Appeared in Proc. of **PODS'94**

Making Object-Oriented Schemas More Expressive

Diego Calvanese
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
calvanese@assi.dis.uniroma1.it

Maurizio Lenzerini
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lenzerini@assi.dis.uniroma1.it

Abstract

Current object-oriented data models lack several important features that would allow one to express relevant knowledge about the classes of a schema. In particular, there is no data model supporting simultaneously the inverse of the functions represented by attributes, the union, the intersection and the complement of classes, the possibility of using nonbinary relations, and the possibility of expressing cardinality constraints on attributes and relations. In this paper we define a new data model, called $\mathcal{CAR}$, which extends the basic core of current object-oriented data models with all the above mentioned features. A technique is then presented both for checking the consistency of class definitions, and for computing the logical consequences of the knowledge represented in the schema. Finally, the inherent complexity of reasoning in $\mathcal{CAR}$ is investigated, and the complexity of our inferencing technique is studied, depending on various assumptions on the schema.

1 Introduction

Many recent research efforts have been devoted to the definition of object-oriented database models. These are data models based on the notions of object, class, attribute, and method, where classes and attributes are used to describe the structural aspects of objects, while methods are used to represent their dynamic aspects. The analysis developed in this paper concentrates on the structural part of object-oriented data models, and starts with the observation that most of these models do not support several important features that would allow one to express relevant knowledge about the classes of a schema. In particular, there is no data model supporting simultaneously the inverse of the functions represented by attributes, the union, the intersection and the complement of classes, the possibility of using nonbinary relations, and the possibility of expressing cardinality constraints on attributes and relations.

In this paper, we address the problem of adding all these constructs to an object-oriented data model, with the goal of devising suitable reasoning techniques (for subtyping and consistency checking) on schemas.

The main results of our work can be summarized as follows: First, a new data model is defined, called $\mathcal{CAR}$ (for <u>C</u>lasses, <u>A</u>ttributes, <u>R</u>elations), supporting inheritance, disjointness between classes, union of classes, inverse attributes, n-ary relations, and cardinality constraints. Second, a technique is presented for both checking the consistency of a class definition, and for computing the logical consequences of the knowledge represented in the schema. Third, the inherent complexity of reasoning in $\mathcal{CAR}$ is investigated, and the complexity of our reasoning technique is studied, depending on various assumptions on the schema.

With regard to the first aspect, our model can be seen as an extension of a basic core which is present in most of the object-oriented data models proposed in the literature. Such core usually includes classes, isa-relationships between classes as the basis for representing inheritance, and attributes as a means to establish the properties of classes (see [Kim90]). First of all, we add to the basic core the possibility of referring to the complement of a class, in the spirit of [AP86, DL93], and the possibility of denoting the union and the intersection of classes (see [LR89, AK89]). Then, we introduce both the inverse of an attribute, which is rarely considered in the object-oriented framework, and the notion of n-ary relations, similarly to [AK89, AGO91]. Finally, we make available the use of cardinality constraints, which impose restrictions on the number of links of a certain type (attribute or relation) involving every instance of a given class. Note that, although cardinality constraints appear in various forms in most conceptual and semantic database models, they are rarely present in the object-oriented setting. It is worth noting that the use of cardinality constraints allows us to represent several forms of existence and functional dependencies, and is very common in structured knowledge representation languages (semantic networks and frame-based languages), as pointed out, for example, in [FK85].

With regard to the second aspect, reasoning about a schema comprising all types of constructs mentioned above appears to be significantly harder than reasoning about the various constructs separately. It is clear, for example, that the interaction between isa-relationships and cardinality constraints may cause a database schema to exhibit undesirable properties. In particular, it may happen that there exists a class in the schema that is necessarily empty (i.e. has no instances) in all finite database states. It follows that we need to develop suitable techniques for checking the satisfiability of a class in a schema, and, more generally, for checking if a certain property is a logical consequence of the schema. The interaction between cardinality constraints and the other modeling constructs of our model makes the nature of the problem different with respect to the context of relational theory (see [CK86, GM84]), conceptual and semantic data models [LS83, Tha92, LN90, CL94], or knowledge representation languages [DLNN91, BDS93, Neb88]. Indeed, the work on relational databases rarely considered cardinality constraints of the form allowed in $\mathcal{CAR}$, whereas the work on conceptual and semantic data modeling usually does not take into account complex class expressions (e.g. union and complement of classes). Finally, the knowledge representation community does not restrict the reasoning process to finite structures, as done in databases in general, and in $\mathcal{CAR}$ in particular.

For this reason, we extend and improve the technique proposed in [CL94] in order to deal with the constructs of the $\mathcal{CAR}$ data model. In the method we propose, the reasoning process is split into two phases, where the first one deals with the logical information regarding the mutual relationships between classes, attributes and relationships in the schema, and the second one is based on the idea of representing the cardinality constraints in terms of a special system of linear disequations. We show that the technique can be turned into a correct and terminating algorithm for checking both class satisfiability and logical implication, thus proving that the reasoning problem for the $\mathcal{CAR}$ data model is decidable.

With respect to the third aspect, we study the computational complexity of the problem of reasoning on a $\mathcal{CAR}$ schema, showing EXPTIME-hardness in the general case, and singling out meaningful cases where our method can be implemented efficiently.

The paper is organized as follows. In Section 2, we define the $\mathcal{CAR}$ data model. In Section 3, we describe our technique for reasoning on a $\mathcal{CAR}$ schema in order to check consistency of definitions and to compute logical consequences. In Section 4, we analyze the computational complexity of the reasoning problems and techniques, discussing upper and lower bounds in the general case, as well as efficiency in special cases.

```
class Person
    attributes name: String;
               date_of_birth: String
endclass

class Professor
    isa Person
    attributes teaches: Course
endclass

class Student
    isa Person
    attributes student_id: String
endclass

class Grad_Student
    isa Student
endclass

class Course
    attributes taught_by: Professor
endclass

class Adv_Course
    isa Course
endclass

class Enrollment
    attributes enrolls: Student;
               enrolled_in: Course
endclass
```

Figure 1: An object-oriented schema

2 The $\mathcal{CAR}$ data model

Generally speaking, object-oriented data models are based on the notions of object, class, and attribute. A class is an abstraction for a set of objects with common characteristics. An attribute is a property of a class expressed in terms of another class. The inheritance of properties from one class to another can be realized by establishing inclusion between classes. Figure 1 shows an example of an object-oriented schema modeling part of the reality at an university, in which both basic and advanced courses are offered. Professors and students are persons, and therefore inherit the attributes `name` and `date_of_birth`. Professors teach courses, and the enrollment of students in courses is represented by the class `Enrollment`.

The expressivity provided by this basic core of constructs is usually not sufficient to model real world situations properly. The $\mathcal{CAR}$ data model we propose extends the basic core by offering the following possibilities:

- to specify more complex relations that exist between classes, such as disjointness or the fact that one class is a subset of the union of other classes;
- to explicitly refer to inverse attributes;
- to use (n-ary) relations with complex properties;
- to express cardinality constraints both for attributes and for the participation of objects in relations.

In the following subsection we discuss and motivate each of the above additions to the basic core of the data model, and in the rest of the section we define syntax and semantics of $\mathcal{CAR}$.

2.1 Motivations

In order to permit the inheritance of properties, object-oriented models allow to specify inclusion between classes. To model correctly the reality, however, it is often necessary to express more complex relations that hold between classes. This can be accomplished by allowing the specification of generic intersections and unions of classes and their complements, both as superclasses and as attribute domains. Referring to the schema of Figure 1, we may want, for example, to express that the classes `Person` and `Course` are disjoint, or that students cannot be professors. A more precise specification should also include the fact that courses can be taught by either professors or graduate students. The following additions to the basic schema take this into account.

```
class Student
    isa Person ∧ ¬Professor
    attributes student_id: String
endclass

class Course
    isa ¬Person
    attributes taught_by: Professor ∨ Grad_Student;
endclass
```

Also, most object-oriented models to not capture the fact that an attribute of a class may be the inverse of another attribute. The two attributes are actually treated as distinct, and it is not possible to express their mutual relationship in the schema. In our running example this is the case with `teaches` and `taught_by`. One feature of the $\mathcal{CAR}$ data model is to allow one to specify that an attribute is actually the inverse of the function represented by another attribute. The following example illustrates this for the attributes of the class `Course`.

```
class Course
    isa ¬Person
    attributes (inv teaches): Professor ∨ Grad_Student;
               (inv enrolled_in): Enrollment
endclass
```

When representing a real world situation it may happen that certain classes have been introduced in order to represent relations that hold between other classes that appear in the schema. This happens for example with the class `Enrollment`, which represents a relationship between courses and students. In such cases it is more appropriate to include directly in the model the possibility to define relations. A relation over a set of predefined roles, where each role is a symbol identifying one component of the relation, is an abstraction for a set of tuples, each one mapping every role to a single object. The following modification to our schema shows the substitution of the class `Enrollment` with a relation. It is also important to allow to express more or less complex conditions about the participation of classes in the relationship. The constraints imposed on the relation `Enrollment` force it to hold just between students and courses, and moreover they ensure that only graduate students enroll in advanced courses. We show also the use of a ternary relation, in this case `Exam`, between students, professors and courses.

```
relation Enrollment(enrolled_in,enrolls)
  constraints
    (enrolled_in:Course);
    (enrolls:Student);
    (enrolled_in:¬Adv_Course) ∨ (enrolls:Grad_Student)
endrelation

relation Exam(of,by,in)
  constraints
    (of:Student);
    (by:Professor);
    (in:Course)
endrelation
```

Another important modeling capability is provided by cardinality constraints, which generalize both existence and functional dependencies. They allow one to express constraints on the number of connections that instances of a class may have, either through attributes or through roles of relations. Figure 2 shows the final specification we obtain when adding all of the modifications discussed above, together with cardinality constraints, to the original schema of Figure 1. The precise syntax we use and the underlying semantics are explained in detail in the next subsections. The cardinality constraints associated with the attribute `taught_by` specify that each course is taught by exactly one person (which is either a professor or a graduate student). The cardinality constraints about the participation of courses in the relationship `Enrollment` are specified in the participates in part of the definition of `Course`, and they express that each course enrolls between 5 and 100 students. An advanced course must satisfy the stronger constraint to enroll at most 20 students. Professors, must teach one or two courses. Note that to express such a constraint properly, it is necessary to use a cardinality

```
class Person
    attributes name : (1, 1) String;
               date_of_birth : (1, 1) String
endclass

class Professor
    isa Person
    attributes (inv taught_by) : (1, 2) Course
endclass

class Student
    isa Person ∧ ¬Professor
    attributes student_id : (1, 1) String
    participates in Enrollment[enrolls] : (1, 6)
endclass

class Grad_Student
    isa Student
    attributes (inv taught_by) : (0, 1) Course
    participates in Enrollment[enrolls] : (2, 3)
endclass

class Course
    attributes taught_by : (1, 1) Professor ∨ Grad_Student
    participates in Enrollment[enrolled_in] : (5, 100)
endclass

class Adv_Course
    isa Course
    attributes taught_by : (1, 1) Professor
    participates in Enrollment[enrolled_in] : (5, 20)
endclass

relation Enrollment(enrolled_in,enrolls)
  constraints
    (enrolled_in : Course);
    (enrolls : Student);
    (enrolled_in : ¬Adv_Course) ∨ (enrolls : Grad_Student)
endrelation

relation Exam(of,by,in)
  constraints
    (of : Student);
    (by : Professor);
    (in : Course)
endrelation
```

Figure 2: A $\mathcal{CAR}$ schema

constraint on the inverse of the attribute taught_by. Each student must be enrolled in at least one and at most 6 courses, and such cardinality constraints are again expressed in the participates in part of the class definition. For graduate students these cardinality constraints are refined, and, finally, the constraints in the attributes part force each graduate student to teach at most one course.

2.2 Syntax

A $\mathcal{CAR}$ *schema* $\mathcal{S}$ is a collection of class and relation definitions over an alphabet $\mathcal{B}$, which is a set of symbols partitioned into the set $\mathcal{C}$ of *class* symbols, the set $\mathcal{A}$ of *attribute* symbols, the set $\mathcal{R}$ of *relation* symbols, and the set $\mathcal{U}$ of *role* symbols. Elements of these sets are denoted respectively with C, A, R and U with possible subscripts. In the following we make use of the notions of class-literal, class-clause, and class-formula, for which we use respectively L, γ and F with possible subscripts as meta-symbols. A *class-literal* is either a class symbol C or an expression of the form $\neg C$. A *class-clause* is an expression of the form $L_1 \vee \cdots \vee L_m$ (where each L_i is a class-literal), and a *class-formula* is an expression of the form $\gamma_1 \wedge \cdots \wedge \gamma_n$ (where each γ_j is a class-clause).

A *class definition* is given in terms of a set of properties. There are three types of properties, namely, isa, attributes, and relation participation.

The form of a class definition is:

$$
\begin{array}{l}
\underline{\text{class}}\ C \\
\quad \underline{\text{isa}}\ F \\
\quad \underline{\text{attributes}}\ att_1 : (u_1, v_1)\, F_1; \\
\qquad\qquad\quad \vdots \\
\qquad\qquad\quad att_p : (u_p, v_p)\, F_p \\
\quad \underline{\text{participates in}}\ R_1[U_1] : (x_1, y_1); \\
\qquad\qquad\qquad\quad \vdots \\
\qquad\qquad\qquad\quad R_q[U_q] : (x_q, y_q) \\
\underline{\text{endclass}};
\end{array}
$$

where

- C is the class to be defined;
- each att_i is either an attribute symbol $A_i \in \mathcal{A}$, or an expression of the form (inv A_i); we assume that each att_i appears at most once in each class definition;
- u_i, x_i are nonnegative integers, and v_i, y_i are either nonnegative integers or the special value ∞.

Let us discuss the informal meaning of a class definition. The isa part of the definition of C contains information about the logical relationship that exists between the set of instances of C and the sets of instances of other classes that appear in the schema. Intuitively, isa F in the definition of C means that, in every database state, every instance of C is also an instance of the class-formula F. Given a set of objects and an assignment of elements of this set to each class symbol, the instances of a class-formula are determined inductively in the intuitive way. For example, an instance of a class-clause $L_1 \vee \cdots \vee L_m$ is simply an instance of the class obtained as union of the instances of each L_i.

The attributes part of the definition of C defines the attributes of C, constraining both their type and the allowed cardinalities. In particular, the attribute specification $att_i : (u_i, v_i)\, F_i$ specifies that every instance $\tilde{c}$ of C is related to instances of F_i by means of the function represented by att_i, and, moreover, there are at least u_i and at most v_i instances of F_i associated to

$\tilde{c}$ by att_i. In the case where att_i is of the form ($\underline{\text{inv}}\ A_i$), the function represented by att_i is the inverse of the function represented by A_i.

The participates in part of the definition of C specifies the cardinality constraints for the participation in relations that the instances of C must satisfy. More precisely, the relation participation specification $R_i[U_i] : (x_i, y_i)$ constraints every instance $\tilde{c}$ of C to participate in at least x_i and at most y_i tuples of R_i in the role U_i.

A *relation definition* specifies both the set of roles associated to the relation and the constraints that hold for the classes participating in it. We make use of the notions of *role-literal*, which is an expression of the form $(U : F)$, and *role-clause* (for which we use the meta-symbol ρ with possible subscripts), which has the form $(U_1 : F_1) \vee \cdots \vee (U_s : F_s)$. We may assume without loss of generality that the role symbols $U_1, \ldots, U_s$ in a role-clause are pairwise distinct.

The form of a relation definition is:

$$\begin{array}{l} \underline{\text{relation}}\ R(U_1, \ldots, U_K) \\ \quad \underline{\text{constraints}}\ \rho_1; \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad \rho_t \\ \underline{\text{endrelation}}; \end{array}$$

where

- R is the relation to be defined;
- $U_1, \ldots, U_K$ are pairwise distinct role symbols;
- all role symbols that appear in $\rho_1, \ldots, \rho_t$ are contained in $\{U_1, \ldots, U_K\}$.

Intuitively, the definition of R specifies that $U_1, \ldots, U_K$ are the roles of R (the set of such roles is denoted by $rol(R)$); moreover, each role-clause $(U_{k_1} : F_1) \vee \cdots \vee (U_{k_s} : F_s)$ in the constraints part of the definition specifies that for every database state and for every tuple $\tilde{r}$ that is an instance of R, there is at least one role U_{k_i} such that the value associated with the U_{k_i}-component of $\tilde{r}$ is an instance of F_i.

2.3 Formal semantics and reasoning

We specify the formal semantics of a $\mathcal{CAR}$ schema as follows. An *interpretation* (corresponding to a database state) $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a $\mathcal{CAR}$ schema $\mathcal{S}$ over an alphabet $\mathcal{B}$, consists of a nonempty *finite* set $\Delta^{\mathcal{I}}$ (the *universe* of $\mathcal{I}$), and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) that maps every class in $\mathcal{B}$ to a subset of $\Delta^{\mathcal{I}}$, every attribute to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every relation to a set of labeled tuples over $\Delta^{\mathcal{I}}$. In particular, if R is a relation such that $rol(R) = \{U_1, \ldots, U_K\}$, then $R^{\mathcal{I}}$ is a set of labeled tuples of the form $\langle U_1 : \tilde{c}_1, \ldots, U_K : \tilde{c}_K \rangle$, where $\tilde{c}_1, \ldots, \tilde{c}_K \in \Delta^{\mathcal{I}}$. A *labeled tuple* over a generic set $\mathcal{D}$ is a function from a subset of $\mathcal{U}$ to $\mathcal{D}$, and is denoted with $\langle U_1 : d_1, \ldots, U_k : d_k \rangle$. We write $t[U]$ to denote the value associated with the U-component of the labeled tuple t.

Given an interpretation $\mathcal{I}$, the extension of class-literals, class-clauses and class-formulae in $\mathcal{I}$ is:

$$\begin{array}{rcl} (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\ (L_1 \vee \cdots \vee L_m)^{\mathcal{I}} & = & L_1^{\mathcal{I}} \cup \cdots \cup L_m^{\mathcal{I}} \\ (\gamma_1 \wedge \cdots \wedge \gamma_n)^{\mathcal{I}} & = & \gamma_1^{\mathcal{I}} \cap \cdots \cap \gamma_n^{\mathcal{I}}. \end{array}$$

Also, the extension of the inverse of an attribute A is:

$$(\underline{\text{inv}}\ A)^{\mathcal{I}} = \left\{ (\tilde{a}, \tilde{b}) \;\middle|\; (\tilde{b}, \tilde{a}) \in A^{\mathcal{I}} \right\}.$$

An interpretation $\mathcal{I}$ is said to be a *model* (corresponding to a legal database state) of $\mathcal{S}$ if it satisfies every definition in $\mathcal{S}$, where $\mathcal{I}$ satisfies a definition δ if the following conditions hold:

- if δ is a class definition (say of class C), then
  - for the class-formula F in the isa part of δ, it holds that $C^{\mathcal{I}} \subseteq F^{\mathcal{I}}$;
  - for every $att_i : (u, v)\ F_i$ in the attributes part of δ, for every $\tilde{c} \in C^{\mathcal{I}}$, for every $(\tilde{c}, \tilde{c}') \in att_i^{\mathcal{I}}$, it holds that $\tilde{c}' \in F_i^{\mathcal{I}}$, and there are least u and at most v pairs $(\tilde{c}, \tilde{d})$ in $att_i^{\mathcal{I}}$;
  - for every $R[U] : (x, y)$ in the participates in part of δ, for every $\tilde{c} \in C^{\mathcal{I}}$, it holds that there at least x and at most y tuples $\tilde{r}$ in $R^{\mathcal{I}}$ such that $\tilde{c} = \tilde{r}[U]$;
- if δ is a relation definition (say of $R(U_1, \ldots, U_K)$), then for every labeled tuple $\tilde{r} \in R^{\mathcal{I}}$, and for every role-clause $(U_{k_1} : F_1) \vee \cdots \vee (U_{k_s} : F_s)$ in the constraints part of δ, there exists at least one role-literal $(U_{k_i} : F_i)$ such that $\tilde{r}[U_{k_i}] \in F_i^{\mathcal{I}}$.

The combination of constructs of the $\mathcal{CAR}$ data model makes this model powerful enough to capture most of the object-oriented and semantic data models presented in the literature. In particular, it is possible to show that the data schemas expressed in object-oriented models encompassing the notion of type (e.g. record and set, as in [AK89]) can be reformulated in terms of the $\mathcal{CAR}$ data model. It is worth noting that IRIS [LV87] is one example of data model that is more powerful than $\mathcal{CAR}$ in the specification of cardinality constraints. Indeed, IRIS allows one to impose cardinality constraints holding between a class and a projection of a relation, while $\mathcal{CAR}$ does not provide any means for referring to projections of relations. However, observe that no complete inferencing mechanism is known for IRIS, while we present in the next section a sound and complete reasoning method for $\mathcal{CAR}$.

It is easy to see that every $\mathcal{CAR}$ schema is satisfied by any interpretation that assigns the empty set of instances to every class, every relationship and every attribute. It may happen, however, that cardinality constraints, isa relationships or their interaction force a class to be empty in every model. This observation leads us to introduce the concept of *class satisfiability*, intended to capture the intuition that we should be able to populate a class in a schema without violating any of the constraints represented in the schema. In particular, a class C is said to be *satisfiable* in a $\mathcal{CAR}$ schema $\mathcal{S}$, if $\mathcal{S}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}}$ is nonempty. A $\mathcal{CAR}$ schema $\mathcal{S}$ *logically implies* a definition δ if every model of $\mathcal{S}$ satisfies δ. Class satisfiability and logical implication are the basis for reasoning about a schema, and the solution of these problems has many applications, like schema validation, inheritance computation, type checking and type inference.

3 Reasoning in $\mathcal{CAR}$

In this section we present a method for verifying the satisfiability of a class in a $\mathcal{CAR}$ schema. This method can also be extended to solve the logical implication problem in the $\mathcal{CAR}$ model, but, due to space limitations, we do not deal with this aspect in the present paper.

The method for class satisfiability extends the technique proposed in [CL94] in order to cope with the higher expressivity of the $\mathcal{CAR}$ data model. Moreover, the class satisfiability algorithm is divided into two separate phases: In the first one we neglect all cardinality constraints and take only into account the knowledge that derives from the other aspects of the schema. In the second one we consider the cardinality constraints present in the schema, and we use them to build a system of linear disequations whose solutions are strictly related to the models of the schema. The way in which we perform this second phase represents an improvement over the algorithm proposed in [CL94], and works in worst case deterministic exponential time (compared to the double exponential time algorithm suggested in [CL94]).

The unknowns in the system $\Psi_{\mathcal{S}}$ derived from a $\mathcal{CAR}$ schema $\mathcal{S}$ are intended to represent the number of instances of each class, each relation and each attribute in a possible model of $\mathcal{S}$, while the disequations take into account the constraints on the number of instances deriving from the cardinality constraints in $\mathcal{S}$. Unfortunately, because of classes that may have common instances, it is not possible to adopt the most natural approach, which would be to use one unknown for each class, attribute and relation, as done in [LN90] for a simple data model that assumes that all classes are pairwise disjoint. We will overcome this problem by introducing the notion of expansion of a $\mathcal{CAR}$ schema.

Subsection 3.1 introduces such a notion, while subsection 3.2 describes how to derive a system of linear disequations with the desired properties, thus setting up a method for deciding the satisfiability of a class in a $\mathcal{CAR}$ schema.

3.1 Expansion of a $\mathcal{CAR}$ schema

In order to define the expansion of a $\mathcal{CAR}$ schema $\mathcal{S}$, we introduce the notions of compound class, compound attribute and compound relation (relative to $\mathcal{S}$).

A *compound class* $\bar{C}$ is a subset of $\mathcal{C}$. Intuitively it represents those objects that are instances of all classes in $\bar{C}$ and are not instances of the classes in $\mathcal{C} \setminus \bar{C}$. Each compound class $\bar{C}$ induces a truth assignment $\Phi_{\bar{C}}$ for the classes in $\mathcal{C}$: the value assigned by $\Phi_{\bar{C}}$ to a class C is *true* if $C \in \bar{C}$ and is $false$ otherwise. This truth assignment can be extended in the obvious way to class-literals, class-clauses and class-formulae. Given a class-formula F, if $\Phi_{\bar{C}}(F) = true$ then we say that $\bar{C}$ *realizes* F. A compound class $\bar{C}$ is said to be *consistent* (with respect to $\mathcal{S}$), if for every class C in $\bar{C}$, $\bar{C}$ realizes F_C, where F_C is the class-formula in the isa part of the definition of C.

For each attribute $A \in \mathcal{A}$, in the expansion of $\mathcal{S}$ we represent explicitly the association with all possible pairs of compound classes. This can be accomplished by defining a *compound attribute corresponding to* A as an indexed pair $\langle \bar{C}_1, \bar{C}_2 \rangle_A$, where $\bar{C}_1$ and $\bar{C}_2$ are compound classes. A compound attribute $\bar{A} = \langle \bar{C}_1, \bar{C}_2 \rangle_A$ is said to be *consistent* (with respect to $\mathcal{S}$) if

- $\bar{C}_1$ and $\bar{C}_2$ are consistent;
- for every class $C \in \bar{C}_1$, if $A : (u,v)\, F$ is in the attributes part of the definition of C, then F is realized by $\bar{C}_2$;
- for every class $C \in \bar{C}_2$, if (inv A) : $(u,v)\, F$ is in the attributes part of the definition of C, then F is realized by $\bar{C}_1$.

With $\bar{\mathcal{A}}(A)$ we denote the set of all consistent compound attributes corresponding to attribute A.

Similarly, we define compound relations, which represent explicitly for each role the association to a compound class. A *compound relation* is a labeled tuple over the set of compound classes. In particular, if R is a relation with $rol(R) = \{U_1, \ldots, U_K\}$, a *compound relation corresponding to* R is a labeled tuple of the form $\langle U_1 \colon \bar{C}_1, \ldots, U_K \colon \bar{C}_K \rangle_R$, where $\bar{C}_1, \ldots, \bar{C}_K$ are compound classes. It is said to be *consistent* if all compound classes associated to its roles are consistent, and if for each role-clause $(U_{k_1} \colon F_1) \vee \cdots \vee (U_{k_s} \colon F_s)$ in the constraints part of the definition of R, for at least one $i \in 1..s$, the class-formula F_i is realized by $\bar{C}_{k_i}$. With $\bar{\mathcal{R}}(R)$ we denote the set of all consistent compound relations $\langle U_1 \colon \bar{C}_1, \ldots, U_K \colon \bar{C}_K \rangle_R$ corresponding to relation R.

It is easy to see that whether a compound class, compound attribute or compound relation is consistent can be checked in polynomial time with respect to its size and the size of the schema.

Definition 3.1 *The* expansion $\bar{\mathcal{S}}$ *of a* $\mathcal{CAR}$ *schema* $\mathcal{S}$ *is constituted by:*

- *the set of all consistent compound classes, denoted with* $\bar{\mathcal{C}}$*;*
- *the set of all consistent compound attributes, denoted with* $\bar{\mathcal{A}}$*, obtained as the union of all* $\bar{\mathcal{A}}(A)$*, where* $A \in \mathcal{A}$*;*
- *the set of all consistent compound relations, denoted with* $\bar{\mathcal{R}}$*, obtained as the union of all* $\bar{\mathcal{R}}(R)$*, where* $R \in \mathcal{R}$*;*
- *the set* $\mathcal{N}_{att}$ *of cardinality constraints between compound classes and attributes obtained as follows: for each attribute or inverse attribute att and for each consistent compound class* $\bar{C}$*, if for some class* $C \in \bar{C}$ *and some u (nonnegative integer) and v (nonnegative integer or* ∞*),* $att : (u, v)\, F$ *is in the* <u>*attributes*</u> *part of the definition of C, then* $\bar{C} \Rightarrow att : (u_{max}, v_{min})$ *is in* $\mathcal{N}_{att}$*, where*

$$\begin{aligned} u_{max} &= max\{u \mid \exists C \in \bar{C} \text{ such that } att:(u,v)\,F \\ &\qquad \text{is in the definition of } C\}, \text{ and} \\ v_{min} &= min\{v \mid \exists C \in \bar{C} \text{ such that } att:(u,v)\,F \\ &\qquad \text{is in the definition of } C\}; \end{aligned}$$

- *the set* $\mathcal{N}_{rel}$ *of cardinality constraints between compound classes and relations obtained as follows: for each relation R with* $rol(R) = \{U_1, \ldots, U_K\}$*, for* $k \in 1..K$*, and for each consistent compound class* $\bar{C}$*, if for some class* $C \in \bar{C}$ *and some x (nonnegative integer) and y (nonnegative integer or* ∞*),* $R[U] : (x, y)$ *is in the* <u>*participates in*</u> *part of the definition of C, then* $\bar{C} \Rightarrow R[U] : (x_{max}, y_{min})$ *is in* $\mathcal{N}_{rel}$*, where*

$$\begin{aligned} x_{max} &= max\{x \mid \exists C \in \bar{C} \text{ such that } R[U]:(x,y) \\ &\qquad \text{is in the definition of } C\}, \text{ and} \\ y_{min} &= min\{y \mid \exists C \in \bar{C} \text{ such that } R[U]:(x,y) \\ &\qquad \text{is in the definition of } C\}. \end{aligned}$$

In order to capture the intuitions given above, we specify the formal semantics of compound classes, compound attributes and compound relations by extending interpretations to the expansion of a $\mathcal{CAR}$ schema. Let $\mathcal{S}$ be a $\mathcal{CAR}$ schema, $\mathcal{I}$ an interpretation of $\mathcal{S}$ and $\bar{\mathcal{S}}$ the expansion of $\mathcal{S}$.

If $\bar{C}$ is a compound class relative to $\mathcal{S}$, then its extension $\bar{C}^{\mathcal{I}}$ is defined as follows:

$$\bar{C}^{\mathcal{I}} = \left\{ \tilde{c} \in \Delta^{\mathcal{I}} \;\middle|\; \begin{array}{l} (\forall C \in \bar{C}. \tilde{c} \in C^{\mathcal{I}}) \;\wedge \\ (\forall C \in \mathcal{C} \setminus \bar{C}. \tilde{c} \notin C^{\mathcal{I}}) \end{array} \right\}.$$

If $\bar{A} = \langle \bar{C}_1, \bar{C}_2 \rangle_A$ is a compound attribute corresponding to an attribute A, then its extension $\bar{A}^{\mathcal{I}}$ is defined as follows:

$$\bar{A}^{\mathcal{I}} = \left\{ (\tilde{c}_1, \tilde{c}_2) \in A^{\mathcal{I}} \;\middle|\; \tilde{c}_i \in \bar{C}_i^{\mathcal{I}},\ i = 1, 2 \right\}.$$

If $\bar{R} = \langle U_1 : \bar{C}_1, \ldots, U_K : \bar{C}_K \rangle_R$ is a compound relation corresponding to a relation R, then its extension $\bar{R}^{\mathcal{I}}$ is defined as follows:

$$\bar{R}^{\mathcal{I}} = \left\{ \langle U_1 : \tilde{c}_1, \ldots, U_K : \tilde{c}_K \rangle \in R^{\mathcal{I}} \;\middle|\; \tilde{c}_i \in \bar{C}_i^{\mathcal{I}},\ i \in 1..K \right\}.$$

Conversely, given the extensions of all compound classes, compound attributes and compound relations of $\bar{\mathcal{S}}$, we can immediately derive the extensions of the classes, attributes and relations of $\mathcal{S}$ as follows:

$$C^{\mathcal{I}} = \bigcup_{\bar{C} \mid C \in \bar{C}} \bar{C}^{\mathcal{I}} \qquad A^{\mathcal{I}} = \bigcup_{\bar{A} \in \bar{\mathcal{A}}(A)} \bar{A}^{\mathcal{I}} \qquad R^{\mathcal{I}} = \bigcup_{\bar{R} \in \bar{\mathcal{R}}(R)} \bar{R}^{\mathcal{I}}$$

Notice that the way we interpret compound classes (attributes, relations) forces their extensions to be pairwise disjoint in all interpretations. This is crucial for deriving from the expansion of the schema a suitable system of disequations, as shown later. However, the price we have to pay for this property is the worst case exponential number of different compound classes (attributes, relations) that form the expansion.

A model of a $\mathcal{CAR}$ schema $\mathcal{S}$ will also be called a model of the expansion $\bar{\mathcal{S}}$ of $\mathcal{S}$. The following lemma states the conditions for an interpretation $\mathcal{I}$ of $\mathcal{S}$ to be a model of $\bar{\mathcal{S}}$.

Lemma 3.2 *An interpretation* $\mathcal{I}$ *of a* $\mathcal{CAR}$ *schema* $\mathcal{S}$ *is a model of the expansion* $\bar{\mathcal{S}}$ *of* $\mathcal{S}$*, if and only if it satisfies the following conditions:*

(A) For each compound class, compound attribute or compound relation $\bar{X}$ *that is not consistent, and therefore not in the expansion, it holds that* $\bar{X}^{\mathcal{I}} = \emptyset$*.*

(B) For each attribute or inverse attribute att, for each consistent compound class $\bar{C}$ *and for each* $\tilde{c} \in \bar{C}^{\mathcal{I}}$*, if* $\bar{C} \Rightarrow att : (u, v)$ *is in* $\mathcal{N}_{att}$*, then it holds that* $u \leq \left|\{(\tilde{c}, \tilde{c}') \in att^{\mathcal{I}}\}\right| \leq v$*.*

(C) For each relation R with $rol(R) = \{U_1, \ldots, U_K\}$*, for* $k \in 1..K$*, for each consistent compound class* $\bar{C}$ *and for each* $\tilde{c} \in \bar{C}^{\mathcal{I}}$*, if* $\bar{C} \Rightarrow R[U_k] : (x, y)$ *is in* $\mathcal{N}_{rel}$*, then it holds that* $x \leq \left|\{\tilde{r} \in R^{\mathcal{I}} \mid \tilde{r}[U_k] = \tilde{c}\}\right| \leq y$*.*

3.2 Characterization of class satisfiability

Lemma 3.2 guarantees that, in order to check the satisfiability of a class C in $\mathcal{S}$ it is sufficient to verify if $\bar{\mathcal{S}}$ admits an interpretation satisfying conditions (A), (B) and (C), and in which the extension of at least one compound class containing C is nonempty. This suggests us to perform the class satisfiability check in two phases:

(1) we construct the expansion $\bar{\mathcal{S}}$ of $\mathcal{S}$;

(2) we derive from $\bar{\mathcal{S}}$ a system $\Psi_{\mathcal{S}}$ of linear disequations and search for particular solutions of $\Psi_{\mathcal{S}}$ which guarantee the satisfiability of C.

We analyze phase (1) in detail in the following section, where we show that it can be performed in worst case deterministic exponential time.

With respect to phase (2), the system is obtained by introducing one unknown $\mathrm{Var}(\bar{X})$ for each consistent compound class, compound attribute or compound relation $\bar{X}$. $\Psi_{\mathcal{S}}$ consists of the following disequations:

- for each unknown $\mathrm{Var}(\bar{X})$ we introduce the disequation
$$\mathrm{Var}(\bar{X}) \geq 0;$$

- for each attribute or inverse attribute *att* and for each consistent compound class $\bar{C}$, if $\bar{C} \Rightarrow att : (u, v) \in \mathcal{N}_{att}$, then we introduce the disequations
$$u \cdot \mathrm{Var}(\bar{C}) \;\; \leq \;\; S(att, \bar{C}) \;\; \leq \;\; v \cdot \mathrm{Var}(\bar{C}),$$
where
$$\begin{aligned} S(A, \bar{C}) &= \sum_{\langle \bar{C}, \bar{C}_2 \rangle_A \in \bar{\mathcal{A}}(A)} \mathrm{Var}(\langle \bar{C}, \bar{C}_2 \rangle_A), \text{ and} \\ S((\underline{\mathrm{inv}}\ A), \bar{C}) &= \sum_{\langle \bar{C}_1, \bar{C} \rangle_A \in \bar{\mathcal{A}}(A)} \mathrm{Var}(\langle \bar{C}_1, \bar{C} \rangle_A); \end{aligned}$$

- for each relation R with $rol(R) = \{U_1, \ldots, U_K\}$, for $k \in 1..K$, and for each consistent compound class $\bar{C}$, if $\bar{C} \Rightarrow R[U_k] : (x, y) \in \mathcal{N}_{rel}$, then we introduce the disequations
$$x \cdot \mathrm{Var}(\bar{C}) \;\; \leq \sum_{\substack{\bar{R} \in \bar{\mathcal{R}}(R) \text{ such} \\ \text{that } \bar{R}[U_k] = \bar{C}}} \mathrm{Var}(\bar{R}) \;\; \leq \;\; y \cdot \mathrm{Var}(\bar{C}).$$

The system $\Psi_{\mathcal{S}}$ of disequations we obtain from the expansion of the $\mathcal{CAR}$ schema $\mathcal{S}$ is linear and admits only nonnegative solutions. The following theorem relates the existence of particular solutions of this system to the satisfiability of a class in $\mathcal{S}$. We call a solution of $\Psi_{\mathcal{S}}$ *acceptable* if for all compound attributes $\bar{A} = \langle \bar{C}_1, \bar{C}_2 \rangle_A$ of $\bar{\mathcal{S}}$, the value assigned to $\mathrm{Var}(\bar{A})$ is 0, whenever the value assigned to $\mathrm{Var}(\bar{C}_1)$ or $\mathrm{Var}(\bar{C}_2)$ is 0, and if for all compound relations $\bar{R} = \langle U_1 : \bar{C}_1, \ldots, U_K : \bar{C}_K \rangle_R$ of $\bar{\mathcal{S}}$, the value assigned to $\mathrm{Var}(\bar{R})$ is 0, whenever the value assigned to $\mathrm{Var}(\bar{C}_k)$ is 0 for some $k \in 1..K$.

We are now ready to prove the main result concerning the satisfiability of a class in a $\mathcal{CAR}$ schema.

Theorem 3.3 *A class C_s of a $\mathcal{CAR}$ schema $\mathcal{S}$ is satisfiable, if and only if*
$$\Psi'_{\mathcal{S}} = \Psi_{\mathcal{S}} \bigcup \left\{ \sum_{\bar{C} \in \mathcal{C} \text{ such that } C_s \in \bar{C}} Var(\bar{C}) \geq 1 \right\}$$
admits an acceptable integer solution.

In order to make use of this result and show that we can effectively decide the satisfiability of a class in a $\mathcal{CAR}$ schema, we have to guarantee that verifying the existence of acceptable integer solutions for a system of disequations is decidable. We show in the next section that this is indeed the case. Summarizing the results concerning the decidability of both points (1) and (2) above, we can conclude this section by stating that class satisfiability in $\mathcal{CAR}$ is decidable.

4 Computational complexity of reasoning

In this section we analyze the computational complexity of class satisfiability in a $\mathcal{CAR}$ schema. In subsection 4.1 we give lower bounds, both for the general case and for restricted cases, while in subsection 4.2 we discuss the complexity of an algorithm based on the decision procedure described above. Subsection 4.3 contains some considerations that allow us to improve the performance of the algorithm, and finally, in subsection 4.4, we discuss meaningful cases where our method works in polynomial time with respect to the size of the schema.

4.1 Lower bounds

The problem of verifying the satisfiability of a class in a $\mathcal{CAR}$ schema is inherently complex. The following theorem shows that the general case is provably intractable, even without fully exploiting the expressivity of the model.

Theorem 4.1 *Deciding if a class is satisfiable in a $\mathcal{CAR}$ schema $\mathcal{S}$ is EXPTIME-hard with respect to the size of $\mathcal{S}$, even if no relation definition appears in $\mathcal{S}$ and each number in $\mathcal{S}$ is either* 0 *or* 1.

Proof (sketch). The proof is based on the reduction of string acceptance by EXPTIME Turing Machines (TM) to class satisfiability in a $\mathcal{CAR}$ schema. Since the TM works in exponential time, both the time instants of the computation and the tape positions of the TM can be encoded with a polynomial number of classes of the schema. Two attributes are used to represent spatial and temporal successors, and their inverses represent the spatial and temporal predecessors. There are additional classes for encoding the input symbols, the states of the TM and the position of the head on the tape. The definitions of the classes account for the input symbols on the tape at time 0 and for the changes induced on the tape and on the state of the TM by the allowed transitions. It is possible to show that the TM accepts the input string if and only if the class corresponding to its accepting state is satisfiable in the schema. □

If we limit ourselves to an even less expressive model, admitting only cardinality constraints and simple isa-relationships, reasoning on the schema is still hard. To show this, we define a *negation-free* schema as one in which for every class and relation definition of the schema, the symbol "¬" does not appear in any class-formula in the definition. Similarly, we define a *union-free* schema as one in which for every class and relation definition, all class-clauses and role-clauses that appear in the definition are constituted by only one class-literal and role-literal, respectively.

Theorem 4.2 *Deciding if a class is satisfiable in a $\mathcal{CAR}$ schema which is both union-free and negation-free, is NP-hard with respect to the size of the schema, even if no relation definition appears in the schema.*

Proof (sketch). The proof is based on the reduction of Intersection Pattern (see [GJ79], problem SP9). It exploits the fact that disjointness between classes can be expressed introducing suitable cardinality constraints. □

4.2 Upper bounds

The method proposed in the previous section splits the process of checking the satisfiability of a class in a $\mathcal{CAR}$ schema in two phases.

Phase (1) requires to determine which of the compound classes, compound attributes and compound relations are consistent. The most trivial way to do this is by enumerating the exponentially many compound classes (attributes, relations) and checking for each one in linear time if it is consistent.

With respect to phase (2), we observe that the construction of the system of disequations from the expansion is immediate. The proof of the following theorem suggests an algorithm for verifying the existence of acceptable integer solutions in polynomial time.

Theorem 4.3 *Checking if the system $\Psi_{\mathcal{S}}$ of disequations corresponding to a $\mathcal{CAR}$ schema $\mathcal{S}$ admits acceptable integer solutions can be done in polynomial time with respect to the size of $\Psi_{\mathcal{S}}$.*

Proof (sketch). The form of $\Psi_{\mathcal{S}}$ guarantees that if it admits a positive solution, then it also admits an integer solution (see [LN90]). Moreover, it has been shown (see for example [Pap81]) that for a system of linear disequations, the existence of an integer solution implies the existence of a solution in which all values assigned to the unknowns of the system are bounded by a number whose binary representation is polynomial in the size of the system. This result can be used to derive suitable disequations that can be added to $\Psi_{\mathcal{S}}$, in order to obtain a new system $\Psi'_{\mathcal{S}}$ with the following properties:

- $\Psi_{\mathcal{S}}$ and $\Psi'_{\mathcal{S}}$ have the same unknowns and their sizes are polynomially correlated;
- all solutions of $\Psi'_{\mathcal{S}}$ are acceptable;
- $\Psi'_{\mathcal{S}}$ admits a solution if and only if $\Psi_{\mathcal{S}}$ admits an acceptable integer solution.

Therefore, verifying the existence of an acceptable integer solution of $\Psi_{\mathcal{S}}$ amounts to verifying the existence of a generic solution of $\Psi'_{\mathcal{S}}$. Using linear programming techniques, this can be done in polynomial time in the size of $\Psi'_{\mathcal{S}}$, which means in polynomial time in the size of $\Psi_{\mathcal{S}}$. □

Note, however, that in the worst case the size of $\Psi_{\mathcal{S}}$ is exponential in the size of $\mathcal{S}$. Summing up, we obtain an algorithm for class satisfiability that works in worst case exponential time.

Theorem 4.4 *Class satisfiability in $\mathcal{CAR}$ can be decided in worst case deterministic exponential time.*

Since phase (2) of the proposed reasoning method can be performed in polynomial time with respect to the size of the system, and since it relies on well established linear programming techniques, we will not analyze it further in detail, and in the rest of the section we concentrate on phase (1) above.

The complexity of constructing the system of disequations depends obviously on the *size* of the expansion of the schema, but also on the *time* needed to construct it, which means to determine which are the consistent compound classes, attributes and relations. The lower bound established in the previous subsection tells us that there will be instances of the problem where the whole process takes exponential time.

The size of the expansion is determined by the number of consistent compound classes, compound attributes and compound relationships: the number $|\bar{\mathcal{C}}|$ of consistent compound classes in general grows exponentially with the number of classes in the schema; the number of compound attributes is simply related to $|\bar{\mathcal{C}}|$; with respect to compound relations, we notice that their number depends on $|\bar{\mathcal{C}}|$, but in general grows also exponentially with the maximum arity of relations.

In practice, the maximum arity of relations is a small number (usually less than or equal to 3), independent of the size of the schema, and we could assume it to be constant. The next theorem ensures that in many significant cases, even from a theoretical point of view, the arity of relations does not constitute a problem. In these cases the exponential increase in the number of compound relations due to this aspect can be avoided by substituting every relation whose arity is K with a new class and K binary relations. The newly introduced classes are pairwise disjoint and disjoint from the other classes in the schema, and therefore each of them gives rise to just one compound class in the resulting expansion.

Theorem 4.5 *Let $\mathcal{S}$ be a $\mathcal{CAR}$ schema in which all role-clauses in the definitions of all nonbinary relations are constituted by just one role-literal. Then $\mathcal{S}$ can be transformed in linear time into a $\mathcal{CAR}$ schema $\mathcal{S}'$ containing only binary relations and such that class satisfiability is preserved.*

In the following we concentrate only on the aspects related to the number of compound classes in the expansion of the schema.

4.3 Strategies for performing phase (1)

In order to propose strategies for optimizing the construction of the expansion, we analyze more in detail the factors that contribute to the complexity of reasoning. We can distinguish between two categories of instances of the problem:

(α) those instances where the number of compound classes in the expansion is necessarily exponential in the number of classes (where "necessarily" means that it cannot be reduced without affecting the result of the class satisfiability check);

(β) those instances corresponding to schemas where the number of compound classes in the expansion is polynomial in the number of classes (or can be made polynomial without affecting the result of the class satisfiability check).

If the schema we have to deal with falls into category (α), any procedure for class satisfiability that is based on the construction of the expansion is deemed to work in exponential time. Therefore, the following considerations are devoted to describe a strategy that, when possible, does better than the trivial method of checking all compound classes for consistency. This strategy may lead to an efficient treatment of the problem, in particular when the schema to be analyzed falls in category (β),

A preliminary analysis shows that there will be instances of the problem where, even if the size of the expansion is polynomial, nevertheless it takes exponential time to discard all inconsistent compound classes, and there will also be instances where both the size of the expansion and the time to construct it can be kept polynomial. Actually, we argue that in most practical situations, this last case is the most likely to occur. We propose a heuristics for optimizing the selection of consistent compound classes, which performs this process in two steps: In the first step, a preselection on the compound classes is performed, using an efficient and possibly incomplete algorithm that allows us to discard a priori as many of them as possible. In the second step each of the remaining compound classes is then checked for consistency, and only the consistent ones are taken into account. Since this test can be performed in linear time for each compound class, the complexity of the second step essentially depends on the number of compound classes that remain to be considered after the preselection.

The preselection is done by considering each pair of classes in turn, and trying to extract from the schema as much information as possible concerning both inclusion and disjointness for the classes in each pair. This information is used on one hand for discarding inconsistent compound classes, and on the other hand for singling out those consistent compound classes that can be ignored without influencing the correctness of satisfiability checking. Of course there is a tradeoff between the need to be efficient in the preselection and the need to obtain from it as much information as possible.

We propose to construct two data structures, one, called *disjointness table*, for storing pairs of classes that are disjoint in every model of the schema, and one, called *inclusion table*, for storing pairs of classes such that the first class of the pair is necessarily included in the second. Basically, the preselection step consists in filling in the entries of the two tables. We have determined two criteria that we can follow to do this:

(a) to consider inclusion or disjointness that logically follows from the isa part of the class definitions in the schema;

(b) to consider inclusion or disjointness that may be assumed without influencing the satisfiability checking.

Regarding criterion (a), the simplest strategy is to fill the entries of the tables simply considering inclusion and disjointness that are explicitly present in the schema. If, for example, the class-formula in the isa part of the definition of a class C_1 contains a class-clause consisting only of the class-literal $\neg C_2$, this allows us immediately to exclude all compound classes containing both C_1 and C_2. A more sophisticated method is to consider inclusion or disjointness of two classes that are deducible from the whole set of isa parts of definitions of classes. However, if we pose no restrictions on the class-formulae in the isa parts of class-definitions, the problem of determining when inclusion or disjointness between classes logically follows from the schema, can easily be shown to be NP-complete. Nevertheless, since we are performing a preselection, it may be sufficient to use an efficient and sound procedure that does not guarantee completeness [Dal92].

With respect to criterion (b), we propose a simple syntactic method that can be applied in order to introduce disjointness between classes without influencing satisfiability of classes in the schema. Let $\mathcal{G}_\mathcal{S}$ be the undirected graph constructed from a schema $\mathcal{S}$ in the following way:

1. for each class C in $\mathcal{S}$, introduce one node n_C in $\mathcal{G}_\mathcal{S}$;
2. for each pair of classes C_1 and C_2 in $\mathcal{S}$, introduce an arc between n_{C_1} and n_{C_2} if at least one of the following conditions is satisfied:
   - C_2 appears positively (i.e. not preceded by the symbol "$\neg$)" in the class-formula in the isa part of the definition of C_1;
   - there is a class C in $\mathcal{S}$ such that both C_1 and C_2 appear positively in some class-formula in the attributes part of the definition of C;
   - there is a relation R in $\mathcal{S}$ such that C_1 appears positively in a class-formula F_1, C_2 appears positively in a class-formula F_2, and F_1 and F_2 are associated to the same role in some role-clauses in the constraints part of the definition of R (note that F_1 and F_2 may coincide);
3. for each pair of classes C_1 and C_2 in $\mathcal{S}$, remove (if present) the arc between n_{C_1} and n_{C_2} if C_1 and C_2 have been determined to be disjoint by applying criterion (a) above.

Theorem 4.6 *Let $\mathcal{S}$ be a $\mathcal{CAR}$ schema and let $\mathcal{G}_\mathcal{S}$ be the graph derived from $\mathcal{S}$ as stated above. Then the $\mathcal{CAR}$ schema $\mathcal{S}'$, obtained from $\mathcal{S}$ by imposing disjointness between each pair of classes that are not connected through a path in $\mathcal{G}_\mathcal{S}$, is satisfiable if and only if $\mathcal{S}$ is so.*

Once we have constructed the disjointness and inclusion tables, the information therein is used to cut down the number of compound classes that have to be checked for consistency. Each entry in the tables allows us to exclude all compound classes that do not satisfy the inclusion or disjointness condition specified by the entry, and therefore excludes three quarters of the compound classes.

4.4 Special Cases

In the case of union-free schemas, the construction of the inclusion and disjointness tables represents an optimal strategy in the following sense: On one hand all possible deductions on the isa parts of class definitions can be computed in polynomial time, and on the other hand, by applying the method referred to in theorem 4.6, we can complete the disjointness table in such a way that the number of disjointness assertions is maximized. It is possible to show that in this case all inconsistent compound classes are determined by using the information contained in the two tables, and, moreover, all possible disjointness assumptions between classes are derived, in the sense that introducing further ones may influence the correctness of class satisfiability checking.

A meaningful case where the number of compound classes can be reduced dramatically, is when the disjointness assertions induce a partition of the classes into a number of clusters, such that classes belonging to different clusters are disjoint. The clusters can easily be determined by constructing a graph with one node for each class and an arc between two nodes if and only if the corresponding classes are not disjoint. Each cluster corresponds to a connected component of this graph. In such a case, all compound classes we have to consider are formed only of classes of the same cluster, and the set of compound classes becomes the union of the sets of compound classes obtained separately for each cluster. If we can ensure that for each cluster this number is polynomial, we obtain a system of disequations whose size is polynomial in the size of the original schema. This is the case, for example, if the size of each cluster is logarithmic in the total number of classes.

Another special case to be considered is that of union-free schemas where the classes are organized in *generalization hierarchies*. Generalization hierarchies are treelike structures representing inclusion, where it is assumed that classes in different groups are pairwise disjoint, and within one group the same holds for all classes at the same depth in the tree [BCN92]. In this case, each group corresponds to one cluster, and for each cluster the number of compound classes equals the number of classes, since it corresponds to the number of paths from the root of the tree to a class. It follows that our method, when applied to schemas of this type, works in polynomial time. This is particularly important, if one considers that most object-oriented data models assume, either implicitly or explicitly, an organization of classes based on generalization hierarchies (see for example how isa-relationships are treated in [AK89]).

5 Conclusions

We have presented the object-oriented data model $\mathcal{CAR}$, supporting several modeling constructs that enhance the expressive power of the structural component of the model but are usually missing in current data models. We believe that the main contributions of our work are the design of the inferencing technique associated with the model, and the characterization of the computational complexity of both the reasoning problems and the technique. Our results show that reasoning in $\mathcal{CAR}$ is decidable by a worst-case exponential time algorithm, and that there are significant cases where our technique exhibits good computational properties. It is worth noticing that one of the reasons of the high complexity of the inference problem is the possibility of refining cardinality constraints for subclasses. Despite the high worst-case complexity, the presence of such constraints does not make the problem undecidable. By exploiting results such as those in [GM84, LV87], one can show that no sound and complete inference methods exist, for a data model obtained by extending $\mathcal{CAR}$ with cardinality constraints imposed on projections of relations. This supports the claim that the modeling constructs of $\mathcal{CAR}$ represent an optimal compromise between expressive power and decidability of schema reasoning.

Acknowledgements

We would like to thank Paolo Nobili for his contribution to the complexity results. This work was partly funded by the ESPRIT BRA Compulog II, and the Italian CNR under Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo, LDR Ibridi.

References

[AGO91] A. Albano, G. Ghelli, and R. Orsini. A relationship mechanism for strongly typed Object-Oriented database programming languages. In *Proc. of VLDB-91*, pages 565–575, Barcelona, 1991.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. of ACM SIGMOD*, pages 159–173, 1989.

[AP86] P. Atzeni and D. S. Parker Jr. Formal properties of net-based knowledge representation schemes. In *Proc. of ICDE-86*, pages 700–706, Los Angeles, 1986.

[BCN92] C. Batini, S. Ceri, and S. B. Navathe. *Conceptual Database Design, an Entity-Relationship Approach*. Benjamin and Cummings Publ. Co., Menlo Park, California, 1992.

[BDS93] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. In *Proc. of IJCAI-93*, pages 704–709, Chambery, France, 1993. Morgan Kaufmann, Los Altos.

[CK86] S. S. Cosmadakis and P. C. Kanellakis. Functional and inclusion dependencies - A graph theoretical approach. In P. C. Kanellakis and F. P. Preparata, editors, *Advances in Computing Research, Vol. 3*, pages 163–184. JAI Press, 1986.

[CL94] Diego Calvanese and Maurizio Lenzerini. On the interaction between ISA and cardinality constraints. In *Proc. of ICDE-94*, pages 204–213, Houston, 1994. IEEE Computer Society Press.

[Dal92] Mukesh Dalal. Tractable instances of some hard deduction problems. In *Proc. of ECAI-92*, pages 354–363, Vienna, 1992. John Wiley & Sons.

[DL93] Giuseppe Di Battista and Maurizio Lenzerini. Deductive entity-relationship modeling. *IEEE Trans. on Knowledge and Data Engineering*, 5(3):439–450, 1993.

[DLNN91] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Proc. of KR-91*, pages 151–162. Morgan Kaufmann, Los Altos, 1991.

[FK85] Richard Fikes and Tom Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, 1985.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness.* W.H. Freeman and Company, San Francisco, 1979.

[GM84] J. Grant and J. Minker. Numerical dependencies. In H. Gallaire, J. Minker, and J.-M. Nicolas, editors, *Advances in Database Theory II.* Plenum Publ. Co., New York, 1984.

[Kim90] Won Kim. *Introduction to Object-Oriented Databases.* The MIT Press, 1990.

[LN90] Maurizio Lenzerini and Paolo Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.

[LR89] C. Lecluse and P. Richard. Modeling complex structures in objects-oriented databases. In *Proc. of PODS-89*, pages 362–369, 1989.

[LS83] Maurizio Lenzerini and Gaetano Santucci. Cardinality constraints in the entity-relationship model. In C. G. Davis and others, editors, *Proc. of ER-83*, pages 529–549. North-Holland Publ. Co., Amsterdam, 1983.

[LV87] Peter Lyngbaek and Victor Vianu. Mapping a semantic database model to the relational model. In *Proc. of ACM SIGMOD*, pages 132–142, 1987.

[Neb88] Bernhard Nebel. Computational complexity of terminological reasoning in BACK. *AIJ*, 34(3):371–383, 1988.

[Pap81] Christos H. Papadimitriou. On the complexity of integer programming. *J. of the ACM*, 28(4):765–768, 1981.

[Tha92] Bernhard Thalheim. Fundamentals of cardinality constraints. In G. Pernoul and A. M. Tjoa, editors, *Proc. of ER-92*, pages 7–23. Springer-Verlag, 1992.