Contents

1 DESCRIPTION LOGICS FOR CONCEPTUAL DATA MODELING

Diego Calvanese, Maurizio Lenzerini, Daniele Nardi

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{calvanese,lenzerini,nardi}@dis.uniroma1.it

Abstract: The article aims at establishing a logical approach to class-based data modeling. After a discussion on class-based formalisms for data modeling, we introduce a family of logics, called Description Logics, which stem from research on Knowledge Representation in Artificial Intelligence. The logics of this family are particularly well suited for specifying data classes and relationships among classes, and are equipped with both formal semantics and inference mechanisms. We demonstrate that several popular data modeling formalisms, including the Entity-Relationship Model, and the most common variants of object-oriented data models, can be expressed in terms of specific logics of the family. For this purpose we use a unifying Description Logic, which incorporates all the features needed for the logical reformulation of the data models used in the various contexts. We also discuss the problem of devising reasoning procedures for the unifying formalism, and show that they provide valuable supports for several important data modeling activities.

1.1 INTRODUCTION

Data modeling is the activity of specifying the structure of the data to be managed within an application. In the last two decades, data modeling has been the subject of a large body of work in several areas, including Databases, Information Systems, Software Engineering, and Knowledge Representation.

In particular, recent approaches to conceptual data modeling advocate the use of abstract formalisms for describing data, mostly based on the notion of class [HK87]. In this paper, we concentrate on such class-based formalisms for data modeling, with the aim of demonstrating that they can be profitably reconstructed within a logical framework. We argue that the reasoning techniques available in the logical framework provide valuable support for the data modeling activity.

Generally speaking, a *class* denotes a subset of the domain of discourse, and a class-based representation formalism allows one to express several kinds of relationships and constraints (e.g. subclass constraints) holding among classes [MM92]. Moreover, class-based formalisms aim at taking advantage of the class structure in order to provide various information, such as whether an element belongs to a class, whether a class is a subclass of another class, and more generally, whether a given constraint holds among a given set of classes.

Two main families of class-based formalisms for data modeling are addressed in this paper. The first one originates in the field of databases and in particular from the work on semantic data models (see e.g. [HK87]). The second one arises from the work on types in programming languages and object-oriented systems (see e.g. [KL89]).

In the past, there have been several attempts to establish relationships among class-based formalisms used in knowledge representation (e.g. semantic networks and frames [Leh92; Sow91]) and the above two families of class-based formalisms. One significant aspect of this work is the identification of reasoning problems, where one can take advantage of techniques for reasoning on hierarchical structures that have been developed in different areas.

The relationship between frame-based languages and types has been addressed in [BHR90; LNS91; Bor92], while in [BS92; PSS92; Bor95; ACS96] frame-based languages are used to enrich the deductive capabilities of data models. The analysis of the above cited works makes it clear that, although a number of steps have been accomplished, several difficulties arise in identifying a common framework, which is expressive enough to capture the essential features of various class-based formalisms, while still providing techniques for the associated reasoning problems. Other formalisms have been recently proposed with the aim of integrating the object-oriented and the logic programming paradigms. A notable example of this effort is F-Logic [KLW95], which provides an elegant framework equipped with a sound and complete resolution based proof procedure. However, the goal of these proposals is to provide a sophisticated environment for computing with objects, rather than a system supporting reasoning over a conceptual specification. For this reason, they cannot be easily compared to the approach proposed here.

In this paper, we make a step towards a unified view of class-based languages by establishing a relationship between semantic and object-oriented data models, rephrasing them in terms of the knowledge representation framework of Description Logics. Specifically, building on the results of [CLN94], we present a class-based representation formalism, of the family of Description Logics [SS91; DLNN91], and show that it is able to capture the most popular data modeling formalisms presently used in Databases and Information System Analysis, providing powerful reasoning techniques.

In Description Logics, structured knowledge is described by means of so called *concepts* and *roles*, which denote unary and binary predicates, respectively. Starting from a set of atomic symbols one can build complex concept and role expressions by applying suitable constructors which characterize a Description Logic. Formally, concepts are interpreted as subsets of a domain and roles as binary relations over that domain, and all constructors are equipped with a precise set-theoretic semantics. The most common constructors include boolean operators on concepts, and quantification over roles. For example, the concept $\mathtt{Person} \sqcap \forall \mathtt{child}.\mathtt{Male}$, denotes the set of individuals that are instances of the concept `Person` and are connected through the role `child` only to instances of the concept `Male`, while the concept $\exists\mathtt{child}$ denotes all individuals that are connected through role `child` to some individual. Further constructors that have been considered important include more general forms of quantification, number restrictions, which allow one to state limits on the number of connections that an individual may have via a certain role, and constructors on roles, such as intersection, concatenation and inverse. A Description Logic knowledge base, expressing the intensional knowledge about the modeled domain, is built by stating inclusion and/or equality assertions between concepts, which have to be satisfied by the models of the knowledge base. The assertions are used to specify necessary and/or necessary and sufficient conditions for individuals to be instances of certain concepts. Reasoning on such knowledge bases includes detecting inconsistencies in the knowledge base itself, determining whether a concept can be populated in a model of the knowledge base, and checking subsumption, i.e. whether all instances of a concept are necessarily also instances of another concept in all models of the knowledge base.

The Description Logic we propose, called $\mathcal{ALCQI}$, features a rich combination of constructors, including qualified number restrictions, inverse roles and inclusion assertions of a general form. We show that these features make $\mathcal{ALCQI}$ powerful enough to provide a unified framework for object-oriented languages and semantic data models. This is done by establishing a precise correspondence with the Entity Relationship model [Che76], and with an object-oriented language in the style of [AK89]. Moreover, we demonstrate that the formalism proposed in this paper provides important features that are currently missing in each family, although their relevance has often been stressed.

Because of the expressive power of $\mathcal{ALCQI}$, the computational complexity of reasoning becomes high, but the relevant reasoning tasks remain nonetheless decidable. We consider this feature very important, because it makes this language an actual knowledge representation and data modeling language and not simply a formal framework for comparing apparently different approaches.

The paper is organized as follows. In the next section we present the Description Logic $\mathcal{ALCQI}$. In Sections 1.3, and 1.4 we deal with semantic data models and object-oriented data models, respectively, showing that their basic features are captured by knowledge bases in $\mathcal{ALCQI}$. Section 1.5 describes how reasoning in $\mathcal{ALCQI}$ supports data modeling. The final section contains some concluding remarks.

1.2 DESCRIPTION LOGICS

The basic elements of Description Logics are *concepts* and *roles*, which denote classes and binary relations, respectively. Arbitrary concept and role expressions (in the following simply called concepts and roles) are formed by starting from two sets of *atomic concepts* and *atomic roles* and applying concept and role *constructors*.

1.2.1 Syntax and Semantics of the Logic $\mathcal{ALCQI}$

We introduce now the Description Logic $\mathcal{ALCQI}$, in which concepts and roles are formed according to the following syntax:

$$\begin{array}{rcl} C, C' & \longrightarrow & A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \\ & & \forall R.C \mid \exists R.C \mid \exists^{\geq n} R.C \mid \exists^{\leq n} R.C \\ R & \longrightarrow & P \mid P^- \end{array}$$

where, A and P, denote atomic concepts and atomic roles respectively, C and R denote arbitrary concepts and roles, and n denotes a positive integer. We also use the following abbreviations to increase readability:

$$\begin{array}{rll} \bot & \text{for} & A \sqcap \neg A \quad \text{where } A \text{ is any atomic concept} \\ \top & \text{for} & A \sqcup \neg A \quad \text{where } A \text{ is any atomic concept} \\ \exists^{=n} R.C & \text{for} & \exists^{\geq n} R.C \sqcap \exists^{\leq n} R.C \\ \exists^{\geq n} R & \text{for} & \exists^{\geq n} R.\top \quad \text{(similarly for } \exists^{\leq n} R \text{ and } \exists^{=n} R) \end{array}$$

Among the constructors used in forming concept expressions we find the basic set operators, namely set complement ($\neg$), union ($\sqcup$), and intersection ($\sqcap$) that are denoted as *negation*, *disjunction*, and *conjunction*, respectively. Description

Logics admit a restricted form of quantification which is realized through so called *quantified role restrictions*, that are composed by a quantifier (existential or universal), a role and a concept expression. Quantified role restrictions allow one to represent the relationships existing between the objects in two classes, and the forms considered in $\mathcal{ALCQI}$ are general enough to capture the most common ways of establishing such relationships. For example, one can characterize the set of objects all of whose children are male as $\forall$`child.Male`, as well as the set of objects that have at least one male child as $\exists$`child.Male`. *Number restrictions* are used to constrain the number of instances that are in a certain relationship. For example, $\exists^{=2}$`child.Male` characterizes the set of parents with exactly two male children. The form used here (called *qualified number restrictions* [HB91]) is a very general one, allowing one to pose restrictions on the number of instances connected through a certain role, counting only those objects that satisfy a certain condition. Observe that the restricted forms of cardinality restrictions where the involved number is equal to 1, express functionality ($\exists^{\leq 1}R$) and existence constraints ($\exists^{\geq 1}R$). Finally, in $\mathcal{ALCQI}$ we have one role constructor, namely *inverse role*, that allows us to denote the inverse of a given relation. One can for example state with $\exists^{\leq 2}\mathtt{child}^-$ that someone has at most two parents, by making use of the inverse of role `child`. It is worth noticing, that in a language without the inverse of roles, in order to express such a constraint one must make use of two distinct roles (e.g. `child` and `parent`) that cannot be put in the proper relation to each other.

From the semantic point of view, concepts are interpreted as subsets of a domain and roles as binary relations over that domain. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a set $\mathcal{A}$ of atomic concepts and a set $\mathcal{P}$ of atomic roles consists of a nonempty *finite* set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) that maps every atomic concept $A \in \mathcal{A}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ (the set of *instances* of A) and every atomic role $P \in \mathcal{P}$ to a subset $P^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ (the set of *instances* of P). The interpretation function can then be extended to arbitrary concepts and roles as follows: ($\sharp S$ denotes the cardinality of the set S)

$$
\begin{array}{rcl}
(\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(C_1 \sqcup C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \forall o'.\, (o,o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \exists o'.\, (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \\
(\exists^{\geq n} R.C)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq n\} \\
(\exists^{\leq n} R.C)^{\mathcal{I}} & = & \{o \in \Delta^{\mathcal{I}} \mid \sharp\{o' \mid (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq n\} \\
(P^-)^{\mathcal{I}} & = & \{(o,o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o',o) \in P^{\mathcal{I}}\}
\end{array}
$$

Notice that $\exists^{\geq 1} R.C$ is equivalent to $\exists R.C$.

1.2.2 Knowledge Bases in $\mathcal{ALCQI}$

Using concept expressions of $\mathcal{ALCQI}$, intensional knowledge about classes and relations can be expressed through a knowledge base. An $\mathcal{ALCQI}$ *knowledge base* is constituted by a finite set of *assertions* of the following form:

$$\begin{array}{rcll} A & \dot{\preceq} & C & \textit{(inclusion assertion)} \\ A & \doteq & C & \textit{(equality assertion)} \end{array}$$

where A is an atomic concept and C is an arbitrary $\mathcal{ALCQI}$ concept expression.

An inclusion assertion $A \dot{\preceq} C$ specifies (by means of C) only *necessary* conditions for an object to be an instance of the concept A, and thus corresponds naturally to the constraints imposed on classes by a schema expressed in a traditional database model. In such models an object can never be inferred to be an instance of a certain class, unless this is explicitly stated. In contrast, an equality assertion specifies both *necessary and sufficient* conditions for the instances of the class, and thus corresponds to the concept of *view* used in databases. Observe, however, that views are usually not considered to be part of the schema, but are built on top of it in order to define how the data present in the schema is used [BDNS94] (see [CDGL95] for a different "view" on this aspect).

We pose no restrictions at all on the form of the assertions in a knowledge base. In particular:

1. Each atomic concept may appear more than once on the left side of an assertion.

2. The assertions may contain *(terminological) cycles*, i.e. the concept in the right part of an assertion may refer (either directly or indirectly) to the atomic concept in the left part of the assertion.

Both these assumptions have a strong impact on the expressiveness of our formalism, although this is paid by an increased computational complexity of reasoning. Making assumption (1) is equivalent to allowing for the use of so called *free inclusion assertions* [BDS93], which have the form $C_1 \dot{\preceq} C_2$, with C_1 and C_2 arbitrary concept expressions. Assumption (2) is seldom made in existing concept-based knowledge representation systems, since terminological cycles increase the computational complexity of reasoning [Baa96], and can be interpreted under different semantics [Neb91; Sch94; DGL94]. From a data modeling perspective it is however unrealistic to assume the absence of cycles.

The semantics of a knowledge base is specified through the notion of satisfaction of assertions. An interpretation $\mathcal{I}$ *satisfies* the inclusion assertion $A \dot{\preceq} C$ if $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$, and it satisfies the equality $A \doteq C$ if $A^{\mathcal{I}} = C^{\mathcal{I}}$. An interpretation is a *model* of a knowledge base if it satisfies all terminological assertions in it.

The fundamental reasoning tasks considered in the context of Description Logics at the intensional level (which is the only one we address here) are *knowledge base satisfiability*, *concept satisfiability*, and *logical implication*. A knowledge base $\mathcal{K}$ is satisfiable if it admits a model; a concept C is satisfiable in $\mathcal{K}$ if $\mathcal{K}$ admits a model in which C has a nonempty interpretation; $\mathcal{K}$ logically implies an inclusion assertion $C_1 \dot{\preceq} C_2$ if in all models of $\mathcal{K}$ the interpretation of C_1 is a subset of the interpretation of C_2.

We would like to remark that we are restricting our attention to finite interpretations (and thus models), which are the only ones of interest in data modeling. The assumption of dealing with finite structures is, however, by no means common in Description Logics, and needs to be taken explicitly into account when devising reasoning procedures [CLN94; Cal96a]. In fact, the constructs present in $\mathcal{ALCQI}$, and in particular functionality, inverse roles and cycles in the knowledge base may interact in such a way that a knowledge base admits no finite model, although it admits one with an infinite domain [CKV90; CLN94]. In other words, $\mathcal{ALCQI}$ lacks the *finite model property*, and reasoning in the finite and in the unrestricted case are different.

1.3 SEMANTIC DATA MODELS

Semantic data models were introduced primarily as formalisms for database schema design. They provide a means to model databases in a much richer way than traditional data models supported by Database Management Systems, and are becoming more and more important because they are adopted in most of the recent database and information system design methodologies and Computer Aided Software Engineering tools.

A rich variety of semantic data models have been introduced with various degrees of expressiveness (see [HK87] for an extensive survey). They provide by means of classes an explicit representation of objects with their attributes and the relationships to other objects. An important feature of semantic data models is the possibility to specify subtype/supertype relationships (also called *is-a*) which provide for the inheritance of properties.

For simplicity of presentation, we concentrate our attention on one of the most widespread semantic data models, which has by now become a standard, extensively used in the design phase of commercial applications, namely the *Entity-Relationship* (ER) model. The ER model was introduced in [Che76], and subsequently several variants and extensions have been proposed, which differ in minor aspects in expressiveness and in notation [Teo89; BCN92; Tha92; Tha93]. In the ER model the domain of interest is modeled by means of an ER schema, usually represented in a graphical notation which is particularly useful for an easy visualization of the data dependencies. However, in the following we introduce a first-order formalization, which includes the most important features present in the different variants and makes it possible to establish a precise correspondence with Description Logics.

1.3.1 The Entity-Relationship Model

The basic elements of the ER model are entities, relationships, and attributes. An *entity set* (or simply *entity*) denotes a set of objects, called its *instances*, that have common properties. Elementary properties are modeled through *attributes*, whose values belong to one of several predefined *domains*, such as `Integer`, `String`, or `Boolean`. Properties that are due to relations to other entities are modeled through the participation of the entity in relationships. A *relationship set* (or simply *relationship*) denotes a set of tuples (also called its instances), each of which represents an association among a different combination of instances of the entities that participate in the relationship. Since each entity can participate in a relationship more than once, the notion of *ER-role* is introduced, which represents such a participation and to which a unique name is assigned. The *arity* of a relationship is the number of its ER-roles. *Cardinality constraints* can be attached to an ER-role in order to restrict the number of times each instance of an entity is allowed to participate via that ER-role in instances of the relationship. Such constraints can be used to specify both existence dependencies and functionality of relations [CK86]. They are often used only in a restricted form, where the minimum cardinality is either 0 or 1 and the maximum cardinality is either 1 or ∞. Cardinality constraints in the form considered here have been introduced already in [Abr74] and subsequently studied in [GM84; LN90; Fer91; YPS94; Tha92; CL94b; CL94a]. Additionally, so called *is-a* relations are used to represent inclusion assertions between entities, and therefore the inheritance of properties from a more general entity to a more specific one. We do not consider *keys* in our formalization, which are essential for a mapping of ER schemas into relational schemas, but lose their relevance when reasoning on a conceptual specification. For a treatment of reasoning on keys in a logic based framework see [BW97; CDGL95]. We define now in a more formal way syntax and semantics of ER schemata.

An *ER schema* $\mathcal{S}$ is constructed starting from pairwise disjoint sets of *entity* symbols, *relationship* symbols (each with an arity), *ER-role* symbols, *attribute* symbols, and *domain* symbols. Each domain symbol D has an associated predefined *basic domain* $D^{\mathcal{B}_D}$, and we assume the basic domains to be pairwise

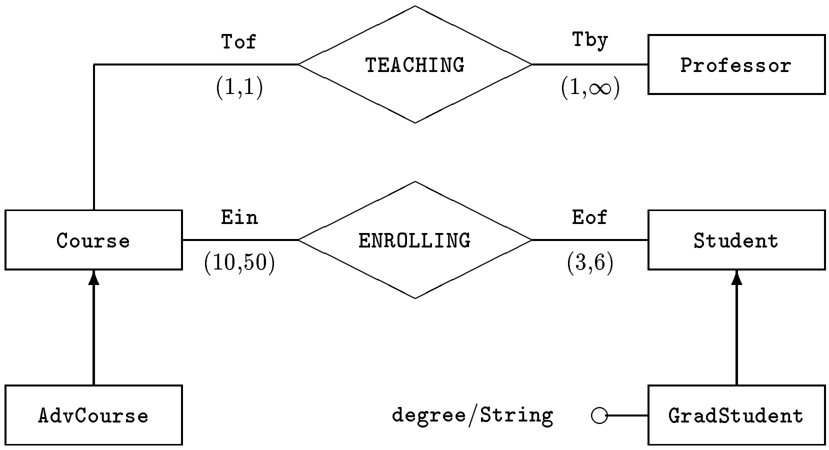

Figure 1.1 An ER schema

disjoint. For each entity symbol a set of attribute symbols is defined, and to each such attribute a unique domain symbol is associated. Each relationship symbol of *arity* k has k associated ER-role symbols, each with an associated entity symbol, and defines a relationship between these entities. We assume that each ER-role symbol belongs to a unique relationship, thus determining also a unique entity. The *cardinality constraints* are represented by two functions $cmin_{\mathcal{S}}$, from ER-role symbols to nonnegative integers, and $cmax_{\mathcal{S}}$, from ER-role symbols to positive integers union the special symbol ∞. *is-a* relations between entities are modeled by means of a binary relation $\preceq_{\mathcal{S}}$. We do not need to make any special assumption on the form of $\preceq_{\mathcal{S}}$, such as requiring that it is acyclic or injective.

In the commonly accepted graphical ER notation, entities are represented as boxes, whereas relationships are represented as diamonds. An attribute is shown as a circle attached to the entity for which it is defined. ER-roles are graphically depicted by connecting the relationship to the participating entities and labeling the connection with the associated cardinality constraints. An is-a relation between two entities is denoted by an arrow from the more specific to the more general entity.

Example 1 Figure 1.1 shows a simple ER schema modeling the situation at an university. The entity `Course` represents courses which enroll students and are taught by professors. Cardinality constraints are used to impose limits on the number of students that may be enrolled in a course (between 10 and 50) and on the number of courses that each student may attend (between 3 and 6), and to express that each course is taught by exactly one professor, who in turn must teach at least one course. The entities `AdvCourse` and `GradStudent` are specializations of `Course` and `Student`, respectively. ■

The semantics of an ER schema can be given by specifying which database states are consistent with the information structure represented by the schema. Formally, a *database state* $\mathcal{B}$ corresponding to an ER schema $\mathcal{S}$ is constituted by a nonempty *finite* set $\Delta^{\mathcal{B}}$, assumed to be disjoint from all basic domains, and a function $\cdot^{\mathcal{B}}$ that maps

- every domain symbol D to the corresponding basic domain $D^{\mathcal{B}_{\mathcal{D}}}$,
- every entity E to a subset $E^{\mathcal{B}}$ of $\Delta^{\mathcal{B}}$,
- every attribute A to a partial function $A^{\mathcal{B}}$ from $\Delta^{\mathcal{B}}$ to the union, for all domains D, of $D^{\mathcal{B}_{\mathcal{D}}}$, and
- every relationship R to a set $R^{\mathcal{B}}$ of labeled tuples over $\Delta^{\mathcal{B}}$.

A *labeled tuple* over a domain $\Delta^{\mathcal{B}}$ is a function from a set of ER-roles to $\Delta^{\mathcal{B}}$. The labeled tuple T that maps ER-role U_i to o_i, for $i \in \{1, \ldots, k\}$, is denoted $[U_1\colon o_1, \ldots, U_k\colon o_k]$. We also write $T[U_i]$ to denote o_i, and call it the *U_i-component of T*. The elements of $E^{\mathcal{B}}$, $A^{\mathcal{B}}$, and $R^{\mathcal{B}}$ are called *instances* of E, A, and R respectively.

A database state is considered acceptable if it satisfies all integrity constraints that are part of the schema. This is captured by the notion of legal database state. A database state $\mathcal{B}$ is *legal for* an ER schema $\mathcal{S}$, if it satisfies the following conditions:

- For each pair of entities E_1, E_2 with $E_1 \preceq_{\mathcal{S}} E_2$, it holds that $E_1^{\mathcal{B}} \subseteq E_2^{\mathcal{B}}$.
- For each entity E, if E has an attribute A with domain D, then for each instance $e \in E^{\mathcal{B}}$, $A^{\mathcal{B}}(e)$ is defined and in $D^{\mathcal{B}_{\mathcal{D}}}$.
- For each relationship R of arity k between entities $E_1, \ldots, E_k$, to which R is connected by means of ER-roles $U_1, \ldots, U_k$ respectively, all instances of R are of the form $[U_1\colon e_1, \ldots, U_k\colon e_k]$, where $e_i \in E_i^{\mathcal{B}}$, for $i \in \{1, \ldots, k\}$.
- For each ER-role U associated to relationship R and entity E, and for each instance e of E, it holds that

$$cmin_{\mathcal{S}}(U) \leq \sharp\{r \in R^{\mathcal{B}} \mid r[U] = e\} \leq cmax_{\mathcal{S}}(U).$$

Notice that an is-a relation is interpreted as set containment between the extensions of the involved entities. In addition, it is worth emphasizing that the definition of database state reflects the usual assumption that database states are finite structures (see also [CKV90]). In fact, the basic domains are not required to be finite, but for each legal database state for a schema, only the finite set of values of attributes of elements of $\Delta^{\mathcal{B}}$ is actually of interest.

1.3.2 Formalizing Entity-Relationship Schemata in Description Logics

We now show that the semantics of the ER model can be captured in Description Logics, by defining a translation ϕ from ER schemata to $\mathcal{ALCQI}$ knowledge bases, and then establishing a correspondence between legal database states and models of the derived knowledge base.

The knowledge base $\phi(\mathcal{S})$ derived from an ER schema $\mathcal{S}$ is defined as follows: It contains an atomic concept $\phi(A)$ for each domain symbol, entity symbol, or relationship symbol A in $\mathcal{S}$, an atomic role $\phi(P)$ for each attribute symbol or ER-role symbol P in $\mathcal{S}$, and the set of assertions of $\phi(\mathcal{S})$ contains the following elements:

- For each pair of entities E_1, E_2 such that $E_1 \preceq_{\mathcal{S}} E_2$, the assertion

$$\phi(E_1) \ \dot{\preceq}\ \phi(E_2).$$

- For each entity E with attributes $A_1, \ldots, A_h$ with domains $D_1, \ldots, D_h$ respectively, the assertion

$$\begin{aligned}\phi(E) \ \dot{\preceq}\ & \forall\phi(A_1).\phi(D_1) \sqcap \cdots \sqcap \forall\phi(A_h).\phi(D_h) \sqcap \\ & \exists^{=1}\phi(A_1) \sqcap \cdots \sqcap \exists^{=1}\phi(A_h).\end{aligned}$$

- For each relationship R of arity k between entities $E_1, \ldots, E_k$, to which R is connected by means of ER-roles $U_1, \ldots, U_k$ respectively, the assertions

$$\begin{aligned}\phi(R) \ &\dot{\preceq}\ \forall\phi(U_1).\phi(E_1) \sqcap \cdots \sqcap \forall\phi(U_k).\phi(E_k) \sqcap \\ & \qquad \exists^{=1}\phi(U_1) \sqcap \cdots \sqcap \exists^{=1}\phi(U_k) \\ \phi(E_i) \ &\dot{\preceq}\ \forall(\phi(U_i))^{-}.\phi(R), \qquad \text{for } i \in \{1, \ldots, k\}.\end{aligned}$$

- For each ER-role U associated to relationship R and entity E,
  - if $m = cmin_{\mathcal{S}}(U) \neq 0$, the assertion

$$\phi(E) \ \dot{\preceq}\ \exists^{\geq m}(\phi(U))^{-}.$$

  - if $n = cmax_{\mathcal{S}}(U) \neq \infty$, the assertion

$$\phi(E) \ \dot{\preceq}\ \exists^{\leq n}(\phi(U))^{-}.$$

$$
\begin{array}{rcl}
\texttt{TEACHING} & \dot{\preceq} & \forall\texttt{Tof.Course} \sqcap \exists^{=1}\texttt{Tof} \sqcap \forall\texttt{Tby.Professor} \sqcap \exists^{=1}\texttt{Tby} \\
\texttt{ENROLLING} & \dot{\preceq} & \forall\texttt{Ein.Course} \sqcap \exists^{=1}\texttt{Ein} \sqcap \forall\texttt{Eof.Student} \sqcap \exists^{=1}\texttt{Eof} \\
\texttt{Course} & \dot{\preceq} & \forall\texttt{Tof}^{-}.\texttt{TEACHING} \sqcap \exists^{=1}\texttt{Tof}^{-} \sqcap \\
 & & \forall\texttt{Ein}^{-}.\texttt{ENROLLING} \sqcap \exists^{\geq 10}\texttt{Ein}^{-} \sqcap \exists^{\leq 50}\texttt{Ein}^{-} \\
\texttt{AdvCourse} & \dot{\preceq} & \texttt{Course} \\
\texttt{Professor} & \dot{\preceq} & \forall\texttt{Tby}^{-}.\texttt{TEACHING} \\
\texttt{Student} & \dot{\preceq} & \forall\texttt{Eof}^{-}.\texttt{ENROLLING} \sqcap \exists^{\geq 3}\texttt{Eof}^{-} \sqcap \exists^{\leq 6}\texttt{Eof}^{-} \\
\texttt{GradStudent} & \dot{\preceq} & \texttt{Student} \sqcap \forall\texttt{degree.String} \sqcap \exists^{=1}\texttt{degree}.
\end{array}
$$

Figure 1.2 The knowledge base corresponding to the ER schema in Figure 1.1

- For each pair of symbols C_1, C_2 such that C_1 is either a relationship or a domain symbol, C_2 is either an entity, a relationship, or a domain symbol, and $C_1 \neq C_2$, the assertion

$$\phi(C_1) \ \dot{\preceq} \ \neg\phi(C_2). \qquad (1.1)$$

Example 1 (cont.) We illustrate the translation on the ER schema of Figure 1.1. The knowledge base that captures its semantics is shown in Figure 1.2, where the disjointness assertions (1.1) are omitted for brevity. ■

Regarding the transformation provided above we observe the following:

- Since an ER schema expresses only necessary conditions for objects of the domain to be instances of entities, the translation makes only use of inclusion assertions, while equality assertions are not necessary.

- Each relationship is *reified*, i.e. is modeled by means of a concept, whose instances represent the tuples of the relationship. The assertions enforce that for each role $\phi(U)$ of the relationship each such instance is connected to exactly one object that represents the U-component of the corresponding tuple.

- Both inverse roles and number restrictions are exploited to capture the semantics of ER schemata. More precisely, functionality of the roles associated to the ER-roles is needed to represent the fact that each instance of a relationship is connected via each ER-role to exactly one instance of the associated entity, while we use number restrictions on the corresponding inverse roles to represent cardinality constraints of the ER schema.

- Even when the ER schema is acyclic, the resulting knowledge base contains cycles.
- By means of the inverse constructor, a binary relationship could be treated in a simpler way by choosing a traversal direction and mapping the relationship directly to a role.

In order to show that the translation preserves the semantics of the ER schema we define a mapping between database states corresponding to the ER schema and finite interpretations of the knowledge base derived from it. Due to reification of relationships, this mapping is however not one-to-one and we first need to characterize those interpretations of the knowledge base that directly correspond to database states.

We say that an interpretation $\mathcal{I}$ of the knowledge base $\phi(\mathcal{S})$ derived from an ER schema $\mathcal{S}$ is *relation-descriptive*, if for every relationship R of $\mathcal{S}$, with ER-roles $U_1, \ldots, U_k$, for every $d, d' \in (\phi(R))^{\mathcal{I}}$, we have that

$$(\bigwedge_{1 \leq i \leq k} (\forall d'' \in \Delta^{\mathcal{I}}.\, (d, d'') \in (\phi(U_i))^{\mathcal{I}} \leftrightarrow (d', d'') \in (\phi(U_i))^{\mathcal{I}})) \rightarrow d = d'.$$

Intuitively, the condition states that there are no two instances of a concept corresponding to a relationship that represent the same tuple. Notice that this condition is implicit in the semantics of the ER model (where the extension of a relationship is a *set* of tuples), while it does not necessarily hold once relationships are reified. It also cannot be imposed in $\mathcal{ALCQI}$ by suitable assertions. However, when reasoning on the knowledge base corresponding to an ER schema, it is sufficient to restrict the attention to relation-descriptive models. Indeed, if a concept expression C of the knowledge base $\phi(\mathcal{S})$ obtained from an ER schema $\mathcal{S}$ is satisfiable in $\phi(\mathcal{S})$, then there is a relation-descriptive model of $\phi(\mathcal{S})$ in which $\mathcal{C}$ has a nonempty extension [Cal96b]. This can be exploited since relation-descriptive models of an $\mathcal{ALCQI}$ knowledge base $\phi(\mathcal{S})$ can be put in correspondence with legal database states for $\mathcal{S}$. More precisely, the correspondence can be established by defining two mappings $\alpha_{\mathcal{S}}$ and $\beta_{\mathcal{S}}$ as follows:

- $\alpha_{\mathcal{S}}$ is a mapping from database states corresponding to $\mathcal{S}$ to relation-descriptive interpretations of $\phi(\mathcal{S})$, with the following properties (let $\mathcal{B}$ be a database state and $\mathcal{I} = \alpha_{\mathcal{S}}(\mathcal{B})$):
  - If $\mathcal{B}$ is a legal database state, then $\mathcal{I}$ is a model of $\phi(\mathcal{S})$.

- $\Delta^{\mathcal{I}}$ is the union of $\Delta^{\mathcal{B}}$, the set of (reified) tuples, and the set of domain values that appear in $\mathcal{B}$ as a value of some attribute.
- For each entity E, $(\phi(E))^{\mathcal{I}} = E^{\mathcal{B}}$.

- $\beta_{\mathcal{S}}$ is a mapping from relation-descriptive interpretations of $\phi(\mathcal{S})$ to database states corresponding to $\mathcal{S}$, with the following properties (let $\mathcal{I}$ be an interpretation of $\phi(\mathcal{S})$ and $\mathcal{B} = \beta_{\mathcal{S}}(\mathcal{I})$):
  - If $\mathcal{I}$ is a model of $\phi(\mathcal{S})$, then $\mathcal{B}$ is a legal database state.
  - $\Delta^{\mathcal{B}}$ is the set of all objects in $\Delta^{\mathcal{I}}$ that are not instances of any atomic concept corresponding to a relationship or a basic domain.
  - For each entity E, $E^{\mathcal{B}} = (\phi(E))^{\mathcal{I}}$.

The existence of the mappings $\alpha_{\mathcal{S}}$ and $\beta_{\mathcal{S}}$, allows us to reduce the problem of checking properties that hold for an ER schema to the problem of reasoning on the corresponding $\mathcal{ALCQI}$ knowledge base [Cal96b]. Indeed, given a relation-descriptive model $\mathcal{I}$ of $\phi(\mathcal{S})$ in which a concept $\phi(E)$ corresponding to an entity E is satisfiable, $\beta_{\mathcal{S}}(\mathcal{I})$ is a legal database state in which entity E is populated. Conversely, applying $\alpha_{\mathcal{S}}$ to such a legal database state, we obtain a suitable model of $\phi(\mathcal{S})$.

1.3.3 Extending the Expressiveness of the Modeling Language

Semantic data models in general, and the ER model in particular, do not provide several features and modeling primitives which would prove useful in order to represent complex dependencies between data. The richness of constructs that is typical of Description Logics, and the correspondence between the two formalisms established in the previous section, makes it possible however, to add such constructs to the basic model and take them fully into account when reasoning on a schema. We provide now several meaningful examples of possible additions to the basic ER model that arise as a natural consequence of the correspondence with $\mathcal{ALCQI}$.

1.3.3.1 Arbitrary Boolean Constructs on Entities. The only direct relationship between entities that can be expressed in the basic ER model is the is-a relation. A common extension is by so called *generalization hierarchies* (see e.g. [BCN92]), which allow one to express that the extension of an entity should be the disjoint union of the extensions of other entities. Such construct can easily be translated by making use of union and negation of Description Logics.

Example 1 (cont.) The fact that each professor is either an associate or a full professor can be expressed by a simple generalization hierarchy stating that the entity Professor is the generalization of the two entities AssProf and FullProf. We can translate this hierarchy in $\mathcal{ALCQI}$ by adding to the assertions in Figure 1.2 the following:

$$
\begin{array}{rcl}
\texttt{Professor} & \dot{\preceq} & \texttt{AssProf} \sqcup \texttt{FullProf} \\
\texttt{AssProf} & \dot{\preceq} & \texttt{Professor} \sqcap \neg\texttt{FullProf} \\
\texttt{FullProf} & \dot{\preceq} & \texttt{Professor}
\end{array}
\qquad \blacksquare
$$

In general, each level in a generalization hierarchy expressing that entity E is the generalization of the disjoint entities $E_1, \ldots, E_n$ can be translated to the following assertions:

$$
\begin{array}{rcl}
\phi(E) & \dot{\preceq} & \phi(E_1) \sqcup \cdots \sqcup \phi(E_n) \\
\phi(E_1) & \dot{\preceq} & \phi(E) \sqcap \neg\phi(E_2) \sqcap \neg\phi(E_3) \sqcap \cdots \sqcap \neg\phi(E_n) \\
\phi(E_2) & \dot{\preceq} & \phi(E) \sqcap \neg\phi(E_3) \sqcap \neg\phi(E_4) \sqcap \cdots \sqcap \neg\phi(E_n) \\
 & \vdots & \\
\phi(E_{n-1}) & \dot{\preceq} & \phi(E) \sqcap \neg\phi(E_n) \\
\phi(E_n) & \dot{\preceq} & \phi(E)
\end{array}
$$

More generally, exploiting the possibility to construct general boolean expressions in $\mathcal{ALCQI}$ we can make use of arbitrary boolean combinations of entities in a schema, thus expressing in particular disjointness and disjunction of entities, which are forms of negative and incomplete knowledge [DL93].

1.3.3.2 Refinement of Properties along an ISA Hierarchy. Another important extension that should be considered is the possibility to specify more complex forms of refinement of properties of entities along ISA hierarchies, than the mere addition of attributes. This is already an essential feature of the more recent object-oriented models. In particular, cardinality constraints could be refined by restricting the range of values, and the participation in relationships can be restricted. One may require for specific instances of an entity that the objects they are related to via a certain relationship belong to a more specific entity than the one directly associated to the ER-role. Such forms of constraints can be naturally expressed in $\mathcal{ALCQI}$ by making use of universal quantification over roles.

Example 1 (cont.) The following assertion imposes that advanced courses are followed by at least 5 and at most 15 students (thus refining the limits that hold for arbitrary courses), and that all these students are graduate:

$$
\texttt{AdvCourse} \ \dot{\preceq}\ \exists^{\geq 5}\texttt{Ein}^- \sqcap \exists^{\leq 15}\texttt{Ein}^- \sqcap \forall\texttt{Ein}^-.\forall\texttt{Eof.GradStudent} \qquad \blacksquare
$$

1.3.3.3 Definitions of Classes by Means of Complex Properties. In the ER model (and more generally in semantic data models) one can specify only necessary conditions that the instances of entities (or more generally classes) must satisfy. This means that in a database that conforms to the schema one cannot deduce that a certain object is an instance of an entity unless this fact is explicitly stated. When modeling a complex domain, however, in order to capture more precisely the intended semantics, one would like to be able to define classes of objects through necessary and sufficient conditions, or even to state just sufficient conditions for an object to be an instance of a class. By using the different types of assertions of $\mathcal{ALCQI}$ such conditions can be easily imposed and become part of the schema. In addition, the availability of the various constructors allows one to express relatively complex conditions.

Example 1 (cont.) A sufficient condition for a student to be considered graduate is that he has a degree. This can be specified by the following assertion, which is used together with the assertion in Figure 1.2 that specifies the necessary conditions.

$$\texttt{Student} \sqcap \exists \texttt{degree} \ \dot{\preceq} \ \texttt{GradStudent} \qquad \blacksquare$$

1.4 OBJECT-ORIENTED DATA MODELS

Object-oriented data models have been proposed with the goal of devising database formalisms that could be integrated with object-oriented programming systems [Kim90]. They are the subject of an active area of research in the Database field, and are based on the following features:

- They rely on the notion of object identifier at the extensional level (as opposed to traditional data models which are value-oriented) and on the notion of class at the intensional level.
- The structure of the classes is specified by means of typing and inheritance.

As in the previous section, we introduce a language for specifying object-oriented schemata, which includes the common features of object-oriented data models, and discuss its relationship with other class-based formalisms by showing that schemata expressed in this language can be correctly represented as knowledge bases in $\mathcal{ALCQI}$.

1.4.1 An Object-Oriented Data Model

We define a simple object-oriented language in the style of the most popular models featuring complex objects and object identity. Although we do not refer

```
Class Teacher type-is
  Union Professor, GradStudent
  End

Class GradStudent is-a Student type-is
  Record
    degree: String
  End

Class Course type-is
  Record
    enrolls: Set-of Student,
    taughtby: Teacher
  End
```

Figure 1.3 An object-oriented schema

to any specific formalism, our model is inspired by the one presented in [AK89], where a formal characterization is presented, but embodies the basic features of the static part of the ODMG standard [CB97]. We recall that we restrict our attention to the structural component of object-oriented models and do not consider those aspects related to the definition of methods associated to the classes.

An *object-oriented schema* is defined over a finite set of *class names*, denoted by the letter C and a finite set of *attribute names*, denoted by the letter A. An object-oriented $\mathcal{S}$ schema is a finite set of *class declarations* of the form:

$$\underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_k\ \underline{\text{type-is}}\ T,$$

with exactly one such declaration for each class C, and where T denotes a *type expression* built according to the following syntax:

$$\begin{array}{rcl} T & \longrightarrow & C \mid \\ & & \underline{\text{Union}}\ T_1, \ldots, T_k\ \underline{\text{End}} \mid \\ & & \underline{\text{Set-of}}\ T \mid \\ & & \underline{\text{Record}}\ A_1 \colon T_1, \ldots, A_k \colon T_k\ \underline{\text{End}}. \end{array}$$

Example 2 Figure 1.3 shows a fragment of the object-oriented schema corresponding to the Entity-Relationship schema of Figure 1.1. ■

Each class declaration imposes constraints on the instances of the class it refers to. The is-a part of a class declaration allows one to specify inclusion between the sets of instances of the involved classes, while the type-is part specifies through a type expression the structure assigned to the objects that are instances of the class.

The meaning of an object-oriented schema is given by specifying the characteristics of an instance of the schema. The definition of instance makes use of the notions of object identifier and value.

Let us first characterize the set of values that can be constructed from a set of symbols, called *object identifiers*. Given a finite set $\mathcal{O}$ of symbols, the set $\mathcal{V}_{\mathcal{O}}$ of *values* over $\mathcal{O}$ is inductively defined as follows:

- $\mathcal{O} \subseteq \mathcal{V}_{\mathcal{O}}$.
- If $v_1, \ldots, v_k \in \mathcal{V}_{\mathcal{O}}$ then $\{\!|v_1, \ldots, v_k|\!\} \in \mathcal{V}_{\mathcal{O}}$.
- If $v_1, \ldots, v_k \in \mathcal{V}_{\mathcal{O}}$ then $[\![A_1\!: v_1, \ldots, A_k\!: v_k]\!] \in \mathcal{V}_{\mathcal{O}}$.
- Nothing else is in $\mathcal{V}_{\mathcal{O}}$.

A *database instance* $\mathcal{J}$ of a schema $\mathcal{S}$ is constituted by

- a *finite* set $\mathcal{O}^{\mathcal{J}}$ of object identifiers;
- a mapping $\pi^{\mathcal{J}}$ assigning to each class name a subset of $\mathcal{O}^{\mathcal{J}}$;
- a mapping $\rho^{\mathcal{J}}$ assigning a value in $\mathcal{V}_{\mathcal{O}^{\mathcal{J}}}$ to each object in $\mathcal{O}^{\mathcal{J}}$.

Although the set $\mathcal{V}_{\mathcal{O}^{\mathcal{J}}}$ of values that can be constructed from a set $\mathcal{O}^{\mathcal{J}}$ of object identifiers is infinite, for a database instance one needs only to consider a finite subset of $\mathcal{V}_{\mathcal{O}^{\mathcal{J}}}$, since finite are the structures that can be stored in a database. For an object-oriented schema $\mathcal{S}$ and an instance $\mathcal{J}$ of $\mathcal{S}$, this finite set is called the set $\mathcal{V}_{\mathcal{J}}$ of *active values* with respect to $\mathcal{J}$, and is constituted by the union of

- the set $\mathcal{O}^{\mathcal{J}}$ of object identifiers and
- the set of values assigned by $\rho^{\mathcal{J}}$ to the elements of $\mathcal{O}^{\mathcal{J}}$, including those values that are not explicitly associated with object identifiers, but are used to form other values. ■

The interpretation of type expressions in $\mathcal{J}$ is defined through an *interpretation function* $\cdot^{\mathcal{J}}$ that assigns to each type expression a subset of $\mathcal{V}_{\mathcal{O}^{\mathcal{J}}}$ such that the following conditions are satisfied:

$$
\begin{aligned}
C^{\mathcal{J}} &= \pi^{\mathcal{J}}(C)\\
(\underline{\text{Union}}\ T_1, \ldots, T_k\ \underline{\text{End}})^{\mathcal{J}} &= T_1^{\mathcal{J}} \cup \cdots \cup T_k^{\mathcal{J}}\\
(\underline{\text{Set-of}}\ T)^{\mathcal{J}} &= \{\{\!|v_1, \ldots, v_k|\!\} \mid k \geq 0, v_i \in T^{\mathcal{J}},\\
&\qquad \text{for } i \in \{1, \ldots, k\}\}\\
(\underline{\text{Record}}\ A_1\!: T_1, \ldots, A_k\!: T_k\ \underline{\text{End}})^{\mathcal{J}} &= \{[\![A_1\!: v_1, \ldots, A_h\!: v_h]\!] \mid h \geq k,\\
&\qquad v_i \in T_i^{\mathcal{J}},\ \text{for } i \in \{1, \ldots, k\},\\
&\qquad v_j \in \mathcal{V}_{\mathcal{O}^{\mathcal{J}}},\ \text{for } j \in \{k+1, \ldots, h\}\}.
\end{aligned}
$$

Notice that the instances of a class of type record may have more components than those specified in the type of the class. Thus we are using an open semantics for records, which is typical of object-oriented data models (see e.g. [AK89]).

In order to characterize object-oriented data models we define which instances are admissible for a schema. A database instance $\mathcal{J}$ of an object-oriented schema $\mathcal{S}$ is said to be *legal* (with respect to $\mathcal{S}$) if for each declaration

$$\underline{\text{Class}}\ C\ \underline{\text{is-a}}\ C_1, \ldots, C_n\ \underline{\text{type-is}}\ T$$

in $\mathcal{S}$, it holds that $C^{\mathcal{J}} \subseteq C_i^{\mathcal{J}}$ for each $i \in \{1, \ldots, n\}$, and that $\rho^{\mathcal{J}}(C^{\mathcal{J}}) \subseteq T^{\mathcal{J}}$.

Therefore, for a legal database instance, the type expressions that are present in the schema determine the (finite) set of active values that must be considered. The construction of such values is limited by the depth of type expressions.

1.4.2 Formalizing Object-Oriented Schemata in Description Logics

We establish a relationship between $\mathcal{ALCQI}$ and the object-oriented language presented above. This is done by providing a mapping from object-oriented schemata into $\mathcal{ALCQI}$ knowledge bases. Since the interpretation domain for an $\mathcal{ALCQI}$ knowledge base consists of atomic objects, whereas each instance of an object-oriented schema is assigned a possibly structured value (see the definition of $\mathcal{V}_{\mathcal{O}}$), we need to explicitly represent some of the notions that underlie the object-oriented language. In particular, while there is a correspondence between concepts and classes, one must explicitly account for the type structure of each class. This can be accomplished by introducing in $\mathcal{ALCQI}$ a concept `AbstractClass`, to represent the classes, and two concepts `RecType` and `SetType` to represent the corresponding types. The associations between classes and types induced by the class declarations, as well as the basic characteristics of types, are modeled by means of roles: the (functional) role `value` models the association between classes and types, and the role `member` is used for specifying the type of the elements of a set. Moreover, the concepts representing types are assumed to be mutually disjoint, and disjoint from the concepts representing classes. These constraints are expressed by suitable inclusion assertions in the knowledge base.

More formally, the knowledge base $\psi(\mathcal{S})$ corresponding to an object-oriented schema $\mathcal{S}$ contains the predefined atomic concepts `AbstractClass`, `RecType`, and `SetType`, and one concept $\psi(C)$ for each class name C in $\mathcal{S}$. It contains also the predefined atomic roles `value` and `member`, and one atomic role $\psi(A)$ for each attribute name A in $\mathcal{S}$.

Before specifying the set of assertions of $\psi(\mathcal{S})$ we specify how the function ψ maps each type expression into a concept expression as follows:

- Every class C is mapped into an atomic concept $\psi(C)$.
- Every type expression <u>Union</u> $T_1,\ldots,T_k$ <u>End</u> is mapped into $\psi(T_1) \sqcup \cdots \sqcup \psi(T_k)$.
- Every type expression <u>Set-of</u> T is mapped into $\mathtt{SetType} \sqcap \forall\mathtt{member}.\psi(T)$.
- Every attribute A is mapped into an atomic role $\psi(A)$, and every type expression <u>Record</u> $A_1\!:T_1,\ldots,A_k\!:T_k$ <u>End</u> is mapped into

$$\begin{aligned}\mathtt{RecType} \sqcap\ & \forall\psi(A_1).\psi(T_1) \sqcap \exists^{=1}\psi(A_1) \sqcap \cdots \sqcap \\ & \forall\psi(A_k).\psi(T_k) \sqcap \exists^{=1}\psi(A_k).\end{aligned}$$

Using ψ we define the knowledge base $\psi(\mathcal{S})$ corresponding to $\mathcal{S}$ as constituted by the inclusion assertions

$$\begin{aligned}\mathtt{AbstractClass} &\ \dot{\preceq}\ \exists^{=1}\mathtt{value} \\ \mathtt{RecType} &\ \dot{\preceq}\ \forall\mathtt{value}.\bot \\ \mathtt{SetType} &\ \dot{\preceq}\ \forall\mathtt{value}.\bot \sqcap \neg\mathtt{RecType}\end{aligned}$$

and for each class declaration

<u>Class</u> C <u>is-a</u> $C_1,\ldots,C_n$ <u>type-is</u> T

in $\mathcal{S}$, an inclusion assertion

$$\psi(C)\ \dot{\preceq}\ \mathtt{AbstractClass} \sqcap \psi(C_1) \sqcap \cdots \sqcap \psi(C_n) \sqcap \forall\mathtt{value}.\psi(T).$$

Example 2 (cont.) We illustrate the translation on the fragment of object-oriented schema in Figure 1.3. The corresponding $\mathcal{ALCQI}$ knowledge base is shown in Figure 1.4. ■

Some remarks on the above translation are in order.

- As for the ER model the resulting knowledge base contains inclusion assertions, but not equality assertions.
- The relationship between a class and the associated type expression is reified, i.e. explicitly represented through the role `value`. Moreover, the type structure of each class is represented in terms of the concepts `RecType` and `SetType`, that explicitly characterize records and sets, respectively. The record attributes are directly mapped into functional roles, while set elements are associated to a set through the role `member`.

$$
\begin{array}{rcl}
\texttt{Course} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \\
& & \forall\texttt{value}.(\texttt{RecType} \sqcap \exists^{=1}\texttt{enrolls} \sqcap \exists^{=1}\texttt{taughtby} \sqcap \\
& & \qquad \forall\texttt{enrolls}.(\texttt{SetType} \sqcap \forall\texttt{member}.\texttt{Student}) \sqcap \\
& & \qquad \forall\texttt{taughtby}.\texttt{Teacher}) \\
\texttt{Teacher} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \forall\texttt{value}.(\texttt{GradStudent} \sqcup \texttt{Professor}) \\
\texttt{GradStudent} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \texttt{Student} \sqcap \\
& & \forall\texttt{value}.(\texttt{RecType} \sqcap \forall\texttt{degree}.\texttt{String} \sqcap \exists^{=1}\texttt{degree}) \\
\texttt{AbstractClass} & \dot{\preceq} & \exists^{=1}\texttt{value} \\
\texttt{RecType} & \dot{\preceq} & \forall\texttt{value}.\bot \\
\texttt{SetType} & \dot{\preceq} & \forall\texttt{value}.\bot \sqcap \neg\texttt{RecType}
\end{array}
$$

Figure 1.4 The $\mathcal{ALCQI}$ knowledge base corresponding to the object-oriented schema in Figure 1.3

- Inverse roles are not needed for the formalization of object-oriented data models.
- The use of number restrictions is limited to the value 1, which corresponds to existence constraints and functionality.

Below we discuss the effectiveness of the translation ψ. First of all observe that the knowledge base $\psi(\mathcal{S})$ resulting from the translation of an object-oriented schema $\mathcal{S}$ may admit models that do not have a direct counterpart among legal database instances of $\mathcal{S}$. More precisely, by characterizing the interpretations as directed labeled graphs [CLN97], one finds that the concepts that translate type expressions involving record and set structures admit certain cyclic models in which values, which have a tree-like structure, have no direct counterpart.

Example 3 Consider the object-oriented schema $\mathcal{S}$, containing a single class declaration

$$\underline{\text{Class}}\ C\ \underline{\text{type-is}}\ \underline{\text{Record}}\ a_1 : \underline{\text{Record}}\ a_2 : \underline{\text{Record}}\ a_3 : C\ \underline{\text{End}}\ \underline{\text{End}}\ \underline{\text{End}}$$

which is translated to

$$
\begin{array}{rcl}
C & \dot{\preceq} & \texttt{AbstractClass} \sqcap \\
& & \forall\texttt{value}.(\texttt{RecType} \sqcap \exists^{=1}a_1 \sqcap \\
& & \qquad \forall a_1.(\texttt{RecType} \sqcap \exists^{=1}a_2 \sqcap \\
& & \qquad\qquad \forall a_2.(\texttt{RecType} \sqcap \exists^{=1}a_3 \sqcap \forall a_3.C))).
\end{array}
$$

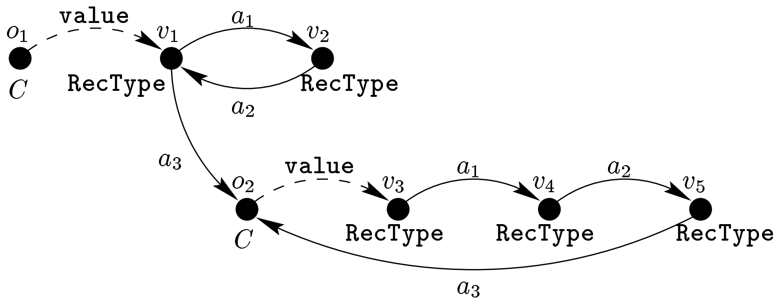

Figure 1.5 A model containing cycles

Figure 1.5 shows a model of $\psi(\mathcal{S})$ represented as a graph. For clarity, we have named the instances of C, and hence of `AbstractClass`, with o and the instances of `RecType` with v. Observe the two different types of cycles in the graph. The cycle involving individuals o_2, v_3, v_4, and v_5 does not cause any problems since it contains an arc labeled with `value`, which is not part of the structure constituting a complex value. In fact, v_3 represents the record value $[\![a_1\colon [\![a_2\colon [\![a_3\colon o_2]\!]]\!]]\!]$. On the other hand, the cycle involving v_1 and v_2 represents (together with o_2 connected via a_3 to v_1) a record of infinite depth. ∎

To establish the correspondence between legal instances of object-oriented schemata and finite interpretation of their translations one needs to unfold "bad" cycles of the form above (which do not include any individual corresponding to an object identifier) into finite trees of depth bound by the size of the object-oriented schema. More precisely, let $\mathcal{I}$ be a finite interpretation of $\psi(\mathcal{S})$. We call *unfolded version* of $\mathcal{I}$ the interpretation obtained by unfolding the bad cycles in $\mathcal{I}$ and generating new individuals only for the instances of `RecType` and `SetType` occurring in the unfolded cycles. For a nonnegative integer m, we call *m-unfolded version* of $\mathcal{I}$, denoted as $\mathcal{I}_{|m}$, the interpretation obtained by truncating at depth m each infinite tree generated in the process of unfolding.

The correctness of $\psi(\mathcal{S})$ can now be established by showing that, for every object-oriented schema $\mathcal{S}$ of depth m, there exist two mappings:

- $\alpha_{\mathcal{S}}$, from instances of $\mathcal{S}$ into finite interpretations of $\psi(\mathcal{S})$, and
- α_v, from active values of instances of $\mathcal{S}$ into domain elements of the finite interpretations of $\psi(\mathcal{S})$

which provide the desired correspondence between the interpretation structures. In particular, $\alpha_{\mathcal{S}}$ and α_v can be defined in such a way that: (i) if $\mathcal{J}$ is a legal instance of $\mathcal{S}$, then $\alpha_{\mathcal{S}}(\mathcal{J})$ is a finite model of $\psi(\mathcal{S})$, and (ii) for each type expression T of $\mathcal{S}$ and each active value $v \in \mathcal{V}_{\mathcal{J}}$, $v \in T^{\mathcal{J}}$ if and only if $\alpha_v(v) \in (\psi(T))^{\alpha_{\mathcal{S}}(\mathcal{J})}$.

Conversely, one can define two mappings:

- $\beta_{\mathcal{S}}$, from finite interpretations of $\psi(\mathcal{S})$ into instances of $\mathcal{S}$, and
- β_v, from domain elements of the m-unfolded versions of the finite interpretations of $\psi(\mathcal{S})$ into active values of instances of $\mathcal{S}$,

such that: (i) for each finite model $\mathcal{I}$ of $\psi(\mathcal{S})$, $\beta_{\mathcal{S}}(\mathcal{I})$ is a legal instance of $\mathcal{S}$, and (ii) for each concept $\psi(T)$, which is the translation of a type expression T of $\mathcal{S}$, and each $d \in \Delta^{\mathcal{I}_{|m}}$, $d \in (\psi(T))^{\mathcal{I}_{|m}}$ if and only if $\beta_v(d) \in T^{\beta_{\mathcal{S}}(\mathcal{I})}$.

Similarly to the ER model, the existing of the above mappings allows us to reduce the problem of checking properties of classes in an object-oriented schema to the problem of reasoning on the corresponding knowledge base.

1.4.3 Extending the Expressiveness of the Modeling Language

The ability to represent any object-oriented schema as a Description Logic knowledge base makes it feasible to consider several extensions of the object-oriented formalism useful for the purpose of conceptual modeling. First of all, the same considerations developed for the ER model with regard to the use of arbitrary boolean constructs on classes can be applied also in the object-oriented setting, which provides disjunction but does not admit any form of negation. Second, one can analogously exploit the ability of expressing class definitions in addition to the inclusion statements that are typical of object-oriented modeling languages [BN94]. However, there are several additional features that can be specifically addressed in the framework of object-oriented languages, as shown below.

1.4.3.1 Cardinality Constraints. Cardinality Constraints that are typical of semantic data models become expressible in object-oriented schemata.

Example 2 (cont.) We can specify numerical restrictions in the definition of a course, which can enroll between 2 and 30 students.

$$
\begin{array}{lll}
\texttt{Course} & \dot{\preceq} & \texttt{AbstractClass} \sqcap \\
& & \forall\texttt{value}.(\texttt{RecType} \sqcap \exists^{=1}\texttt{enrolls} \sqcap \exists^{=1}\texttt{taughtby} \sqcap \\
& & \qquad \forall\texttt{enrolls}.(\texttt{SetType} \sqcap \forall\texttt{member}.\texttt{Student} \sqcap \\
& & \qquad\qquad \exists^{\geq 2}\texttt{member} \sqcap \exists^{\leq 30}\texttt{member}) \sqcap \\
& & \qquad \forall\texttt{taughtby}.\texttt{Teacher})
\end{array}
$$

■

Notice that, the usage of cardinality restrictions in the example above actually corresponds to the ability to constrain the cardinality of sets.

1.4.3.2 General Restrictions on the Values of Attributes. The usage in object-oriented modeling languages of constructs corresponding to quantified role restrictions is limited to the translation of record and set structures. Adding general forms of quantified role restrictions amounts to admitting more flexible structures, whose closer counterpart are possibly frames in knowledge representation systems (see e.g. [BS85; CLN94]). In a frame structure one can for example specify that a slot, which is the counterpart of a record attribute, can have restrictions on the number and type of fillers. In addition, instead of introducing set types explicitly for those attributes whose values are sets, one can directly use multivalued attributes and express the constraints on the type and the number of instances using the constructors of $\mathcal{ALCQI}$.

Example 4 We can specify, that a graduate student must have at least one degree, but maybe more, and express the relationship between courses and students by means of the attribute `enrolls` which we now consider multivalued.

$$\begin{array}{rcl} \texttt{GradStudent} & \dot{\preceq} & \texttt{Student} \sqcap \exists\texttt{degree} \\ \texttt{Course} & \dot{\preceq} & \forall\texttt{enrolls.Student} \sqcap \exists^{\geq 2}\texttt{enrolls} \sqcap \exists^{\leq 30}\texttt{enrolls} \sqcap \\ & & \forall\texttt{taughtby.}(\texttt{Professor} \sqcup \texttt{GradStudent}) \sqcap \exists^{=1}\texttt{taughtby} \end{array} \quad \blacksquare$$

1.4.3.3 Inverse of Roles. We have already observed that inverse roles are not necessary for the formalization of object-oriented data models. However, by admitting inverse of roles in the language one gains the ability to put constraints using a relation in both directions, as it is customary in semantic data models.

Example 2 (cont.) We can state that each professor should teach at least one course by means of the assertion:

$$\texttt{Professor} \ \dot{\preceq}\ \exists\texttt{taughtby}^{-}\texttt{.Course} \qquad \blacksquare$$

Indeed, the possibility of referring to the inverse of an attribute is often ruled out in object-oriented models. However, this strongly limits the expressive power of the data model, as pointed out in recent papers (see e.g. [AGO91]). On the other hand, we recall that inverse roles have been included in the ODGM standard [CB97].

1.5 SUPPORT FOR DATA MODELING

The logical formalization of data models in terms of Description Logics is a first step towards the development of logic-based modeling tools that can support the database designer in his activity, by taking over certain tasks that in traditional CASE tools are left to the responsibility of the designer. In this section we first discuss the tasks that can be performed by a Description Logic system by relying on its reasoning capabilities, thus identifying the essential reasoning services that the system should provide. We then briefly discuss a technique for reasoning on $\mathcal{ALCQI}$ knowledge bases. Finally, we discuss how reasoning in $\mathcal{ALCQI}$ does indeed capture the reasoning services for the semantic and object-oriented schemata described in the paper.

1.5.1 Reasoning Tasks in Data Modeling

Traditional database modeling tools support the designer with a user friendly graphical environment and provide means to access different kinds of repositories that store information associated to the elements of the developed schemata. However, these tools do not provide any support for higher level activities related to managing the complexity of schemata. In particular, the burden of checking relevant properties of schemata, such as consistency or redundancy, is left to the responsibility of the designer.

There are different ways in which $\mathcal{ALCQI}$ knowledge bases may enter the database modeling process and may be used by a system to support this activity.

- A knowledge base may be the result of a direct translation by a CASE tool from a database schema expressed in one of the traditional (object-oriented or semantic) data models. Notice that due to the expressiveness of $\mathcal{ALCQI}$ several additional constraints that cannot be directly expressed in the source data model may be expressed by the designer and included in the knowledge base, in order to be considered by the reasoning procedures.

- A knowledge base may be constructed and managed by the design tool in a way completely transparent to the designer in order to perform various kinds of checks on the schemata.

- $\mathcal{ALCQI}$ may be used as a unifying formalism in which to express schemata in different data models for the purpose of integration. In this case, additional assertions may be added to those resulting from the individual schemata in order to express so called *interschema constraints* [CL93; Hul97].

Once a formalization of a schema in terms of $\mathcal{ALCQI}$ is provided, the checking of properties which ensure correctness and optimality of a design, can be turned into reasoning tasks in $\mathcal{ALCQI}$. Such properties are addressed in the following.

$$\begin{array}{rcl}
\texttt{SUPERVISING} & \dot{\preceq} & \forall\texttt{Sby.Professor} \sqcap \exists^{=1}\texttt{Sby} \sqcap \forall\texttt{Sof.GradStudent} \sqcap \exists^{=1}\texttt{Sof} \\
\texttt{Professor} & \dot{\preceq} & \exists^{\geq 1}\texttt{Tby}^{-}.\forall\texttt{Tof.AdvCourse} \sqcap \forall\texttt{Sby}^{-}.\texttt{SUPERVISING} \sqcap \exists^{\leq 2}\texttt{Sby}^{-} \\
\texttt{GradStudent} & \dot{\preceq} & \exists^{\geq 2}\texttt{Ein}^{-}.\forall\texttt{Eof.AdvCourse} \sqcap \exists^{\leq 4}\texttt{Ein}^{-}.\forall\texttt{Eof.AdvCourse} \sqcap \\
 & & \forall\texttt{Sof}^{-}.\texttt{SUPERVISING} \sqcap \exists^{=1}\texttt{Sof}^{-} \\
\texttt{AdvCourse} & \dot{\preceq} & \forall\texttt{Eof}^{-}.\forall\texttt{Ein.GradStudent}
\end{array}$$

Figure 1.6 An inconsistent extension of the knowledge base shown in Figure 1.2

1.5.1.1 Schema Consistency. A *schema* is *consistent*, if there is a (nonempty) database that satisfies (all constraints specified in) the schema. Although the problem of checking schema consistency arises already in relatively simple data models (e.g. the ER model without is-a relations [LN90]), it becomes much more difficult to solve if the expressiveness of the formalism is increased.

Example 5 Consider augmenting the ER schema in Figure 1.1 with a new relationship `SUPERVISING`, which is linked with role `Sby` to `Professor` and with role `Sof` to `GradStudent`, each with suitable cardinality constraints. Additionally we want to impose more realistic constraints on the relationship `ENROLLING` between `AdvCourse` and `GradStudent`. These additional constraints, which cannot be stated in the ER model, are expressed (together with the translation of the relationship `SUPERVISING`) as assertions in $\mathcal{ALCQI}$ in Figure 1.6. They express that advanced courses may enroll only graduate students, and each such student must be enrolled in at least 2 and at most 4 advanced courses. We impose also that each professor must teach at least one advanced course.

The formalization above seems reasonable. It turns out, however, that the knowledge base which is the union of the assertions in Figures 1.2 and 1.6 (and of the disjointness assertions stemming from the translation of the ER schema) is unsatisfiable. In order to understand why, one has to consider that the cardinality constraints on the participation of an entity in a relationship pose restrictions on the number of their instances. For example, they enforce that in every legal database state, the number of instances of `GradStudent` is the same as the number of instances (i.e. reified tuples) in `SUPERVISING`. When such constraints appear along a cycle in the schema it may happen that they cannot be satisfied together. For this to be the case, the assumption that the database contains a finite number of objects is essential. In the example, one such cycle is constituted by: `AdvCourse`, `ENROLLING`, `GradStudent`, `SUPERVISING`, `Professor`, `TEACHING`, and back to `AdvCourse`. ∎

$$
\begin{array}{rcl}
\texttt{SUPERVISING} & \dot{\preceq} & \forall\texttt{Sby.GradStudent} \sqcap \exists^{=1}\texttt{Sby} \sqcap \forall\texttt{Sof.Student} \sqcap \exists^{=1}\texttt{Sof} \\
\texttt{GradStudent} & \dot{\preceq} & \exists^{\leq 1}\texttt{Sby}^{-} \sqcap \forall\texttt{Sby}^{-}\texttt{.SUPERVISING} \\
\texttt{Student} & \dot{\preceq} & \exists^{\geq 1}\texttt{Sof}^{-} \sqcap \forall\texttt{Sof}^{-}\texttt{.SUPERVISING}
\end{array}
$$

Figure 1.7 An extension of the knowledge base shown in Figure 1.2 which forces equivalence of two classes

It is worth emphasizing, that situations like the one exemplified above are not unusual in practice. They are generally difficult to discover until the data base population stage, where handling them can be rather inconvenient.

1.5.1.2 Class Consistency. A *class* is *consistent*, if it has a nonempty extension in some database that satisfies the schema. The inconsistency of a class may be due to a design error or due to over-constraining. In any case, the designer can be forced to remove the inconsistency, either by correcting the error, or by relaxing some constraints, or by deleting the class, thus removing redundancy from the schema. Observe also that while schema consistency follows from consistency of all classes in the schema (in fact from consistency of at least one class), the converse is in general not true.

1.5.1.3 Class Equivalence. Two classes are *equivalent* if they denote the same set of instances in all databases that satisfy the schema. Determining equivalence of two classes allows for their merging, thus reducing the complexity of the schema. It is worth emphasizing that such an operation is in practice a very difficult task. Moreover, the ability to introduce definitions, allows for a hierarchical structuring of the schema which can be used to support a refinement approach to schema design, while ensuring the overall consistency.

1.5.1.4 Class Subsumption. A class C_1 is *subsumed by* a class C_2 if in all databases that satisfy the schema the extension of C_1 is a subset of the extension of C_2. Subsumption allows one to deduce properties for one class from those of another one. It is also the basis for a *classification* of all the classes that appear in a schema within a lattice. Such a classification, as in any object-oriented approach, can be exploited in several ways within the modeling process [BN94].

Example 6 Suppose we extend now the ER schema in Figure 1.1 as shown in Figure 1.7, linking the relationship `SUPERVISING`, with role `Sby` to `GradStudent` and with role `Sof` to `Student`. The cardinality constraints express that each student has at least one supervisor and that each graduate student supervises at most one student. Now the constraints we have imposed force the two classes `Student` and `GradStudent` to have the same number of instances, and since the extension of `GradStudent` is a subset of the extension of `Student` (and the domain is assumed to be finite), the two classes are in fact equivalent. We can also say that `Student` is subsumed by `GradStudent` in this augmented schema. Observe that this property may have consequences on other classes in the schema. If, for example we introduce a new subclass `BasicCourse` of `Course`, and require by means of

$$\texttt{BasicCourse} \ \dot{\preceq} \ \texttt{Course} \sqcap \forall \texttt{Ein}^{-}.\forall \texttt{Eof}.\neg \texttt{GradStudent} \sqcap \exists^{\geq 1} \texttt{Ein}^{-}$$

that basic courses enroll only students which are not graduate, and there should be at least one enrolled student, then `BasicCourse` is inconsistent. ■

1.5.1.5 Logical Consequence. A property is a *logical consequence* of a schema if it holds in all databases that satisfy the schema. The properties that should be considered are those of the same form as the constraints that can be expressed in the schema definition language. Determining logical consequence is at the basis of all types of reasoning that a Description Logic system can provide. In particular, all reasoning tasks we have considered above can be rephrased in terms of logical consequence. For example, a class A is inconsistent in a schema if and only if the constraint $A \dot{\preceq} \perp$ is a logical consequence of the assertions in the schema. Logical consequence is useful on the one hand to reduce the complexity of the schema by removing those constraints that logically follow from other ones, and on the other hand it can be used to explicit properties that are implicit in the schema, thus enhancing its readability.

1.5.2 Realization of Reasoning

In order to provide the above mentioned services, we first address the decidability of the relevant reasoning tasks in $\mathcal{ALCQI}$ knowledge bases, and then illustrate how the method can be applied to perform reasoning tasks that are specific to semantic data models and object-oriented schemata. In this way reasoning can effectively support the designer in the data modeling activity.

1.5.2.1 Reasoning in $\mathcal{ALCQI}$. The richness of constructs available in $\mathcal{ALCQI}$, which in addition to the basic Description Logic constructs includes inverse of roles, qualified number restrictions and cyclic assertions, makes reasoning on a knowledge base a nontrivial task. We recall that in a data modeling setting we are interested in finite model reasoning.

First, we observe that all reasoning tasks can immediately be reduced to the fundamental problem of checking the satisfiability of a single atomic concept.

In fact, a knowledge base $\mathcal{K}$ is satisfiable if and only if $\top$ is satisfiable in $\mathcal{K}$, and $\mathcal{K}$ implies the inclusion assertion $C_1 \dot{\preceq} C_2$ if and only if $C_1 \sqcap \neg C_2$ is not satisfiable in $\mathcal{K}$. Moreover, it is sufficient to treat the satisfiability of an atomic concept, since an arbitrary concept expression C is satisfiable in $\mathcal{K}$, if and only if a newly introduced atomic concept A is satisfiable in $\mathcal{K} \cup \{A \doteq C\}$.

Checking the satisfiability of a single atomic concept A in an $\mathcal{ALCQI}$ knowledge base $\mathcal{K}$ can be done by exploiting the technique developed in [Cal96a]. The method extends the one developed in [CLN94] for a simpler logic, and is based on the construction of a particular system $\Psi^A_{\mathcal{K}}$ of linear inequations, and the search for particular (*acceptable*) solutions of $\Psi^A_{\mathcal{K}}$ (see [Cal96b] for full details). The size of $\Psi^A_{\mathcal{K}}$ is in the worst case doubly exponential in the size of $\mathcal{K}$, while the search for acceptable solutions can be done in polynomial time in the size of the system. Thus we obtain a decision procedure for all reasoning tasks that works in deterministic double exponential time.

We notice that in the case discussed in [CLN94], where the schema contains no equality assertions and certain constructs are used only in a restricted way (negation is allowed only in front of atomic concepts, and all number restrictions are non-qualified, i.e. of the form $\exists^{\geq n} R$ or $\exists^{\leq n} R$) the size of the system can be kept single exponential in the size of $\mathcal{K}$. Such a case is of particular interest, since the basic constructs of semantic and object-oriented data models can be expressed in this language. A discussion on the optimization techniques for the proposed framework is presented in [CL94a].

1.5.2.2 Reasoning in Semantic and Object-Oriented Data Models. The decidability of reasoning in $\mathcal{ALCQI}$ together with the characterization of semantic and object-oriented data models developed in Sections 1.3.2 and 1.4.2 provides a method to reason in these data models.

In particular, the consistency of an entity E in an ER schema can be rephrased as the problem of checking whether the concept corresponding to E is satisfiable in the translation of the schema. Analogously, subtyping in object-oriented schemata, i.e. checking whether a type denotes a subset of another type in every legal instance of a schema can be accomplished by checking whether subsumption between the translated type expressions follows from the translation of the schema. Moreover, type consistency, i.e. checking whether a type is consistent in a legal instance of the schema, can be reduced to concept consistency.

A number of attempts have been previously made to characterize reasoning problems and devise reasoning techniques for semantic [CFP84; LN90; DL93; HK87] and object-oriented data models [AK89; BS92; BN94]. There is also significant work on reasoning about dependencies in the relational model (see e.g. [CFP84; CK86; GM85; CKV90; AHV95]), although it has no tight relationship to the framework addressed here.

With respect to reasoning on Entity-Relationship schemata, previous work [CTF88; LN90] addressed less expressive formalisms, and therefore the reasoning techniques for Description Logics can provide extensions towards a a significantly more expressive framework. Similar considerations apply with respect to the proposals to perform type consistency and inheritance reasoning on object-oriented data models. As an example, the type consistency and type subsumption algorithms in [BN94] for an expressive object-oriented formalism, which allows for the definition of classes by means of necessary and sufficient conditions, could further be extended in order to take into account also union types and inverse attributes.

By identifying reasoning in $\mathcal{ALCQI}$ as the basis for reasoning on semantic and object-oriented data models we are able to combine the features of the two approaches, while retaining the decidability of the reasoning problems. Moreover, the combination of two language constructs is for the first time taken into consideration with regard to reasoning in data models, namely negation and disjunction. Clearly, such a generality leads to a high computational complexity and in order to make the proposed approach feasible in a practical setting further analysis is required, that is beyond the scope of the present paper. Nonetheless, the proposed approach does provide a uniform basis for reasoning on semantic and object-oriented data models.

1.6 CONCLUSIONS

In this paper we have presented a unified view of the formalisms for conceptual data modeling, by adopting the language of Description Logics as a common basis. Such logics, which originated from the formalization of frame-based systems and semantic networks, provide enough expressive power to give a translation of the most popular semantic and object-oriented data modeling languages. A feature of Description Logics is that both the expressiveness of the language and the associated reasoning capabilities can be related to the set of constructs that are admitted in the language. Therefore, the comparison of data modeling formalisms has been done in terms of the constructs that are needed in order to enable the translation in Description Logics.

The use of Description Logics as a framework for data modeling has a number of consequences that we have addressed in the paper. First of all, it shows that semantic and object-oriented data modeling formalisms have several commonalities and a few distinguishing features. Second, we have seen how both modeling frameworks can be enriched by including several constructs of Description Logics, thus providing additional expressive capabilities. Third, we have discussed how the reasoning methods developed for Description Logics can provide the basic reasoning services that are needed to support the data modeling process.

Acknowledgments

This work has been partly funded by ESPRIT LTR Project "Foundations of Datawarehouse Quality (DWQ)" No.22469, and by Progetto Strategico "Informatica nella Pubblica Amministrazione", Sottoprogetto PROGRESS of the Italian Research Council.

References

[Abr74] J. R. Abrial. Data semantics. In J. W. Klimbie and K. L. Koffeman, editors, *Data Base Management*, pages 1–59. North-Holland Publ. Co., Amsterdam, 1974.

[ACS96] Alessandro Artale, Francesca Cesarini, and Giovanni Soda. Describing database objects in a concept language environment. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):345–351, 1996.

[AGO91] A. Albano, G. Ghelli, and R. Orsini. A relationship mechanism for strongly typed Object-Oriented database programming languages. In *Proc. of the 17th Int. Conf. on Very Large Data Bases (VLDB-91)*, pages 565–575, 1991.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1995.

[AK89] Serge Abiteboul and Paris Kanellakis. Object identity as a query language primitive. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 159–173, 1989.

[Baa96] Franz Baader. Using automata theory for characterizing the semantics of terminological cycles. *Annals of Mathematics and Artificial Intelligence*, 18:175–219, 1996.

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

[BDNS94] Martin Buchheit, Francesco M. Donini, Werner Nutt, and Andrea Schaerf. Terminological systems revisited: Terminology = schema + views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94)*, pages 199–204, 1994.

[BDS93] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[BHR90] K. H. Bläsius, U. Hedstück, and C.-R. Rollinger, editors. *Sorts and Types in Artificial Intelligence*. Number 418 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[BN94] Sonia Bergamaschi and Bernhard Nebel. Acquisition and validation of complex object database schemata supporting multiple inheritance. *Applied Intelligence*, 4(2):185–203, 1994.

[Bor92] Alexander Borgida. From type systems to knowledge representation: Natural semantics specifications for description logics. *Journal of Intelligent and Cooperative Information Systems*, 1(1):93–126, 1992.

[Bor95] Alexander Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[BS92] Sonia Bergamaschi and Claudio Sartori. On taxonomic reasoning in conceptual design. *ACM Transactions on Database Systems*, 17(3):385–422, 1992.

[BW97] Alexander Borgida and Grant E. Weddell. Adding functional dependencies to description logics. In *Proc. of the 5th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-97)*, 1997.

[Cal96a] Diego Calvanese. Finite model reasoning in description logics. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 292–303. Morgan Kaufmann, Los Altos, 1996.

[Cal96b] Diego Calvanese. *Unrestricted and Finite Model Reasoning in Class-Based Representation Formalisms*. PhD thesis, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1996.

[CB97] R.G.G Cattell and Douglas K. Barry, editors. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, Los Altos, 1997.

[CDGL95] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Structured objects: Modeling and reasoning. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases (DOOD-95)*, number 1013 in Lecture Notes in Computer Science, pages 229–246. Springer-Verlag, 1995.

[CFP84] Marco A. Casanova, Ronald Fagin, and Christos H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, 1984.

[Che76] P. P. Chen. The Entity-Relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.

[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.

[CKV90] S. S. Cosmadakis, P. C. Kanellakis, and M. Vardi. Polynomial-time implication problems for unary inclusion dependencies. *Journal of the ACM*, 37(1):15–46, January 1990.

[CL93] Tiziana Catarci and Maurizio Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, 2(4):375–398, 1993.

[CL94a] Diego Calvanese and Maurizio Lenzerini. Making object-oriented schemas more expressive. In *Proc. of the 13th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-94)*, pages 243–254. ACM Press and Addison Wesley, 1994.

[CL94b] Diego Calvanese and Maurizio Lenzerini. On the interaction between ISA and cardinality constraints. In *Proc. of the 10th IEEE Int. Conf. on Data Engineering (ICDE-94)*, pages 204–213. IEEE Computer Society Press, 1994.

[CLN94] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. A unified framework for class based representation formalisms. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 109–120. Morgan Kaufmann, Los Altos, 1994.

[CLN97] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Foundations of class-based representation formalisms. Technical report, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", 1997.

[CTF88] Marco A. Casanova, Luiz Tucherman, and Antonio L. Furtado. Enforcing inclusion dependencies and referencial integrity. In *Proc. of the 14th Int. Conf. on Very Large Data Bases (VLDB-88)*, pages 38–49, 1988.

[DGL94] Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of the 11th European Conf. on Artificial Intelligence (ECAI-94)*, pages 411–415, 1994.

[DL93] Giuseppe Di Battista and Maurizio Lenzerini. Deductive entity-relationship modeling. *IEEE Transactions 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 *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*, pages 151–162. Morgan Kaufmann, Los Altos, 1991.

[Fer91] S. Ferg. Cardinality concepts in entity-relationship modeling. In *Proc. of the 10th Int. Conf. on the Entity-Relationship Approach (ER-91)*, pages 1–30, 1991.

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

[GM85] John Grant and Jack Minker. Inferences for numerical dependencies. *Theoretical Computer Science*, 41:271–287, 1985.

[HB91] Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. Technical Report RR-91-03, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Kaiserslautern, Germany, 1991. An abridged version appeared in *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-91)*.

[HK87] R. B. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[Hul97] Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-97)*, 1997.

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

[KL89] Won Kim and Frederick H. Lochovsky, editors. *Object-Oriented Concepts, Databases, and Applications*. ACM Press and Addison Wesley, New York, 1989.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of Object-Oriented and frame-based languages. *Journal of the ACM*, 42(3), 1995.

[Leh92] Fritz Lehmann, editor. *Semantic Networks in Artificial Intelligence.* Pergamon Press, Oxford, 1992.

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

[LNS91] Maurizio Lenzerini, Daniele Nardi, and Maria Simi, editors. *Inheritance Hierarchies in Knowledge Representation and Programming Languages.* John Wiley & Sons, 1991.

[MM92] R. Motschnig-Pitrik and J. Mylopoulous. Classes and instances. *Journal of Intelligent and Cooperative Information Systems*, 1(1), 1992.

[Neb91] Bernhard Nebel. Terminological cycles: Semantics and computational properties. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 331–361. Morgan Kaufmann, Los Altos, 1991.

[PSS92] Barbara Piza, Klaus-Dieter Schewe, and Joachim W. Schmidt. Term subsumption with type constructors. In Y. Yesha, editor, *Proc. of the Int. Conf. on Information and Knowledge Management (CIKM-92)*, pages 449–456, 1992.

[Sch94] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR-94)*, pages 509–520. Morgan Kaufmann, Los Altos, 1994.

[Sow91] John F. Sowa, editor. *Principles of Semantic Networks.* Morgan Kaufmann, Los Altos, 1991.

[SS91] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[Teo89] J. T. Teorey. *Database Modeling and Design: The Entity-Relationship Approach.* Morgan Kaufmann, Los Altos, 1989.

[Tha92] Bernhard Thalheim. Fundamentals of cardinality constraints. In G. Pernoul and A. M. Tjoa, editors, *Proc. of the 11th Int. Conf. on the Entity-Relationship Approach (ER-92)*, pages 7–23. Springer-Verlag, 1992.

[Tha93] Bernhard Thalheim. *Fundamentals of the Entity Relationship Model.* Springer-Verlag, 1993.

[YPS94] Xian Ye, Christine Parent, and Stefano Spaccapietra. Cardinality consistency of derived objects in DOOD systems. In *Proc. of the 13th Int. Conf. on the Entity-Relationship Approach (ER-94)*, number 881 in Lecture Notes in Computer Science, pages 278–295. Springer-Verlag, 1994.

Index