Incoherence and Subsumption for recursive views and queries in Object-Oriented Data Models

Domenico Beneventano, Sonia Bergamaschi[a]

Dipartimento di Scienze dell'Ingegneria - Modena
CIOC-CNR - Bologna

Abstract

Object-oriented data models are being extended with recursion to gain expressive power. This complicates both the incoherence detection problem which has to deal with recursive classes descriptions and the optimization problem which has to deal with recursive queries on complex objects. In this paper, we propose a theoretical framework able to face the above problems. In particular, it is able to validate and automatically classify in a database schema, organized in an inheritance taxonomy, (recursive) classes, views and queries. The framework adopts the ODL formalism (an extension of the *Description Logics* developed in the area of Artificial Intelligence) which is able to express the semantics of complex objects data models and to deal with cyclic references at the schema and instance level. It includes *subsumption* algorithms, which perform automatic placement in a specialization hierarchy of (recursive) views and queries, and *incoherence* algorithms, which detect *incoherent* (i.e., always empty) (recursive) classes, views and queries. As different styles of semantics: greatest fixed-point, least fixed-point and descriptive can be adopted to interpret recursive views and queries, first of all we analyze and discuss the choice of one or another semantics and, secondly, we give the subsumption and incoherence algorithms for the three different semantics. We show that subsumption computation and incoherence detection appear to be feasible since in almost all practical cases they can be solved in polynomial time algorithms.

Key words: Object-Oriented DataModels, Descriptive Logics, Recursive Views and Queries, Incoherence, Subsumption.

1 Introduction

Recently database models that are more expressive than the relational data model have been proposed. Object-oriented data models proposed for Object-Oriented Databases (*OODBs*)

[a]Corresponding author's address: Dipartimento di Scienze dell'Ingegneria
Via Campi 213/B, I-41100 - Modena, Italy
E-mail: {benevent,sonia}@dsi.unimo.it
Phone: +39 59 374057 - Fax: +39 59 364132

[6, 26], are based on the notions of object, class, attribute, inheritance and method, where classes and attributes are used to describe the structural aspects, while methods are used to represent behavioural aspects. Complex object data models (*CODMs*), recently proposed in the database area [2, 3, 4, 7, 14, 28, 29] describe only structural aspects. They represent well–known notions such as types, complex values, classes, objects with identity and inheritance. *Description Logics languages - DLs* [1], derived from the KL-ONE model [40], have been proposed in the 80's in the Artificial Intelligence research area. DLs are fragments of first order logic: they enable *concepts* to be expressed, that can be viewed as logical formulas built using unary and binary predicates, and contain one free variable (to be filled with instances of the concept). They bear similarities with CODMs as they are concerned with only structural aspects; concepts roughly correspond to database classes and are organized in inheritance taxonomies. An additional feature of DLs with respect to CODMs is that concepts are differentiated in *primitive* and *defined*: a primitive concept description represents necessary conditions (thus corresponding to the usual database class semantics); a defined concept description represents necessary and sufficient conditions (thus corresponding to a database view semantics). By exploiting defined concepts semantics, and, given a *type as set* semantics to concept descriptions, it is possible to provide reasoning techniques: to compute *subsumption* relations (i.e. "isa" relationships implied by concepts descriptions) and to detect *incoherent* (i.e. always empty) concepts. The research on DLs has provided reasoning techniques to determine incoherence and subsumption of concepts and has assessed the complexity of these inferences for a variety of *acyclic* , i.e. not allowing recursive descriptions, DLs (see e.g.[23]).

These techniques are profitable for database design activities, as will be briefly argued in the following. In fact, if we map a database schema including only classes (no view) into one of the DLs supported by a system, we are able to automatically detect incoherent classes. Incoherence of a schema can thus be avoided by preventing the introduction of incoherent classes. A more active role can be performed with the introduction of views. Given a new view, it can be automatically *classified* (i.e., its right place in an already existing taxonomy can be found) by determining the set of its most specific subsumer views (*subsumers*) and the set of its most generalized specialization views (*subsumees*). Thus, besides a passive consistency check, *minimality* of the schema with respect to inheritance can easily be computed. In [15], the effectiveness of these techniques in a database environment with respect to well-known conceptual data models (E/R [22], Taxis[32], Daplex[39]), was shown: these models were mapped in a suitable acyclic DL and polynomial subsumption and coherence algorithms were given. Another contribution for the E/R model can be found in [10].

The expressiveness of CODMs gives rise to new problems for this mapping, as many of their features are not supported by implemented DLs. First of all, CODMs support the representation and management of *cyclic classes*, i.e., classes which directly or indirectly refer to themselves. (No DL-based system support cycles and only in [8, 34, 35] cycles in DLs were investigated from a theoretical point of view.). Secondly, most of the CODMs introduce a clear cut distinction between values and objects with identity and, thus, between value types and object classes (No DL supports this distinction). Thirdly, CODMs support additional type constructors, such as tuple, set and sequence which force to develop more complex controls to detect incoherence and compute subsumption relations.

[1] *DLs* are also known as *Concept Languages* or *Terminological Logics*.

Recently, a description logics (ODL = *Object Description Logics*) for CODMs overcoming these problems was proposed in [14]. The ODL formalism represents the structural part of OODB data models and incorporates: value-types, complex values, classes, objects with identity, and inheritance, in full analogy with CODMs. The main extensions of ODL, with respect to CODMs, are: the *conjunction* operator, which permits the expression of multiple *inheritance* as part of a class description, and, secondly, the capability of expressing *base* and *virtual* classes. Base classes correspond to ordinary classes used in database systems and virtual classes (corresponding to defined concepts semantics) are similar to database *views* [2]. Cyclic classes (base and virtual) are allowed and a *greatest fixed-point* semantics was adopted to interpret cyclic classes. A theoretical framework for database design based on ODL, subsumption computation and coherence detection was proposed in the same paper [14].

Coherence checking and subsumption computation can also be profitable for query optimization. In fact, a query has the semantics of a virtual class, as it expresses a set of necessary and sufficient conditions. If we restrict the query language (*DML*) to the subset of queries expressible with the schema description language (*DDL*), thus imposing DML=DDL, we can perform incoherence detection and subsumption computation for queries. This choice was made in the first works on query optimization based on acyclic DLs such as CANDIDE [11], CLASSIC [17] and BACK [36].

Coherence checking and subsumption computation may be classified as *semantic query optimization* techniques [27, 21, 38], as they perform a transformation of a query into a semantically equivalent one, that minimizes query execution costs. In fact, if the query is detected as incoherent a null answer can immediately be returned without accessing the database. Furthermore, provided the query is coherent, it can be temporarily classified (by subsumption computation) in the schema with respect to pre-existing views. If views are materialized, we obtain query optimization: in fact, as a result, either an *equivalent* view or the set of *immediate subsumers* and *immediate subsumees* is returned. In the former case, the answer set is simply the set of all objects that are instances of the view equivalent to the query. In the latter case, the union of the sets of instances of immediate subsumee views are included in the answer set and only objects that are instances of the immediate subsumer views, but not instances of the immediate subsumee views, have to be checked against the query condition.

As a matter of fact, usually, query languages are more expressive than DDLs. This holds for Relational Databases and is still true for OODBs, even if the high expressivity of CODMs shortened the gap between DDL and DML. Assuming a query language to be more expressive than the DDL, we can ideally introduce a separation of a query into a *clean* part, that can be represented in the DDL, in conjunction with a *dirty* part that goes beyond the DDL expressiveness, as was firstly proposed for OODBs in [18]. In this way, coherence checking and subsumption computation can be performed only over the clean part of a given query (i.e., on a query subsuming the target one). It follows that if queries are formulated in the general query language the set of queries detected as incoherent and the subsumption relations computed could be incomplete. Obviously, for the subset of queries with no dirty part, coherence check and subsumption computation are complete.

On the other hand, the complexity of query languages in OODBs makes a general solution for coherence and subsumption computation impossible. The point is, increasing the expressiveness

[2]Views in OODB have recently been called virtual classes by Abiteboul and Bonner [1].

of formalisms leads to an increased complexity of reasoning activities, (see e.g.[23] for acyclic DLs). It is particularly true if cyclic descriptions are allowed. In [35], for example, it was proved that subsumption computation for DLs including both cyclic defined concepts and equality comparisons between attributes is undecidable.

Recursive [3] queries are relevant for OODB query languages. For instance, object-oriented recursive queries are important in engineering DBs [20]. Unfortunately, while many works have been developed in deductive relational databases [24, 31, 5], very few proposals have been presented for OODB query languages. Recently, a significant contribution in OODB query languages was made in [16], where a form of recursion which corresponds to the notion of linear recursive queries was introduced.

The aim of this paper is to show the effectiveness of incoherence detection and subsumption computation for two tasks. The first one is the validation of a database schema, including (cyclic) classes and views, and of (recursive) queries. The second one is the automatic classification of (cyclic) views and (recursive) queries in the schema. We use ODL as a common formalism for the schema and for the queries.

To have greater insight into cyclic views and queries semantics we analyze and discuss the adoption of different candidate styles of semantics: *greatest fixed-point*, *least fixed-point*, and *descriptive semantics* and the consequences on subsumption computation and incoherence detection. We will show that the three semantics lead to different interpretations of cyclic views and queries: least fixed-point semantics captures (the recursive definition of) all the acyclic object structures; greatest fixed-point semantics captures (the recursive definition of) both cyclic and acyclic object structures; descriptive semantics represents the many interpretations that come between the two above mentioned.

We observe that, if we assume database schemata including both classes and views, we may choose any of the above mentioned semantics to interpret a database view. On the other hand, only the two fixed-point semantics are suitable to interpret a cyclic query as they allow to uniquely determine a query answer set. Incoherence and subsumption for cyclic views and queries are thus studied assuming that different semantics for the schema (classes + views) and the queries be adopted. This general assumption leads to the development of a theoretical framework including incoherence and subsumption algorithms for each of the three considered semantics.

In order to evaluate the complexity of these problems, according to the approach developed in AI environment [8, 34, 35] for cyclic DLs and for ODL in database environment [13], we will reduce subsumption and coherence for each of the different styles of semantics (see Appendix) to properties of regular languages by specifying a mapping from ODL schemata to nondeterministic finite automata. We will show that, even if from a theoretical point of view these problems are intractable, they can be efficiently solved by, firstly, transforming a schema in a "canonical" form and, secondly, performing subsumption computation and incoherence detection on a canonical schema. In fact, the intractability previously mentioned is due to very rare worst-case (see [34]) of canonical transformation, but in practice a schema is generally formulated in such a way as to be "almost" canonical.

As far as queries are concerned, and consequently query optimization, our work extends the framework of the papers based on acyclic DLs [11, 17, 18], as we are able to express *recursive*

[3]In the following cyclic and recursive will be used as synonymous.

queries by means of cyclic virtual classes. Furthermore, the analysis and discussion of different semantics give rise to interesting comparisons with the semantics of cyclic virtual classes and the semantics of the fixed-point operators introduced in object-oriented query languages to express recursive queries [16].

Our work generalizes the theoretical framework proposed in [14] for database design which is based on greatest fixed-point semantics, as we analyze and solve incoherence and subsumption problems for cyclic virtual classes with respect to other styles of semantics, say least fixed-point and descriptive. In particular, for least-fixed point semantics, we can partially reuse the algorithms developed for the greatest-fixed point semantics in [14] and we need to develop a new algorithm to detect incoherences. For descriptive semantics, we can reuse the greatest-fixed point incoherence algorithm, but we need to extend the definition of the canonical form of a schema and to develop a new subsumption algorithm.

With respect to the works on cyclic DLs [8, 34, 35], a more complex mapping from schemata to automata and new techniques to detect structured value–types incoherence were developed.

The rest of the paper is organized as follows. In the first subsection we introduce an example to show the effectiveness of the approach. The second section contains: the specification of ODL; the definitions of the legal instance of a database and the formal definitions of subsumption and coherence for the different semantics proposed. Moreover, it throws light in the meaning and the consequences of adopting one or another semantics for database views and queries. Section 3 contains the algorithms to solve the subsumption and incoherence problems under the different semantics. Appendix A solves the computational problems of subsumption and incoherence with the help of finite automata theory and discusses their complexity.

1.1 The Company Domain Example

In order to illustrate our approach, let us consider the following description of part of the organizational structure of a Company (see figure 1), where classes are represented by ellipses, explicit specialization relations by solid arrows, computed subsumption relations by dashed arrows and attributes by oriented arcs (notice that some attributes are not drawn to make the figure more readable).

Departments are described by a name and a category (from 5 to 7), employ only employees and have at least one employee with a typing function. Employees have a name, work in a department and have a skill level (from 1 to 10). Medium level employees (`Mdl-Employee`) are *exactly* those employees having a medium level (from 5 to 10), whereas Advanced level employees (`Adl-Employee`) are *exactly* those employees having a high level (from 8 to 10). Offices are *exactly* those departments which employ only clerks; clerks, in their turn, are *exactly* those medium level employees who work in an office. Secretaries are *exactly* those advanced level employees working in a T–office and T-offices are, in their turn, *exactly* those particular departments which employ only secretaries and have at least one secretary with a typing function.

Intuitively, descriptions without *exactly* introduce base classes (indicated by an asterisk in figure 1), whereas those with *exactly* introduce virtual classes (views). Moreover, note that `Clerk` and `Department`, `Office` and `Secretary` are cyclic views with mutual references. Using an OODB–like syntax, the Company domain can be described as in Table 1.

From an extensional point of view, this means that, for example, we have to explicitly fill the

$$
\begin{array}{rcl}
\underline{\textbf{class}}\ \texttt{Department} & = & \big[\texttt{dname}\colon \texttt{String}, \texttt{category}\colon 5\text{–}7, \\
 & & \ \texttt{employs}\colon \{\texttt{Employee}\}, \texttt{typing}\colon \texttt{Employee}\big] \\
\underline{\textbf{class}}\ \texttt{Employee} & = & \big[\texttt{name}\colon \texttt{String}, \texttt{worksin}\colon \texttt{Department}, \texttt{level}\colon 1\text{–}10\big] \\
\underline{\textbf{view}}\ \texttt{Mdl-Employee} & = & \underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \big[\texttt{level}\colon 5\text{–}10\big] \\
\underline{\textbf{view}}\ \texttt{Adl-Employee} & = & \underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \big[\texttt{level}\colon 8\text{–}10\big] \\
\underline{\textbf{view}}\ \texttt{Clerk} & = & \underline{\textbf{isa}}\ \texttt{Mdl-Employee}\ \underline{\textbf{and}}\ \big[\texttt{worksin}\colon \texttt{Office}\big] \\
\underline{\textbf{view}}\ \texttt{Office} & = & \underline{\textbf{isa}}\ \texttt{Department}\ \underline{\textbf{and}}\ \big[\texttt{employs}\colon \{\texttt{Clerk}\}\big] \\
\underline{\textbf{view}}\ \texttt{Secretary} & = & \underline{\textbf{isa}}\ \texttt{Adl-Employee}\ \underline{\textbf{and}}\ \big[\texttt{worksin}\colon \texttt{T-Office}\big] \\
\underline{\textbf{view}}\ \texttt{T-Office} & = & \underline{\textbf{isa}}\ \texttt{Department}\ \underline{\textbf{and}}\ \big[\texttt{employs}\colon \{\texttt{Secretary}\}, \\
 & & \qquad\qquad\qquad\qquad \texttt{typing}\colon \texttt{Secretary}\big]
\end{array}
$$

Table 1: The Company domain schema in OODB–like syntax

base class `Employee`, while every object which is inserted as an employee having a level from 5 to 10 can be automatically inserted in the view `Mdl-Employee`. Moreover, from an intensional point of view, views enable *subsumption relations* (i.e. specialization relations implicitly hidden in class descriptions) to be computed. In figure 1 a computed subsumption relation between `Adl-Employee` and `Mdl-Employee` is represented by a dashed arrow. In fact, even if it has not been explicitly asserted, `Adl-Employee` is a sub-class of the class `Mdl-Employee` as each of its objects is also an object of the class `Mdl-Employee`.

In order to show as coherence checking and subsumption computation may be useful for query optimization let us introduce some queries on the above schema.

First of all, in order to show a simple example of incoherence detection, let us introduce the query $Q1$ "select the T-Office with a typist of level 4":

$$\underline{\textbf{query}}\ Q_1 \quad = \quad \underline{\textbf{isa}}\ \texttt{T-Office}\ \underline{\textbf{and}}\ \big[\texttt{typing}\colon\ \underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ [\texttt{level}\colon 4]\big]$$

The query is detected as incoherent. In fact, the set of employees having a `level` interval (4-4) is disjoint from the set of objects belonging to the class `Secretary` (`level` interval (8-10)) which must fill the `typing` property of `T-Office` (see `T-Office` definition). A result, in terms of query optimization, is that a null answer can be immediately returned without accessing the database.

As an example of automatic query classification (by subsumption computation), let us introduce the query Q_2 "select the employees having a level greater or equal to 6":

$$\underline{\textbf{query}}\ Q_2 \quad = \quad \underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \big[\texttt{level}\colon 6\text{–}\infty\big]$$

Classifying the query Q_2 into the classes taxonomy returns the `Mdl-Employee` view as the only immediate subsumer and the `Adl-Employee` view as the only immediate subsumee (see figure 2). Using this result, all instances of `Adl-Employee` are returned and only the instances of `Mdl-Employee` that are not instances of `Adl-Employee` are checked against the query–condition ("level greater or equal to 6") and included in the answer-set if they satisfy this condition.

Assuming `Adl-Employee` and `Mdl-Employee` as materialized views, an interesting query optimization result is thus obtained.

The problem is more complicated if we want to detect incoherences and to compute subsumption relations in the presence of recursive views and queries, as, from a theoretical point of view, we have to select (the most appropriate) among different semantics (least fixed-point, greatest fixed-point and descriptive) to interpretet them.

As different semantics lead to different automatic insertion of objects in virtual cyclic classes, it follows that the sets of computed subsumption relations and detected incoherences are different for different semantics.

As an example, let us introduce the following recursive query Q_3 "select the employees having a level greater or equal to 6 working in a department which employs only employees which *recursively* have the same properties" (i.e. an employee having a level greater or equal to 6 but working in a department which employs at least one employee having a level less than 6 does not belong to the answer set of query Q_3):

$$\underline{\textbf{query}}\ Q_3 \;=\; \underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \Big[\texttt{level}{:}\,6\text{--}\infty,\ \texttt{worksin}{:}\ \ \underline{\textbf{isa}}\ \texttt{Department}\ \underline{\textbf{and}}\ [\texttt{employs}{:}\,\{Q_3\}]\Big]$$

For the different semantics we can have different answers and query optimization results (see figure 3). By adopting descriptive semantics, the query Q_3 is pushed down in the schema hierarchy as a new subsumption relation is computed with respect to the `Mdl-Employee` view. By adopting least fixed-point semantics, the query Q_3 is pushed a little more lower down in the schema hierarchy as a further subsumption relation is computed between the cyclic view `Clerk` and Q_3. By adopting greatest fixed-point semantics Q_3 is classified in the schema hierarchy between the view `Clerk` (its lowest generalization) and the view `Secretary` (its highest specialization).

More generally, figure 3 shows the results of subsumption and coherence computation of the example schema with respect to the three semantics and the employees hierarchy. We can see that the adoption of descriptive semantics in most cases prevents computing subsumption relations for cyclic views (see `Secretary` and `Clerk`); with least fixed-point semantics the view `Secretary` has been detected as incoherent; with greatest fixed-point semantics no view has been detected as incoherent and a new subsumption relation has been computed between the cyclic views `Secretary` and `Clerk`.

Finally, in order to show an example of a query going beyond our DDL expressiveness[4], let us introduce the following query: "select the employees having a level greater or equal to 6 working as typists":

$$\underbrace{\underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \Big[\texttt{level}{:}\,6\text{--}\infty\Big]}_{\text{clean part}}\ \underline{\textbf{and}}\ \underbrace{\Big(\underline{\textbf{self}} = \texttt{worksin.typing}\Big)}_{\text{dirty part}}$$

The dirty part of the above query exploits *self* reference and *equality path* as usually happens in OODB query languages: it selects only the employees who play the typing role in the

[4]Allowing cycles and equality comparisons between attributes leads to undecidability [35] of subsumption computation.

department where they work. Note that, in this case, even if we are able to perform subsumption computation only over the clean part of a query, we can obtain an optimization result as the above query is subsumed by the `Mdl-Employee` view.

The last two queries show that subsumption and incoherence are not computable as they are hidden in the dirty part:

$$\underline{\textbf{isa}}\ \texttt{Employee}\ \underline{\textbf{and}}\ \Big(\texttt{level}\ =\ \texttt{worksin.category}\Big)$$

$$\underline{\textbf{isa}}\ \texttt{Adl-Employee}\ \underline{\textbf{and}}\ \Big(\texttt{level}\ =\ \texttt{worksin.category}\Big)$$

Intuitively, the first query ("select the employees having a level equal to the category of the department where they work") is subsumed by the view `Mdl-Employee`; the second query ("select the advanced level employees having a level equal to the category of the department where they work") is incoherent.

2 The ODL formalism

We adopt the ODL formalism proposed in [14]; a reader familiar to ODL can skip subsections 2.1 and 2.2. Since a greatest fixed-point semantics was adopted in [14], we generalize the theoretical framework proposed there by adding least fixed-point and descriptive semantics.

2.1 Atomic Types

Let $\mathcal{D}$ be the countably infinite set of atomic values (which are denoted by $d_1, d_2, \ldots$), e.g., the union of the set of integers, the set of strings, and the booleans. No distinction will be made between atomic values and their encoding.

Let $\mathbf{B}$ be a countable set of designators for atomic types that contains $\mathcal{D}$, and let $\mathcal{I}_{\mathbf{B}}$ be the (fixed) standard interpretation function from $\mathbf{B}$ to $2^{\mathcal{D}}$ such that for all $d \in \mathcal{D}$: $\mathcal{I}_{\mathbf{B}}[d] = \{d\}$. Let "$\sqcap$" be an operation on $\mathbf{B}$ defined by:

$$B' \sqcap B'' = B \ \text{ iff } \ \mathcal{I}_{\mathbf{B}}[B'] \cap \mathcal{I}_{\mathbf{B}}[B''] = \mathcal{I}_{\mathbf{B}}[B].$$

We say that $\mathbf{B}$ is a *system of atomic types* iff $\mathbf{B}$ is complete with respect to $\sqcap$. The special type, denoted by $\perp$, that has an empty interpretation, $\mathcal{I}_{\mathbf{B}}[\perp] = \emptyset$, will be called *empty type*; this type must be part of $\mathbf{B}$ because $\mathcal{I}_{\mathbf{B}}[d_1] \sqcap \mathcal{I}_{\mathbf{B}}[d_2] = \emptyset$ if $d_1 \neq d_2$. We say that $\mathbf{B}$ is a *PTIME system* iff $B' \sqcap B'' = B$ can be decided in polynomial time. In the following we assume that the system of atomic types has this property. Note that this assumption does not allow the rapresentation of enumeration types but is a prerequisite to guarantee polynomial subsumption and incoherence algorithms.

Sometimes we will also talk about systems of atomic types with a particular simple structure, namely, systems such that for each subset $\mathbf{X} \subseteq \mathbf{B}$ with $\bigsqcap \mathbf{X} = B$, there are two elements $B', B'' \in \mathbf{X}$ such that $B' \sqcap B'' = B$. Such systems of atomic types are called *binary compact*.

Consider the following set of designators for atomic types, which we use in all our examples: $\mathbf{B} = \{\texttt{Int}, \texttt{String}, \texttt{Bool}, \texttt{Real}, i_1\text{–}j_1, i_2\text{–}j_2, \ldots, d_1, d_2, \ldots\}$, where the $i_k\text{–}j_k$'s denote all possible ranges of integers, and the d_k's denote all the elements [5] of $\texttt{Int} \cup \texttt{String} \cup \texttt{Bool}$. Assuming

[5] Note that a d_k is an element, whose interpretation is a singleton set.

the standard interpretation of the type designators, $\mathbf{B}$ is obviously a binary compact system of atomic types.

2.2 Database Schema

We suppose a countable set $\mathbf{A}$ of *attributes* (denoted by $a_1, a_2, \ldots$) and a countable set $\mathbf{C}$ of class names (denoted by C, C') such that $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are pairwise disjoint.

$\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ [6] denotes the set of all *finite type descriptions* (S, S', ...), also briefly called *types*, over given $\mathbf{A}, \mathbf{B}, \mathbf{C}$, that are built according to the following abstract syntax rule (assuming $a_i \neq a_j$ for $i \neq j$):

$$
\begin{array}{lll}
S \rightarrow & \top & \text{top type} \\
 & B & \text{atomic type} \\
 & C & \text{class name} \\
 & \{S\} & \text{set type} \\
 & \langle S \rangle & \text{sequence type} \\
 & [a_1 \colon S_1, \ldots, a_k \colon S_k] & \text{tuple type} \\
 & S \sqcap S' & \text{types conjunction} \\
 & \triangle S & \text{object type}
\end{array}
$$

Most of the type constructors are common in CODM; $\triangle$ is unusual and represents an object set forming constructor, i.e. $\triangle S$ denotes the objects whose state is of type S.

Given a finite set $\mathbf{C_B}$ of *base class* names (denoted by $P, P' \ldots$) a *base schema* σ_B is a total function from $\mathbf{C_B}$ to $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C_B})$. Intuitively, σ_B associates base class names to their descriptions.

Given a finite set $\mathbf{C_V}$ of *virtual class* names (denoted by $V, V' \ldots$) disjoint from $\mathbf{C_B}$, a *virtual schema* σ_V is a total function from $\mathbf{C_V}$ to $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$, where $\mathbf{C} = \mathbf{C_B} \cup \mathbf{C_V}$. Like σ_B, σ_V associates virtual class names to their descriptions.

The union of σ_B and σ_V, denoted by σ, is a total function from $\mathbf{C}$ to $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$; note that we have constrained base class descriptions not to refer to any virtual class. The Company domain described in Section 1.1, for instance, would lead to the ODL schema in Table 2.

Cyclic classes are permitted: in fact, as a class name may appear in type descriptions, we can have *circular references*, that is, class names which make direct or indirect references to themselves. Formally, cyclic classes are recognized by means of the notion of *dependence*. We say that C *depends on* C', written $C \hookrightarrow C'$, where $C, C' \in \mathbf{C}$, if C' is contained in the type $\sigma(C)$; the transitive closure of $\hookrightarrow$ is denoted by $\overset{+}{\hookrightarrow}$. We say that $C \in \mathbf{C}$ is *cyclic*, iff $C \overset{+}{\hookrightarrow} C$.

A key feature of ODL is the expression of the *isa* relation by the conjunction operator (compare tables 1 and 2). We say that C *inherits directly from* C', written $C \prec\!\!| C'$ iff $\sigma(C) = S_1 \sqcap \ldots \sqcap S_n$, $n > 0$, and $C' = S_i$ for some i, $1 \leq i \leq n$. We require that the inheritance relation expressed by conjunctions in a class name definition be cycle–free. A schema is *inheritance well-formed* iff the transitive closure of $\prec\!\!|$, which is denoted by $\prec_\sigma$, is a strict partial order. The reflexive and transitive closure is denoted by $\preceq_\sigma$. Reconsidering our Company example, it can easily be verified that this schema is inheritance well-formed.

[6] In the following, we also will write $\mathbf{S}$ instead of $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ when $\mathbf{A}, \mathbf{B}$, and $\mathbf{C}$ are clear from the context.

$$
\begin{array}{rcl}
\mathbf{C_B} & = & \{\texttt{Department}, \texttt{Employee}\} \\
\sigma_B(\texttt{Department}) & = & \triangle[\texttt{dname}\colon \texttt{String}, \texttt{category}\colon 5\text{--}7, \\
 & & \quad \texttt{employs}\colon \{\texttt{Employee}\}, \texttt{typing}\colon \texttt{Employee}] \\
\sigma_B(\texttt{Employee}) & = & \triangle[\texttt{name}\colon \texttt{String}, \texttt{worksin}\colon \texttt{Department}, \texttt{level}\colon 1\text{--}10] \\
 & & \\
\mathbf{C_V} & = & \{\texttt{Mdl-Employee}, \texttt{Adl-Employee}, \texttt{Clerk}, \texttt{Office} \\
 & & \ \texttt{Secretary}, \texttt{T-Office}\} \\
\sigma_V(\texttt{Mdl-Employee}) & = & \texttt{Employee} \sqcap \triangle[\texttt{level}\colon 5\text{--}10] \\
\sigma_V(\texttt{Adl-Employee}) & = & \texttt{Employee} \sqcap \triangle[\texttt{level}\colon 8\text{--}10] \\
\sigma_V(\texttt{Clerk}) & = & \texttt{Mdl-Employee} \sqcap \triangle[\texttt{worksin}\colon \texttt{Office}] \\
\sigma_V(\texttt{Office}) & = & \texttt{Department} \sqcap \triangle[\texttt{employs}\colon \{\texttt{Clerk}\}] \\
\sigma_V(\texttt{Secretary}) & = & \texttt{Adl-Employee} \sqcap \triangle[\texttt{worksin}\colon \texttt{T-Office}] \\
\sigma_V(\texttt{T-Office}) & = & \texttt{Department} \sqcap \triangle[\texttt{employs}\colon \{\texttt{Secretary}\}, \\
 & & \qquad\qquad\qquad \texttt{typing}\colon \texttt{Secretary}]
\end{array}
$$

Table 2: The Company domain schema in ODL syntax

2.3 Database Instances

In this subsection a formal semantics for the ODL type system is given. The semantics is *set–theoretical*, similar to [19] and adopted in the O_2 system [28, 29]. We will formally define legal states of a database by introducing the notions of *possible instance* and *legal instance* of a schema with respect to descriptive, least fixed-point and greatest fixed-point semantics.

We assume a countable set $\mathcal{O}$ of *object identifiers* (denoted by $o, o', \ldots$) disjoint from $\mathcal{D}$ and define the set $\mathcal{V}(\mathcal{O})$ of all *values over* $\mathcal{O}$ (denoted by v, v') as follows (assuming $p \geq 0$ and $a_i \neq a_j$ for $i \neq j$):

$$
\begin{array}{rcll}
v & \to & d & \text{atomic value} \\
 & & o & \text{object identifier} \\
 & & \{v_1, \ldots, v_p\} & \text{set value} \\
 & & \langle v_1, \ldots, v_p \rangle & \text{sequence value} \\
 & & [a_1\colon v_1, \ldots, a_p\colon v_p] & \text{tuple value}
\end{array}
$$

Given a set of object identifiers $\mathcal{O}$, a *domain* δ over $\mathcal{O}$ is a total function from $\mathcal{O}$ to $\mathcal{V}(\mathcal{O})$. Intuitively, object identifiers are assigned values by the function δ. For instance, we may have the assignments of values to objects of table 3, related to Company database schema.

Now we define an *interpretation function*, which associates a set of values to every value-type and a set of objects to every class type. Given a type system $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ and a domain δ over $\mathcal{O}$, the *interpretation function* $\mathcal{I}$ of $\mathbf{S}$ over δ is a function from $\mathbf{S}$ to $2^{\mathcal{V}(\mathcal{O})}$ such that:

$$
\mathcal{I}[\top] \;=\; \mathcal{V}(\mathcal{O})
$$

$\mathcal{O}$	$=$	$\{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9\}$
$\delta(o_1)$	$=$	$[\mathtt{name}: \textit{“Mark”}, \mathtt{worksin}: o_2, \mathtt{level}: 3]$
$\delta(o_3)$	$=$	$[\mathtt{name}: \textit{“Robert”}, \mathtt{worksin}: o_2, \mathtt{level}: 6]$
$\delta(o_5)$	$=$	$[\mathtt{name}: \textit{“Andy”}, \mathtt{worksin}: o_4, \mathtt{level}: 7]$
$\delta(o_7)$	$=$	$[\mathtt{name}: \textit{“Peter”}, \mathtt{worksin}: o_6, \mathtt{level}: 8]$
$\delta(o_9)$	$=$	$[\mathtt{name}: \textit{“Franz”}, \mathtt{worksin}: o_8, \mathtt{level}: 9]$
$\delta(o_2)$	$=$	$[\mathtt{dname}: \textit{“Administration”}, \mathtt{category}: 5, \mathtt{employs}: \{o_1, o_3\}, \mathtt{typing}: o_3]$
$\delta(o_4)$	$=$	$[\mathtt{dname}: \textit{“Development”}, \mathtt{category}: 6, \mathtt{employs}: \{o_5\}, \mathtt{typing}: o_5]$
$\delta(o_6)$	$=$	$[\mathtt{dname}: \textit{“Research”}, \mathtt{category}: 7, \mathtt{employs}: \{o_7\}, \mathtt{typing}: o_9]$
$\delta(o_8)$	$=$	$[\mathtt{dname}: \textit{“Project”}, \mathtt{category}: 7, \mathtt{employs}: \{\}, \mathtt{typing}: o_9]$

Table 3: A possible database state

$$
\begin{aligned}
\mathcal{I}[B] &= \mathcal{I}_{\mathbf{B}}[B] \\
\mathcal{I}[C] &\subseteq \mathcal{O} \\
\mathcal{I}[\{S\}] &= \Big\{\{v_1, \ldots, v_p\} \Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\} \\
\mathcal{I}[\langle S\rangle] &= \Big\{\langle v_1, \ldots, v_p\rangle \Big| v_i \in \mathcal{I}[S], 0 \leq i \leq p\Big\} \\
\mathcal{I}\Big[[a_1\colon S_1, \ldots, a_p\colon S_p]\Big] &= \Big\{[a_1\colon v_1, \ldots, a_q\colon v_q] \Big| \, p \leq q, v_i \in \mathcal{I}[S_i], 0 \leq i \leq p, \\
&\qquad\qquad v_j \in \mathcal{V}(\mathcal{O}), p+1 \leq j \leq q\Big\} \\
\mathcal{I}[S \sqcap S'] &= \mathcal{I}[S] \cap \mathcal{I}[S'] \\
\mathcal{I}[\triangle S] &= \Big\{o \in \mathcal{O} \Big| \delta(o) \in \mathcal{I}[S]\Big\}.
\end{aligned}
$$

Note that the interpretation of tuples implies an open world semantics for tuple types similar to the one adopted by Cardelli [19]:

$$
\begin{aligned}
[\mathtt{name}: \textit{“Mark”}, \mathtt{worksin}: o_2, \mathtt{level}: 3] &\in \mathcal{I}\Big[[\mathtt{name}\colon \mathtt{String}]\Big] \\
\Big[\mathtt{name}\colon [\mathtt{dname}\colon \textit{“Research”}, \mathtt{sname}\colon \textit{“DB”}]\Big] &\in \mathcal{I}\Big[[\mathtt{name}\colon [\mathtt{dname}\colon \mathtt{String}]]\Big]
\end{aligned}
$$

The above–defined interpretation function is not sufficient to ensure that values and objects be *legal*, i.e. satisfy for class names their descriptions. For this purpose, we have to further constrain the interpretation function as follows.

Given a schema $\sigma = \sigma_B \cup \sigma_V$ on $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ with $\mathbf{C} = \mathbf{C_B} \cup \mathbf{C_V}$ and a domain δ over $\mathcal{O}$, let $\mathcal{I}$ be an interpretation of $\mathbf{S}$ over δ. We say that $\mathcal{I}$ is an *instance* of the base schema σ_B iff the set $\mathcal{O}$ is *finite*, and for all $P \in \mathbf{C_B}$:

$$\mathcal{I}[P] \subseteq \mathcal{I}[\sigma_B(P)].$$

	$\mathcal{I}_1$ *(greatest)*	$\mathcal{I}_2$ *(descriptive)*	$\mathcal{I}_3$ *(least)*
`Clerk`	$\{o_5, o_7, o_9\}$	$\{o_5, o_9\}$	$\{o_9\}$
`Office`	$\{o_4, o_6, o_8\}$	$\{o_4, o_8\}$	$\{o_8\}$
`Secretary`	$\{o_7, o_9\}$	$\{o_7, o_9\}$	$\{\}$
`T-Office`	$\{o_6, o_8\}$	$\{o_6, o_8\}$	$\{\}$

Table 4: Examples of possible instances for cyclic virtual classes

From the above definition, we see that the interpretation of a base class *is included* in the interpretation of its description, i.e. it has to be provided by the user, according to the given description.

Now, given an instance of the base schema σ_B, we have to determine the extension of a virtual class starting from the extensions of base classes. To this purpose, we first introduce the notion of a possible instance.

We say that an instance of a base schema σ_B is a *possible instance* of a schema σ iff for all $V \in \mathbf{C_V}$:

$$\mathcal{I}[V] \;=\; \mathcal{I}[\sigma_V(V)].$$

As an example of a possible instance, let us consider the domain δ of table 3 and the following instance $\breve{\mathcal{I}}$ of the base schema σ_B:

$$\begin{aligned} \breve{\mathcal{I}}[\texttt{Employee}] &= \left\{\ o_1, o_3, o_5, o_7, o_9\ \right\} \\ \breve{\mathcal{I}}[\texttt{Department}] &= \left\{\ o_2, o_4, o_6, o_8\ \right\} \end{aligned}$$

The instances of the acyclic virtual classes `Mdl-Employee` and `Adl-Employee` can be uniquely computed as:

$$\begin{aligned} \mathcal{I}[\texttt{Mdl-Employee}] &= \left\{\ o_3, o_5, o_7, o_9\ \right\} \\ \mathcal{I}[\texttt{Adl-Employee}] &= \left\{\ o_7, o_9\ \right\} \end{aligned}$$

On the other hand, we can have many possible instances for cyclic virtual classes, as shown in table 4.

In order to formally characterize the set of possible instances, given a schema σ and a domain δ, let $\Psi_{\breve{\mathcal{I}}}$ be the set of interpretations with the same instance $\breve{\mathcal{I}}$ of the base schema σ_B:

$$\Psi_{\breve{\mathcal{I}}} = \left\{\mathcal{I} \mid \mathcal{I}[P] = \breve{\mathcal{I}}[P] \;\text{ for all } P \in \mathbf{C_B}\right\}$$

Let "$\trianglelefteq$" be a relation over $\Psi_{\check{\mathcal{I}}}$ such that for all $\mathcal{I}, \mathcal{I}' \in \Psi_{\check{\mathcal{I}}}$:

$$\mathcal{I} \trianglelefteq \mathcal{I}' \quad \text{iff} \quad \mathcal{I}[V] \subseteq \mathcal{I}'[V] \text{ for all } V \in \mathbf{C_V}.$$

Then $(\Psi_{\check{\mathcal{I}}}, \trianglelefteq)$ is a partial ordering which forms, together with componentwise set union as the least upper bound (*lub*) and componentwise set intersection as the greatest lower bound (*glb*), a complete lattice. The least element of $\Psi_{\check{\mathcal{I}}}$ is $\mathcal{I}_\perp$ ($\mathcal{I}_\perp[V] = \emptyset$, for all V) and the greatest element is $\mathcal{I}_\top$ ($\mathcal{I}_\top[V] = \mathcal{O}$, for all V).

Let Γ be a function mapping interpretation into interpretation of the set $\Psi_{\check{\mathcal{I}}}$ as follows:

$$\Gamma : \Psi_{\check{\mathcal{I}}} \longrightarrow \Psi_{\check{\mathcal{I}}}$$

such that, for all $V \in \mathbf{C_V}$:

$$\Gamma(\mathcal{I})[V] = \mathcal{I}[\sigma(V)]$$

i.e., given from an interpretation $\mathcal{I}$, the application of Γ determines the new interpretation of a virtual class V ($\Gamma(\mathcal{I})[V]$) as the interpretation of its description ($\mathcal{I}[\sigma(V)]$). Therefore, it is easy to see that a *fixed-point* of Γ, i.e. an interpretation $\mathcal{I}$ with the property $\Gamma(\mathcal{I}) = \mathcal{I}$, is a possible instance of σ.

Consequently, by applying some results concerning fixed-points , (see [30] and [37]) we have:

- since the function Γ is monotone, (for all $\mathcal{I}, \mathcal{I}' \in \Psi_{\check{\mathcal{I}}}$, $\mathcal{I} \trianglelefteq \mathcal{I}'$ implies $\Gamma(\mathcal{I}) \trianglelefteq \Gamma(\mathcal{I}')$) the set of fixed-points of Γ, denoted as $\mathbf{P}_{\check{\mathcal{I}}}$, is nonempty and forms a complete lattice, and thus a greatest fixed-point and a least fixed-point always exist.

- since the function Γ is upward continuous, (for any increasing chain $\mathcal{I}_0 \trianglelefteq \mathcal{I}_1 \trianglelefteq \mathcal{I}_2 \trianglelefteq \ldots$ we have $\Gamma(lub(\{\mathcal{I}_i; i \geq 0\})) = lub(\{\Gamma(\mathcal{I}_i); i \geq 0\})$), its least fixed-point can be computed as:

$$lfp(\Gamma) = \bigcup_{i \geq 0} \Gamma^i(\mathcal{I}_\perp)$$

- since the function Γ is downward continuous, (for any decreasing chain $\mathcal{I}_0 \trianglerighteq \mathcal{I}_1 \trianglerighteq \mathcal{I}_2 \trianglerighteq \ldots$ we have $\Gamma(glb(\{\mathcal{I}_i; i \geq 0\})) = glb(\{\Gamma(\mathcal{I}_i); i \geq 0\})$), its greatest fixed-point is computed as:

$$gfp(\Gamma) = \bigcap_{i \geq 0} \Gamma^i(\mathcal{I}_\top)$$

Therefore, one relevant consequence is that we can give three different characterizations of the semantics of schemata with cyclic virtual classes. Given a schema σ, a domain δ and a partial possible instance $\check{\mathcal{I}}$, we say that a *legal instance* of the schema σ is:

- *one* of the possible instances $\mathcal{I}$ from the set $\mathbf{P}_{\check{\mathcal{I}}}$

  descriptive semantics;

- the *least* instance (*lfp*–instance) $\mathcal{I}$ from the set $\mathbf{P}_{\check{\mathcal{I}}}$

  least fixed-point semantics (*lfp*–semantics);

- the *greatest* instance (*gfp*–instance) $\mathcal{I}$ from the set $\mathbf{P}_{\check{\mathcal{I}}}$

  greatest fixed-point semantics (*gfp*–semantics).

For schemata with no cyclic virtual classes the set $\mathbf{P}_{\check{\mathcal{I}}}$ consists of a single element, that is, the lfp–instance and the gfp–instance coincide. In a general schema, by adopting lfp–semantics or gfp–semantics we select as the legal instance a precise instance (the least or the greatest instance, respectively). This means that the interpretation of cyclic virtual classes is uniquely computable for a given possible instance of the base schema, as in the acyclic case. On the other hand, by adopting the descriptive semantics we loose the unique computation property and admit one of the possible instances as a legal instance.

The choice of one or another semantics (and thus of one or another legal instance) gives rise to relevant differences in the sets of computed subsumption relations and incoherent types, as subsumption and incoherence definitions are based on legal instances.

We can generalize the framework by allowing that for each cyclic virtual class of a schema one or another of the above semantics be chosen. In this way, $\mathbf{C_V}$ is partitioned into the sets $\mathbf{C}^l_{\mathbf{V}}$, $\mathbf{C}^g_{\mathbf{V}}$ and $\mathbf{C}^d_{\mathbf{V}}$, where $\mathbf{C}^l_{\mathbf{V}}$ consists of names for views under lfp–semantics (denoted by V^l), $\mathbf{C}^g_{\mathbf{V}}$ consists of names for views interpreted under gfp–semantics (denoted by V^g), and $\mathbf{C}^d_{\mathbf{V}}$ consists of names for views under descriptive semantics (denoted by V^d).

Given a schema $\mu = \sigma_B \cup \sigma_V$ on $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ with $\mathbf{C} = \mathbf{C_B} \cup \mathbf{C_V}$ and $\mathbf{C_V} = \mathbf{C}^l_{\mathbf{V}} \cup \mathbf{C}^g_{\mathbf{V}} \cup \mathbf{C}^d_{\mathbf{V}}$, let us suppose that one of the following *subschemata* exists:

- a virtual schema σ^g over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C_B} \cup \mathbf{C}^g_{\mathbf{V}})$ such that $\sigma^g(V^g) = \mu(V^g)$, for all $V^g \in \mathbf{C}^g_{\mathbf{V}}$;
- a virtual schema σ^l over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C_B} \cup \mathbf{C}^l_{\mathbf{V}})$ such that $\sigma^l(V^l) = \mu(V^l)$, for all $V^l \in \mathbf{C}^l_{\mathbf{V}}$

In this way, in the schema μ a view V^l (V^g) must refer to only views in $\mathbf{C}^l_{\mathbf{V}}$ ($\mathbf{C}^g_{\mathbf{V}}$), whereas no limitations on the description of a V^d view are introduced.

Given a domain δ, a schema μ and a possible instance $\check{\mathcal{I}}$ of σ_B, let $\mathcal{I}^g$ the gfp–instance of σ^g and let $\mathcal{I}^l$ the lfp–instance of σ^l. We say that a possible instance $\mathcal{I}$ of the schema μ is an *heterogeneous legal instance* of μ if:

1. $\mathcal{I}[V^g] = \mathcal{I}^g[V^g]$, for all $V^g \in \mathbf{C}^g_{\mathbf{V}}$;
2. $\mathcal{I}[V^l] = \mathcal{I}^l[V^l]$, for all $V^l \in \mathbf{C}^l_{\mathbf{V}}$.

In other words, an *heterogeneous semantics* is a descriptive semantics where we limit the legal instances of the schema by conditions 1 and 2.

2.4 Incoherence and Subsumption

The notion of coherence is associated to the possibility of populating a type with at least one value for some legal instance of the schema. In order to express this formally, let us define *incoherence* of a type S of a schema σ for the three different semantics:

$$\begin{array}{rcl} S \text{ is } \textit{des–incoherent} & \text{iff} & \mathcal{I}[S] = \emptyset \text{ for all legal instance } \mathcal{I} \text{ of } \sigma, \\ S \text{ is } \textit{lfp–incoherent} & \text{iff} & \mathcal{I}[S] = \emptyset \text{ for all lfp–legal instance } \mathcal{I} \text{ of } \sigma, \\ S \text{ is } \textit{gfp–incoherent} & \text{iff} & \mathcal{I}[S] = \emptyset \text{ for all gfp–legal instance } \mathcal{I} \text{ of } \sigma. \end{array}$$

In the following, we will indicate with *incoherence* one of the above-defined incoherence definitions in order to express properties which hold for incoherence in all the three semantics.

A schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ is called coherent iff for all $C \in \mathbf{C}$, C is coherent. Note that for the coherence of a schema σ over $\mathbf{S}$ it is required that every class name C be coherent, but not necessarily that every type description $S \in \mathbf{S}$ be coherent, since an incoherent type used in a set or sequence type does not give rise to an incoherence. In fact, $\{\perp\}$ and $\langle\perp\rangle$ are coherent types denoting the empty set and the empty sequence, respectively.

Let us formally express the *subsumption relation* between two types S, S' of a schema σ for the three different semantics:

$$
\begin{array}{lll}
S \sqsubseteq^d_\sigma S' & \text{iff} & \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all legal instances } \mathcal{I} \text{ of } \sigma, \\
S \sqsubseteq^l_\sigma S' & \text{iff} & \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all } \mathit{lfp}\text{–legal instances } \mathcal{I} \text{ of } \sigma, \\
S \sqsubseteq^g_\sigma S' & \text{iff} & \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all } \mathit{gfp}\text{–legal instances } \mathcal{I} \text{ of } \sigma.
\end{array}
$$

In the following, we will denote with $\sqsubseteq_\sigma$ one of the above defined subsumption relations in order to express properties which hold for $\sqsubseteq_\sigma$ in all the three semantics. First of all, it follows immediately that $\sqsubseteq_\sigma$ is a pre-order (i.e., transitive and reflexive but antisymmetric) that induces an *equivalence relation* $\simeq_\sigma$ on types:

$$S \simeq_\sigma S' \text{ iff } S \sqsubseteq_\sigma S' \text{ and } S' \sqsubseteq_\sigma S.$$

Finally, let us analyze the relationships between subsumption, coherence and inheritance. Obviously, a type S is incoherent iff $S \simeq_\sigma \perp$. The relation between inheritance and subsumption is intuitive: given a schema σ and $C, C' \in \mathbf{C}$:

$$\text{If } C \preceq_\sigma C' \text{ then } C \sqsubseteq_\sigma C'.$$

In fact, a class C that inherits from another class C', is always interpreted as a subset of the interpretation of C'. The converse does not necessarily hold. The main reason is the presence of virtual classes. Another reason is the fact that a class name can be incoherent; in fact, in a coherent schema we can give a partial converse of the above implication: given a coherent schema σ, a class name $C \in \mathbf{C}$ and a base class name $P \in \mathbf{C_B}$:

$$C \sqsubseteq_\sigma P \text{ iff } C \preceq_\sigma P.$$

In other words, in a coherent schema including only base classes the computed subsumption relations coincide with those explicitly given as inheritance relations.

Summarizing the relationships between subsumption and incoherence in the three different semantics, we have:

$$
\begin{array}{lclclcl}
S \sqsubseteq^l_\sigma S' & \overset{\Leftarrow}{\not\Rightarrow} & S \sqsubseteq^d_\sigma S' & (1) \qquad & S \simeq^l_\sigma \perp & \overset{\Leftarrow}{\not\Rightarrow} & S \simeq^d_\sigma \perp \quad (4) \\
S \sqsubseteq^g_\sigma S' & \overset{\Leftarrow}{\not\Rightarrow} & S \sqsubseteq^d_\sigma S' & (2) \qquad & S \simeq^g_\sigma \perp & \overset{\Leftarrow}{\Rightarrow} & S \simeq^d_\sigma \perp \quad (5) \\
S \sqsubseteq^l_\sigma S' & \overset{\Leftarrow}{\not\Rightarrow} & S \sqsubseteq^g_\sigma S' & (3) \qquad & S \simeq^l_\sigma \perp & \overset{\Leftarrow}{\not\Rightarrow} & S \simeq^g_\sigma \perp \quad (6)
\end{array}
$$

(1) and (2) state that subsumption w.r.t. descriptive semantics implies subsumption w.r.t. a least or greatest fixed-point semantics. In fact, adopting descriptive semantics all the possible instances (least and greatest included) have to be considered. (3) compares subsumption w.r.t. the two fixed-point semantics. Since in the least fixed-point semantics some cyclic virtual classes are incoherent and hence subsumed by every class, we know that $\sqsubseteq^l_\sigma$ does not imply $\sqsubseteq^g_\sigma$. The converse, however, holds. (4) (5) (6) refer to incoherence and are immediate; notice that a type is incoherent w.r.t. descriptive semantics if and only if it is incoherent w.r.t. gfp–semantics.

Let us generalize the framework, by assuming an heterogeneous semantics to be adopted for a schema μ. Incoherence and subsumption definitions for a schema μ adopting an heterogeneus semantics are straightforward extensions of the above definitions. Given a type S of a schema μ, we have:

$$\begin{array}{rl} S \text{ is } \textit{het–incoherent} & \text{iff} \;\; \mathcal{I}[S] = \emptyset \text{ for all heterogeneus legal instance } \mathcal{I} \text{ of } \mu \\ S \sqsubseteq^h_\mu S' & \text{iff} \;\; \mathcal{I}[S] \subseteq \mathcal{I}[S'] \text{ for all heterogeneus legal instance } \mathcal{I} \text{ of } \mu. \end{array}$$

Note that, by adopting heterogeneous semantics we restrict the set of legal instances of a schema under descriptive semantics. Consequently, we have a set of computed subsumption relations and incoherent types which is greater than the one that would have been obtained by adopting a descriptive semantics for the whole schema.

The problem is now to detect incoherence and compute subsumption assuming an heterogeneous semantics. The solution is given in Table 5 where it can be seen that incoherence and subsumption relationships under an heterogeneous semantics can be turned into incoherence and subsumption relationships under the three given semantics.

The relationships between virtual classes interpreted under the same semantics are immediate, thus we comment the ones between classes interpreted under different semantics:

- $V^l_2 \sqsubseteq^h_\mu V^d_1 \Leftrightarrow V^l_2 \sqsubseteq^l_\sigma V^d_1$ and $V^l_2 \sqsubseteq^h_\mu V^g_1 \Leftrightarrow V^l_2 \sqsubseteq^l_\sigma V^g_1$

  $\Rightarrow$: By adopting the heterogeneous semantics for the schema μ we also consider the lfp–instances of the schema μ.

  $\Leftarrow$: If $V^l_2 \sqsubseteq^l_\sigma V^d_1$, then for all the heterogeneous legal instance $\mathcal{I}$, $\mathcal{I}[V^l_2]$ will always be a subset of $\mathcal{I}[V^d_1]$, as $\mathcal{I}[V^l_2] = \mathcal{I}^l[V^l_2]$; hence $V^l_2 \sqsubseteq^l_\sigma V^d_1 \Rightarrow V^l_2 \sqsubseteq^h_\mu V^d_1$. The same holds for the other case.

- $V^g_2 \sqsubseteq^h_\mu V^d_1 \Leftrightarrow V^g_2 \sqsubseteq^d_\sigma V^d_1$, $V^g_2 \sqsubseteq^h_\mu V^l_1 \Leftrightarrow V^g_2 \sqsubseteq^d_\sigma V^l_1$ and
  $V^d_2 \sqsubseteq^h_\mu V^l_1 \Leftrightarrow V^d_2 \sqsubseteq^d_\sigma V^l_1$

  $\Rightarrow$: If $V^g_2 \sqsubseteq^h_\mu V^d_1$, then for all the legal instance $\mathcal{I}$, $\mathcal{I}[V^g_2]$ will always be a subset of $\mathcal{I}[V^d_1]$, as $\mathcal{I}[V^g_2] \subseteq \mathcal{I}^g[V^g_2]$; hence $V^g_2 \sqsubseteq^h_\mu V^d_1 \Rightarrow V^g_2 \sqsubseteq^d_\sigma V^d_1$. The same holds for the other two cases.

  $\Leftarrow$: By adopting the descriptive semantics we include all the heterogeneous legal instances.

- $V^d_2 \sqsubseteq^h_\mu V^g_1 \Leftrightarrow V^d_2 \sqsubseteq^g_\sigma V^g_1$

  $\Rightarrow$: By adopting the heterogeneous semantics for the schema μ we also consider the gfp–instances of the schema μ.

$V^d \simeq^h_\mu \perp$ iff $V^d \simeq^d_\mu \perp$	$V^l \simeq^h_\mu \perp$ iff $V^l \simeq^l_\mu \perp$	$V^g \simeq^h_\mu \perp$ iff $V^g \simeq^g_\mu \perp$

$V_2 \sqsubseteq^h_\mu V_1$	V_1^d	V_1^l	V_1^g
V_2^d	$V_2^d \sqsubseteq^d_\mu V_1^d$	$V_2^d \sqsubseteq^d_\mu V_1^l$	$V_2^d \sqsubseteq^g_\mu V_1^g$
V_2^l	$V_2^l \sqsubseteq^l_\mu V_1^d$	$V_2^l \sqsubseteq^l_\mu V_1^l$	$V_2^l \sqsubseteq^l_\mu V_1^g$
V_2^g	$V_2^g \sqsubseteq^d_\mu V_1^d$	$V_2^g \sqsubseteq^d_\mu V_1^l$	$V_2^g \sqsubseteq^g_\mu V_1^g$

Table 5: Incoherence and subsumption under an heterogeneous semantics

$\Leftarrow$: If $V_2^d \sqsubseteq^g_\sigma V_1^g$, then for all the heterogeneous legal instance $\mathcal{I}$, $\mathcal{I}[V_2^d]$ will always be a subset of $\mathcal{I}[V_1^g]$, as $\mathcal{I}[V_1^g] = \mathcal{I}^g[V_2^g]$.

2.5 Comparing different semantics for cyclic views

In the following we will discuss, by means of some examples, the adoption of one or another semantics to interpret cyclic views and the related effects on incoherence detection and subsumption computation. Most of the considerations apply as well to recursive queries as are referred in general to the semantics of a cyclic virtual class.

By adopting lfp–semantics for a database schema, we choose *the least possible instance* as the legal instance. This instance is computed for virtual classes starting from an empty set of objects, thus, *cyclic objects*, i.e., objects which directly or indirectly make references to themselves (see, for example, objects o_5 and o_4 in table 3) cannot be inserted in the interpretation of a cyclic virtual class (see $\mathcal{I}_3$ in table 4).

For this reason, a cyclic virtual class is coherent, only if *all* its circular references use a set or sequence type constructors: in this case, in fact, it can be populated with *acyclic objects* which make references to empty set or empty sequence. For example, the class `Office`, (and consequently the class `Clerk`), is coherent, as it can be populated starting from object o_8 which has an empty set of employees (see tables 3 and 4).

Hence, bearing in mind that in an instance of a schema the set of objects is finite, with lfp–semantics we capture the recursive definition of acyclic finite object structures [7].

[7]A similar result is discussed in [12]: "a least fixed point solution of recursive type definition does not allow

By adopting gfp–semantics for a database schema, we choose *the largest possible instance* as the legal instance: the extensions of the cyclic virtual classes contain all the objects, both cyclic and acyclic, agreeing with class descriptions [8] (see $\mathcal{I}_1$ in table 4). Thus a gfp–legal instance *assigns to each virtual class the set of all possible objects which can belong to the extension of the class*. As a consequence the relation $\sqsubseteq^g_\sigma$ defines all the possible subsumption relations which can occur between virtual classes. With reference to our Company schema, by adopting gfp–semantics no class has been eliminated as incoherent and two new subsumption relations have been computed between `Secretary` and `Clerk` (see figure 3) and between `T-Office` and `Office`.

By adopting descriptive semantics, we select *one of the possible instances* as the legal instance. As a result, the adoption of the descriptive semantics does not allow, in most cases, the computation of implicit subsumption relations between cyclic virtual classes, say V, V', having a similar description, as it is possible to find one legal instance $\mathcal{I}$ of the schema which does not satisfy $\mathcal{I}[V] \subseteq \mathcal{I}[V']$. See for example the possible instances shown in table 4: if we select $\mathcal{I}_2$ as the descriptive legal instance, we have $\mathcal{I}_2[\texttt{Secretary}] \not\subseteq \mathcal{I}_2[\texttt{Clerk}]$ and $\mathcal{I}_2[\texttt{T-Office}] \not\subseteq \mathcal{I}_2[\texttt{Office}]$.

This consideration suggests that the adoption of descriptive semantics for cyclic views is advantageous in the case we want to maintain in the schema (some) cyclic views with equivalent descriptions but with independent interpretations. As an example, let us add to the Company domain the views `Technician`, having a description equal to `Clerk`, and `Laboratory` having a description equal to `Office`:

$$
\begin{aligned}
\sigma_V(\texttt{Technician}) &= \texttt{Mdl-Employee} \sqcap \triangle[\texttt{worksin}\colon \texttt{Laboratory}] \\
\sigma_V(\texttt{Laboratory}) &= \texttt{Department} \sqcap \triangle[\texttt{employs}\colon \{\texttt{Technician}\}]
\end{aligned}
$$

Assuming a descriptive semantics for the above views means that we can insert somebody as a `Technician` if and only if he/she is an employee working in a `Laboratory`, without being committed to conclude that he/she is a `Clerk` as well. In some sense, descriptive semantics seems to assign more relevance to view names of cyclic views with respect to lfp and gfp–semantics.

To show the difference from an extensional point of view between gfp and descriptive semantics, let us add the following objects to the domain δ of table 3:

$$
\begin{aligned}
\delta(o_{10}) &= [\texttt{dname} : \textit{“Store”}, \texttt{category} : 8, \texttt{employs} : \{o_{11}\}, \texttt{typing}\colon o_{11}] \\
\delta(o_{11}) &= [\texttt{name} : \textit{“Boris”}, \texttt{worksin} : o_{10}, \texttt{level} : 9]
\end{aligned}
$$

If we insert the object o_{10} into `Department` and if we explicitly insert the cyclic object o_{11} into the `Secretary` view, then, under both the semantics, o_{10} is automatically inserted into the `T-Office` view, but, while adopting gfp–semantics o_{10} and o_{11} are also automatically inserted into the `Office` and `Clerk` cyclic views respectively, these insertions are not necessarily performed by adopting descriptive semantics. For example, the following interpretation:

$$
\begin{aligned}
\mathcal{I}_4[\texttt{Clerk}] &= \{o_9\} & \mathcal{I}_4[\texttt{Office}] &= \{o_8\} \\
\mathcal{I}_4[\texttt{Secretary}] &= \{o_{11}\} & \mathcal{I}_4[\texttt{T-Office}] &= \{o_{10}\}
\end{aligned}
$$

cycles in the data”.

[8]A gfp–semantics is adopted in [28], where a relation called *refinement* is defined which is identical to our subsumption relation if we assume only virtual classes.

is a possible instance, and hence a descriptive legal instance, where $o_{11} \notin \mathcal{I}_4[\texttt{Clerk}]$ and $o_{10} \notin \mathcal{I}_4[\texttt{Office}]$. In other words, the interpretations of cyclic virtual classes under descriptive semantics are in some way arbitrary as the one of base classes.

Finally, let us consider lfp–semantics behaviour from an extensional point of view. First of all, we note that if we try to insert the same objects o_{10} and o_{11} in the `Department` class and in the `Secretary` view, respectively, the least possible instance is not modified with respect to $\mathcal{I}_3$ of table 4. In fact, the insertion of o_{11} in the `Secretary` view is rejected, due to lfp–semantics, as we try to insert a cyclic object. On the other hand, if we consider the above interpretation $\mathcal{I}_4$, which is a legal descriptive instance, we note that this instance is *the least possible instance* $\mathcal{I}$ *such that* $o_{11} \in \mathcal{I}[\texttt{Secretary}]$. We can characterize this particular class of possible instances by introducing a variation in lfp–instance computation.

In fact, up to now we assumed to compute the least possible instance starting from the empty set, i.e., starting from $\mathcal{I}_\perp[V] = \emptyset$, $\forall V$. Let us suppose to compute, instead, the least possible instance starting from a given $\mathcal{I}_{start}$ for a target virtual class. With reference to the above example, $\mathcal{I}_{\text{start}}[\texttt{Secretary}] = \{o_{11}\}$ and $\mathcal{I}_{\text{start}}[V] = \emptyset$, for every virtual class of the schema, different from `Secretary`. This least possible instance exists if the objects in $\mathcal{I}_{start}[C]$ satisfy the description of the class C.

Formally, we can prove that if $\mathcal{I}_{start} \trianglelefteq \Gamma(\mathcal{I}_{\text{start}})$ then $\bigcup_{i\geq 0} \Gamma^i(\mathcal{I}_{\text{start}})$ is a fixed–point of Γ (and hence a possible instance of the schema); more precisely, this is the least fixed–point of Γ which is greater or equal to $\mathcal{I}_{start}$.

Finally, let us comment now the very subtle differences between base classes and cyclic virtual classes with a descriptive interpretation. As an example, we take a modified description of the `Secretary` view including an explicit inheritance relation between secretaries and clerks:

$$\sigma_V(\texttt{Secretary}) \quad = \quad \texttt{Clerk} \sqcap \triangle\Big[\texttt{worksin:T-Office}\Big]$$

we obtain $\texttt{T-Office} \sqsubseteq^d_\sigma \texttt{Office}$, if `Office` is a virtual class, but this subsumption relation does not hold if the class `Office` is a base class. Furthermore, from an extensional point of view, if we insert o_{11} into the `Secretary` view (and then o_{10} is inserted into the `T-Office` view) for the explicit inheritance relation, o_{11} is inserted as well into the `Clerk` view. Consequently, if `Office` is a view, the object o_{10} must be inserted into the view `Office` and thus $\texttt{T-Office} \sqsubseteq^d_\sigma \texttt{Office}$. On the other hand, if `Office` is a base class, o_{10} is not automatically inserted in the `Office` class, consequently $o_{10} \notin \mathcal{I}[\texttt{Office}]$ but $o_{10} \in \mathcal{I}[\texttt{T-Office}]$, thus `Office` does not subsume `T-Office`.

2.6 Comparing different semantics for recursive queries

Focusing our attention on recursive queries we have to choose a semantics that uniquely defines their answer set. The choice of descriptive semantics is thus unfeasible as we could have many different answers to the same query; the surviving candidates are gfp–semantics and lfp–semantics.

Let us firstly recall the form of recursion introduced in [16] for OODB query languages, which corresponds to the notion of linear recursion for logic queries. It is expressed by means of a recursion operator, named ρ, and makes it possible to express most recursive queries encountered in real applications. We will show that the same recursive queries can be expressed by ODL equivalent formulation of the ρ operator.

The ρ operator has the following behaviour: given an object o, instance of a class C, and a property a of o, $o.\rho(a)$ defines a set of objects that are either values of property a of o, or values of property a of some object o' which is a value of property a of object o, etc. Objects in this set are implicitly bound to the class C which is the domain of property a. In other words, $\rho(a)$ is the *transitive closure* of the attribute a and includes both cyclic and acyclic objects. The set of objects evaluated by the $\rho(a)$ operator thus, intuitively, corresponds to a gfp–interpretation.

Let us give an ODL equivalent formulation of the $\rho(a)$ operator. Given a cyclic base class C which directly makes reference to itself by an attribute a: $\sigma(C) = \triangle[a\colon C, \ldots]$, a generic recursive query Q "select from the class C the objects such that all the recursive objects of its attribute a satisfy the condition represented by a type S" can be expressed in ODL as:

$$\sigma(Q) = C \sqcap \triangle\Big[a\colon Q \sqcap S\Big]$$

Query Q states that the value domain of the attribute a, with respect to the *transitive closure*, must be of type S. More precisely, if Q is interpreted under gfp–semantics, we have:

$$\mathcal{I}^g[Q] = \Big\{o \in C \mid \; \forall o' \in o.\rho(a)\colon o' \text{ satisfies } S\Big\}.$$

Let us show that the choice of interpreting a query under lfp or gfp–semantics depends on the meaning of the query itself. As an example let us introduce a class of cities connected by flights of different airlines and its extension.

$$\sigma_B(\texttt{City}) \;\; = \;\; \triangle\Big[\texttt{connected-to}\colon \{\texttt{City}\}, \texttt{airline}\colon \texttt{String}\Big]$$

$$\begin{aligned}
\delta(o_1) &= [\texttt{connected-to}\colon \{o_2, o_3\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_2) &= [\texttt{connected-to}\colon \{\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_3) &= [\texttt{connected-to}\colon \{o_2\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_4) &= [\texttt{connected-to}\colon \{o_2, o_7\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_5) &= [\texttt{connected-to}\colon \{o_3, o_6\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_6) &= [\texttt{connected-to}\colon \{o_5\}, \texttt{airline}\colon \text{“}AA\text{”}] \\
\delta(o_7) &= [\texttt{connected-to}\colon \{\}, \texttt{airline}\colon \text{“}AC\text{”}]
\end{aligned}$$

From city o_1 for example, there are two connections by aerline "AA" (o_2 and o_3); no "AA" flight leaving from o_2 exists.

The ρ operator applied to the property `connected-to` starting from the object o_5 gives: $o_5.\rho(\texttt{connected-to}) = \{o_3, o_6, o_5, o_2\}$.

Let Q_4 be a recursive query: "Select the cities served only by the "AA" airline which are connected by tours".

$$\sigma_V(Q_4) \;\; = \;\; \texttt{City} \sqcap \triangle\Big[\texttt{connected-to}\colon \{Q_4\}, \texttt{airline}\colon \text{“}AA\text{”}\Big]$$

If we want to include in the query answer set the cities which are traversed more than once (as o_5 and o_6), then the query Q_4 must be interpreted under gfp–semantic, having:

$$\begin{aligned}\mathcal{I}^g[Q_4] &= \Big\{o \in \texttt{City} \mid o.\texttt{airline} = \text{“}AA\text{''} \wedge \forall o' \in o.\rho(\texttt{connected-to})\colon o'.\texttt{airline} = \text{“}AA\text{''}\Big\} \\ &= \Big\{o_1, o_2, o_3, o_5, o_6\Big\}\end{aligned}$$

On the other hand, if we want to know all those cities from which tours completely covered by “AA” airline start, but excluding the cities which are traversed more than once, query Q_4 must be aswered under lfp–semantics, having:

$$\begin{aligned}\mathcal{I}^l[Q_4] &= \Big\{o \in \texttt{City} \,|\, o.\texttt{airline} = \text{“}AA\text{''} \wedge \forall o' \in o.\rho(\texttt{connected-to})\colon o' \neq o \wedge o'.\texttt{airline} = \text{“}AA\text{''}\Big\} \\ &= \Big\{o_1, o_2, o_3\Big\}\end{aligned}$$

In conclusion, by means of the above generic ODL query formulation and by a gfp–interpretation we can directly express the transitive closure of an attribute. On the other hand, by means of the above generic ODL query formulation and by its lfp–interpretation we can directly express the transitive closure of an attribute (or of a path), but limiting to acyclic objects.

Up to now we discussed the implications of selecting one or another fixed-point semantics for recursive queries. Let us consider now the most general framework: we have an heterogeneous semantics for the schema (classes + views) and we have queries, which can refer to base classes, other queries and to views, interpreted by a fixed-point semantics.

The resulting framework (classes + views + queries) will be interpreted again under an heterogeneous semantics and for a generic query Q we have that:

$Q \in \mathbf{C}^l_{\mathbf{V}}$ (denoted by Q^l): if the query is interpreted under the lfp–semantics and its description refers only to views and queries interpreted under lfp–semantics.

$Q \in \mathbf{C}^g_{\mathbf{V}}$ (denoted by Q^g): if the query is interpreted under the gfp–semantics and its description refers only to views and queries interpreted under gfp–semantics.

$Q \in \mathbf{C}^d_{\mathbf{V}}$ (denoted by Q^d): if the query is interpreted under the gfp–semantics (lfp–semantics) and its description refers to descriptive views and/or queries interpreted under a fixed-point semantics. Note that, on the basis of this definition, the answer set of a query Q^d is unique.

In this way, given a query Q, interpreted under a fixed-point semantics, we firstly classify it as Q^l or Q^g or Q^d on the basis of its description and, then, perform query validation and query classification under an heterogeneous semantics on the basis of the incoherence and subsumption relationships given in Table 5.

As an example, let the following recursive query Q_5 be interpreted under gfp–semantics:

$$\begin{aligned}\sigma_Q(Q_5) \;=\; \texttt{Department} \sqcap \triangle\Big[&\texttt{employs}\colon\{\texttt{Clerk} \sqcap \texttt{Secretary}\}, \\ &\texttt{typing}\colon \triangle[\texttt{worksin}\colon Q_5]\Big]\end{aligned}$$

Let us assume that `Secretary` and `T-Office` be interpreted under gfp–semantics: $\texttt{Secretary}^g$, $\texttt{T-Office}^g$. As Q_5 uses the view `Clerk`, we can have the following cases:

- If `Clerk` is interpreted under des–semantics ($\texttt{Clerk}^d$), then we obtain Q^d_5. Thus, if for example $\mathcal{I}[\texttt{Clerk}^d] = \{o_5, o_9\}$ (see Table 4), we obtain its answer set as $\mathcal{I}[Q^d_5] = \{o_4, o_8\}$.
- If `Clerk` is interpreted under lfp–semantics ($\texttt{Clerk}^l$), then we obtain Q^d_5. As an example of query answer set, if $\mathcal{I}[\texttt{Clerk}^l] = \{o_9\}$ (see Table 4), we obtain $\mathcal{I}[Q^d_5] = \{o_8\}$.

- If `Clerk` is interpreted under gfp–semantics ($\texttt{Clerk}^g$), then we must consider Q_5^g; as an example of query answer set, if $\mathcal{I}[\texttt{Clerk}^g] = \{o_5, o_7, o_9\}$ (see Table 4), we obtain $\mathcal{I}[Q_5^g] = \{o_4, o_6, o_8\}$.

To show Q_5 classification, let us consider: $\texttt{Clerk}^l$, $\texttt{Office}^l$, $\texttt{Secretary}^g$ and $\texttt{T-Office}^g$. Note that no subsumption relations there exists between $\texttt{Clerk}^l$ and $\texttt{Secretary}^g$ ($\texttt{Office}^l$ and $\texttt{T-Office}^g$). In this case we must consider Q_5^d and thus we can determine that `Office` and `T-Office` are its subsumees as we have $Q_5^d \sqsubseteq_\mu^d \texttt{Office}^l$ and $Q_5^d \sqsubseteq_\mu^g \texttt{T-Office}^g$. On the other hand, if we consider `Secretary` and `T-Office` interpreted under descriptive semantic, classification of query Q_5 changes as, even if Q_5 refers to the view `Secretary`, we have $Q_5^d \not\sqsubseteq_\mu^d \texttt{T-Office}^d$.

3 Algorithms

In [14] algorithms for detecting incoherence and computing subsumption between types of an ODL schema, adopting the gfp–semantics were given. In this section we extend the framework there proposed by providing algorithms to detect incoherence and subsumption also for the lfp and the descriptive semantics. Note that these problems are computationally intractable from a purely theoretical point of view (see Appendix), but if we provide algorithms on a *simplified schema description* (named *canonical schema*) we can solve both in time polynomial in the size of the canonical schema. Therefore, in analogy with [14], instead of directly specifying algorithms for incoherence and subsumption, first we transform a schema into a canonical one and then define algorithms on the canonical schema. The algorithm for the transformation of a schema σ into a canonical schema ν is given in [9]; the main points of the transformation are:

1. Every base class P of the schema σ is transformed into a virtual class V of the schema ν, expressing it as the conjunction of an *imaginary atomic* class $\hat{P}$, which is a base class without any description, i.e. $\nu(\hat{P}) = \triangle\top$, and its original description ($\sigma(P)$), thus obtaining $\nu(V) = \hat{P} \sqcap \sigma(P)$. Note that the interpretation of $\hat{P}$ is an arbitrary set of objects of the domain. For example, the base class `Employee` is transformed as follows:

$$\begin{aligned}\nu(\texttt{Employee}) &= \widehat{\texttt{Employee}} \sqcap \sigma(\texttt{Employee})\\ &= \widehat{\texttt{Employee}} \sqcap \triangle[\texttt{name}\colon \texttt{String}, \texttt{level}\colon 1\text{–}10,\\ &\qquad \texttt{worksin}\colon \texttt{Department}]\end{aligned}$$

2. Every class description is transformed by substituting only the names of the classes from which it inherits with their descriptions, expanding the value–type names and applying suitable conjunction rules. For example, as $\sigma(\texttt{Mdl-Employee}) = \texttt{Employee} \sqcap \triangle[\texttt{level}\colon 5\text{–}10]$, we have:

$$\begin{aligned}\nu(\texttt{Mdl-Employee}) &= \widehat{\texttt{Employee}} \sqcap \triangle[\texttt{name}\colon \texttt{String}, \texttt{level}\colon 5\text{–}10,\\ &\qquad \texttt{worksin}\colon \texttt{Department}]\end{aligned}$$

   It is worthwhile to observe that in $\nu(\texttt{Mdl-Employee})$ we lose the inheritance relation with respect to `Employee`.

3. A new name is assigned to each new type obtained by the transformations. Thus, the canonical transformation of `Employee` is:

$$\begin{aligned} \nu(\texttt{Employee}) &= \widehat{\texttt{Employee}} \sqcap \triangle T_0 \\ \nu(T_0) &= [\texttt{name}\colon \texttt{String}, \texttt{worksin}\colon \texttt{Department}, \texttt{level}\colon 1\text{–}10] \end{aligned}$$

The key point of the transformation is step 2, as it allows to *solve*, exploiting conjunction semantics, conflicts resolution in the case of inheritance: if a local attribute is also inherited from one (or more) classes, the resulting domain type of the attribute is obtained as the conjunction of the given types (see the `level` attribute of `Mdl-Employee`).

It was proved in [33] that, in a schema without virtual cyclic classes, the above transformation *preserves the extensions of classes*. Thus, the incoherences detected and the subsumption relations computed with respect to a canonical schema *are the same as the ones* of the original schema. In particular, this implies that the inheritance relations given in σ are preserved, as they can be computed as subsumption relations in the canonical schema ν. In fact, given $C \prec_\sigma C'$ in σ, we obtain a subsumption relation in ν between the canonical descriptions of C and C', say $\nu(C) \sqsubseteq_\nu \nu(C')$. Note that, as C and C' are virtual classes in ν, from $\nu(C) \sqsubseteq_\nu \nu(C')$ it follows that $C \sqsubseteq_\nu C'$. For example, the inheritance relation `Mdl-Employee` $\prec_\sigma$ `Employee` is computed in the canonical schema (`Mdl-Employee` $\sqsubseteq_\nu$ `Employee`).

Things are much more complex if we deal with a schema including virtual cyclic classes. The question is: is it always possible to perform a transformation of such a schema into a canonical form preserving its classes extensions? The answer is in the affirmative for gfp–semantics, as was proved in [14]. In the following, we will prove that it is always in the affirmative for lfp–semantics. In particular, we will show that the obtained canonical schema is equal to the one obtained for gfp–semantics, but it is interpreted under lfp–semantics. Unfortunately, the answer is in the negative for descriptive semantics: there exist cyclic schemata for which a canonical transformation preserving the extensions of the classes of the given schema is not possible.

Let us give now the formal definition of a canonical schema as proposed in [14]. Let $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ the set of types $\mathbf{N} = \mathbf{C_B} \cup \mathbf{C_V} \cup \mathbf{T}$, where $\mathbf{C_B}$ is a set of imaginary atomic classes, $\mathbf{C_V}$ a set of virtual class names, $\mathbf{T}$ a set of value-type names. A *canonical schema* ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ has the following form (for all $\hat{P} \in \mathbf{C_B}, V \in \mathbf{C_V}, T \in \mathbf{T}$):

$$\begin{aligned} \nu(\hat{P}) &= \triangle\top \\ \nu(V) &= \textstyle\bigsqcap \hat{P}_i \sqcap \triangle N \\ \nu(T) &= \top \text{ or } B \text{ or } \{N\} \text{ or } \langle N \rangle \text{ or } [a_1\colon N_1, \ldots, a_p\colon N_p] \end{aligned}$$

where $N, N_j \in \mathbf{N} \setminus \mathbf{C_B}, 1 \leq j \leq p, \hat{P}_i \in \mathbf{C_B}, i \geq 0$.

An interpretation $\mathcal{I}$ is defined as in section 2.3, but with the extension of the interpretation to deal with value-type name: $\mathcal{I}[T] \subseteq \mathcal{V}(\mathcal{O}) \setminus \mathcal{O}$. We say that an interpretation $\mathcal{I}$ is a possible instance of the schema ν iff, for all $\hat{P} \in \mathbf{C_B}, V \in \mathbf{C_V}, T \in \mathbf{T}$, we have:

$$\mathcal{I}[\hat{P}] \subseteq \mathcal{I}[\nu(\hat{P})], \quad \mathcal{I}[V] = \mathcal{I}[\nu(V)], \quad \mathcal{I}[T] = \mathcal{I}[\nu(T)].$$

To interpret a canonical schema ν we can adopt any of the different semantics. Let us start by considering gfp–semantics (all the results on gfp–semantics are taken from [14, 9]).

We say that a schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ is a *gfp–conservative extension* of a schema σ over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ iff $\mathbf{C} \subseteq \mathbf{N}$ and for any gfp–legal instance $\mathcal{I}$ of σ a gfp–legal instance $\mathcal{I}'$ of ν exists and *vice versa* such that $\mathcal{I}[C] = \mathcal{I}'[C]$ for all $C \in \mathbf{C}$. Note that this implies that:

- for all $C \in \mathbf{C}$: C is gfp–incoherent in σ iff it is gfp–incoherent in ν;
- for all $C, C' \in \mathbf{C}$: $C \sqsubseteq^g_\sigma C'$ iff $C \sqsubseteq^g_\nu C'$.

A schema ν is called a *gfp–canonical extension* of σ iff ν is canonical and ν is a gfp–conservative extension of σ.

Any schema σ over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ can be effectively transformed into a schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ which is a gfp–canonical extension of σ.

Using gfp–canonical schemata, detecting incoherence and computing subsumption is easy. As a matter of fact, both problems can be solved in time polynomial in the size of the canonical schema.

Algorithm 1 (gfp–Incoherence) *Let $I^0 = \{\bot\}$ and define I^{i+1} in the following way:*

$$I^{i+1} = I^i \cup \Big\{N \in \mathbf{N} | \nu(N) \in \Big\{\bot, [\ldots, a' : N', \ldots], \sqcap P_i \sqcap \triangle N' \mid N' \in I^i\Big\}\Big\}.$$

Compute $I^0, \ldots, I^i, \ldots$ until $I^i = I^{i+1}$ and set $\widetilde{I^g_\nu} = I^i$.

Since I^i grows monotonically with increasing i and $\mathbf{N}$ is finite, there exists a natural number n such that $I^n = I^{n+1}$, i.e, the algorithm terminates. Further, note that the maximal n is $|\mathbf{N}|$, i.e., the algorithm is polynomial in ν.

In a canonical schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$, for all $N \in \mathbf{N}$, N is gfp–incoherent iff $N \in \widetilde{I^g_\nu}$.

For example, with reference to the schema of Table 2, the query Q_1 of the introduction is detected as incoherent because after its canonical transformation attribute `level` has values in the value–type $4 \sqcap 10$–∞, which is equal to $\bot$, i.e., we have:

$$\begin{array}{rcl}
\nu(Q_1) & = & \widehat{\texttt{Department}} \sqcap \triangle T_1 \\
\nu(T_1) & = & [\texttt{dname}: \texttt{String}, \texttt{category}: 5\text{–}7, \texttt{employs}: T_2, \texttt{typing}: V_1] \\
\nu(T_2) & = & \{\texttt{Secretary}\} \\
\nu(V_1) & = & \widehat{\texttt{Employee}} \sqcap \triangle T_3 \\
\nu(T_3) & = & [\texttt{name}: \texttt{String}, \texttt{worksin}: \texttt{T-Office}, \texttt{level}: \bot]
\end{array}$$

Therefore: $I^0 = \{\bot\}$, $I^1 = I^0 \cup \{T_3\}$, $I^2 = I^1 \cup \{V_1\}$, $I^3 = I^2 \cup \{T_1\}$, $\widetilde{I^g_\nu} = I^4 = I^3 \cup \{Q_1\}$. Note that since sets or sequences may be empty, algorithm 1 does not consider these type constructors.

For gfp–subsumption computation, let us introduce the relation $\leq \subseteq \mathbf{N} \times \mathbf{N}$ and let us define $S \sqsubseteq_\leq S'$ as follows:

$$\begin{array}{rcl}
B \sqsubseteq_\leq B' & \text{iff} & B \sqcap B' = B \\
\{N\} \sqsubseteq_\leq \{N'\} & \text{iff} & N \leq N' \\
\langle N \rangle \sqsubseteq_\leq \langle N' \rangle & \text{iff} & N \leq N' \\
[\ldots, a_i: N_i, \ldots] \sqsubseteq_\leq [\ldots, a'_j: N'_j, \ldots] & \text{iff} & \forall j \exists i\colon (a_i = a'_j \wedge N_i \leq N'_j) \\
\sqcap_i \hat{P}_i \sqcap \triangle N \sqsubseteq_\leq \sqcap_j \hat{P}'_j \sqcap \triangle N' & \text{iff} & N \leq N' \wedge \forall j \exists i\colon \hat{P}_i = \hat{P}'_j.
\end{array}$$

ALGORITHM 2 (GFP–SUBSUMPTION) *Let* $\leq^0 = \Big\{(N,N') \mid N,N' \in \mathbf{C_V} \vee N,N' \in \mathbf{T} \vee N = N' \in \mathbf{C_B}\Big\}$ *and define* $\leq^{i+1}$ *in the following way:*

$$\leq^{i+1} \; = \; \leq^i \; \cap \Big\{(N,N') \in \mathbf{N} \times \mathbf{N} \Big| \, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N \in \widetilde{I^g_\nu} \vee \nu(N') = \top \Big\}.$$

Compute $\leq^0, \ldots, \leq^i, \ldots$ *until* $\leq^i = \leq^{i+1}$ *and set* $\lesssim^g_\nu \; = \leq^i$.

As above, a natural number n always exists such that $\leq^n = \leq^{n+1}$. This time, the maximal n is $|\mathbf{N}|^2$, i.e., also the subsumption algorithm is polynomial in ν.

We can assert that, in a canonical schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$, for all $N,N' \in \mathbf{N}$: $N \sqsubseteq^g_\nu N'$ iff $N \lesssim^g_\nu N'$.

Finally, using the property that a canonical extension of a schema is a conservative extension, we can say that: let σ be a schema over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$, and let ν be a schema over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ which is a gfp–canonical extension of σ,

- for all $C \in \mathbf{C}$: C is gfp–incoherent in σ iff $C \in \widetilde{I^g_\nu}$;
- for all $C, C' \in \mathbf{C}$: $C \sqsubseteq^g_\sigma C'$ iff $C \lesssim^g_\nu C'$.

In order to briefly explain the subsumption algorithm, let us observe first that subsumption computation for a schema without cyclic classes can be performed with exactly three iterations of the algorithm, whereas cyclic references increase the number of iterations. Algorithm starts assuming at step $\leq^0$ that each class name subsumes all the other class names and each value-type name subsumes all the other value-type names. At step $\leq^1$, we can drop many subsumption relations: those between non-homogeneous value-types. At step $\leq^2$ we can drop other subsumption relations, for instance, it is detected that `Department` and `Employee` are not subsumed by any other class. Thus, by iterating we can only decide negative relations as we eliminate subsumption relations.

To show a computation with more than three iterations, let us introduce a new virtual class `Typist` and consider subsumption computation between `Typist` and `Clerk`:

$$\sigma_V(\texttt{Typist}) \;=\; \texttt{Mdl-Employee} \sqcap \triangle\Big[\texttt{worksin}\colon \texttt{Office} \sqcap \triangle[\texttt{typing}\colon \texttt{Typist}]\Big]$$

Obtained the canonical form of `Typist` and `Clerk` as follows:

$$\begin{array}{rcl}
\nu(\texttt{Clerk}) & = & \widehat{\texttt{Employee}} \sqcap \triangle T_4 \\
\nu(T_4) & = & [\texttt{name}\colon \texttt{String}, \\
 & & \;\texttt{level}\colon 5\text{–}10 \\
 & & \;\texttt{worksin}\colon \texttt{Office}] \\
\nu(\texttt{Office}) & = & \widehat{\texttt{Department}} \sqcap \triangle T_5 \\
\nu(T_5) & = & [\texttt{dname}\colon \texttt{String}, \\
 & & \;\texttt{category}\colon 5\text{–}7, \\
 & & \;\texttt{employs}\colon T_6, \\
 & & \;\texttt{typing}\colon \texttt{Employee}] \\
\nu(T_6) & = & \{\texttt{Clerk}\}
\end{array}
\qquad
\begin{array}{rcl}
\nu(\texttt{Typist}) & = & \widehat{\texttt{Employee}} \sqcap \triangle T_7 \\
\nu(T_7) & = & [\texttt{name}\colon \texttt{String}, \\
 & & \;\texttt{level}\colon 5\text{–}10 \\
 & & \;\texttt{worksin}\colon V_2] \\
\nu(V_2) & = & \widehat{\texttt{Department}} \sqcap \triangle T_8 \\
\nu(T_8) & = & [\texttt{dname}\colon \texttt{String}, \\
 & & \;\texttt{category}\colon 5\text{–}7, \\
 & & \;\texttt{employs}\colon T_6, \\
 & & \;\texttt{typing}\colon \texttt{Typist}]
\end{array}$$

To decide subsumption between `Clerk` and `Typist` we need seven iterations as, at each step since step 6, we delete subsumption relations:

$$\begin{array}{rcl} T_0 & \not\leq^1 & T_4, T_7 \\ \texttt{Employee} & \not\leq^2 & \texttt{Clerk}, \texttt{Typist} \\ T_5 & \not\leq^3 & T_8 \\ \texttt{Office} & \not\leq^4 & V_2 \\ T_4 & \not\leq^5 & T_7 \\ \texttt{Clerk} & \not\leq^6 & \texttt{Typist} \end{array}$$

and we can stop at step 7, since we cannot delete any further subsumption relation. Thus we obtain: `Typist` $\lesssim^g_\nu$ `Clerk` and `Clerk` $\not\lesssim^g_\nu$ `Typist`.

Now we can provide algorithms for lfp–and descriptive semantics by extending the above framework.

For lfp–semantics we can adopt the same canonical and subsumption algorithm developed for gfp–semantics, but we need a new algorithm to detect incoherence under lfp–semantics.

We say that a schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$ is a *lfp–conservative extension* of a schema σ over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{C})$ iff $\mathbf{C} \subseteq \mathbf{N}$ and for any lfp–legal instance $\mathcal{I}$ of σ an lfp–legal instance $\mathcal{I}'$ of ν exists and *vice versa* such that $\mathcal{I}[C] = \mathcal{I}'[C]$ for all $C \in \mathbf{C}$. It can be proved that a schema ν which is a gfp–canonical extension of σ is also an lfp–canonical extension of the schema σ.

A gfp–incoherent type is also an lfp–incoherent type; moreover, for lfp–semantics there is an additional source of incoherence due to circular references to type constructors different from sequence and set. This further control can be performed by the following algorithm:

Algorithm 3 (lfp–Incoherence) *Let* $I^0 = \mathbf{N} \setminus \mathbf{C}_\mathbf{B}$ *and define*

$$I^{i+1} = I^i \cap \Big\{ N \in \mathbf{N} | \nu(N) \in \Big\{ [\ldots, a' \colon N', \ldots], \sqcap \hat{P}_i \sqcap \triangle N' \mid N' \in I^i \Big\} \Big\}.$$

Compute $I^0, \ldots, I^i, \ldots$ *until* $I^i = I^{i+1}$ *and set* $\widetilde{I^l_\nu} = I^i$.

In conclusion, we have that in a canonical schema ν over $\mathbf{S}(\mathbf{A},\mathbf{B},\mathbf{N})$, for all $N \in \mathbf{N}$, N is lfp–incoherent iff $N \in \widetilde{I^l_\nu} \cup \widetilde{I^g_\nu}$.

To give an intuition, let us observe that the above algorithm works only on virtual classes, in fact, at the first step the names of base classes are excluded. Furthermore, the set of type names I^{i+1} decreases at each iteration step by the elimination of set and sequence types and of the types which refer to type names not present in the set of types obtained at the previous step (I^i). As an example, let us show that the `Typist` virtual class is detected as lfp–incoherent by the algorithm (note that it is gfp–coherent):

$$\begin{array}{rcl} I^0 & = & N \setminus \{\texttt{Employee}, \texttt{Department}\} \\ I^1 & = & I^0 \setminus \{T_6\} \\ I^2 & = & I^1 \setminus \{T_5\} \\ I^3 & = & I^2 \setminus \{\texttt{Office}\} \\ I^4 & = & I^3 \setminus \{T_4\} \\ I^5 & = & I^4 \setminus \{\texttt{Clerk}\} \\ I^6 & = & I^5 \end{array}$$

We obtain $\texttt{Typist} \in \widetilde{I^l_\nu}$ and thus $\texttt{Typist}$ is detected as lfp–incoherent.

A remarkable result is that now we are able to compute lfp–subsumption by still adopting algorithm 2 (gfp–Subsumption) and by simply substituting the set of incoherent type names $\widetilde{I^g_\nu}$ with $\widetilde{I^g_\nu} \cup \widetilde{I^l_\nu}$.

If we denote by $\lesssim^l_\nu$ the subsumption relationships computed in such a way, we obtain that, in a canonical schema ν, $N \sqsubseteq^l_\nu N'$ iff $N \lesssim^l_\nu N'$.

Finally, let σ be a schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$, and let ν be a schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ which is a lfp–canonical extension of σ. Then:

- for all $C \in \mathbf{C}$: C is lfp–incoherent in σ iff $C \in \widetilde{I^l_\nu} \cup \widetilde{I^g_\nu}$;
- for all $C, C' \in \mathbf{C}$: $C \sqsubseteq^l_\sigma C'$ iff $C \lesssim^l_\nu C'$.

For detecting incoherence under descriptive semantics we can adopt algorithm 1 (GFP–INCOHERENCE) as a type is incoherent w.r.t. descriptive semantics if and only if it is incoherent w.r.t. gfp–semantics ($\widetilde{I^d_\nu} = \widetilde{I^g_\nu}$).

On the other hand, subsumption computation under descriptive semantics introduces a new problem, as there exist schemata including virtual cyclic classes for which the above canonical transformation is not extension preserving. We observe that this problem is due to *the elimination of conjunctions between general types.*

Let us show that, with reference to the class `Employee` and `Clerk`, a canonical transformation ν under descriptive semantics is not a conservative extension of the schema σ. First of all, it is easy to verify that any possible instance $\mathcal{I}$ (and thus legal under descriptive semantics) of σ of table 4 is a legal instance of ν. However, the *vice versa* does not hold; in fact, given the domain of table 4 the following interpretation $\mathcal{I}'$:

$$\begin{array}{rclcrcl} \mathcal{I}'[\texttt{Employee}] & = & \{o_9\} & \qquad & \mathcal{I}'[\texttt{Department}] & = & \{o_6, o_8\} \\ \mathcal{I}'[\texttt{Clerk}] & = & \{o_7, o_9\} & & \mathcal{I}'[\texttt{Office}] & = & \{o_6, o_8\} \end{array}$$

is a possible instance of ν, but is not a possible instance of σ as, due to the inheritance relation $\texttt{Clerk} \prec_\sigma \texttt{Employee}$ we must have $\mathcal{I}[\texttt{Clerk}] \subseteq \mathcal{I}[\texttt{Employee}]$. In terms of subsumption relations, we have that `Clerk` is subsumed by `Employee` in the schema σ ($\texttt{Clerk} \sqsubseteq^d_\sigma \texttt{Employee}$), but `Clerk` is not subsumed by `Employee` in the canonical schema ν ($\texttt{Clerk} \not\sqsubseteq^d_\nu \texttt{Employee}$).

Generalizing with respect to the above example, we can individuate the source of the problem in the presence of the same *cyclic path* between a class and one of its superclasses: the class `Clerk`, for example, has in common with the class `Employee` the cyclic path `worksin. employs. worksin.`... The problem arises only under descriptive semantics (not under the fixed–point semantics) as only adopting descriptive semantics two cyclic virtual classes with the same description might have independent interpretations. Note that, this problem occurs also between classes without inheritance relations, as in the case of the classes `Typist` and `Clerk`: in fact, given the domain of table 4 the following interpretation $\mathcal{I}'$:

$$\begin{array}{rclcrcl} \mathcal{I}'[\texttt{Typist}] & = & \{o_9\} & \qquad & \mathcal{I}'[\texttt{Department}] & = & \{o_6, o_8\} \\ \mathcal{I}'[\texttt{Clerk}] & = & \{o_7, o_9\} & & \mathcal{I}'[\texttt{Office}] & = & \{o_6, o_8\} \end{array}$$

is a possible instance of ν, but it is not a possible instance of σ. A formal definition of cyclic schemata which cannot be transformed into a conservative canonical extension under descriptive semantics will be given in Appendix.

Thus we need to extend for descriptive semantics, the definition of canonical schemata. An *extended canonical schema* is a pair $(\nu, \ll_\nu)$ where ν is a canonical schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ and $\ll_\nu$ is a relation on $\mathbf{C_V}$. A legal instance $\mathcal{I}$ of ν is a legal instance of $(\nu, \ll_\nu)$ iff $\mathcal{I}[V] \subseteq \mathcal{I}[V'],\ \forall V, V' : V \ll_\nu V'$.

Any schema σ can be effectively transformed into an extended canonical schema $(\nu, \ll_\nu)$ which is a descriptive canonical extension of σ. In fact, in order to obtain the relation $\ll_\nu$, it is sufficient to add at step 2 of the canonical transformation algorithm the following statement: insert in $\ll_\nu$ the pair (N, N') whenever, during the canonical transformation, N' contained in N is substituted by its description. Therefore, at first $\ll_\nu = \preceq_\sigma$ but this set of relations grows with the transformations made.

To give a rough idea, let us consider the canonical form of the virtual class `Typist`: as V_2 is obtained from the canonical transformation of the type $\texttt{Office} \sqcap \triangle[\texttt{typing}\colon \texttt{Typist}]$, we insert into $\ll_\nu$ (besides the inheritance relations) the pair $\{(V_2, \texttt{Office})\}$.

Note that, we can adopt the modified algorithm as a general canonical transformation algorithm for all the three semantics; in this case the relation $\ll_\nu$ will be used only for descriptive semantics.

Using the extended canonical schema $(\nu, \ll_\nu)$, computing subsumption under descriptive semantics can be performed as for gfp–semantics, on the basis of the relation $\sqsubseteq_\leq$.

Algorithm 4 (Descriptive–Subsumption) *Let $\leq^0 = \ll_\nu$ and define $\leq^{i+1}$ in the following way:*

$$\leq^{i+1} \;=\; \leq^i \;\cup \left\{ (N, N') \in \mathbf{N} \times \mathbf{N} \,\middle|\, \nu(N) \sqsubseteq_{\leq^i} \nu(N') \vee N \in \widetilde{I^d_\nu} \vee \nu(N') = \top \right\}.$$

Compute $\leq^0, \ldots, \leq^i, \ldots$ until $\leq^i = \leq^{i+1}$ and set $\lesssim^d_\nu \; = \leq^i$.

As above, a natural number n always exists such that $\leq^n = \leq^{n+1}$; the maximal n is $|\mathbf{N}|^2$, i.e., also the descriptive–subsumption algorithm is polynomial in ν.

In the extended canonical schema, for all $N, N' \in \mathbf{N}$ we have that: $N \sqsubseteq^d_\nu N'$ iff $N \lesssim^d_\nu N'$. Let σ be a schema over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$, and let $(\nu, \ll_\nu)$ be a an extended canonical schema which is a descriptive canonical extension of σ. Then:

- for all $C \in \mathbf{C}$: C is des–incoherent in σ iff $C \in \widetilde{I^g_\nu}$;
- for all $C, C' \in \mathbf{C}$: $C \sqsubseteq^d_\sigma C'$ iff $C \lesssim^d_\nu C'$.

Let us briefly comment the subsumption computation under descriptive semantics. First of all, notice the difference of subsumption computation between gfp and descriptive semantics. In algorithm 2 the set of subsumption relations decreases from one iteration to the subsequent one until we reach a fix point: we start from subsumption relations between all the couples of type names and then we eventually eliminate, on the basis of types descriptions, subsumption relations which do not hold. In algorithm 4 we start from a set of subsumption relations coinciding exactly with the set of explicitly given inheritance relations and we eventually increase this set on the basis of types descriptions. For example, with reference to the virtual class `Typist`, we begin with $V_2 \leq^0 \texttt{Office}$, then, at the first iteration, we add $T_7 \leq^1 T_4$ and then we obtain $\texttt{Typist} \leq^2 \texttt{Clerk}$.

4 Conclusions

We adopted the ODL formalism to describe complex object databases schemata including cyclic views and to express recursive queries. We solved the problems of detecting incoherence and computing subsumption for cyclic views and queries with respect to different styles of semantics: greatest fixed-point, least fixed-point and descriptive.

The greatest and least fixed-point semantics are the only suitable semantics for cyclic queries as they uniquely define queries answer sets. The greatest fixed-point semantics allows to directly express the transitive closure of an attribute. The least fixed-point semantics allows to express the transitive closure of an attribute but limiting to acyclic objects.

On the other hand, as in a database schemata including classes and views, a cyclic view may be interpreted by any of the above mentioned semantics, we considered subsumption and incoherence for cyclic views and queries in the most general framework of heterogeneus schemata.

We showed that, even if from a purely theoretical point of view (see Appendix), the computation of subsumption and incoherence control are intractable, these problems can be efficiently solved by transforming a schema in a canonical form. These results imply that computing the canonical extension of a schema is difficult or that the canonical extension of a schema has a worst-case size that is exponential in the size of the original schema. A worst-case example is given in [34]. However the intractability previously mentioned rarely occurs in practice as a schema is generally formulated in such a way as to be "almost" canonical—an observation also made in the area of knowledge representation [34], where we have to deal with quite similar problems. Hence, we can conclude that transforming a schema to its canonical extension is feasible for most cases that appear in practice.

This research can be continued by trying to extend the result to cyclic views and queries in larger languages, for instance, by handling an existentially quantified set operator.

References

[1] S. Abiteboul and A. Bonner. Objects and views. In *SIGMOD*, pages 238–247. ACM Press, 1991.

[2] S. Abiteboul and S. Grumbach. Col: a logic-based language for complex objects. In S. Ceri, J.W. Schmidt, and M. Missikoff, editors, *EDBT '88 - Lecture Notes in Computer Science N.303*, pages 271–293. Springer-Verlag, 1988.

[3] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, 1987.

[4] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *SIGMOD*, pages 159–173. ACM Press, 1989.

[5] R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *IEEE Transactions on Software Enginering*, 14(7), July 1988.

[6] M. Atkinson et al. The object-oriented database system manifesto. In *1nd Int. Conf. on Deductive and Object-Oriented Databases*. Springer-Verlag, 1989.

[7] P. Atzeni, editor. *LOGIDATA$^+$: Deductive Databases with Complex Objects*. Springer-Verlag: LNCS n. 701, Heidelberg - Germany, 1993.

[8] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *8th National Conference of the American Association for Artificial Intelligence*, volume 2, pages 621–626, Boston, Mass., USA, 1990.

[9] J.P. Ballerini and S. Bergamaschi. Automatic building and validation of complex object database schemata. Technical Report 5/124, CNR - Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, July 1993.

[10] G. Di Battista and M. Lenzerini. Deductive entity relationship modeling. *IEEE Trans. on Knowledge and Data Engineering*, 5(3):439–450, June 1993.

[11] H.W. Beck, S.K. Gala, and S.B. Navathe. Classification as a query processing technique in the CANDIDE data model. In *5th Int. Conf. on Data Engineering*, pages 572–581, Los Angeles, CA, 1989.

[12] C. Beeri. Formal models for object-oriented databases. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Deductive and Object-Oriented Databases*, page 405:430. Elsevier Science Publisher, B.V. - North-Holland, 1990.

[13] D. Beneventano and S. Bergamaschi. Subsumption for complex object data models. In J. Biskup and R. Hull, editors, *4th Int. Conf. on Database Theory - Berlin*, pages 357–375, Heidelberg, Germany, October 1992. Springer-Verlag.

[14] S. Bergamaschi and B. Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem Solving Technologies*, 4:185–203, 1994.

[15] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, September 1992.

[16] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-oriented query languages: The notion and the issues. *IEEE Trans. Knowl. and Data Engineering*, 4(3):223–236, June 1992.

[17] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *SIGMOD*, pages 58–67, Portland, Oregon, 1989.

[18] M. Buchheit, M. A. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented database. In *EDBT*, pages 348–353, 1994.

[19] L. Cardelli. A semantics of multiple inheritance. In *Semantics of Data Types - Lecture Notes in Computer Science N. 173*, pages 51–67. Springer-Verlag, 1984.

[20] R.G.G. Cattell and J. Skeen. Object operations benchmark. *ACM Transactions on Database Systems*, 17(1), 1990.

[21] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[22] P. Chen. The Entity-Relationship model - towards a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[23] F.M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. In J. Allen, R. Fikes, and E. Sandewall, editors, *KR '91 - 2nd Int. Conf on Principles of Knowledge Representation and Reasoning*, pages 151–162, Cambridge - MA, April 1991. Morgan Kauffmann Publishers, Inc.

[24] H. Gallaire, J. Minker, and J.M. Nicholas. Logic and databases: a deductive approach. *ACM Computing Surveys*, 16(2), 1984.

[25] Lewis H.R. and H.P. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, Englewood Cliffs, N.J., 1981.

[26] W. Kim and F. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications*. Addison-Wesley, Reading (Mass.), 1989.

[27] J. J. King. Quist: a system for semantic query optimization in relational databases. In *7th Int. Conf. on Very Large Databases*, pages 510–517, 1981.

[28] C. Lecluse and P. Richard. Modelling complex structures in object-oriented databases. In *Symp. on Principles of Database Systems*, pages 362–369, Philadelphia, PA, 1989.

[29] C. Lécluse and P. Richard. The O_2 data model. In *Int. Conf. On Very Large Data Bases*, pages 411–422, Amsterdam, 1989.

[30] J.W. Lloyd. *Logic Programming.* Springer-Verlag, Berlin, 1987.

[31] J. Minker and J.M. Nicholas. On recursive axiom in deductive databases. *Information Systems*, 8(1):1–13, 1983.

[32] J. Mylopoulos, P.A. Bernstein, and H.K.T. Wong. A language facility for designing database-intensive applications. *ACM Transactions on Database Systems*, 5(2):185–207, 1980.

[33] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence.* Springer-Verlag, Berlin, Heidelberg, New York, 1990.

[34] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2), 1990.

[35] B. Nebel. Terminological cycles: Semantics and computational properties. In J.F. Sowa, editor, *Principles of Semantic Networks*, chapter 11, pages 331–362. Morgan Kaufmann Publishers, Inc., 1991.

[36] B. Nebel and C. Peltason. Terminological reasoning and information management. Technical Report 85, Tech. Univ, Berlino, October 1990.

[37] D.A. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, Boston, 1986.

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

[39] D.W. Shipman. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, 1981.

[40] W.A. Woods and J.G. Schmolze. The kl-one family. In F.W. Lehman, editor, *Semantic Networks in Artificial Intelligence*, pages 133–178. Pergamon Press, 1992. Pubblished as a Special issue of *Computers & Mathematics with Applications*, Volume 23, Number 2-9.

A Appendix: Mapping schemata to finite automata

In the following we show that coherence and subsumption, for each of the different styles of semantics, can be reduced to properties of regular languages by specifying a mapping from ODL schemata to nondeterministic finite automata. A similar approach was adopted for cyclic DLs in [8, 34, 35] and firstly proposed in database environment in [13],

The main difference with respect to the cited papers on cyclic DLs is that, due to the increased expressiveness of ODL with respect to DLs (i.e. distinction between objects, complex values; type constructors as tuple, set and sequence) we developed a significatively more complex mapping from schemata to automata and new techniques to detect incoherences between structured value-types (see sectionA.1). In particular, this last problem is not dealt with in [8, 34, 35].

Furthermore, from a theoretical point of view, the mapping from schemata to automata allowed us to analyze and formalize a problem of cyclic schemata which arises adopting descriptive semantics: the impossibility of an extension preserving canonical transformation for particular cyclic schemata.

Let Σ be a finite *alphabet*. The set of all finite words over Σ will be denoted by Σ^* and the empty word by ϵ. The set of all infinite words over Σ will be denoted by Σ^ω.

A *Nondeterministic Finite Automaton* $\mathcal{A} = (\mathcal{Q}, \Sigma, \delta, q_0, \mathcal{F})$ consists of a finite set of *states* $\mathcal{Q}$, a finite *alphabet* Σ, and a finite set of *transitions* $\delta \subseteq \mathcal{Q} \times (\Sigma \cup \{\epsilon\}) \times 2^{\mathcal{Q}}$, an *initial state* q_0 and a set of *final states* $\mathcal{F}$.

A *finite path* from q_0 to q_n, with $q_0, q_n \in \mathcal{Q}$, is a sequence $q_0, \alpha_1, q_1, \alpha_2, \ldots, \alpha_n, q_n$, where for each i, $1 \leq i \leq n$, we have $(q_{i-1}, \alpha_i, q_i) \in \delta$; the finite word $\alpha_1\alpha_2 \ldots \alpha_n$ is called the *label* of this path. The path consisting of q alone, is a path from q to q, called *empty path*, with label ϵ.

The finite word w is *accepted* by $\mathcal{A}$ iff w is a label of a path from the initial state q_0 to a final state $q_i \in \mathcal{F}$. The set of all finite words which are accepted by $\mathcal{A}$ is called *language accepted* by $\mathcal{A}$ and will be denoted by $\mathcal{L}(\mathcal{A})$ (see, e.g., [25]).

An *infinite path* from q_0 is an infinite sequence $q_0, \alpha_1, q_1, \alpha_2, \ldots,$, where for each i, $i \geq 1$, we have $\langle q_{i-1}, \alpha_i, q_i \rangle \in \delta_\sigma$; the finite or infinite word $w = \alpha_1\alpha_2 \ldots$ is the label of this path. We denote with $\mathcal{U}(\mathcal{A})$ the set of all words which are labels of infinite paths starting with the initial state q_0 of $\mathcal{A}$.

Given a schema σ over $\mathbf{S}$, the set of type expressions *used as subexpressions* in the range of σ is denoted by $\mathbf{S}_\sigma$. We associate to σ a set of automata $\mathcal{A}_\sigma = (\mathcal{Q}_\sigma, \Sigma_\sigma, \delta_\sigma, q_0, \mathcal{F})$, where

$$\begin{array}{rcl} \Sigma_\sigma & = & \mathbf{A} \cup \Big\{\{\}, \Diamond, \triangle\Big\} \\ \mathcal{Q}_\sigma & = & \mathbf{C} \cup \mathbf{S}_\sigma \end{array}$$

and the transitions between states are defined by:

$$\begin{array}{rcl} \Big\langle B, \epsilon, B' \Big\rangle \in \delta_\sigma & \text{iff} & B \sqcap B' = B \\ \Big\langle C, \epsilon, S \Big\rangle \in \delta_\sigma & \text{iff} & \sigma(C) = S \\ \Big\langle \{S\}, \{\}, S \Big\rangle \in \delta_\sigma & \text{iff} & \{S\} \in \mathbf{S}_\sigma \\ \Big\langle \langle S \rangle, \Diamond, S \Big\rangle \in \delta_\sigma & \text{iff} & \langle S \rangle \in \mathbf{S}_\sigma \\ \Big\langle [\ldots, a_i : S_i, \ldots], a_i, S_i \Big\rangle \in \delta_\sigma & \text{iff} & [\ldots, a_i : S_i, \ldots] \in \mathbf{S}_\sigma \end{array}$$

$$\Big\langle (S \sqcap S'), \epsilon, S \Big\rangle, \Big\langle (S \sqcap S'), \epsilon, S' \Big\rangle \in \delta_\sigma \quad \text{iff} \quad (S \sqcap S') \in \mathbf{S}_\sigma$$
$$\Big\langle (\triangle S), \triangle, S \Big\rangle \in \delta_\sigma \quad \text{iff} \quad (\triangle S) \in \mathbf{S}_\sigma$$

The automata $\mathcal{A}_\sigma$ have a size that is linear in the size of the schema σ, which is defined as the sum of the sizes of the expressions $\sigma(C)$ for all $C \in \mathbf{C}$. The automaton $\mathcal{A}_\sigma$ with initial state q_0 and final states $\mathcal{F}$ will be denoted by $\mathcal{A}_\sigma(q_0, \mathcal{F})$. The expression $\Sigma^*(\{\}|\langle\rangle)\Sigma^*$ denotes the set of finite words containing $\{\}$ or $\langle\rangle$ and the expression $\Sigma^*(\{\}|\langle\rangle)\Sigma^\omega$ denotes the set of infinite words containing $\{\}$ or $\langle\rangle$.

A.1 Incoherence Detection

Viewing a database schema as a finite automaton, we can give a simple criterion for incoherence detection. For this purpose, we define the notion of immediately incoherent, deriving from a syntactic type incompatibility. Given $\mathbf{S}$, we define the set of couples $\Phi \subseteq \mathbf{S} \times \mathbf{S}$ of types *immediately incoherent* as follows:

$$\begin{aligned}
\Big\langle S, \bot \Big\rangle \in \Phi \quad &\text{iff} \quad S \in \mathbf{S}_\sigma \\
\Big\langle B, S \Big\rangle \in \Phi \quad &\text{iff} \quad S' \in \mathbf{B} \ \wedge \ B \sqcap S' = \bot \\
\Big\langle C, S' \Big\rangle \in \Phi \quad &\text{iff} \quad S' \notin \mathbf{C} \ \wedge \ S' \neq \triangle S'' \\
\Big\langle \{S\}, S' \Big\rangle \in \Phi \quad &\text{iff} \quad S' \neq \{S''\} \\
\Big\langle \langle S \rangle, S' \Big\rangle \in \Phi \quad &\text{iff} \quad S' \neq \langle S'' \rangle \\
\Big\langle [\ldots, a_i : S_i, \ldots], S' \Big\rangle \in \Phi \quad &\text{iff} \quad S' \neq [\ldots, a_j^{'} : S_j^{'}, \ldots] \\
\Big\langle \triangle S, S' \Big\rangle \in \Phi \quad &\text{iff} \quad S' \neq \triangle S'' \ \wedge \ S' \notin \mathbf{C}
\end{aligned}$$

Based on this notion, incoherence of a type can be characterized as follows.

Theorem 1 *Given a schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ with $\mathbf{B}$ binary compact. For all $S \in \mathbf{S}_\sigma$:*

- *S is gfp–incoherent and des–incoherent iff*

$$\exists \langle S_1, S_2 \rangle \in \Phi : \ \mathcal{L}(\mathcal{A}_\sigma(S, \{S_1\})) \cap \mathcal{L}(\mathcal{A}_\sigma(S, \{S_2\})) \not\subseteq \Sigma^*(\{\}|\langle\rangle)\Sigma^*.$$

- *S is lfp–incoherent iff*

$$S \text{ is gfp–incoherent} \quad \text{or} \quad \mathcal{U}(\mathcal{A}_\sigma(S, \{\})) \not\subseteq \Sigma^*(\{\}|\langle\rangle)\Sigma^\omega.$$

The common condition can be intuitively understood as: a type expression is incoherent if a type information is conflicting, i.e., following identical paths in the automaton we reach states that are immediately incoherent. Since sets or sequences can be empty, we will allow for incoherent types inside set or sequence operators. For example, the query Q_1 in the introduction is incoherent, as for the couple immediately incoherent $4, 10–\infty\rangle$ we have:

$$\triangle\texttt{employs} \triangle \texttt{level} \in \mathcal{L}(\mathcal{A}_\sigma(Q_1, \{4\})) \cap \mathcal{L}(\mathcal{A}_\sigma(Q_1, \{10–\infty\}))$$

Due to the hypothesis of binary compactness of $\mathbf{B}$, we can perform the check of two types at a time. Remember that a system of atomic types is binary compact if for each subset $\mathbf{X} \subseteq \mathbf{B}$ with $\bigsqcap \mathbf{X} = B$, there are two elements $B', B'' \in \mathbf{X}$ such that $B' \sqcap B'' = B$. If this property is not satisfied we could have $\mathbf{X} = \{B_1, B_2, B_3\}$ with $\bigsqcap \mathbf{X} = \bot$ and $B_1 \sqcap B_2 \neq \bot$, $B_2 \sqcap B_3 \neq \bot$ e $B_1 \sqcap B_3 \neq \bot$. In this case the incoherent type $B_1 \sqcap B_2 \sqcap B_3$ would not be detected by theorem 1.

In particular, notice that if a class name C is defined by a value–type description S , i.e., in the automaton there is an ϵ-path from C to a type S not prefixed by $\triangle$, we have $\langle C, S\rangle \in \Phi$ and $\mathcal{L}(\mathcal{A}_\sigma(C, \{C\})) \cap \mathcal{L}(\mathcal{A}_\sigma(C, \{S\})) = \epsilon$; hence the class name C is incoherent.

For lfp–semantics there is an additional source of incoherence (which gives rise to the second condition of theorem 1) owing to circular references to type constructors different from sequence and set. This condition checks that any infinite path starting with S contains at least one transition labelled with $\{\}$ or $\lozenge$, i.e., the circular references used in the description of S contain at least one sequence or set type constructor. For example, figure 1 shows that the only infinite path starting with `Clerk` contains a $\{\}$ transition, and thus `Clerk` is lfp–coherent, as well as `Office`. On the other hand, an infinite path without $\{\}$ or $\lozenge$ symbols (the one involving `worksin` and `typing`) starts in `Secretary`, so that `Secretary` is lfp–incoherent, as well as `T-Office`.

A.2 Subsumption Computation

Like incoherence, subsumption can be computed using properties of regular languages accepted by our automaton. In order to give a characterization of subsumption in terms of regular languages, we first have to modify the automaton $\mathcal{A}_\sigma$, to consider also the incoherent types. First of all, $\langle \bot, a, \bot\rangle$ is added to δ for all $a \in \Sigma \cup \{\epsilon\}$. For any state q such that q is gfp–incoherent (and thus des–incoherent), the transition $\langle S, \epsilon, \bot\rangle$ is added to δ. The new automaton will be denoted by $\mathcal{A}^g_\sigma$, and with $\mathcal{A}^l_\sigma$ we denote the automaton $\mathcal{A}^g_\sigma$ where the transitions $\langle S, \epsilon, \bot\rangle$ relative to further lfp–incoherent types are added to δ. Formally, given $\mathcal{A}_\sigma = (\mathcal{Q}, \Sigma, \delta_\sigma, q_0, \mathcal{F})$ related to a schema σ, we define:

$$\begin{aligned}
\mathcal{A}^g_\sigma \;&=\; (\mathcal{Q}, \Sigma, \delta^g_\sigma, q_0, \mathcal{F}) \;\text{ where} \\
&\quad\;\; \delta^g_\sigma = \delta_\sigma \cup \Big\{\langle \bot, \alpha, \bot\rangle \mid \alpha \in \Sigma \cup \{\epsilon\}\Big\} \cup \Big\{\langle S, \epsilon, \bot\rangle \mid S \simeq^l_\sigma \bot\Big\} \\
\mathcal{A}^l_\sigma \;&=\; (\mathcal{Q}, \Sigma, \delta^l_\sigma, q_0, \mathcal{F}) \;\text{ where} \\
&\quad\;\; \delta^l_\sigma = \delta_\sigma \cup \Big\{\langle \bot, \alpha, \bot\rangle \mid \alpha \in \Sigma \cup \{\epsilon\}\Big\} \cup \Big\{\langle S, \epsilon, \bot\rangle \mid S \simeq^l_\sigma \bot\Big\}
\end{aligned}$$

Besides, let $q_0, w_0, q_1, w_1, q_1, w_2, \ldots$ denote the infinite path starting at q_0 which reaches q_1 infinitely often where w_j are nonempty labels of paths from q_0 to q_1 for $j = 0$ and from q_1 to q_1 for $j > 0$.

Theorem 2 *Given a schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ for all $S, S' \in \mathbf{S}_\sigma$:*

- *$S \sqsubseteq^g_\sigma S'$ iff*
  1. *$\mathcal{L}(\mathcal{A}^g_\sigma(S, \{E, \bot\})) \supseteq \mathcal{L}(\mathcal{A}^g_\sigma(S', \{E\})) \quad \forall E \in (\mathbf{B} \cup \mathbf{C}_\mathbf{B}) \cap \mathbf{S}_\sigma$;*
  2. *$\mathcal{U}(\mathcal{A}^g_\sigma(S, \{\})) \supseteq \mathcal{U}(\mathcal{A}^g_\sigma(S', \{\}))$.*
- *$S \sqsubseteq^l_\sigma S'$ iff*

1. $\mathcal{L}(\mathcal{A}^l_\sigma(S, \{E, \bot\})) \supseteq \mathcal{L}(\mathcal{A}^l_\sigma(S', \{E\})) \ \ \forall E \in (\mathbf{B} \cup \mathbf{C_B}) \cap \mathbf{S}_\sigma$;
2. $\mathcal{U}(\mathcal{A}^l_\sigma(S, \{\})) \supseteq \mathcal{U}(\mathcal{A}^l_\sigma(S', \{\}))$.

- $S \sqsubseteq^d_\sigma S'$ *iff*
  1. $\mathcal{L}(\mathcal{A}^g_\sigma(S, \{E, \bot\})) \supseteq \mathcal{L}(\mathcal{A}^g_\sigma(S', \{E\})) \ \ \forall E \in (\mathbf{B} \cup \mathbf{C_B}) \cap \mathbf{S}_\sigma$;
  2. *For all* $C \in \mathbf{C_V}$ *and all infinite paths of the form* $S'w_0Cw_1Cw_2C\ldots$ *exists* $k \geq 0$ *such that* $w_0w_1\ldots w_k \in \mathcal{L}(\mathcal{A}^g_\sigma(S, \{C, \bot\}))$.

As mentioned before, for schemata with no virtual cyclic class, the three semantics coincide, thus the following corollary holds.

Corollary 1 *Given a schema* σ *over* $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ *with no virtual cyclic class, for all* $S, S' \in \mathbf{S}_\sigma$*:*

$$S \sqsubseteq_\sigma S' \quad \textit{iff} \quad \mathcal{L}(\mathcal{A}^g_\sigma(S, \{E, \bot\})) \supseteq \mathcal{L}(\mathcal{A}^g_\sigma(S', \{E\})) \ \ \forall E \in (\mathbf{B} \cup \mathbf{C_B}) \cap \mathbf{S}_\sigma$$

The corollary can be intuitively understood as: the languages $\mathcal{L}(\mathcal{A}^g_\sigma(S, \{E\}))$, with $E \in \mathbf{B} \cup \mathbf{C_B}$ stand for the *constraints* which the schema imposes on S with respect to base classes and atomic types. A value is admissible in the extension of S if and only if it satisfies all of these constraints. If S has to satisfy more constraints than S', i.e., $\mathcal{L}(\mathcal{A}^g_\sigma(S, \{E\})) \supseteq \mathcal{L}(\mathcal{A}^g_\sigma(S', \{E\})) \ \forall E \in (\mathbf{B} \cup \mathbf{C_B}) \cap \mathbf{S}_\sigma$, the extension of S will become smaller than the extension of S', thus $S \sqsubseteq_\sigma S'$. For example, for the computation of `Adl-Employee` $\sqsubseteq_\sigma$ `Mdl-Employee` we have:

$$\mathcal{L}(\mathcal{A}^g_\sigma(\texttt{Adl-Employee}, \{8\text{–}10\})) = \{\triangle\texttt{level}\} \supset \mathcal{L}(\mathcal{A}^g_\sigma(\texttt{Mdl-Employee}, \{8\text{–}10\})) = \emptyset$$
$$\mathcal{L}(\mathcal{A}^g_\sigma(\texttt{Adl-Employee}, \{5\text{–}10\})) = \mathcal{L}(\mathcal{A}^g_\sigma(\texttt{Mdl-Employee}, \{5\text{–}10\})) = \{\triangle\texttt{level}\}$$

Note that, the second relation is due to the transition $\langle 8\text{–}10, \epsilon, 5\text{–}10\rangle$: in fact $8\text{–}10 \sqcap 5\text{–}10 = 8\text{–}10$.

Besides, to take into account that, for all S, $\bot \sqsubseteq_\sigma S$, $\mathcal{A}^g_\sigma(q_0, \mathcal{F})$ has been defined so that $\mathcal{L}(\mathcal{A}^g_\sigma(\bot, \{\bot\})) = \Sigma^* \cup \{\epsilon\}$ and $\bot$ has been added as a final state to $\mathcal{A}^g_\sigma(S, \{E\})$.

In theorem 2, we need a further condition to represent constraints with respect to cyclic virtual classes for the three different semantics. In fact, we can have cyclic virtual classes which make no reference to base classes and atomic types, as in the following example:

$$\begin{aligned} \sigma(X) &= \triangle\Big[\texttt{worksin}\colon Y\Big] \\ \sigma(Y) &= \triangle\Big[\texttt{employs}\colon \{X\}\Big] \end{aligned}$$

where it is easy to see that point 1 of theorem 2 is useless, as it would give $S \sqsubseteq_\sigma X$ and $S \sqsubseteq_\sigma Y$, for each type S. For gfp–semantics, point 2 takes into account these cyclic virtual classes[9] by checking that for every infinite path starting with the type S there is one with the same label starting with the type S'. Thus, with respect to the example schema, we would have `Clerk` $\sqsubseteq^g_\sigma X$ and `Office` $\sqsubseteq^g_\sigma Y$. The same holds for *lfp*–semantics, if we consider the $\mathcal{A}^l_\sigma$ automaton.

For descriptive semantics, infinite paths are still important but it is not sufficient to only consider their labels. The states which are infinitely often reached by this path, which are cyclic virtual classes, are also significant, thus point 2 applies the same type of check as point 1, but with reference to cyclic virtual classes. In fact, in descriptive semantics, the cyclic virtual classes are in some way similar to the base classes, so we must also check the constraints with respect to cyclic virtual classes. Therefore, in our example schema no subsumption relationship is detected for the X and Y classes.

[9]Note that point 2 deals with all cyclic classes, and thus repeats some checks in point 1.

A.3 Computational Complexity

In this section we shall briefly discuss the computational complexity of incoherence control and subsumption computation in a schema, referring to the computational complexity results reported in the papers [8, 14, 34, 35].

Theorem 2 states that the problem of determining the subsumption w.r.t. the gfp–semantics and lfp–semantics can be reduced to two problems: (1) inclusion of regular languages (defined by nondeterministic finite state automata) and (2) inclusion of infinite words. The first problem can be solved by means of a PSPACE algorithm. As for the second problem, [8] shows how the inclusion of infinite words can be reduced to problems of inclusion for regular languages and can therefore be solved by means of a PSPACE algorithm. In conclusion the problem of determining subsumption w.r.t. the gfp–semantics and *lfp*–semantics can be reduced to problems of inclusion for regular languages and is therefore a PSPACE problem. In [8] the other direction of these reductions has also been shown: an inclusion problem can be reduced to a subsumption problem and so it is shown that the problem of subsumption is PSPACE–complete.

For descriptive semantics point 2 of theorem 2 is no longer expressed as an inclusion of infinite words. However, [8] shows how this problem can be reduced to a problem of languages of infinite words accepted by Büchi automata and can therefore be solved by means of a PSPACE algorithm. The same result was obtained in [35] characterizing the equivalence w.r.t. the descriptive semantics by means of deterministic automata. Therefore, the problem of determining subsumption w.r.t. the descriptive semantics is also a PSPACE problem. However, whether this problem is PSPACE–hard or simpler is still open.

The complexity of the problem of checking coherence for ODL schemata w.r.t. the gfp–semantics was assessed in [14], where a theorem states that it is NP-hard in the general case and polynomial when the system of atomic types is binary compact. As a type is des–incoherent if and only if it is gfp–incoherent, the same complexity result can be assumed for descriptive semantics. We can assert the same complexity results for lfp–semantics as the second condition of theorem 1 does not increase the problem complexity (it can be solved in polynomial time).

A.4 Canonical Transformation

By using languages accepted by automaton, we can formally define cyclic schemata which cannot be transformed into a descriptive–canonical extension.

For this purpose, given a class name C, we denote by $\mathcal{A}_{\sigma\backslash C}$ the automaton $\mathcal{A}_\sigma$ where the transitions starting from C are eliminated. Formally, given $\mathcal{A}_\sigma = (\mathcal{Q}, \Sigma, \delta_\sigma, q_0, \mathcal{F})$ related to a schema σ and $C \in \mathcal{Q}$, we define:

$$\begin{aligned} \mathcal{A}_{\sigma\backslash C} &= (\mathcal{Q}, \Sigma, \delta_{\sigma\backslash C}, q_0, \mathcal{F}) \text{ where} \\ & \quad \delta_{\sigma\backslash C} = \delta_\sigma \setminus \left\{ \langle C, \alpha, q\rangle \mid \alpha \in \Sigma \cup \{\epsilon\}, q \in \mathcal{Q} \right\} \end{aligned}$$

A schema σ over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{C})$ can be effectively transformed into a schema ν over $\mathbf{S}(\mathbf{A}, \mathbf{B}, \mathbf{N})$ which is a descriptive–canonical extension of σ if for each $C, C' \in \mathbf{C}$ such that $C \sqsubseteq^d_\sigma C'$ we have that

$$\mathcal{U}(\mathcal{A}_{\sigma\backslash C}(C', \{\})) \cap \mathcal{U}(\mathcal{A}_{\sigma\backslash C'}(C, \{\})) = \emptyset.$$

For example, for the class Typist we have that Typist $\sqsubseteq^d_\sigma$ Clerk and

$$\mathcal{U}(\mathcal{A}_{\sigma \backslash \texttt{Typist}}(\texttt{Clerk}, \{\})) \cap \mathcal{U}(\mathcal{A}_{\sigma \backslash \texttt{Clerk}}(\texttt{Typist}\{\})) = \\ \{\triangle \texttt{worksin} \triangle \texttt{typing} \triangle \texttt{worksin} \ldots\}$$

then we cannot find an extension preserving canonical transformation for this cyclic schemata.

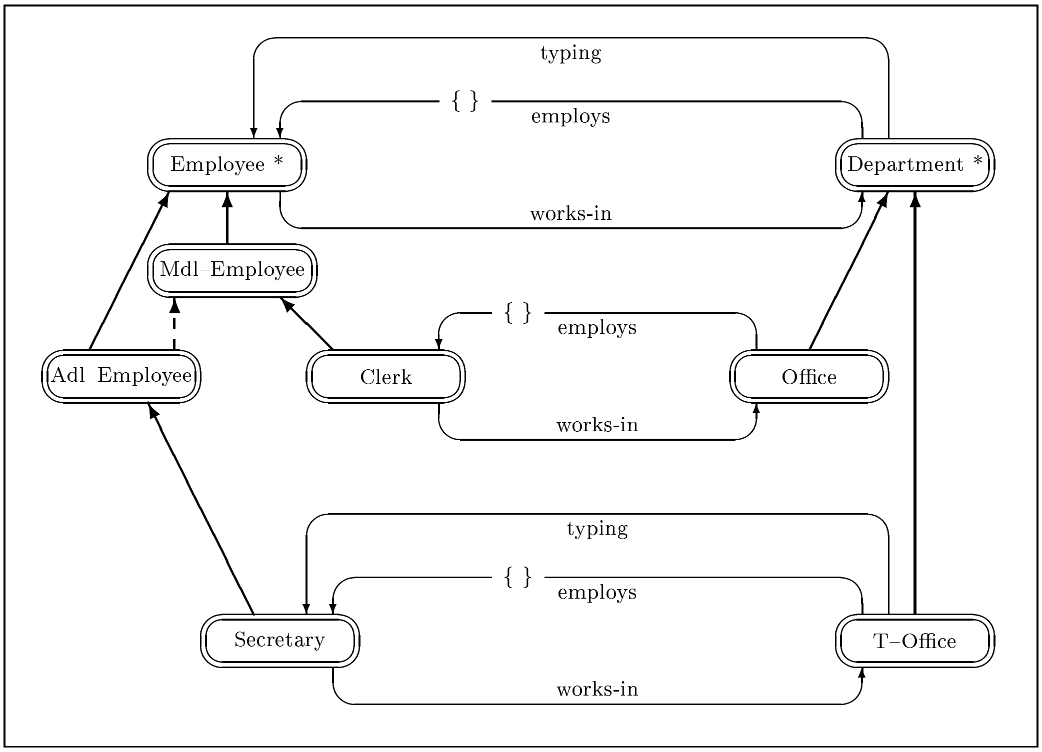

Figure 1: The Company domain

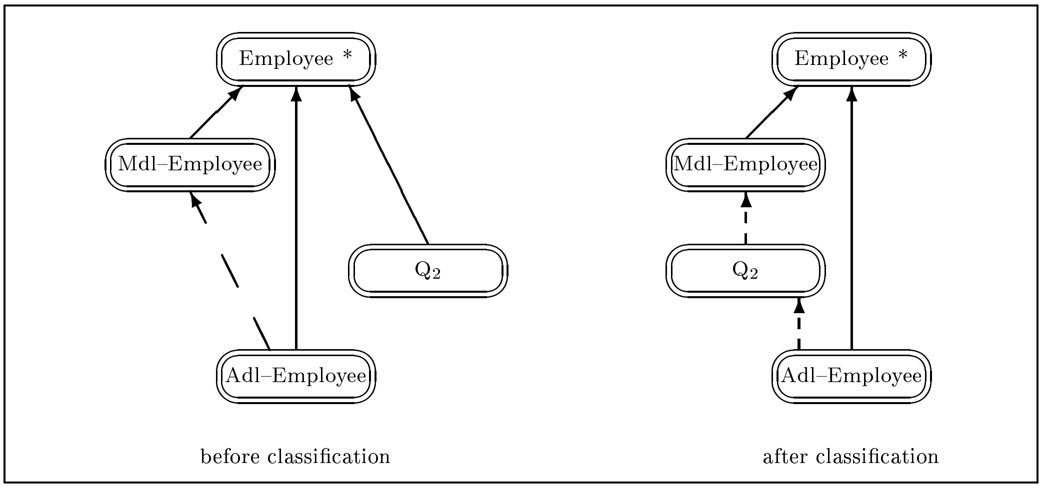

Figure 2: Query classification

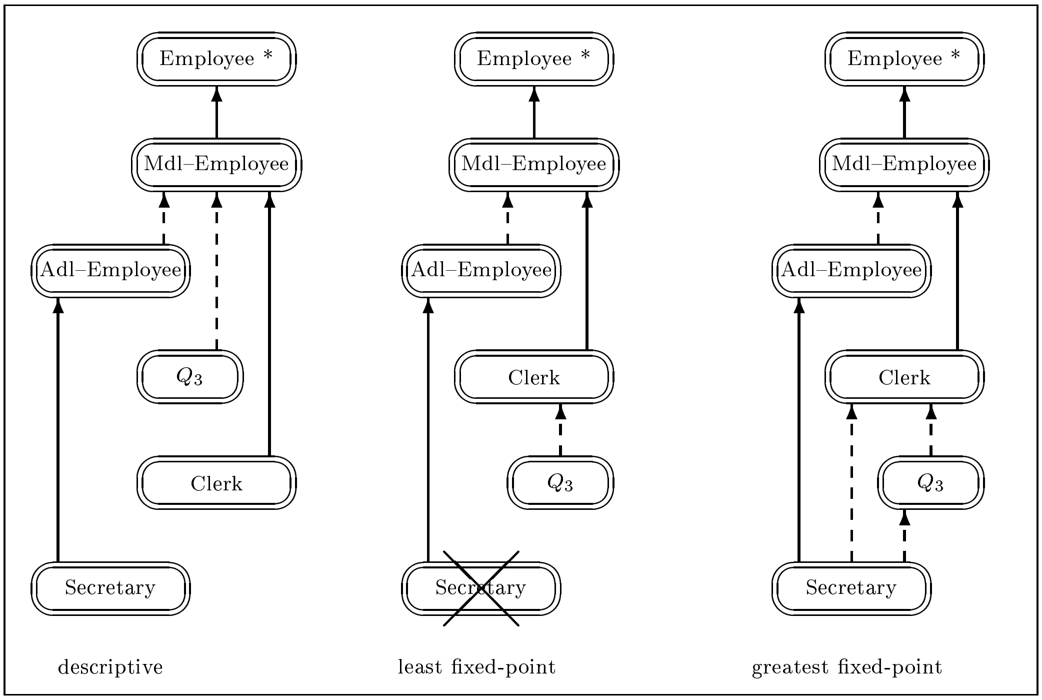

Figure 3: Subsumption and coherence for the different semantics.