules of the Calculus

To formulate the calculus we augment our syntax by *variables* (ranged over by x, y). We will refer to constants and variables alike as *individuals* (denoted by the letters s, t). Our calculus works on syntactic entities called *constraints*[6] that have one of the forms

$$s{:}\,C, \qquad sRt, \qquad spt,$$

where C is a $\mathcal{QL}$ concept, R is an attribute, p is a path, and s, t are individuals. The first constraint says that s is an instance of C, the second that t is an R-filler of s, and the third that s and t are related through p. A *constraint system* is a set S of constraints.

We also extend the semantics. An interpretation $\mathcal{I}$ maps a variable x to an element $x^{\mathcal{I}}$ of its domain. It *satisfies* a constraint $s{:}\,C$ if $s^{\mathcal{I}} \in C^{\mathcal{I}}$, a constraint sRt if $(s^{\mathcal{I}}, t^{\mathcal{I}}) \in R^{\mathcal{I}}$, and a constraint spt if $(s^{\mathcal{I}}, t^{\mathcal{I}}) \in p^{\mathcal{I}}$. We say that a Σ-interpretation $\mathcal{I}$ is a Σ-*model* of a constraint c if it satisfies c. A constraint is Σ-*satisfiable* if it has a Σ-model. The notions of satisfaction, model, and satisfiability are extended to constraint systems as one would expect.

Let c, c' be constraints and Σ be an $\mathcal{SL}$ schema. We write

$$c \models_{\Sigma} c'$$

[6]Not to be confused with the constraints in class declarations!

S1: $F.G \rightarrow \{s\colon A_2\} \cup F.G$

  if $s\colon A_1$ is in F, and $A_1 \sqsubseteq A_2$ is in Σ

S2: $F.G \rightarrow \{t\colon A_2\} \cup F.G$

  if $s\colon A_1$ and sPt are in F, and $A_1 \sqsubseteq \forall P.\, A_2$ is in Σ

S3: $F.G \rightarrow \{s\colon A_1,\ t\colon A_2\} \cup F.G$

  if sPt is in F, and $P \sqsubseteq A_1 \times A_2$ is in Σ

S4: $F.G \rightarrow (F.G)[y/t]$

  if $s\colon A$, sPy, sPt are in F, and $A \sqsubseteq (\leq 1\, P)$ is in Σ

S5: $F.G \rightarrow \{sPy\} \cup F.G$

  if $s\colon \exists(P\colon C)p$ is in G or $s\colon \exists(P\colon C)p \doteq \epsilon$ is in G, and there is no t with sPt in F, and there is an A with $s\colon A$ in F and $A \sqsubseteq \exists P$ in Σ, and y is a fresh variable

Figure 8: The schema rules

if every Σ-model of c is also a Σ-model of c'. This notion of Σ-entailment is naturally extended to constraint systems. The following proposition describes how Σ-entailment is linked to Σ-subsumption.

Proposition 4.1 *Let Σ be an $\mathcal{SL}$ schema, C, D be $\mathcal{QL}$ concepts, and x be a variable. Then*

$$C \sqsubseteq_\Sigma D \quad \textit{iff} \quad x\colon C \models_\Sigma x\colon D.$$

General Assumption. *Throughout this section, Σ denotes a fixed $\mathcal{SL}$ schema.*

Our calculus features four kinds of rules: decomposition, schema, goal, and composition rules. The rules work on pairs of constraint systems $F.G$ (called *pairs* for short). We call F the *facts* and G the *goals*. In order to decide whether $C \sqsubseteq_\Sigma D$, we take a variable x and start with the fact $\{x\colon C\}$ and the goal $\{x\colon D\}$. Applying the rules, we add more facts and goals until no more rule is applicable. Intuitively, the query C is subsumed by the view D iff the final set of facts contains the constraint $x\colon D$.[7] All rules exploit the hierarchical structure of concepts, which is the basic reason for the polynomiality of the procedure.

To formulate the rules we use the following notation. By R^{-1} we denote P^{-1} if $R = P$ and P if $R = P^{-1}$. The pair $(F.G)[y/s]$ is obtained from $F.G$ by replacing every occurrence of y with s.

The rules are presented in Figures 7 to 10. A rule is applicable to a pair if it satisfies the conditions associated with the rule and if it is altered when transformed

[7]We will see later on that this condition has to be refined a bit.

G1: $F.G \;\rightarrow\; F.G \cup \{s\colon C, s\colon D\}$

if $s\colon C \sqcap D$ is in G

G2: $F.G \;\rightarrow\; F.G \cup \{t\colon C\}$

if $s\colon \exists(R\colon C)$ is in G or $s\colon \exists(R\colon C) \doteq \epsilon$ is in G, and sRt is in F

G3: $F.G \;\rightarrow\; F.G \cup \{t\colon C, t\colon \exists p\}$

if $s\colon \exists(R\colon C)p$ is in G or $s\colon \exists(R\colon C)p \doteq \epsilon$ is in G, and sRt is in F

Figure 9: The goal rules

according to the rule. The second requirement is needed to ensure termination of our calculus. As an example, Rule D1 is applicable to a pair $F.G$ if F contains a constraint $s\colon C \sqcap D$ *and* if $s\colon C$ and $s\colon D$ are not both in F.

The *decomposition rules* (Figure 7) work on facts. They break up the initial fact $x\colon C$ into constraints involving only primitive concepts, primitive attributes, and singletons. In breaking up a path, Rules D4 and D6 use fresh variables to represent the objects along the path.

The *schema rules* (Figure 8) also work on facts. They add information derivable from the schema and the current facts. The first four rules are simple: Rules S1 to S3 add membership constraints for individuals in F, and Rule S4 identifies values of functional attributes. Rule S5, however, which might create a new individual, is subject to a tricky control that limits the number of new individuals: it is only applicable if it contributes to a path that is required by a goal.

Also individuals introduced by the decomposition rules can help in building up such a path. Since they carry more specific information than variables created by schema rules, decomposition rules receive priority:

- A schema rule can be applied only if no decomposition rule is applicable.

This control structure contributes to keeping the whole procedure polynomial.

The *goal rules* (Figure 9) work on goals. They guide the evaluation of the view concept D by deriving subgoals from the original goal $x\colon D$. The interesting rules are G2 and G3, since they relate goals to facts: if the goal is to find a path issuing from s whose first step involves the attribute R, then only individuals t are tested which are explicitly mentioned as R-fillers of s in the facts.

The *composition rules* (Figure 10) compose complex facts from simpler ones directed by the goals. This can be understood as a bottom up evaluation of concept D over F.

A pair is *complete* if no rule is applicable. A complete pair obtained from a pair

C1: $F.G \rightarrow \{s\colon C \sqcap D\} \cup F.G$
 if $s\colon C$ and $s\colon D$ are in F, and $s\colon C \sqcap D$ is in G

C2: $F.G \rightarrow \{s\colon \top\} \cup F.G$
 if $s\colon \top$ is in G

C3: $F.G \rightarrow \{s\colon \exists p\} \cup F.G$
 if $p = \epsilon$ or there is a t with spt in F, and $s\colon \exists p$ is in G

C4: $F.G \rightarrow \{s\colon \exists p \doteq \epsilon\} \cup F.G$
 if $p = \epsilon$ or sps is in F, and $s\colon \exists p \doteq \epsilon$ is in G

C5: $F.G \rightarrow \{s(R\colon C)pt\} \cup F.G$
 if there is a t' with sRt', $t'\colon C$ and $t'pt$ in F, such that $s\colon \exists(R\colon C)p$ or $s\colon \exists(R\colon C)p \doteq \epsilon$ is in G

C6: $F.G \rightarrow \{s(R\colon C)t\} \cup F.G$
 if sRt and $t\colon C$ are in F, and $s\colon \exists(R\colon C)$ or $s\colon \exists(R\colon C) \doteq \epsilon$ is in G

Figure 10: The composition rules

$F.G$ by applying the above rules is called a *completion* of $F.G$. Since all rules are deterministic, there exists—up to variable renaming—exactly one completion for a pair of constraint systems.

In Figure 11, we use the calculus to check the concepts C_Q and D_V of Section 3.2 for subsumption. We start with the pair $F_1.G_1$ where:

$$\begin{aligned} F_1 &= \{x\colon \mathsf{Male} \sqcap \mathsf{Patient} \sqcap \\ &\qquad \exists(\mathsf{consults}\colon \mathsf{Female} \sqcap \mathsf{Doctor})(\mathsf{skilled\_in}\colon \top)(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\} \\ G_1 &= \{x\colon \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String}) \sqcap \\ &\qquad \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\}. \end{aligned}$$

Note that C_Q and D_V have been rewritten such that all agreements of paths have the form $\exists p \doteq \epsilon$. Figure 11 shows a sequence of rule applications. In each step we give only the component of the pair that changes. As completion we obtain $F_{21}.G_5$ and see that $x\colon D_V$ is in F_{21}. Hence C_Q is Σ-subsumed D_V.

4.2 Soundness and Completeness of the Calculus

We now show that our calculus indeed gives rise to a polynomial time decision procedure for Σ-subsumption of $\mathcal{QL}$ concepts.

First, notice that in any pair $F.G$ derivable from an initial pair $\{x\colon C\}.\{x\colon D\}$, the set of goals G contains exactly one constraint of the form $s\colon D$. The individual

$$\begin{array}{lcll}
F_2 & = & F_1 \cup \{x\colon \mathsf{Male} \sqcap \mathsf{Patient}, & \\
 & & \quad x\colon \exists(\mathsf{consults}\colon \mathsf{Female} \sqcap \mathsf{Doctor})(\mathsf{skilled\_in}\colon \top)(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \mathrm{D1} \\
F_3 & = & F_2 \cup \{x\colon \mathsf{Male}, x\colon \mathsf{Patient}\} & \mathrm{D1} \\
F_4 & = & F_3 \cup \{x\ (\mathsf{consults}\colon \mathsf{Female} \sqcap \mathsf{Doctor})(\mathsf{skilled\_in}\colon \top)(\mathsf{suffers}^{-1}\colon \top)\ x\} & \mathrm{D5} \\
F_5 & = & F_4 \cup \{x\ \mathsf{consults}\ s_1, y_1\colon \mathsf{Female} \sqcap \mathsf{Doctor}, y_1\ (\mathsf{skilled\_in}\colon \top)(\mathsf{suffers}^{-1}\colon \top)\ x\} & \mathrm{D6} \\
F_6 & = & F_5 \cup \{y_1\colon \mathsf{Female}, y_1\colon \mathsf{Doctor}\} & \mathrm{D1} \\
F_7 & = & F_6 \cup \{y_1\ \mathsf{skilled\_in}\ y_2, y_2\colon \top, y_2\ (\mathsf{suffers}^{-1}\colon \top)\ x\} & \mathrm{D6} \\
F_8 & = & F_7 \cup \{y_2\ \mathsf{suffers}^{-1}\ x, x\colon \top\} & \mathrm{D7} \\
F_9 & = & F_8 \cup \{x\ \mathsf{suffers}\ y_2, y_1\ \mathsf{consults}^{-1}\ x, y_2\ \mathsf{skilled\_in}^{-1}\ y_1\} & 3\times\mathrm{D2} \\
F_{10} & = & F_9 \cup \{x\colon \mathsf{Person}\} & \mathrm{S1} \\
F_{11} & = & F_{10} \cup \{y_2\colon \mathsf{Disease}\} & \mathrm{S2} \\
F_{12} & = & F_{11} \cup \{y_1\colon \mathsf{Person}, y_2\colon \mathsf{Topic}\} & \mathrm{S3} \\
G_2 & = & G_1 \cup \{x\colon \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String}), & \\
 & & \quad x\colon \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \mathrm{G1} \\
G_3 & = & G_2 \cup \{x\colon \mathsf{Patient}, x\colon \exists(\mathsf{name}\colon \mathsf{String})\} & \mathrm{G1} \\
G_4 & = & G_3 \cup \{y_1\colon \mathsf{Doctor}, y_1\colon \exists(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top)\} & \mathrm{G3} \\
G_5 & = & G_4 \cup \{y_2\colon \exists(\mathsf{suffers}^{-1}\colon \top)\} & \mathrm{G3} \\
F_{13} & = & F_{12} \cup \{x\ \mathsf{name}\ y_3\} & \mathrm{S5} \\
F_{14} & = & F_{13} \cup \{y_3\colon \mathsf{String}\} & \mathrm{S2} \\
F_{15} & = & F_{14} \cup \{y_3\ \mathsf{name}^{-1}\ x\} & \mathrm{D2} \\
F_{16} & = & F_{15} \cup \{y_1\ (\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top)\ x\} & \mathrm{C5} \\
F_{17} & = & F_{61} \cup \{x\ (\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top)\ x\} & \mathrm{C5} \\
F_{18} & = & F_{17} \cup \{x\colon \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \mathrm{C4} \\
F_{19} & = & F_{18} \cup \{x\colon \exists(\mathsf{name}\colon \mathsf{String})\} & \mathrm{C6} \\
F_{20} & = & F_{19} \cup \{x\colon \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String})\} & \mathrm{C1} \\
F_{21} & = & F_{20} \cup \{x\colon \mathsf{Patient} \sqcap \exists(\mathsf{name}\colon \mathsf{String}) \sqcap & \\
 & & \quad \exists(\mathsf{consults}\colon \mathsf{Doctor})(\mathsf{skilled\_in}\colon \mathsf{Disease})(\mathsf{suffers}^{-1}\colon \top) \doteq \epsilon\} & \mathrm{C1}
\end{array}$$

Figure 11: A sequence of rule applications

s occurring in $s\colon D$ may be distinct from x due to applications of the rules D3 and S4, which replace a variables by another individual. Moreover, if $s\colon D \in G$, then $s\colon C \in F$. This can be verified by inspecting the rules that alter goals.

General Assumption. *In the sequel of this section, C, D are $\mathcal{QL}$ concepts, x is a variable, $F_C . G_D$ is the completion of $\{x\colon C\} . \{x\colon D\}$, and o is the individual such that $o\colon D$ is in G_D.*

We first observe that the calculus adds only consequences to the set of facts, *i.e.*, it keeps the models of the facts invariant.

Proposition 4.2 (Invariance) *Suppose $F . G$ has been derived from $\{x\colon C\} . \{x\colon D\}$, and $F' . G'$ is obtained from $F . G$ by applying a rule. Then every Σ-model $\mathcal{I}$ of F can be turned into a Σ-model $\mathcal{I}'$ of F' by modifying the interpretation of fresh variables. Moreover, if s, s' are such that $\{s\colon D\} \in G$ and $\{s'\colon D\} \in G'$, then $\mathcal{I}'$ can be chosen such that $s'^{\mathcal{I}'} = s^{\mathcal{I}}$.*

Proof. The claim can be shown by a case analysis considering all rules. □

Corollary 4.3 *Every Σ-model $\mathcal{I}$ of $x\colon C$ can be turned into a Σ-model $\mathcal{I}'$ of F_C by modifying the interpretation of variables. Moreover, $\mathcal{I}'$ can be chosen such that $o^{\mathcal{I}'} = x^{\mathcal{I}}$.*

Proof. This follows by induction from the preceding proposition. □

Corollary 4.4

$$x\colon C \models_\Sigma x\colon D \quad\Longleftrightarrow\quad F_C \models_\Sigma o\colon D$$

Proof. "$\Rightarrow$" Let $\mathcal{I}$ be a Σ-model of F_C. Then $\mathcal{I}$ is a Σ-model of $o\colon C$, since $o\colon C \in F_C$. Let $\mathcal{I}'$ be such that $x^{\mathcal{I}'} = o^{\mathcal{I}}$ and $\mathcal{I}'$ coincides with $\mathcal{I}$ otherwise. Then $\mathcal{I}'$ is a Σ-model of $x\colon C$ and hence of $x\colon D$. By definition, $\mathcal{I}'$ is a Σ-model of $o\colon D$ as well. Since $\mathcal{I}$ and $\mathcal{I}'$ coincide on all symbols, except possibly x, it follows that $\mathcal{I}$ is a Σ-model of $o\colon D$.

"$\Leftarrow$" Let $\mathcal{I}$ be a Σ-model of $x\colon C$. By the preceding corollary, $\mathcal{I}$ can be turned into a Σ-model $\mathcal{I}'$ of F_C with $o^{\mathcal{I}'} = x^{\mathcal{I}}$ by modifying the interpretation of variables. The interpretation $\mathcal{I}'$ is also a Σ-model of $o\colon D$. Since $D^{\mathcal{I}} = D^{\mathcal{I}'}$ and $x^{\mathcal{I}} = o^{\mathcal{I}'}$ it follows that $\mathcal{I}$ is a Σ-model of $x\colon D$. □

Not every constraint system is Σ-satisfiable. Since different constants are interpreted as distinct objects, a constraint of the form $a\colon\{b\}$ is unsatisfiable. For the same reason, different constants cannot be values of a functional attribute at the same time. These observations are captured by the notion of clash.

A *clash* is a constraint system of one of the following forms:

- $\{a\colon\{b\}\}$, where $a \neq b$
- $\{sPa, sPb, s\colon A\}$, where $A \sqsubseteq (\leq 1\ P)$ is in Σ and $a \neq b$.

A constraint system is *clash-free* if it does not contain a clash. Obviously, a constraint system containing a clash is not Σ-satisfiable. We will show that a clash-free set of facts in a complete pair is Σ-satisfiable.

From constraint systems one can derive interpretations in a natural way. Let u be a new symbol and S be a constraint system. The *canonical interpretation* $\mathcal{I}_S$ of S is defined as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}_S} &:= \{s \mid s \text{ is an individual in } S\} \cup \{a \mid a \text{ is a constant}\} \cup \{u\} \\
s^{\mathcal{I}_S} &:= s
\end{aligned}$$

$$\begin{array}{rcl} A^{\mathcal{I}_S} & := & \{s \mid s\colon A \text{ is in } S\} \cup \{u\} \\ P^{\mathcal{I}_S} & := & \{(s,t) \mid sPt \text{ is in } S\} \cup \{(u,u)\} \\ & & \cup \{(s,u) \mid \text{there is no } sPt \text{ in } S, \text{ but for some } A, \\ & & \qquad\qquad s\colon A \text{ is in } S \text{ and } A \sqsubseteq \exists P \text{ is in } \Sigma\}. \end{array}$$

We will be particularly interested in canonical interpretations that are obtained from the facts of a complete pair. As pointed out before, the schema rules are designed in such a way that not every necessary attribute will get a variable as a filler. The role of the new object u is to compensate for this lack.

The idea that our calculus constructs a Σ-model of C is made more precise by the following proposition.

Proposition 4.5 *Let $F.G$ be a complete pair. If F is clash-free, then the canonical interpretation $\mathcal{I}_F$ is a Σ-model of F.*

Proof. We have to verify that $\mathcal{I}_F$ satisfies the Unique Name Assumption as well as every axiom in Σ and every constraint in F.

The Unique Name Assumption is trivially satisfied because every constant symbol is interpreted by itself.

Let us consider the schema axioms. Suppose that Σ contains an axiom $A \sqsubseteq \exists P$. Let $s \in A^{\mathcal{I}_F}$. If $s = u$, then the axiom is satisfied, since $(u,u) \in P^{\mathcal{I}_F}$. If $s \neq u$, then $s\colon A \in F$. We distinguish two cases: (1) there is a constraint sPt in F, or (2) there is no such constraint. In the first case, $(s,t) \in P^{\mathcal{I}_F}$ and in the second, $(s,u) \in P^{\mathcal{I}_F}$. Thus the axiom $A \sqsubseteq \exists P$ is satisfied.

Suppose that Σ contains an axiom of the form $A \sqsubseteq (\leq 1\ P)$. Let $s \in A^{\mathcal{I}_F}$. If $s = u$, then the axiom is satisfied, since by construction of $\mathcal{I}_F$ the symbol u is the only object that is in the relation $P^{\mathcal{I}_F}$ with u. If $s \neq u$, then $s\colon A \in F$. Assume that t and t' are two distinct symbols such that both (s,t) and (s,t') are in $P^{\mathcal{I}_F}$. Then both t and t' are distinct from u, because $(s,u) \in P^{\mathcal{I}_F}$ implies $t'' = u$ for any t'' with $(s,t'') \in P^{\mathcal{I}_F}$. By definition of $\mathcal{I}_F$ it follows that F contains the constraints sPt and sPt'. If t or t' were a variable, then Rule S4 would be applicable to $F.G$, in contradiction to the completeness of $F.G$. Hence, both t and t' are constants, which contradicts the fact that F is clash-free. Consequently, the above assumption is false and the axiom $A \sqsubseteq (\leq 1\ P)$ is satisfied. The other cases require similar reasoning and are therefore dismissed.

Next we consider the different constraints. By definition of $\mathcal{I}_F$, every constraint sPt is satisfied. If F contains a constraint $sP^{-1}t$, then it contains also the constraint tPs, since otherwise Rule D2 would be applicable. Obviously, every constraint $s\colon A$ and $s\colon \top$ is satisfied. Moreover, for every constraint $s\colon \{a\} \in F$ we have that s is a constant, since otherwise Rule D3 would be applicable to $F.G$, and that $s = a$ because F is clash-free.

To prove that more complex constraints are satisfied, we proceed by induction. If F contains $s\colon E \sqcap E'$, then, because of Rule D1, it contains as well $s\colon E$ and $s\colon E'$, which are satisfied by the inductive hypothesis. Hence, $\mathcal{I}_F$ satisfies also $s\colon E \sqcap E'$. If F contains $s(R\colon C)t$, then, because of Rule D7, it contains as well sRt and $t\colon C$, which are satisfied by the inductive hypothesis. Hence, $\mathcal{I}_F$ satisfies also $s(R\colon C)t$. We drop the remaining cases, since the arguments will be similar. □

Proposition 4.6 *Let $\mathcal{I}_{F_C}$ be the canonical interpretation of F_C and $s\colon E$ be a constraint in G_D. If F_C is clash-free, then*

$$\mathcal{I}_{F_C} \textit{ satisfies } s\colon E \quad \Longrightarrow \quad s\colon E \in F_C.$$

Proof. Suppose that F_C is clash-free and that $\mathcal{I}_{F_C}$ satisfies $s\colon E$. The proof is by induction on the structure of E.

We first consider the base cases. Suppose that $E = A$. Then $s\colon A \in F_C$ by definition of $\mathcal{I}_{F_C}$, since $\mathcal{I}_{F_C}$ satisfies $s\colon A$. Suppose that $E = \top$. Then $s\colon \top \in F_C$ because of Rule C2. Suppose that $E = \{a\}$. In order to deal with this case, we have to make use of two observations. First, noticing the fact that our calculus starts off with a pair $\{x\colon C\}.\{x\colon D\}$, and inspecting the rules of the calculus we see that any individual t occurring in a constraint $t\colon E'$ in G_D occurs also in F_C. Second, analyzing the rules again we see that if a constant a occurs in F_C, then F_C contains a constraint $t\colon \{a\}$. Now, if $\mathcal{I}_{F_C}$ satisfies $s\colon \{a\}$, then by definition of $\mathcal{I}_{F_C}$ we have $s\colon = a$. By our first observation, a occurs also in F_C, and by our second observation, F_C contains a constraint $t\colon \{a\}$. Since F_C is the first component of a complete pair, it follows that $t = a$ and hence that $a\colon \{a\} \in F_C$.

Next we analyze in turn each possible complex concept E. If E is of the form $E' \sqcap E''$ then $\mathcal{I}_{F_C}$ satisfies $s\colon E' \sqcap E''$ iff $\mathcal{I}_{F_Q}$ satisfies $s\colon E'$ and $s\colon E''$. By the inductive hypothesis, this is the case iff $s\colon E' \in F_C$ and $s\colon E'' \in F_C$. Since $s\colon E' \sqcap E'' \in G_D$, we have that $s\colon E' \sqcap E'' \in F_C$, since otherwise Rule C1 would be applicable.

If E is of the form $\exists p$ then we have to consider two cases. The case $p = \epsilon$ is captured by Rule C3. If $p \neq \epsilon$, then $\mathcal{I}_{F_C}$ satisfies $s\colon \exists p$ iff $\mathcal{I}_{F_C}$ satisfies spt for an individual t. By an induction on the length of p, which is nested in the current induction, we show that $spt \in F_C$ if $\mathcal{I}_{F_C}$ satisfies spt. Then, because of Rule C3, $s\colon \exists p \in F_C$.

As the base case of the nested induction we consider $p = (R\colon E')$. Then, since $s\colon \exists(R\colon E') \in G_D$ and since $F_C.G_D$ is complete, by the definition of $\mathcal{I}_{F_C}$ it follows that $\mathcal{I}_{F_C}$ satisfies $s(R\colon E')t$ iff $sRt \in F_C$ and $\mathcal{I}_{F_C}$ satisfies $t\colon E'$. By the outer inductive hypothesis this is the case if $t\colon E' \in F_C$ (since $t\colon E' \in G_D$). Then, because of Rule C6, we have $spt \in F_C$.

For the induction step suppose that $p = (R\colon E')p'$. Since $s\colon \exists(R\colon E')p' \in G_D$ and since $F_C.G_D$ is complete, by the definition of $\mathcal{I}_{F_C}$ it follows that $\mathcal{I}_{F_C}$ satisfies

$s(R\colon E')p't$ iff there is a t' such that $sRt' \in F_C$ and $\mathcal{I}_{F_C}$ satisfies both $t'\colon E'$ and $t'p't$. By the inner and the outer inductive hypothesis this is the case if $t'\colon E' \in F_C$ and $t'p't \in F_C$ (since $t'\colon E' \in G_D$ and $t'\colon \exists p' \in G_D$). Then, because of Rule C5, we have $spt \in F_D$, which ends the inner induction proof.

If E is of the form $\exists p \doteq \epsilon$ then we again have to consider two cases. The case $p = \epsilon$ is captured by Rule C4. If $p \neq \epsilon$, then $\mathcal{I}_{F_C}$ satisfies $s\colon \exists p \doteq \epsilon$ iff $\mathcal{I}_{F_C}$ satisfies sps. If $\mathcal{I}_{F_C}$ satisfies sps, then sps is in F_C. This can be shown by an induction analogous to the one above. Then because of Rule C4, we have $s\colon \exists p \doteq \epsilon \in F_C$. □

Now, we only have to put together the previous statements to show the soundness and completeness of the calculus.

Theorem 4.7 (Soundness and Completeness)

$$C \sqsubseteq_\Sigma D \quad \Longleftrightarrow \quad o\colon D \in F_C \;\; \textit{or} \;\; F_C \; \textit{contains a clash.}$$

Proof. By Proposition 4.1, we have $C \sqsubseteq_\Sigma D$ if and only if $x\colon C \models_\Sigma x\colon D$.

"$\Rightarrow$" Suppose $x\colon C \models_\Sigma x\colon D$ and F_C does not contain a clash. By Corollary 4.4, we know that $F_C \models_\Sigma o\colon D$. Since F_C is clash-free, Proposition 4.5 implies that $\mathcal{I}_{F_C}$ is a model of F_C and hence, $\mathcal{I}_{F_C}$ satisfies $o\colon D$. Since clearly $o\colon D$ is in G_D, we conclude by Proposition 4.6 that $o\colon D \in F_C$.

"$\Leftarrow$" If $o\colon D \in F_C$, then $F_C \models_\Sigma o\colon D$, and Corollary 4.4 yields the claim. If F_C contains a clash, then F_C is unsatisfiable and hence, $x\colon C$ is unsatisfiable too, by Corollary 4.3. Thus, C is unsatisfiable and therefore Σ-subsumed by any concept. □

4.3 Complexity Analysis of the Calculus

Now we turn to the complexity of deciding Σ-subsumption. To this end we estimate the number of individuals in $F_C.G_D$. First, observe that any constant in this pair must appear in C. Second, for every variable introduced by a decomposition rule, there is an existentially quantified subconcept of C. Hence, the number of constants plus the number of variables generated by decomposition rules is less or equal to the size of C. We call these individuals *primary individuals.* Moreover, we say that t' is a *successor* of t if F_C contains constraints tP_1s_1, $s_1P_2s_2$, $\ldots, s_{j-1}P_jt'$. Third, since the introduction of variables by the schema rule S5 is controlled by the structure of D, one can show that for every primary individual the number of nonprimary successors is bounded by the size of D. Summarizing, we obtain an upper bound for the number of individuals occurring in $F_C.G_D$.

Proposition 4.8 *The number of individuals occurring in $F_C . G_D$ is at most MN, where M is the size of C and N is the size of D.*

Every application of a rule either adds new constraints or reduces the number of variables. Since each new constraint is a combination of variables and concept or attribute expressions occurring in C, D, or Σ, the number of rule applications adding constraints is polynomially bounded by the size of C, D and Σ. Similarly, the number of rule applications that identify individuals is bounded by the number of individuals generated through the computation. Summarizing, we conclude that any derivation of $F_C . G_D$ comprises polynomially many steps. Moreover, testing whether a rule is applicable and applying a rule can be accomplished in time polynomial in the size of Σ and the current constraint system. Thus, a completion of $\{x\colon C\}.\{x\colon D\}$ can be computed in time polynomial in the size of C, D and Σ. This yields the following theorem.

Theorem 4.9 *Σ-subsumption between $\mathcal{QL}$ concepts can be decided in time polynomial in the size of C, D and Σ.*

For a practical application we do not suggest to directly implement the rule-based calculus described in this paper. The calculus is rather intended as a conceptual framework for studying subsumption problems and proving results about their properties. A description of an optimal implementation technique was beyond the scope of this paper.

4.4 The Complexity of Language Extensions

Based on previous complexity analyses from the areas of query optimization and concept languages we will show that the structural parts of our database schema and query language are designed so as to gain maximal expressiveness without losing tractability. To this end, we will discuss several natural extensions of $\mathcal{SL}$ and $\mathcal{QL}$ for which determining Σ-subsumption is NP-hard or co-NP-hard. We prefer to discuss this topic in terms of concepts rather than classes and queries because of the higher level of abstraction. In most cases, we will not give the proofs for the hardness results, but motivate them intuitively and describe how they are derived from previous work.

Variables on Paths. In some object-oriented query languages one can not only restrict intermediate nodes on paths to classes, as in our language, but also require arbitrary coreferences between them through variables that force intermediate nodes on different paths to be equal (see *e.g.* [KKS92]). We could easily introduce this feature into $\mathcal{QL}$ by allowing for singletons $\{x\}$ consisting of a variable. Such variables are implicitly understood to be existentially quantified. For example, the concept

$$\exists(P\colon\{x\})(P'\colon\{x\})$$

is equivalent to the formula

$$\exists x.\,(\exists y \exists z.\,(P(\gamma,z)\land z \doteq x)\ \land\ (P'(z,y)\land y \doteq x)\ ).$$

Obviously, singletons with variables in $\mathcal{QL}$ concepts allow us to express arbitrary conjunctive queries involving unary and binary predicates and having one free variable. The subsumption problem for such queries is known to be NP-hard [CM93].

However, for subsumption problems "$C \sqsubseteq_\Sigma D$?" where variables occur only in C and D is an ordinary $\mathcal{QL}$ concept, we still have a sound and complete method. To see this, observe that the problem "$C \sqsubseteq_\Sigma D$?" is logically equivalent to a problem "$C' \sqsubseteq_\Sigma D$?" where C' has been obtained from C by replacing the variables with new constants. This transformation of C just amounts to skolemizing the existentially quantified variables. The resulting concept C' is in $\mathcal{QL}$ and we can decide with our calculus whether $C' \sqsubseteq_\Sigma D$.

Universal and Existential Quantification. The key result for our discussion of other NP-hard extensions is due to Donini *et al.* [DHL+92] who showed that the interplay of universal and existential quantification over attributes is a source of complexity that makes subsumption checking intractable. More precisely, they introduced a concept language $\mathcal{L}$ whose elements are built up from primitive concepts A and primitive attributes P according to the syntax rule

$$C, D \ \longrightarrow\ A \mid C \sqcap D \mid \forall P.\,C \mid \exists P.\,C,$$

where a concept $\exists P.\,C$ is interpreted in the same way as $\exists(P\colon C)$. Donini *et al.* proved that deciding subsumption in $\mathcal{L}$ is NP-hard and that adding to $\mathcal{L}$ a construct $\perp$ to denote the empty concept yields a language $\mathcal{L}^{\perp}$ where already unsatisfiability is NP-hard. Note that neither $\mathcal{SL}$ nor $\mathcal{QL}$ contain both universal and existential quantification over attributes as they occur in $\mathcal{L}$. In $\mathcal{SL}$, existential quantification is restricted to concepts of the form $\exists P$ while in $\mathcal{QL}$ only existential, but no universal quantification is allowed.

The reductions in [DHL+92] can be modified to show that natural extensions of our schema language are computationally harmful.

Proposition 4.10 *Extending $\mathcal{SL}$ by any of the following constructs makes Σ-subsumption of $\mathcal{QL}$ concepts* NP*-hard:*

1. *qualified existential quantification over attributes in the form $\exists P.\,A$;*

2. *inverse attributes in concepts of the form* $\forall P^{-1}.\, A$ *and* $\exists P^{-1}$*;*

3. *singletons, i.e., concepts of the form* $\{a\}$*, where* a *is a constant.*

An intuitive reason why in the first case subsumption is hard can be given in terms of our calculus. To keep our procedure polynomial it is crucial that for any individual s in the set of facts, the schema rule S5 creates at most one P-value. However, if our schema contains axioms $A \sqsubseteq \exists P.\, A'$ and $A \sqsubseteq \exists P.\, A''$ then for every fact $s\colon A$ we have to create two P-values, namely a variable y' with the fact $y'\colon A'$ and a variable y'' with the fact $y''\colon A''$. We have to distinguish between y' and y'' because they have different properties. Since the process of variable generation may have to be iterated on y' and y'', we may end up with exponentially many facts.

While the idea underlying Rule S5 is to create only variables if a goal expression requires to do so, this policy is no more sufficient in the presence of inverse attributes. To see this consider the set of axioms $\Sigma_1 := \{A \sqsubseteq \exists P,\ A \sqsubseteq \forall P.\, A',\ A' \sqsubseteq \forall P^{-1}.\, A''\}$. Then A is Σ_1-subsumed by A'', since an object o in A has a P-value o' in A', and hence o, being an P^{-1}-value of o', is in A''. Intuitively, such implicit inclusions between primitive concepts can only be detected by generating variables as values for all attributes in the schema that are marked as necessary. Since this process has to be iterated we may end up with exponentially many variables.

The basic reason for the hardness of case 3 is that by using singletons in schema axioms one can impose conditions on a primitive concept that make it unsatisfiable. Obviously, in all models of $\Sigma_2 := \{A \sqsubseteq \{a\},\ A \sqsubseteq \{b\}\}$ the concept A is empty. For other schemas it is necessary to reason about values of attributes in order to recognize unsatisfiability. For instance, in all models of $\{A \sqsubseteq \exists P,\ A \sqsubseteq \forall P.\{a\},\ A \sqsubseteq \forall P.\{b\}\}$, the concept A is empty because any object in A is required to have a P-value that is simultaneously in $\{a\}$ and in $\{b\}$. For more complex schemata this reasoning process may involve exponentially many attribute values.

As suggested by the informal arguments above, it can be shown that adding inverse attributes or singletons turns even subsumption of atomic concepts into an NP-hard problem. Moreover, the result on singletons can be generalized so that any extension of our schema definition formalism that allows one to specify unsatisfiable primitive concepts makes subsumption of primitive concepts NP-hard. Examples of such extensions are negation—however limited—, relative complements, or axioms stating the disjointness of primitive concepts.

The result in [DHL+92] applies also to our query language.

Proposition 4.11 *Extending* $\mathcal{QL}$ *by universal quantification over attributes in the form* $\forall P.\, C$ *leads to a language where unsatisfiability—and therefore subsumption—is* NP*-hard even when the schema is empty.*

Proof. The claim holds because adding universal quantification to $\mathcal{QL}$ results in an extension of the language $\mathcal{L}^{\perp}$. $\square$

Disjunction and Negation. Next we are going to discuss extensions that lead to co-NP-hard subsumption problems. As one would suppose, allowing for disjunction of concepts in either our schema or our query language has this effect because disjunction together with conjunction and singletons, by which one can express disjoint concepts, gives the power of propositional logic.

Proposition 4.12

- *Extending $\mathcal{SL}$ by disjunctions of concepts of the form $A \sqcup A'$, which are interpreted as $A^{\mathcal{I}} \cup A'^{\mathcal{I}}$, makes Σ-subsumption of $\mathcal{QL}$ concepts* co-NP*-hard.*
- *Extending $\mathcal{QL}$ by disjunctions of concepts of the form $C \sqcup C'$, which are interpreted as $C^{\mathcal{I}} \cup C'^{\mathcal{I}}$, leads to a language where unsatisfiability—and therefore subsumption—is* co-NP*-hard.*

The first part of the preceding proposition can be shown by an immediate reduction of the satisfiability for propositional logic. The second part has been proved by Kasper and Rounds for feature structures [KR86].

Finally, we observe that even limited forms of negation in the query language make subsumption checking intractable. This can be shown shown using results by Schaerf [Sch93] or Lenzerini [Len93].

Proposition 4.13 *Extending $\mathcal{QL}$ by relative complements $A \backslash A'$ of atomic concepts, which are interpreted as $A^{\mathcal{I}} \setminus A'^{\mathcal{I}}$, leads to a language where subsumption is* co-NP*-hard even when the schema is empty.*

For the preceding proposition to hold, schema and agreements in queries are not essential. The statement is even still valid if either constants [Len93] or inverse attributes [Sch93] are given up.

Proposition 4.13 can be applied as well to an extension of $\mathcal{QL}$ by numerical ranges for functional attributes. If Person is a primitive concept and age is a necessary functional attribute on Person, then the expressions $\mathsf{Person} \sqcap \exists \mathsf{age}.[0, 18]$ and $\mathsf{Person} \sqcap \exists \mathsf{age}.[19, \infty]$, describing persons with age below 18 and persons with age above 18, respectively, are related to each other like relative complements. Hence, numerical ranges give rise to a co-NP-hard subsumption problem too.

5 Related Work

Our work relates to several fields of research in databases and Artificial Intelligence. We shortly discuss the relationship to common subexpression analysis, optimization of conjunctive queries, semantic query optimization, queries and views in OODB's, and query optimization in existing OODB systems.

Common Subexpressions. The problem of recognizing that one query is more general than another one has already been addressed in the context of relational databases. Finkelstein [Fin92] presented an algorithm that detects common subexpressions of relational algebra queries. He proposed to compute answers to such subqueries only once and then to reuse them. His approach is too general to permit a polynomial algorithm. In contrast to our work, he did not make use of schema information.

Conjunctive Queries. As pointed out before, $\mathcal{QL}$ concepts are equivalent to a subclass of conjunctive queries. General conjunctive queries are defined by formulas whose prefix has only existential quantifiers and whose matrix is a conjunction of positive function free atoms [CM93, Ull89]. Much effort has been devoted to the *containment* problem for such queries, *i.e.*, to determine whether the answers for one query are also answers for a second. Thus, in our framework, containment is subsumption with respect to the empty schema.

The objective of this work, however, was not reusing queries, but computing for a given query an equivalent one by removing unnecessary conjuncts. It has been shown that deciding containment of conjunctive queries is NP-hard, even if all predicates involved are binary [CM93]. Aho *et al.* [ASU79] and Johnson and Klug [JK83] identified classes of conjunctive queries for which subsumption can be decided in polynomial time. The class described by Aho *et al.* is so restricted that even queries that chain one attribute (like chaining "child" so as to yield "grandchild") are not captured, while the queries studied by Johnson and Klug are defined by means of complicated graph theoretic properties. Neither of these classes comprises the language $\mathcal{QL}$, so that $\mathcal{QL}$ concepts can be seen as a naturally occurring class of conjunctive queries with polynomial containment problem. In the work on conjunctive queries no schema information like in our case has been taken into account.

Recently, Chan [Cha92] has adapted optimization techniques for conjunctive queries to an object-oriented setting. He considered some minimal schema information like subclass relationship and disjointness of classes. Although the containment problem for his language is obviously NP-hard, he did not address the question of complexity.

Semantic Query Optimization. Semantic query optimization exploits semantic knowledge expressed by integrity constraints for constructing query evaluation plans. Semantic optimization techniques were first proposed in the context of relational databases [Kin81, HZ80, Jar84] and dealt with rather simple types of constraints stating *e.g.*, referential integrity and functional dependencies. For deductive databases and general integrity constraints in clausal form, [CGM88, CGM90, GL92, Kow92] describe a rewriting of queries and rules. This technique has also been used for generating cooperative answers [Gaa92]. Several papers [SO89, HLO91] deal with the implementation of semantic query optimizers, especially schemes for deciding which rules and integrity constraints are actually promising profit for a given query.

Within this framework, our method belongs to the category of approaches that exploit constraints of specific kinds, namely those expressible in the abstract schema language $\mathcal{SL}$. It is tailored to the typical needs of an object-oriented data model like value restrictions, existential and functional requirements for attribute fillers and subset relationships.

Queries and Views in OODB's. In [KKS92] an object-oriented query language called XSQL has been defined. Similar to $\mathcal{QL}$, in XSQL intermediate nodes in path expressions can be constrained by classes. XSQL exceeds our language in expressivity. For instance, it provides generation of object identifiers, which we have not considered.

Abiteboul and Bonner [AB91] presented a view mechanism for OODB's where views can be defined as virtual classes that are populated by existing objects. Virtual classes are integrated into the existing class hierarchy by a simple subsumption check. In the COCOON system views are seen as special classes defined by queries. When integrating them into the schema they are checked for subsumption by ad-hoc techniques [SLT91].

Specific OODB views called materialized functions have been investigated in [KMWZ91]. The functions are used for deriving attribute values which are stored in a separate data structure. A similar strategy is pursued in the extended relational database system Postgres [SK91]. Cost models and benchmark results are in [KKM91] and [Han87] (for relational databases).

Object-oriented Query Optimization. Object-oriented database systems like O_2 [OT92, BCD92] and ObjectStore [OHMS92] focus query optimization on the use of physical clustering strategies and indexes. In O_2, indexes are quite flexible by allowing a membership condition and computed attributes specified as path expressions, similar to the role of views presented in this paper. However, the schema of the O_2 database is not taken into account for query optimization. ObjectStore concentrates on indexes for path expressions which allows an easier selection of indexes for a given query. Both OODB systems do not provide automatic maintenance of their indexes. In O_2, the application program must take care of it, and in ObjectStore the database designer has to annotate the schema. Our approach makes this overhead unnecessary since the triggers for view maintenance can be automatically generated from the logical representation of views.

6 Conclusions

We have proposed a query optimization technique for OODB's that takes advantage of the subsumption of queries. We have presented a polynomial algorithm that recognizes subsuming queries by analyzing complex path expressions and subclass relationships in the query definition and exploits the structural knowledge encoded in the database schema. Technically, we applied results from Artificial Intelligence to identify portions of queries and schemas that permit tractable inferences. Subsumption can be particularly useful in an environment where many view are materialized.

Summarizing we used the following ideas:

1. We introduced a generic object-oriented data model with a simple first-order semantics.
2. Queries and class declarations are separated into a *structural* part containing structural membership conditions and a *non-structural* part containing the remaining membership conditions.
3. To guarantee polynomial time performance, the subsumption checker considers only structural information and ignores the non-structural parts. The approach is sound if the more general query can be described entirely in structural terms.
4. Our algorithm incorporates knowledge from the schema level for finding additional subsumptions which are not derivable from query definitions only.
5. The correspondence between concept languages and the first-order semantics of OODB's makes view maintenance methods from deductive databases ([UO92, CW91] applicable.

One way to increase the success of the method is focusing on specific domains of applications, *e.g.*, distributed information systems. There, a couple of users cooperatively work on a set of tasks. Since objects are shared and passed between the subsystems it can be expected that different people work on the same set of objects—specified by a query. For example, each user may want to see the patients leaving the hospital next week. The first user asking this query triggers the normal evaluation. A control component ("trader") memorizes the query and the location of the answer (the view). A new query is then checked for subsumption against such views. Such an environment is currently set up in a quality management project [JJS93] where autonomous data-intensive tools cover certain aspects of quality management in the industrial product life cycle. Since the trader manipulates schema and query descriptions it provides an excellent test bed for the application of the techniques presented in this paper.

There are a couple of open questions with our approach. *First*, the issue of complex answers has not been addressed. In our model, answers are just sets of object identifiers without any derived answer attributes. These attributes are needed by application programs, and by permutation of parameters they entail additional subsumptions between queries. *Second*, it is an open problem what is the best way to evaluate the query against the subsuming view. We are interested in a minimal filter query which intersected with the view results exactly in the subsumed query. Then, it would be sufficient to test the answer candidates for satisfaction of the filter conditions.

The actual performance gain by exploiting subsumptions to views has to be validated in practical experiments. We expect good results since the structural part has been designed to capture frequently used constructs like path expressions. One should also note that the syntax of queries in Section 2 gives a view definition for free: its structural part! The first evaluation of the view creates no significant overhead since it is part of the evaluation of the original query. Afterwards, such views can be used to optimize subsequent queries.

A prototypical implementation of the proposed optimization technique is planned within the ConceptBase system. The calculus of Section 4 can serve as a starting point for developing an efficient subsumption tester for query and view concepts. This module has then to be embedded into the larger context of query modules of ConceptBase. The maintenance of the views will not create much additional work since it can re-use modules for deductive integrity checking already present in ConceptBase.

Acknowledgements. We would like to thank Maurizio Lenzerini and Matthias Jarke for fruitful discussions that contributed much to the contents of this paper. We also thank Alex Borgida and Peter Patel-Schneider for commenting on earlier drafts.

This work was supported in part by the Commission of the European Communities under ESPRIT Basic Research Action 6810 (Compulog 2), by the German Ministry of Research and Technology under grant ITW 92-01 (TACOS), and by the Ministry of Science and Research of Nordrhein-Westfalen.

References

[AB91] S. Abiteboul and A. Bonner. Objects and views. In J. Clifford and R. King, editors, *Proc. 1991 ACM SIGMOD International Conference on Management of Data, Denver (Co, USA*, pages 238–247, NewYork (NY, USA), may 1991. ACM.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, pages 159–173, Portland, Oregon, 1989.

[ASU79] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, 1979.

[BCD92] F. Bancilhon, S. Cluet, and C. Delobel. A query language for O_2. In F. Bancilhon, C. Delobel, and P. Kanellakis, editors, *Building and Object-oriented Database System – The Story of* O_2, pages 234–255. Morgan Kaufmann Publ., 1992.

[CGM88] U.S. Chakravarthy, J. Grant, and J. Minker. Foundations of semantic query optimization for deductive databases. In J. Minker, editor, *Foundations of deductive databases and Logic programming*, pages 243–273. Morgan Kaufmann, 1988.

[CGM90] U.S. Chakravarthy, J. Grant, and J. Minker. Logic based approach to semantic query optimization. *ACM Trans. on Database Systems*, 5(2):162–207, June 1990.

[Cha92] E. P. F. Chan. Containment and minimization of positive conjunctive queries in OODB's. In *Proc. of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 202–211, San Diego, CA, 1992.

[CM93] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th Annual ACM Symposium on Theory of Computing*, pages 77–90, NewYork (NY, USA), May 1993. ACM.

[CW91] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. 17th Int. Conf. on Very Large Databases*, pages 577–589, 1991.

[DHL+92] F. M. Donini, B. Hollunder, M. Lenzerini, D. Nardi, A. Marchetti Spaccamela, and W. Nutt. The complexity of existential quantification in concept languages. *Artificial Intelligence*, 53(2,3):309–327, 1992.

[DLNN91a] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of the 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162. Morgan Kaufmann, 1991.

[DLNN91b] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence IJCAI-91*, Sydney, 1991.

[Fin92] S. Finkelstein. Common eexpression analysis in database applications. In *Proc. 1982 ACM SIGMOD Int. Conf. on Management of Data*, pages 235–245, Orlando, Florida, June 1992.

[Gaa92] T. Gaasterland. *Generating cooperative answers in deductive databases.* PhD thesis, University of Maryland, Dept. of Computer Science, College Park, 1992.

[GL92] T. Gaasterland and J. Lobo. Processing negation and disjunction in logic programs through integrity constraints. Technical report, University of Maryland, 1992.

[Han87] E.N. Hanson. A performance analysis of view materialization strategies. In *Proc. 1987 ACM SIGMOD Int. Conf. on Management of Data*, pages 440–453, San Francisco, CA, May 1987.

[HLO91] H.H.Pang, H. Lu, and B.C. Ooi. An efficient semantic query optimization algorithm. In *Proc. Int. Conf. on DataEngineering*, pages 326–335, 1991.

[HZ80] M. Hammer and S.B. Zdonik. Knowledge based query processing. In *Proc. 6th VLDB Conf.*, pages 137–147, Montreal, Canada, 1980.

[Jar84] M. Jarke. External semantic query simplification: A graph theoretic approach and its implementation in prolog. In *Proc. 1st Int. Conf. on Expert Database Systems*, pages 467–482, Kiawah Isl., SC, 1984.

[JJS93] M. Jarke, M. A. Jeusfeld, and P. Szczurko. Three aspects of intelligent cooperation in the quality life cycle. In *Proc. Intl. Conf. Intelligent and Cooperative Information Systems*, Rotterdam, Netherlands, 1993. Keynote talk.

[JK83] D. S. Johnson and A. Klug. Optimizing conjunctive queries that contain untyped variables. *SIAM Journal of Computation*, 12(4):616–640, 1983.

[JS93] M. Jeusfeld and Martin Staudt. Query optimization in deductive object bases. In Freytag, Maier, and Vossen, editors, *Query Processing for Advanced Database Systems*. Morgan Kaufmann, June 1993.

[Kin81] J.J. King. QUIST: A system for semantic query optimization in relational databases. In *Proc. 7th VLDB Conf.*, pages 510–517, 1981.

[KKM91] A. Kemper, C. Kilger, and G. Moerkotte. Function materialization in object bases. In *Proc. 1991 ACM SIGMOD Int. Conf. on Management of Data*, pages 258–267, Denver, Colorado, May 1991.

[KKS92] M. Kifer, W. Kim, and Y. Sagiv. Querying object-oriented databases. In *Proc. 1992 ACM SIGMOD Int. Conf. on Management of Data*, pages 393–402, San Diego, CA, June 1992.

[KMWZ91] A. Kemper, G. Moerkotte, H.D. Walter, and A. Zachmann. GOM: a strongly typed, persistent object model with polymorphism. In *Proc. of BTW*, pages 198–217, Kaiserlautern, Germany, March 1991. Springer-Verlag, IFB 270.

[Kow92] W.L. Kowarschick. Semantic optimization: What are disjunctive residues useful for ? *SIGMOD RECORD*, 21(3):26–32, September 1992.

[KR86] R. T. Kasper and W. C. Rounds. A logical semantics for feature structures. In *Proc. of the 24th ACL*, pages 257–266, New York, 1986.

[Len93] M. Lenzerini. Personal communication, August 1993.

[OHMS92] J. Orenstein, S. Haradhvala, B. Margulies, and D. Sakahara. Query processing in the ObjectStore database system. In *Proc. 1992 ACM SIGMOD Int. Conf. on Management of Data*, pages 403–412, San Diego, CA, June 1992.

[OT92] O_2-Team. A technical overview of the O_2 system. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases, and CASE*, pages 337–356. John Wiley & Sons, 1992.

[Sch93] A. Schaerf. On the complexity of the instance checking problem in concept languages with existential quantification. In *Proc. 7th International Symposium on Methodologies for Intelligent Systems, Trondheim (Norway)*, 1993.

[SK91] M. Stonebraker and G. Kemnitz. The Postgres next-generation database system. *Communications of the ACM*, 34(10):78–93, October 1991.

[SLT91] M.H. Scholl, C. Laasch, and M. Tresch. Updatable views in object oriented databases. In *Proc. 2nd Int. Conf. on Deductive and Object-Oriented Databases*, pages 189–207, Munich, Germany, December 1991. Springer LNCS 566.

[SNJ93] M. Staudt, H.W. Nissen, and M.A. Jeusfeld. Query by rule, class and concept. *Applied Intelligence*, 1993. Special issue on knowledge base management, to appear.

[SO89] S.T. Shenoy and Z.M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Trans. on Knowledge and Data Engineering*, 1(3):344–361, September 1989.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume 2: The New Technologies.* Computer Science Press, Rockville, MD, 1989.

[UO92] T. Urpi and A. Olive. A method for change computation in deductive databases. In *Proc. 18th VLDB Conf.*, pages 225–237, Vancouver,British Columbia, Canada, 1992.

[WS92] W. A. Woods and J. G. Schmolze. The KL-ONE family. In F.W. Lehmann, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Published as a special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2–9.