Reasoning on UML Class Diagrams

Daniela Berardi, Andrea Calì, Diego Calvanese and Giuseppe De Giacomo
Dipartimento di Informatica e Sistemistica,
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy,
lastname@dis.uniroma1.it.

Abstract

UML is the de-facto standard formalism for software design and analysis. To support the design of large-scale industrial applications, sophisticated CASE tools are available on the market, that provide a user-friendly environment for editing, storing, and accessing multiple UML diagrams. It would be highly desirable to equip such CASE tools with automated reasoning capabilities in order to detect relevant formal properties of UML diagrams, such as inconsistencies or redundancies. With regard to this issue, we consider UML *class diagrams*, which are one of the most important components of UML, and we address the problem of reasoning on such diagrams. We resort to several results developed in the field of Description Logics (DLs), a family of logics that admit decidable reasoning procedures. Our first contribution is to show that reasoning on UML class diagrams is EXPTIME-hard, even under restrictive assumptions; we prove this result by showing a polynomial reduction from reasoning in DL. The second contribution consists in establishing EXPTIME-membership of reasoning on UML class diagrams, provided that the use of arbitrary OCL (first-order) constraints is disallowed. We get this result by using $\mathcal{DLR}_{ifd}$, a very expressive EXPTIME-decidable DL that is able to capture conceptual and object-oriented data models. The last contribution has a more practical flavor, and consists in a polynomial encoding of UML class diagrams in the DL $\mathcal{ALCQI}$, which is the most expressive DL supported so far by DL-based reasoning systems. Though less expressive than $\mathcal{DLR}_{ifd}$, the DL $\mathcal{ALCQI}$ preserves enough semantics to keep reasoning about UML class diagrams sound and complete. Exploiting such an encoding, one can use current standard DL-based reasoning systems as core reasoning engines for equipping CASE tools with reasoning capabilities on UML class diagrams.

Categories and Subject Descriptors: [Software Engineering]: Design Tools and Techniques – *UML Class Diagrams*; [Software Engineering]: Software Verification – *Formal Methods*; [Artificial Intelligence]: Knowledge Representation Formalisms and Methods – *Description Logics*; [Mathematical Logic and Formal Languages]: Mathematical Logic – *Complexity Analysis*
General Terms: Design, Theory, Verification
Additional Key Words and Phrases: Automated Reasoning, CASE Tools, Computational Complexity, Description Logics, UML Class Diagrams

1 Introduction

UML (Unified Modeling Language) is the de-facto standard formalism for the analysis and design of software. One of the most important components of UML are *class diagrams*, which model the information on the domain of interest in terms of objects organized in classes and relationships between them[1]. The use of UML in industrial-scale software applications brings about class diagrams that are large and complex to design, analyze, and maintain. To simplify these tasks, sophisticated CASE tools are commonly adopted,

[1] In this paper we deal with UML class diagrams for the *conceptual perspective*, as opposed to the *implementation perspective*, see, e.g., [29].

e.g., Rational Rose[2], Together[3], Poseidon[4], ArgoUML[5] (see also the OMG home page[6]). Such tools support the designer with a user-friendly graphical environment for editing, storing, and accessing multiple UML class diagrams. However, the expressiveness of the UML constructs and their interaction in different parts of complex diagrams may lead to implicit consequences that can go undetected by the designer, and cause various forms of inconsistencies or redundancies in the diagram. This may result in a degradation of the quality of the design and/or increased development times and costs. Hence, it would be highly desirable to equip CASE tools with capabilities to automatically detect relevant formal properties of UML class diagrams, such as inconsistencies and redundancies.

Several works propose to describe UML class diagrams using various kinds of formal systems [24, 25, 26, 21, 36, 12]. Using such formal systems, one can potentially reason on UML class diagrams, and formally prove properties of interest through inference. Hence, in some sense, such works lay the foundation for next generation CASE tools, able to help the designer in understanding the hidden implications of his choices when building a class diagram. In order to select the appropriate kind of formal tool for UML class diagrams, a fundamental question needs to be addressed: What is the computational complexity of reasoning on UML class diagrams? That is, independently of the particular formal tool adopted for describing such diagrams, how difficult is it to reason about them from the computational point of view?

In this paper we address this question in several ways, resorting to results developed through the years in Description Logics (DLs) [1]. These are logics specifically designed for the conceptual representation of an application domain in terms of classes and relationships between classes that admit decidable reasoning.

Our first contribution in this paper is to show that reasoning on UML class diagrams is EXPTIME-hard even under fairly restrictive assumptions, namely: only binary associations, only minimal multiplicity constraints, generalizations (between classes and associations) with disjointness and completeness constraints. We get this result by exhibiting a polynomial reduction from reasoning in the basic DL $\mathcal{ALC}$[7] [1], which is EXPTIME-complete. In particular we show that every $\mathcal{ALC}$ knowledge base can be expressed as a UML class diagram preserving soundness and completeness of reasoning. This possibility is quite surprising, since UML class diagrams apparently have very limited means to express negative and disjunctive information, namely disjointness and covering constraints in generalization hierarchies. Instead $\mathcal{ALC}$ is equipped with unrestricted negation and disjunction. That is, it is able to treat negative information in the same way as positive one, and to reason by cases to fully take into account disjunctive information.

Our second contribution is to establish EXPTIME-membership of reasoning on UML class diagrams, once we disallow the use of arbitrary OCL (first order) constraints [45], which would make reasoning undecidable, but still allow for covering and disjointness constraints on generalization hierarchies. We get this result by using one of the most expressive EXPTIME-decidable DLs studied so far, namely $\mathcal{DLR}_{ifd}$ [15, 14]. This DL is equipped with means to represent n-ary relations, identification constraints (i.e., keys), and functional dependency constraints on components of n-ary relations. This logic was developed with the aim of being able to capture conceptual and object-oriented data models, and is the final result of a series of studies on DLs for reasoning on conceptual data models and object oriented models [8, 20, 9, 13, 17, 19, 30]. The maturity of these studies is testified in the present paper by the fact that we are able to fully capture every UML class diagram as a $\mathcal{DLR}_{ifd}$ knowledge base: the $\mathcal{DLR}_{ifd}$ knowledge base is such that its models are exactly the possible instantiations of the UML class diagram.

Our third contribution is more practically oriented. Indeed the ability of being able to capture UML class diagrams using a DL suggests that we can use DL-based implemented systems to reason on UML class diagrams. However current state-of-the-art DL-based systems [37, 33] are not able to deal with n-ary relations, identification constraints, or functional dependency constraints. These constructs are needed to fully capture in $\mathcal{DLR}_{ifd}$ the semantics of UML class diagrams. However, due to a specific property of $\mathcal{DLR}_{ifd}$ models, namely the tree model property, we can renounce to them while preserving sound and complete reasoning [15]. On the base of this property, we propose a (polynomial) encoding of UML class diagrams in a simpler DL, called $\mathcal{ALCQI}$ [1], which is still EXPTIME-complete, but lacks the features above

[2]`http://www.rational.com/products/rose/`
[3]`http://www.togethersoft.com/`
[4]`http://www.gentleware.com/`
[5]`http://argouml.tigris.org/`
[6]`http://www.omg.org/`
[7]In this paper when we mention reasoning in a DL, we always intend reasoning over a knowledge base expressed in that DL.

that are problematic from an implementation point of view. Such a logic is essentially the most expressive DL that the current DL-based system can support. The encoding in $\mathcal{ALCQI}$, while not preserving entirely the semantics of UML class diagrams, preserves enough of it to keep reasoning sound and complete. Using this encoding we were able to validate on industrial scale examples, namely the UML class diagrams of the Common Information Model (CIM)[8], the feasibility of the idea of using DL-based systems as core inference engines for reasoning on UML class diagrams.

The rest of the paper is organized as follows. In Section 2 we give some preliminary notions on DLs that we use later on. In Section 3, we briefly discuss UML class diagrams giving a natural formalization in first-order logic. In Section 4 we discuss various forms of reasoning on UML class diagrams and show examples of how they can be usefully exploited in order to detect interesting properties of the diagram. In Section 5 we present our EXPTIME-hardness result for reasoning on UML class diagrams, by showing a polynomial reduction from reasoning in the EXPTIME-complete DL $\mathcal{ALC}$. In Section 6 we show how UML class diagrams not including general OCL constraints, but including covering and disjointness constraints on generalization hierarchies, can be fully captured in the EXPTIME-complete DL $\mathcal{DLR}_{ifd}$, thus giving an EXPTIME upper bound for reasoning on UML class diagrams. In Section 7 we show how UML class diagrams can be expressed in the simpler DL $\mathcal{ALCQI}$, preserving enough semantics to keep reasoning on them sound and complete. In Section 8 we discuss our experience in using state-of-the art DL-based reasoning systems for reasoning on the CIM UML class diagrams. Finally, in Section 9, we draw some conclusions.

2 Description Logics

Description Logics (DLs) are logics tailored towards representing knowledge in terms of classes and relationships between classes. Formally they are a well behaved fragment of first order logic (FOL) equipped with decidable reasoning. In DLs, the domain of interest is modeled by means of *concepts* and *relationships*, which denote classes of objects and relations, respectively. Generally speaking, a DL is formed by three basic components:

- a *description language*, which specifies how to construct complex concept and relation expressions (also called simply concepts and relations), by starting from a set of atomic symbols and by applying suitable constructors,
- a *knowledge specification mechanism*, which specifies how to construct a DL knowledge base, in which properties of concepts and relations are asserted, and
- a set of *automatic reasoning procedures* provided by the DL.

The set of allowed constructors characterizes the expressive power of the description language. Various languages have been considered by the DL community, and numerous works investigate the relationship between expressive power and computational complexity of reasoning (see [23] for a survey). The research on these logics has resulted in a number of automated reasoning systems [38, 40, 31], which have been successfully tested in various application domains (see e.g., [43, 46, 42]).

In this Section we briefly review three Description Logics that we will consider in the rest of the paper, namely $\mathcal{DLR}_{ifd}$ [15], $\mathcal{ALCQI}$ [16] and $\mathcal{ALC}$ [1].

2.1 The Description Logic $\mathcal{DLR}_{ifd}$

$\mathcal{DLR}_{ifd}$ is a DL that is characterized by the ability of representing n-ary relations, functional dependencies on n-ary relations, and identification constraints on concepts [15, 14]. The basic elements of $\mathcal{DLR}_{ifd}$ are *concepts* (unary relations), and *n-ary relations*. Let P and A denote atomic relations (of given arity between 2 and n_{max}) and atomic concepts respectively. $\mathcal{DLR}_{ifd}$ relations, denoted by R, and $\mathcal{DLR}_{ifd}$ concepts, denoted by C, are built according to the following syntax:

$$\begin{array}{rcl} R & ::= & \top_n \mid P \mid (i/n\!:\!C) \mid \neg R \mid R_1 \sqcap R_2 \\ C & ::= & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k\,[i]R) \end{array}$$

[8]`http://www.dmtf.org/standards/standard_cim.php/`

$$
\begin{array}{rcl@{\qquad}rcl}
\top_n^{\mathcal{I}} & \subseteq & (\Delta^{\mathcal{I}})^n & \top_1^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \\
P^{\mathcal{I}} & \subseteq & \top_n^{\mathcal{I}} & A^{\mathcal{I}} & \subseteq & \Delta^{\mathcal{I}} \\
(i/n\!:\!C)^{\mathcal{I}} & = & \{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\} & (\neg C)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(\neg R)^{\mathcal{I}} & = & \top_n^{\mathcal{I}} \setminus R^{\mathcal{I}} & (C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\
(R_1 \sqcap R_2)^{\mathcal{I}} & = & R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}} & (\leq k\,[i]R)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} \mid \sharp\{t \in R_1^{\mathcal{I}} \mid t[i] = a\} \leq k\}
\end{array}
$$

Figure 1: Semantic rules for $\mathcal{DLR}_{ifd}$ (P, R, R_1, and R_2 have arity n)

where i denotes a component of a relation, i.e., an integer between 1 and n_{max}, n denotes the *arity* of a relation, i.e., an integer between 2 and n_{max}, and k denotes a non-negative integer. We consider only concepts and relations that are *well-typed*, which means that (i) only relations of the same arity n are combined to form expressions of type $R_1 \sqcap R_2$ (which inherit the arity n), and (ii) $i \leq n$ whenever i denotes a component of a relation of arity n. We also make use of the following standard abbreviations:

$$
\begin{array}{rcl}
C_1 \sqcup C_2 & \text{for} & \neg(\neg C_1 \sqcap \neg C_2) \\
C_1 \Rightarrow C_2 & \text{for} & \neg C_1 \sqcup C_2 \\
(\geq k\,[i]R) & \text{for} & \neg(\leq k{-}1\,[i]R) \\
\exists[i]R & \text{for} & (\geq 1\,[i]R) \\
\forall[i]R & \text{for} & \neg\exists[i]\neg R
\end{array}
$$

Moreover, we abbreviate $(i/n\!:\!C)$ with $(i\!:\!C)$ when n is clear from the context.

As usual in DLs, a $\mathcal{DLR}_{ifd}$ *Knowledge Base* (KB) is constituted by a finite set of *inclusion assertions*. In $\mathcal{DLR}_{ifd}$, these assertions have one of the forms:

$$R_1 \sqsubseteq R_2 \qquad\qquad C_1 \sqsubseteq C_2$$

with R_1 and R_2 of the same arity.

Besides inclusion assertions, $\mathcal{DLR}_{ifd}$ KBs allow for assertions expressing identification constraints and functional dependencies. An *identification assertion* on a concept C has the form:

$$(\mathbf{id}\ C\ [i_1]R_1, \ldots, [i_h]R_h)$$

where each R_j is a relation, and each i_j denotes one component of R_j. Intuitively, such an assertion states that no two different instances of C agree on the participation to $R_1, \ldots, R_h$. In other words, if a is an instance of C that is the i_j-th component of a tuple t_j of R_j, for $j \in \{1, \ldots, h\}$, and b is an instance of C that is the i_j-th component of a tuple s_j of R_j, for $j \in \{1, \ldots, h\}$, and for each j, t_j agrees with s_j in all components different from i_j, then a and b coincide.

A *functional dependency assertion* on a relation R has the form:

$$(\mathbf{fd}\ R\ i_1, \ldots, i_h \rightarrow j)$$

where $h \geq 2$, and $i_1, \ldots, i_h, j$ denote components of R. The assertion imposes that two tuples of R that agree on the components $i_1, \ldots, i_h$, agree also on the component j.

Note that unary functional dependencies (i.e., functional dependencies with $h = 1$) are ruled out in $\mathcal{DLR}_{ifd}$, since these lead to undecidability of reasoning [15]. Note also that the right hand side of a functional dependency contains a single element. However, this is not a limitation, because any functional dependency with more than one element in the right hand side can always be split into several dependencies of the above form.

As usual in DLs, the semantics of $\mathcal{DLR}_{ifd}$ is specified through the notion of interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of a $\mathcal{DLR}_{ifd}$ KB $\mathcal{K}$ is constituted by an *interpretation domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$ that assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and to each relation R of arity n a subset $R^{\mathcal{I}}$ of $(\Delta^{\mathcal{I}})^n$, such that the conditions in Figure 1 are satisfied. In the figure, $t[i]$ denotes the i-th component of tuple t, and $\sharp S$ denotes the cardinality of the set S. Observe that $\top_1$ denotes the interpretation domain, while $\top_n$, for $n > 1$, does *not* denote the n-Cartesian product of the domain, but only a subset of it that covers all relations of arity n. It follows, from this property, that the "$\neg$" constructor on relations is used to express difference of relations, rather than complement.

To specify the semantics of a KB we first define when an interpretation satisfies an assertion as follows:

- An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $R_1 \sqsubseteq R_2$ (resp. $C_1 \sqsubseteq C_2$) if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$ (resp. $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$).
- An interpretation $\mathcal{I}$ *satisfies* the assertion $(\mathbf{id}\ C\ [i_1]R_1, \ldots, [i_h]R_h)$ if for all $a, b \in C^{\mathcal{I}}$ and for all $t_1, s_1 \in R_1^{\mathcal{I}}, \ldots, t_h, s_h \in R_h^{\mathcal{I}}$ we have that:

$$\left.\begin{array}{l} a = t_1[i_1] = \cdots = t_h[i_h], \\ b = s_1[i_1] = \cdots = s_h[i_h], \\ t_j[i] = s_j[i], \text{ for } j \in \{1, \ldots, h\}, \text{ and for } i \neq i_j \end{array}\right\} \text{ implies } a = b$$

- An interpretation $\mathcal{I}$ *satisfies* the assertion $(\mathbf{fd}\ R\ i_1, \ldots, i_h \rightarrow j)$ if for all $t, s \in R^{\mathcal{I}}$, we have that:

$$t[i_1] = s[i_1],\ \ldots,\ t[i_h] = s[i_h] \quad \text{implies} \quad t[j] = s[j]$$

An interpretation that satisfies all assertions in a KB $\mathcal{K}$ is called a *model* of $\mathcal{K}$.

We say that a KB $\mathcal{K}$ is *satisfiable* if there exists a model of $\mathcal{K}$. A concept C is *satisfiable* in a KB $\mathcal{K}$ if there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}}$ is non-empty. An assertion α is *logically implied* by $\mathcal{K}$ if all models of $\mathcal{K}$ satisfy α. It can be shown that all these reasoning tasks, namely KB satisfiability, concept satisfiability in a KB, and logical implication, are mutually reducible (in polynomial time).

One of the distinguishing features of DLs is that they are designed so as to admit reasoning procedures that are sound and complete with respect to the semantics and decidable. In particular, reasoning (i.e., KB satisfiability, concept satisfiability in a KB, and logical implication) in $\mathcal{DLR}_{ifd}$ is EXPTIME-complete [15, 14].

2.2 The Description Logics $\mathcal{ALCQI}$ and $\mathcal{ALC}$

$\mathcal{ALCQI}$ [1] is a rich DL that represents knowledge in terms of concepts (classes) and roles (binary relations). It can be seen as a fragment of $\mathcal{DLR}_{ifd}$ where relations are restricted to be binary and KBs are restricted to be a finite set of inclusion assertions on concepts only (no inclusion assertions on relations, and no identification assertions, and obviously no functional dependency assertions since they require a relation of arity at least three).

Let P and A denote atomic roles (binary relations) and atomic concepts respectively. $\mathcal{ALCQI}$ roles, denoted by R, and $\mathcal{ALCQI}$ concepts, denoted by C, are built according to the following syntax:

$$\begin{array}{lcl} R & ::= & P \mid P^- \\ C & ::= & A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k\, R.C) \end{array}$$

Additionally, we make use of the standard abbreviations below:

$$\begin{array}{rll} \bot & \text{for} & A \sqcap \neg A \qquad \text{(where } A \text{ is any atomic concept)} \\ \top & \text{for} & \neg\bot \\ C_1 \sqcup C_2 & \text{for} & \neg(\neg C_1 \sqcap \neg C_2) \\ C_1 \Rightarrow C_2 & \text{for} & \neg C_1 \sqcup C_2 \\ (\geq k\, R.C) & \text{for} & \neg(\leq k-1\, R.C) \\ \exists R.C & \text{for} & (\geq 1\, R.C) \\ \forall R.C & \text{for} & \neg\exists R.\neg C \end{array}$$

An $\mathcal{ALCQI}$ KB is constituted by a finite set of *inclusion assertions* of the form $C_1 \sqsubseteq C_2$, with C_1 and C_2 arbitrary concept expressions.

Notably $\mathcal{ALCQI}$ includes *inverse roles* P^-, which allow for talking about the inverse of a relation, and *qualified number restrictions*, which are the most general form of cardinality constraints on roles. The

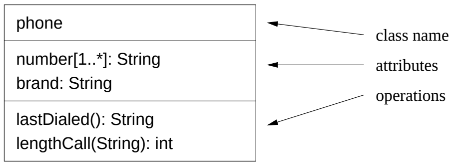

Figure 2: Representation of a class in UML

semantics of $\mathcal{ALCQI}$ constructs and KBs is analogous to that of $\mathcal{DLR}_{ifd}$. In particular the semantic rules for inverse roles and qualified number restrictions are as follows:

$$\begin{array}{rcl}(P^-)^\mathcal{I} & = & \{(a,a') \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \mid (a',a) \in P^\mathcal{I}\} \\ (\leq k\, R.C)^\mathcal{I} & = & \{a \in \Delta^\mathcal{I} \mid \sharp\{a' \in \Delta^\mathcal{I} \mid (a,a') \in R^\mathcal{I} \wedge a' \in C^\mathcal{I}\} \leq k\}\end{array}$$

We can define KB satisfiability, concept satisfiability in a KB, and logical implication, as for $\mathcal{DLR}_{ifd}$. Moreover, as for $\mathcal{DLR}_{ifd}$, reasoning (i.e., KB satisfiability, concept satisfiability in a KB, and logical implication) in a $\mathcal{ALCQI}$ KB is EXPTIME-complete [1].

Finally we turn to $\mathcal{ALC}$ [1]. This is a simpler DL, obtained from $\mathcal{ALCQI}$ by dropping inverse roles and restricting qualified number restrictions to existential restrictions only. The syntax of $\mathcal{ALC}$ concept is thus as follows:

$$C \ ::= \ A \ \mid \ \neg C \ \mid \ C_1 \sqcap C_2 \ \mid \ \exists P.C$$

We also introduce the standard abbreviations:

$$\begin{array}{rcl} C_1 \sqcup C_2 & \text{for} & \neg(\neg C_1 \sqcap \neg C_2) \\ \forall P.C & \text{for} & \neg\exists P.\neg C \end{array}$$

The semantics of the existential restrictions is

$$(\exists P.C)^\mathcal{I} = \{a \in \Delta^\mathcal{I} \mid \exists b.(a,b) \in P^\mathcal{I} \wedge b \in C^\mathcal{I}\}.$$

The semantics of the other constructs is as in $\mathcal{ALCQI}$. As for $\mathcal{ALCQI}$, an $\mathcal{ALC}$ KB is a finite set of inclusion assertions on $\mathcal{ALC}$ concepts. In spite of its simplicity, reasoning in $\mathcal{ALC}$ KBs is EXPTIME-complete, as for $\mathcal{ALCQI}$ [28, 22, 1].

3 UML class diagrams

UML class diagrams allow for modeling, in a declarative way, the static structure of an application domain, in terms of concepts and relations between them. We concentrate on UML class diagrams for the conceptual perspective [45, 29]. In particular, we do not deal with those features that are relevant for the implementation perspective, such as *public*, *protected*, and *private* qualifiers for operations and attributes. We describe the semantics of each construct of UML class diagrams in terms of first order logic (FOL).

3.1 Classes

A *class* in an UML class diagram denotes a set of objects with common features. A class is graphically rendered as a rectangle divided into three parts (see e.g., Figure 2). The first part contains the *name* of the class, which has to be unique in the whole diagram. The second part contains the *attributes* of the class, each denoted by a name, possibly followed by the *multiplicity*, and with an associated *type*[9], for the attribute values. The third part contains the *operations* of the class, i.e., the operations associated to the objects of the class. Note that both the second and the third part are optional. Formally, a class C corresponds to a FOL unary predicate C.

[9]For simplicity, we do not distinguish between classes, i.e., collection of objects, and types, i.e., collections of values, such as integers, reals, ...

Example 3.1 Figure 2 models the class phone, characterized by the attributes number and brand, both of type String, and by the operations lastDialed(), which returns the last number called, and lengthCall(String), which returns the duration time of the call given as input.

An *attribute* a of type T for a class C associates to each instance of C a set of instances of T. Attributes are unique within a class, but two classes may have the same attribute, possibly of different types. An optional multiplicity $[i..j]$ for a specifies that a associates to each instance of C at least i and most j instances of T. When there is no upper bound on the multiplicity, the symbol $*$ is used for j. When the multiplicity is missing, [1..1] is assumed, i.e., the attribute is *mandatory* and *single-valued*. For example, the attribute number[1..*]: String in Figure 2 means that each instance of the class has at least one phone number, and possibly more, and that each phone number is an instance of String. Formally, an attribute a of type T for class C corresponds to a binary predicate a for which the following FOL assertion holds:

$$\forall x, y.\ (C(x) \wedge a(x, y)) \supset T(y)$$

i.e., for each instance x of class C, an object y related to x by a is an instance of T. The multiplicity $[i..j]$ associated to the attribute a can be expressed by

$$\forall x.\ C(x) \supset (i \leq \sharp\{y \mid a(x, y)\} \leq j)$$

where $(i \leq \sharp\{y \mid a(x, y)\} \leq j)$ is an abbreviation for the FOL formula with free variable x expressing that there are at least i and at most j different y's such that $a(x, y)$ holds.

An *operation* of a class is a function from the objects of the class to which the operation is associated, and possibly additional parameters, to objects or values. An operation definition for a class C has the form

$$f(P_1, \ldots, P_m) : R$$

where f is the name of the operation, $P_1, \ldots, P_m$ are the types of the m parameters, and R is the type of the result[10]. Observe that in class diagrams the actual definition of the function is not considered, and what is represented is only the *signature* (i.e., the name of the function and the number and the types of parameters, where the object of invocation is an implicit parameter) and the return type of the function.

Formally, such an operation corresponds to an $(1 + m + 1)$-ary predicate $f_{P_1,\ldots,P_m}$, in which the first argument represents the object of invocation, the next m arguments represent the additional parameters, and the last argument represents the result. Observe that the name of the predicate depends on the whole signature, i.e., it includes the types of the parameters.

The predicate $f_{P_1,\ldots,P_m}$ (in the following referred to simply as f, to improve readability) has to satisfy the following FOL assertions:

$$\begin{array}{l}
\forall x, p_1, \ldots, p_m, r.\ f(x, p_1, \ldots, p_m, r) \supset \bigwedge_{i=1}^{m} P_i(p_i) \\
\forall x, p_1, \ldots, p_m, r, r'.\ f(x, p_1, \ldots, p_m, r) \wedge f(x, p_1, \ldots, p_m, r') \supset r = r' \\
\forall x, p_1, \ldots, p_m, r.\ C(x) \wedge f(x, p_1, \ldots, p_m, r) \supset R(r)
\end{array}$$

The first assertion imposes the correct typing for the parameters, which, observe, depends only on the name of the operation, and not the class to which the operation belongs (in fact, an operation may belong to several classes). The next assertion imposes that invoking the operation on a given object with given parameters determines in a unique way the return value (i.e., the relation corresponding to the operation is in fact a function from the invocation object and the parameters to the result). The last assertion imposes the correct type of the result, depending on the class (and the parameters) to which the operation is applied.

UML allows for the *overloading* of operations, which takes place between two or more functions having the same name but different signatures. *Overriding* takes place when two operations have the same signature, but behave in different ways. In UML class diagrams for the conceptual perspective, where the bodies of operations are not specified, overriding may only show up as a restriction on the type of the result. Observe that the above formalization allows one to have operations with the same name or even with the same name and the same signature in two different classes, and correctly captures overloading and overriding.

[10] Observe that a function returning multiple results can be represented by a function returning a single tuple of results, i.e., a complex value.

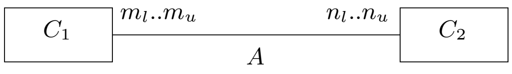

Figure 3: Binary association in UML

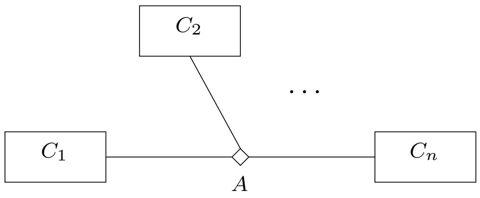

Figure 4: n-ary association in UML

3.2 Associations and aggregations

An *association* in UML is a relation between the instances of two or more classes. Names of associations (as names of classes) are unique in a UML class diagram. A binary association A between two classes C_1 and C_2 is graphically rendered as in Figure 3. The *multiplicity* $n_\ell..n_u$ on the binary association specifies that each instance of the class C_1 can participate at least n_ℓ times and at most n_u times to relation A; $m_\ell..m_u$ has an analogous meaning for the class C_2. When the multiplicity is omitted, it is intended to be $0..*$. Observe that an association can also relate several classes $C_1, C_2, \dots, C_n$, as depicted in Figure 4[11].

Often, an association has a related *association class* that describes properties of the association, such as attributes, operations, etc. A binary association A between two classes C_1 and C_2 with an association class is graphically rendered as in Figure 5, where the class A is the association class related to the association, and r_1 and r_2 are the *role names* of C_1 and C_2 respectively, which specify the role that each class plays within the association A. An association class can also be added to an n-ary association, as in Figure 6.

Example 3.2 The association in Figure 7 models phone calls originating from phones: a phone_call originates from exactly one phone, whereas from a phone 0 or more phone calls can originate. Note that the association origin is characterized by an attribute place of type String.

When the association class is not present, an association A between the instances of classes $C_1, \dots, C_n$, can be formalized as an n-ary predicate A that satisfies the following FOL assertion:

$$\forall x_1, \dots, x_n.\ A(x_1, \dots, x_n) \supset C_1(x_1) \land \dots \land C_n(x_n)$$

An association A between n classes $C_1, \dots, C_n$ that has a related association class is represented by a unary predicate A and n binary predicates $r_1, \dots, r_n$, one for each role name[12], for which the following FOL assertions hold:

$$\begin{array}{l} \forall x, y.\ r_i(x, y) \land A(x) \supset C_i(y), \qquad \text{for } i = 1, \dots, n \\ \forall x.\ A(x) \supset \exists y.\ r_i(x, y), \qquad \text{for } i = 1, \dots, n \\ \forall x, y, y'.\ A(x) \land r_i(x, y) \land r_i(x, y') \supset y = y', \qquad \text{for } i = 1, \dots, n \\ \forall y_1, \dots, y_n, x, x'.\ A(x) \land A(x') \land \bigwedge_{i=1}^{n} (r_i(x, y_i) \land r_i(x', y_i)) \ \supset\ x = x' \end{array}$$

The first assertion types the association; the second and the third ones specify, respectively, that there exists at least one and at most one element playing role r_i for each component of A; the fourth one imposes that there are no two instances of A that represent the same tuple, which is required for the association class to faithfully represent the relation.

[11] In UML, different from other conceptual modeling formalisms, such as Entity-Relationship diagrams [4], multiplicities are look-across cardinality constraints [47]. This makes their use in non-binary associations difficult with respect to both modeling and reasoning.

[12] These binary relations may have the name of the roles of the association, if available in the UML diagram, or an arbitrary name if role names are not available. In any case, we allow for using the same role name in different associations.

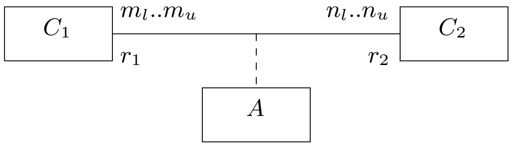

Figure 5: Binary association with association class in UML

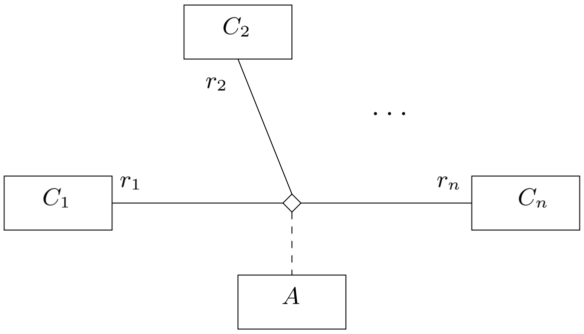

Figure 6: n-ary association with association class in UML

Observe that the formalization for associations differs from the one for attributes, since associations are unique in the diagram, while attributes, being local to classes, are not.

For binary associations without association class (see Figure 3), multiplicities are formalized by the FOL assertions

$$\forall x.\ (n_\ell \leq \sharp\{y \mid A(x,y)\} \leq n_u)$$
$$\forall y.\ (m_\ell \leq \sharp\{x \mid A(x,y)\} \leq m_u)$$

where we have abbreviated FOL formulas expressing cardinality restrictions as before. For binary associations with association class (see Figure 5) the formalization of multiplicities is similar.

A particular kind of binary associations are *aggregations*, which play an important role in UML class diagrams. An aggregation is a binary relation between the instances of two classes, denoting a part-whole relationship, i.e., a relationship that specifies that each instance of a class (the *containing class*) contains a set of instances of another class (the *contained class*). Aggregations have no associated class. An aggregation is graphically rendered as shown in Figure 8, where the diamond indicates the containing class. The aggregation of Figure 8 is represented by a binary predicate G for which the following FOL assertion holds:

$$\forall x, y.\ G(x,y) \supset C_1(x) \wedge C_2(y)$$

where we use the convention that the first argument of the predicate is the containing class. Multiplicities are treated as for binary associations.

Example 3.3 The aggregation in Figure 9 models phone bills containing phone calls: a phone_call is contained in one and only one phone_bill, while a phone_bill contains at least one phone_call.

3.3 Generalization and hierarchies

In UML one can use a *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class. Several generalizations can be grouped together to form a *class hierarchy*, as shown in Figure 10. *Disjointness* and *covering constraints* can also be enforced on a class hierarchy (graphically, by adding suitable labels).

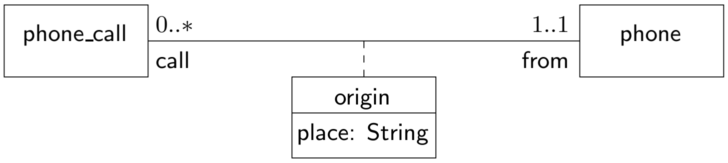

Figure 7: Example of association in UML

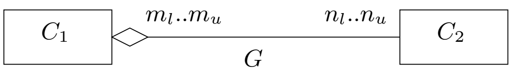

Figure 8: Aggregation in UML

Example 3.4 Figure 11 shows a class hierarchy among the parent class phone and the child classes cell_phone and fixed_phone. In particular, it models the facts that both cell and fixed phones are phones, that no other kind of phones exist and that no phone is at the same time both fixed and cell. Note that, as shown in Figure 13, mobile_calls originate only from cell_phones.

Observe that UML allows for inheritance among association classes, which are treated exactly as all other classes, and for multiple inheritance between classes (including association classes, see Figure 13).

An UML class C generalizing a class C_1 can be formally captured by means of the FOL assertion

$$\forall x.\ C_1(x) \supset C(x)$$

Note that each attribute or operation of C, and each association involving C is correctly inherited by C_1.

A class hierarchy as the one in Figure 10 is formally captured by means of the FOL assertions

$$\forall x.\ C_i(x) \supset C(x), \qquad \text{for } i = 1, \ldots, n$$

Disjointness among $C_1, \ldots, C_n$ is expressed by the FOL assertions

$$\forall x.\ C_i(x) \supset \textstyle\bigwedge_{j=i+1}^{n} \neg C_j(x), \qquad \text{for } i = 1, \ldots, n-1$$

Observe that disjointness of classes is a form of *negative information.*

The *covering constraint* expressing that each instance of C is an instance of at least one of $C_1, \ldots, C_n$ is expressed by

$$\forall x.\ C(x) \supset \textstyle\bigvee_{i=1}^{n} C_i(x)$$

Sometimes, in UML class diagrams, it is assumed that all classes not in the same hierarchy are a priori disjoint. Here we do not force this assumption; instead we allow two classes to have common instances. When needed, disjointness can be enforced by means of explicit disjointness constraints. Similarly, we do not assume that objects in a hierarchy must belong to a single most specific class. Hence, two classes in a hierarchy may have common instances, even when they do not have a common subclass. Again, when needed, suitable covering and disjointness assertions that express the most specific class assumption can be added to a class diagram.

For example, referring to Figure 12, besides the assertions representing the hierarchy, the most-specific-class assumption is captured by means of the FOL assertions

$$\begin{array}{l} \forall x.\ C_1(x) \land C_2(x) \supset C_{12}(x) \\ \forall x.\ C_3(x) \supset \neg C_1(x) \\ \forall x.\ C_3(x) \supset \neg C_2(x) \end{array}$$

3.4 General constraints

Disjointness and covering constraints are in practice the most commonly used constraints in UML class diagrams. However, UML allows for other forms of constraints, specifying class identifiers, functional dependencies for associations, and, more generally through the use of OCL [45], any form of constraint expressible

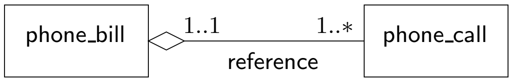

Figure 9: Example of aggregation in UML

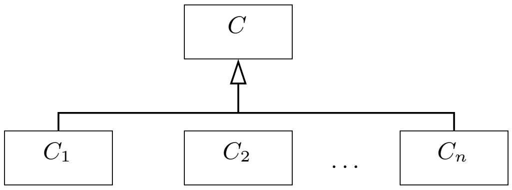

Figure 10: A class hierarchy in UML

in FOL. The use of general OCL constraints moves the semantics of the domain from the diagram to the OCL constraints, and this may compromise the understandability of the diagram. Hence, the use of constraints is typically limited. Also, unrestricted use of OCL constraints makes reasoning on a class diagram undecidable, since it amounts to full FOL reasoning. In the following, we will not consider general constraints.

We conclude the section with an example of a full UML class diagram.

Example 3.5 Figure 13 shows a complete UML class diagram that models phone calls originating from different kinds of phones, and phone bills they belong to[13]. The diagram shows that a mobile_call is a particular kind of phone_call and that the origin of each phone_call is one and only one phone. Additionally, a phone can be only of two different kinds: a fixed_phone or a cell_phone. Mobile calls originate (through the association m_origin) from cell phones. The association m_origin is contained in the binary association origin: hence m_origin inherits the attribute place of association class origin. Finally, a phone_call is referenced in one and only one phone_bill, whereas a phone_bill contains at least one phone_call. In FOL, the diagram is

[13]This diagram is based on an example provided with I.COM, a prototype design tool for conceptual modeling with reasoning support [30].

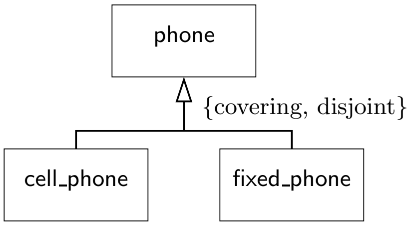

Figure 11: Example of class hierarchy

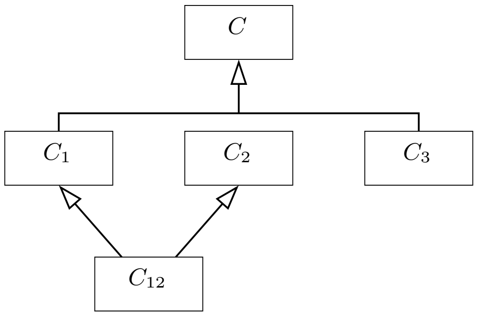

Figure 12: A class hierarchy with most-specific-class assumption

represented as follows.

$$
\begin{array}{l}
\forall x, y.\ \mathsf{origin}(x) \wedge \mathsf{place}(x, y) \supset \mathsf{String}(x)\\
\forall x, y.\ \mathsf{call}(x, y) \wedge \mathsf{origin}(x) \supset \mathsf{phone\_call}(y)\\
\forall x, y.\ \mathsf{from}(x, y) \wedge \mathsf{origin}(x) \supset \mathsf{phone}(y)\\
\forall x.\ \mathsf{origin}(x) \supset \exists y.\ \mathsf{call}(x, y)\\
\forall x.\ \mathsf{origin}(x) \supset \exists y.\ \mathsf{from}(x, y)\\
\forall x, y, y'.\ \mathsf{origin}(x) \wedge \mathsf{call}(x, y) \wedge \mathsf{call}(x, y') \supset y = y'\\
\forall x, y, y'.\ \mathsf{origin}(x) \wedge \mathsf{from}(x, y) \wedge \mathsf{from}(x, y') \supset y = y'\\
\forall x, x', y_1, y_2.\ \mathsf{origin}(x) \wedge \mathsf{origin}(x') \wedge \mathsf{call}(x, y_1) \wedge \mathsf{call}(x', y_1)\\
\quad \wedge \mathsf{from}(x, y_2) \wedge \mathsf{from}(x', y_2) \supset x = x'\\
\forall x.\ 1 \leq \sharp\{y \mid \mathsf{origin}(x, y) \wedge \mathsf{call}(x, y)\} \leq 1\\
\forall x, y.\ \mathsf{call}(x, y) \wedge \mathsf{m\_origin}(x) \supset \mathsf{mobile\_call}(y)\\
\forall x, y.\ \mathsf{from}(x, y) \wedge \mathsf{m\_origin}(x) \supset \mathsf{cell\_phone}(y)\\
\forall x.\ \mathsf{m\_origin}(x) \supset \exists y.\ \mathsf{call}(x, y)\\
\forall x.\ \mathsf{m\_origin}(x) \supset \exists y.\ \mathsf{from}(x, y)\\
\forall x, y, y'.\ \mathsf{m\_origin}(x) \wedge \mathsf{call}(x, y) \wedge \mathsf{call}(x, y') \supset y = y'\\
\forall x, y, y'.\ \mathsf{m\_origin}(x) \wedge \mathsf{from}(x, y) \wedge \mathsf{from}(x, y') \supset y = y'\\
\forall x, x', y_1, y_2.\ \mathsf{m\_origin}(x) \wedge \mathsf{m\_origin(x')} \wedge \mathsf{call}(x, y_1) \wedge \mathsf{call}(x', y_1)\\
\quad \wedge \mathsf{from}(x, y_2) \wedge \mathsf{from}(x', y_2) \supset x = x'\\
\forall x, y, .\ \mathsf{reference}(x, y) \supset \mathsf{phone\_bill}(x) \wedge \mathsf{phone\_call}(y)\\
\forall x.\ 1 \leq \sharp\{y \mid \mathsf{reference}(x, y)\} \leq 1\\
\forall y.\ 1 \leq \sharp\{x \mid \mathsf{reference}(x, y)\}\\
\forall x.\ \mathsf{mobile\_call}(x) \supset \mathsf{phone\_call}(x)\\
\forall x.\ \mathsf{m\_origin}(x) \supset \mathsf{origin}(x)\\
\forall x.\ \mathsf{cell\_phone}(x) \supset \mathsf{phone}(x)\\
\forall x.\ \mathsf{fixed\_phone}(x) \supset \mathsf{phone}(x)\\
\forall x.\ \mathsf{cell\_phone}(x) \supset \neg\mathsf{fixed\_phone}(x)\\
\forall x.\ \mathsf{phone}(x) \supset \mathsf{cell\_phone}(x) \vee \mathsf{fixed\_phone}(x)
\end{array}
$$

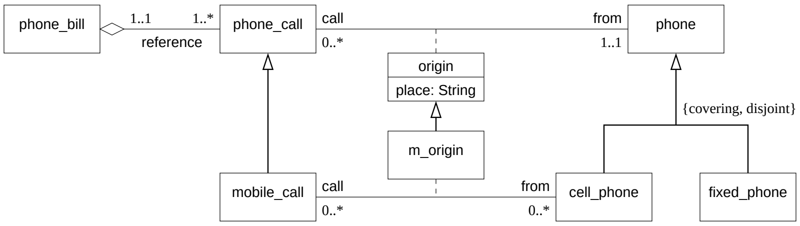

Figure 13: UML class diagram modeling phone calls

Notice that, in the above diagram, one would like to express that each mobile_call is related via the association origin only to instances of cell_phone. Similarly for the other direction of the association. This can be expressed in FOL as follows:

$$\forall x, y.\ \mathsf{mobile\_call}(x) \land \mathsf{origin}(x, y) \supset \mathsf{cell\_phone}(y)$$
$$\forall x, y.\ \mathsf{cell\_phone}(y) \land \mathsf{origin}(x, y) \supset \mathsf{mobile\_call}(x)$$

The association m_origin approximates this, making it explicit in the diagram that mobile_calls and cell_phones are related to each other. It would be possible to express the above constraints directly in the diagram. However, it would require the introduction of additional classes and would make the diagram more complex (see later).

4 Reasoning on UML class diagrams

The design of UML class diagrams modeling complex real world domains is facilitated by automated CASE tools. Currently, CASE tools support the designer with a user friendly graphical environment and provide powerful means to access different kinds of repositories that store information associated to the elements of the developed project. The fact that UML class diagrams can be re-expressed in FOL allows one in principle to go far beyond such a kind of support. Indeed, the designer can use the FOL formalization to formally check relevant properties of class diagrams so as to assess the quality of the diagram according to objective quality criteria. Typical properties of interest are the following (see, e.g., [18, 10]).

Consistency of the whole class diagram A class diagram is *consistent*, if it admits an instantiation, i.e., if its classes can be populated without violating any of the requirements imposed by the diagram. Formally, this means that the corresponding set of FOL assertions admits a model in which at least one class has a nonempty extension. When the diagram is not consistent, the definitions altogether are contradictory, since they do not allow any class to be populated. Observe that the interaction of various types of constraints may make it very difficult to detect inconsistencies.

Class consistency A class is *consistent*, if the class diagram admits an instantiation in which the class has a non-empty set of instances. Intuitively, the class can be populated without violating the requirements imposed by the class diagram. Formally, the set of FOL assertions corresponding to the diagram admits a model in which the class has a nonempty extension. The inconsistency of a class may be due to a design error or due to over-constraining. In any case, the understandability of the diagram is weakened, since the class stands for the empty class, and thus, at the very least, it is inappropriately named. To increase the quality of the diagram, the designer may remove the inconsistency by relaxing some constraints (possibly by correcting errors), or by deleting the class, thus removing redundancy and increasing understandability.

Class subsumption A class C_1 *subsumes* a class C_2, if the class diagram implies that C_1 is a generalization of C_2. Formally, in every model of the set of FOL assertions, the extension of C_1 is a superset of the extension of C_2. Such a subsumption allows one to deduce that properties for C_1 hold also for C_2.

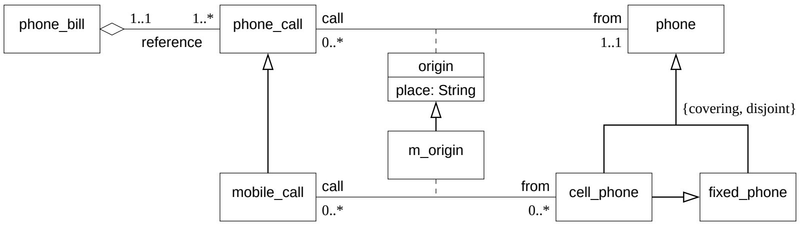

Figure 14: Modified version of the UML class diagram modeling phone calls

This suggests the possible omission of an explicit generalization. Alternatively, if all instances of the more specific class are not supposed to be instances of the more general class, then something is wrong with the diagram, since it is forcing an undesired conclusion. Class subsumption is also the basis for a *classification* of all the classes in a diagram. Such a classification, as in any object-oriented approach, can be exploited in several ways within the modeling process [7].

Class equivalence Two classes are *equivalent* if they denote the same set of instances whenever the requirements imposed by the class diagram are satisfied: in this case one of them is redundant. Determining equivalence of two classes allows for their merging, thus reducing the complexity of the diagram.

Refinement of properties The properties of various classes and associations may interact to yield stricter multiplicities or typing than those explicitly specified in the diagram. Detecting such cases allows the designer for refining the class diagram by making such properties explicit, thus enhancing the readability of the diagram.

Implicit consequences More generally, a property is an *(implicit) consequence* of a class diagram if it holds whenever all requirements imposed by the diagram are satisfied. Formally, this means that the property is logically implied by the FOL assertions corresponding to the class diagram, i.e., the property holds in every model of the assertions. Determining implicit consequences is useful on the one hand to reduce the complexity of the diagram by removing those parts that implicitly follow from other ones, and on the other hand it can be used to make properties explicit, thus enhancing its readability. Note that the above properties can be seen as special cases of implicit consequences.

We illustrate the above properties on our running example.

Example 4.1 Consider the UML class diagram shown in Figure 13. By reasoning on such a diagram, one can deduce that the class mobile_call participates to association m_origin with multiplicity 0..1. Indeed, m_origin is included in origin, hence every tuple of m_origin is-a tuple of origin; moreover, since every phone_call participates exactly once to association origin, necessarily every mobile_call participates at most once to association m_origin, since mobile_call is a subclass of phone_call. This is an example of refinement of a multiplicity.

If we add a generalization to the diagram in Figure 13 that asserts that each cell_phone is a fixed_phone (see Figure 14), we get several undesirable properties. First, the class cell_phone is inconsistent, i.e., it has no instances. Indeed, the disjointness constraint asserts that there are no cell phones that are also fixed phones, and since the empty set is the only set that can be at the same time disjoint from and contained in the class fixed_phone, the class cell_phone is equivalent to it. Second, since the class phone is made up by the union of classes cell_phone and fixed_phone, and since cell_phone is inconsistent, the classes phone and fixed_phone are equivalent, hence one of them is redundant. Finally, since there are no cell phones, there are no pairs in the association m_origin, and so it is inconsistent too. The class mobile_call is not inconsistent since it can be populated by instances that do not participate to association m_origin. Note that, if we added the constraint

$$\forall x, y.\ \mathsf{mobile\_call}(x) \land \mathsf{origin}(x, y) \supset \mathsf{cell\_phone}(y)$$

discussed in Example 3.5, considering the minimal multiplicity 1 of mobile_call in origin, mobile_call would be inconsistent too.

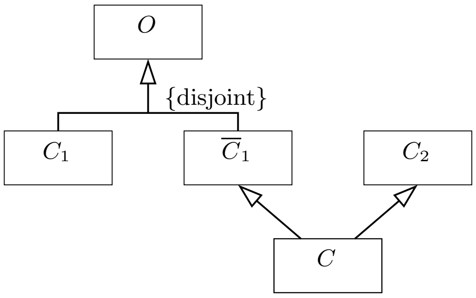

Figure 15: Reduction from class subsumption to class consistency

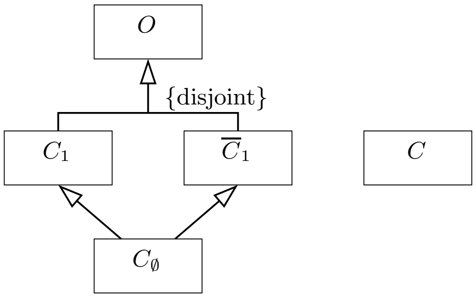

Figure 16: Reduction from class consistency to class subsumption

The example above shows that reasoning is required in order to understand whether the class diagram enjoys required properties. Considering the high complexity of industrial software, it can be very difficult to verify the properties of a UML class diagram and to guarantee that they are preserved during the design of the diagram. Thus, it would be highly desirable to have CASE tools equipped with automated reasoning capabilities to support the designer. One possibility would be to resort to a full FOL theorem prover [3, 35]. While certainly worth exploring, due to the intrinsic undecidability of FOL, such an approach cannot provide completely automated techniques for reasoning on UML class diagrams. Here instead we follow a different approach, and we investigate the intrinsic complexity of reasoning on UML class diagrams, taking into account restricted forms of constraints. We characterize the complexity by resorting to DLs [1]. On the one hand, we show that reasoning on UML class diagrams (that include as constraints only disjointness and covering) is EXPTIME-hard. On the other hand, we show that EXPTIME-decidable DLs can fully capture UML class diagrams with restricted forms of FOL constraints. This demonstrates that DL reasoning algorithms are ideal candidates for being used as core reasoning engines in advanced CASE tools with reasoning support.

5 Hardness of reasoning on UML class diagrams

From the formal point of view, one can see that the reasoning tasks necessary for checking the various properties discussed in Section 4 are mutually reducible to each other. As an example, we show the mutual reducibility between class consistency and class subsumption.

Given a class diagram with classes C_1 and C_2, if we want to check whether C_1 subsumes C_2, then we can add to the class diagram the part depicted in Figure 15, where O, C, and $\overline{C}_1$ are new classes, and check whether C is inconsistent. Indeed, if C_1 subsumes C_2, there can be no object that is both in $\overline{C}_1$, hence not in C_1, and in C_2, and so C is inconsistent. Conversely, if C_1 does not subsume C_2, this means that there is a model $\mathcal{I}$ of the (original) diagram with an object o not in C_1 but in C_2. We can take the extension of $\overline{C}_1$ in $\mathcal{I}$ to include o. Hence C has a nonempty extension in $\mathcal{I}$ and is consistent.

Given a class diagram with a class C, if we want to check whether C is inconsistent, then we can add to the class diagram the part depicted in Figure 16, where O, C_1, $\overline{C}_1$, and $C_\emptyset$ are new classes, and check whether $C_\emptyset$ subsumes C. Indeed, since C_1 and $\overline{C}_1$ are disjoint, $C_\emptyset$ denotes the empty class, and so C is inconsistent if and only if it is subsumed by $C_\emptyset$.

Hence in the following, without loss of generality, we focus on class consistency only. Specifically, we show that class consistency in UML class diagrams with only disjointness and covering constraints is EXPTIME-hard. We prove the claim by a reduction from concept satisfiability in $\mathcal{ALC}$ KBs, which is EXPTIME-hard [1]. We proceed in two steps:

1. First, we show that we can restrict the attention to a syntactically restricted form of $\mathcal{ALC}$ called $\mathcal{ALC}^-$ below.

2. Then, we describe a reduction from atomic concept satisfiability in $\mathcal{ALC}^-$ KBs to class consistency in UML class diagrams.

In the following, we call *primitive* an inclusion assertion of the form $A \sqsubseteq C$, where A is an atomic concept and C is an arbitrary concept. The DL $\mathcal{ALC}^-$ is obtained from $\mathcal{ALC}$ by dropping intersection and allowing only for complex concepts built with at most one construct of $\mathcal{ALC}$, i.e.,

$$C \ ::= \ A \ \mid \ \neg A \ \mid \ A_1 \sqcup A_2 \ \mid \ \exists P.A \ \mid \ \forall P.A$$

where A denotes an atomic concept and P denotes an atomic role. An $\mathcal{ALC}^-$ KB is a finite set of primitive $\mathcal{ALC}^-$ inclusion assertions, i.e., inclusion assertions of the form $A \sqsubseteq C$ where C is an $\mathcal{ALC}^-$ concept.

By exploiting a result in [11] we can reduce concept satisfiability in $\mathcal{ALC}$ KBs to atomic concept satisfiability in $\mathcal{ALC}^-$ KBs.

Lemma 5.1 *Concept satisfiability in an $\mathcal{ALC}$ KB can be linearly reduced to atomic concept satisfiability in a primitive $\mathcal{ALC}$ KB.*

Proof. Let $\mathcal{K}$ be an $\mathcal{ALC}$ KB and C an $\mathcal{ALC}$ concept. By a result in [11], C is satisfiable in $\mathcal{K}$ if and only if $A_T \sqcap C$ is satisfiable in the KB $\mathcal{K}_1$ consisting of the single assertion

$$A_T \ \sqsubseteq \ \bigsqcap_{C_1 \sqsubseteq C_2 \in \mathcal{K}} (\neg C_1 \sqcup C_2) \sqcap \bigsqcap_{1 \leq i \leq n} \forall P_i.A_T$$

where A_T is a new atomic concept and $P_1, \ldots, P_n$ are all atomic roles appearing in $\mathcal{K}$ and C.

Then, in order to reduce the problem to atomic concept satisfiability, we introduce a new atomic concept A_C, and check its satisfiability in $\mathcal{K}_2 = \mathcal{K}_1 \cup \{A_C \sqsubseteq A_T \sqcap C\}$. Indeed, if $\mathcal{K}_1$ admits a model $\mathcal{I}$ such that $(A_T \sqcap C)^\mathcal{I} \neq \emptyset$, then by extending $\mathcal{I}$ so that $A_C^\mathcal{I} = (A_T \sqcap C)^\mathcal{I}$, we get a model of $\mathcal{K}_2$ in which $A_C^\mathcal{I} \neq \emptyset$. Conversely, every model of $\mathcal{K}_2$ with $A_C^\mathcal{I} \neq \emptyset$ is also a model of $\mathcal{K}_1$ with $(A_T \sqcap C)^\mathcal{I} \neq \emptyset$. $\square$

Below we assume, without loss of generality, that primitive $\mathcal{ALC}$ KBs are in negation normal form. Indeed, every primitive $\mathcal{ALC}$ KB can be put in negation normal form in linear time.

Given a primitive $\mathcal{ALC}$ KB $\mathcal{K}$ (in negation normal form), we construct a primitive $\mathcal{ALC}^-$ KB $\mathcal{K}'$ by recursively replacing each $\mathcal{ALC}$ assertion in K that is not already a (primitive) $\mathcal{ALC}^-$ assertion as follows:

1. $A \sqsubseteq C_1 \sqcap C_2$ is replaced by $A \sqsubseteq C_1$ and $A \sqsubseteq C_2$;

2. $A \sqsubseteq C_1 \sqcup C_2$ is replaced by $A \sqsubseteq A_1 \sqcup A_2$, $A_1 \sqsubseteq C_1$ and $A_2 \sqsubseteq C_2$, where A_1 and A_2 are new atomic concepts;

3. $A \sqsubseteq \forall P.C$ is replaced by $A \sqsubseteq \forall P.A_1$ and $A_1 \sqsubseteq C$, where A_1 is a new atomic concept;

4. $A \sqsubseteq \exists P.C$ is replaced by $A \sqsubseteq \exists P.A_1$ and $A_1 \sqsubseteq C$, where A_1 is a new atomic concept.

Notice that the number of such replacements is finite (in fact linear), since for each occurrence of an $\mathcal{ALC}$ construct in $\mathcal{K}$ at most one replacement is done.

Lemma 5.2 *Given a primitive $\mathcal{ALC}$ KB $\mathcal{K}$, the size of the (primitive) $\mathcal{ALC}^-$ KB $\mathcal{K}'$ obtained as above is linear in the size of $\mathcal{K}$.*

Proof. By construction. $\square$

Lemma 5.3 *An atomic concept A_0 is satisfiable in a primitive $\mathcal{ALC}$ KB $\mathcal{K}$ if and only if A_0 is satisfiable in the (primitive) $\mathcal{ALC}^-$ KB $\mathcal{K}'$ obtained as above.*

Proof. We show that A_0 in $\mathcal{K}$ is satisfiable if and only if it is satisfiable in the KB obtained after n replacements, for each $n > 0$. We proceed by induction on n. Let $\mathcal{K}_i$ be the KB obtained from $\mathcal{K}$ after i replacements.

Base case: $\mathcal{K}_0 = \mathcal{K}$ (obvious).

Inductive case: By inductive hypothesis, we have that A_0 is satisfiable in $\mathcal{K}$ if and only if A_0 is satisfiable in $\mathcal{K}_n$. We prove that, given a model $\mathcal{I}$ of $\mathcal{K}_n$ with $A_0^{\mathcal{I}} \neq \emptyset$ we can construct a model $\mathcal{J}$ of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$, and conversely, that every model $\mathcal{J}$ of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$ is also a model of $\mathcal{K}_n$.

1. If the next step to be applied is the replacement of $A \sqsubseteq C_1 \sqcap C_2$ with $A \sqsubseteq C_1$ and $A \sqsubseteq C_2$, then:

$$\mathcal{K}_{n+1} = \mathcal{K}_n \cup \{A \sqsubseteq C_1, A \sqsubseteq C_2\} \setminus \{A \sqsubseteq C_1 \sqcap C_2\}$$

   In this case, the statement is obvious, since $\{A \sqsubseteq C_1 \sqcap C_2\}$ logically implies $\{A \sqsubseteq C_1, A \sqsubseteq C_2\}$ and vice-versa. Therefore $\mathcal{K}_{n+1}$ and $\mathcal{K}_n$ have the same models.

2. If the next step consists in the replacement of $A \sqsubseteq C_1 \sqcup C_2$ by $A \sqsubseteq A_1 \sqcup A_2$, $A_1 \sqsubseteq C_1$ and $A_2 \sqsubseteq C_2$, where A_1 and A_2 are new atomic concepts, we get:

$$\mathcal{K}_{n+1} = \mathcal{K}_n \cup \{A \sqsubseteq A_1 \sqcup A_2, A_1 \sqsubseteq C_1, A_2 \sqsubseteq C_2\} \setminus \{A \sqsubseteq C_1 \sqcup C_2\}$$

   "$\Leftarrow$" Let $\mathcal{I}$ be a model of $\mathcal{K}_n$ with $A_0^{\mathcal{I}} \neq \emptyset$, let $\mathcal{J}$ coincide with $\mathcal{I}$ on all atomic concepts and roles in $\mathcal{K}_n$, and additionally let $A_1^{\mathcal{J}} = C_1^{\mathcal{I}}$ and $A_2^{\mathcal{J}} = C_2^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $A \sqsubseteq C_1 \sqcup C_2$, we have by construction that $\mathcal{J}$ satisfies $A \sqsubseteq A_1 \sqcup A_2$, $A_1 \sqsubseteq C_1$ and $A_2 \sqsubseteq C_2$, and hence is a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$.

   "$\Rightarrow$" Let $\mathcal{J}$ be a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$. Since it satisfies $A \sqsubseteq A_1 \sqcup A_2$, for each instance $a \in A^{\mathcal{J}}$, we have $a \in A_1^{\mathcal{J}}$ or $a \in A_2^{\mathcal{J}}$. In the first case, by $A_1 \sqsubseteq C_1$, we get $a \in C_1^{\mathcal{J}}$; in the second case, by $A_2 \sqsubseteq C_2$, we get $a \in C_2^{\mathcal{J}}$. Therefore, $\mathcal{J}$ satisfies $A \sqsubseteq C_1 \sqcup C_2$, and hence is a model of $\mathcal{K}_n$ as well.

3. If the next step to be applied is to replace $A \sqsubseteq \forall P.C$ by $A \sqsubseteq \forall P.A_1$ and $A_1 \sqsubseteq C$, where A_1 is a new atomic concept, we have:

$$\mathcal{K}_{n+1} = \mathcal{K}_n \cup \{A \sqsubseteq \forall P.A_1, A_1 \sqsubseteq C\} \setminus \{A \sqsubseteq \forall P.C\}$$

   "$\Leftarrow$" Let $\mathcal{I}$ be a model of $\mathcal{K}_n$ with $A_0^{\mathcal{I}} \neq \emptyset$, let $\mathcal{J}$ coincide with $\mathcal{I}$ on all atomic concepts and roles in $\mathcal{K}_n$, and additionally let $A_1^{\mathcal{J}} = C^{\mathcal{I}}$. Since $\mathcal{I}$ satisfies $A \sqsubseteq \forall P.C$, we have by construction that $\mathcal{J}$ satisfies $A \sqsubseteq \forall P.A_1$ and $A_1 \sqsubseteq C$, and hence is a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$.

   "$\Rightarrow$" Let $\mathcal{J}$ be a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$. Since it satisfies $A \sqsubseteq \forall P.A_1$, for each instance $a \in A^{\mathcal{J}}$, if a is connected via role P to an instance a', then $a' \in A_1^{\mathcal{J}}$. By $A_1 \sqsubseteq C$, we have that $a' \in C^{\mathcal{J}}$. Therefore $\mathcal{J}$ satisfies $A \sqsubseteq \forall P.C$, and hence is a model of K_n as well.

4. If the next step to be applied is to replace $A \sqsubseteq \exists P.C$ by $A \sqsubseteq \exists P.A_1$ and $A_1 \sqsubseteq C$, where A_1 is a new atomic concept, we have:

$$\mathcal{K}_{n+1} = \mathcal{K}_n \cup \{A \sqsubseteq \exists P.A_1, A_1 \sqsubseteq C\} \setminus \{A \sqsubseteq \exists P.C\}$$

   "$\Leftarrow$" Let $\mathcal{I}$ be a model of $\mathcal{K}_n$ with $A_0^{\mathcal{I}} \neq \emptyset$, let $\mathcal{J}$ coincide with $\mathcal{I}$ on all atomic concepts and roles in $\mathcal{K}_n$, and additionally let $A_1^{\mathcal{J}} = C^{\mathcal{J}}$. Since $\mathcal{I}$ satisfies $A \sqsubseteq \exists P.C$, we have by construction that $\mathcal{J}$ satisfies $A \sqsubseteq \exists P.A_1$ and $A_1 \sqsubseteq C$, and hence is a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$.

   "$\Rightarrow$" Let $\mathcal{J}$ be a model of $\mathcal{K}_{n+1}$ with $A_0^{\mathcal{J}} \neq \emptyset$. Since it satisfies $A \sqsubseteq \exists P.A_1$, there exists an instance $a \in A^{\mathcal{J}}$ that is connected via role P to an instance $a' \in A_1^{\mathcal{J}}$. By $A_1 \sqsubseteq C$, we have that $a' \in C^{\mathcal{J}}$. Therefore $\mathcal{J}$ satisfies $A \sqsubseteq \exists P.C$, and hence is a model of $\mathcal{K}_n$ as well.

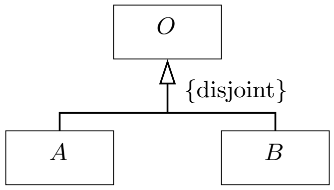

Figure 17: UML encoding of the assertion $A \sqsubseteq \neg B$

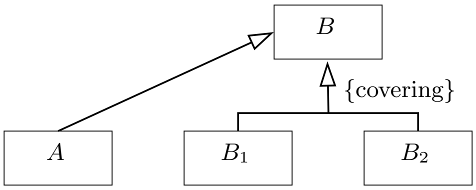

Figure 18: UML encoding of the assertion $A \sqsubseteq B_1 \sqcup B_2$

□

Next, we reduce concept satisfiability in a primitive $\mathcal{ALC}^-$ KB $\mathcal{K}'$ to class consistency in an UML class diagram $\mathcal{D}$. For each atomic concept A in $\mathcal{K}'$, we introduce a class A in $\mathcal{D}$. Additionally, we add a class O that generalizes (possibly indirectly) all classes in $\mathcal{D}$. O is also used to specify disjointness among classes (see later). For each atomic role P, we introduce an association P (with related association class), involving the class O twice. Intuitively, using O in such a way, we do not constrain in any way the classes to which the component instances of P may belong. More classes and associations, as well as generalizations between O and the new classes, are added below as needed.

The assertions in the $\mathcal{ALC}^-$ KB $\mathcal{K}'$ are encoded in the class diagram as follows:

1. For each assertion of the form $A \sqsubseteq B$, we introduce a generalization between the classes A and B (where A is the subclass).
2. For each assertion of the form $A \sqsubseteq \neg B$, we construct the hierarchy in Figure 17, exploiting the superclass O to express disjointness between A and B.
3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$, we introduce an auxiliary class B, and construct the hierarchy in Figure 18. Intuitively, being B a covering of B_1 and B_2, and A a subclass of B, it follows that A is a subclass of the union of B_1 and B_2.

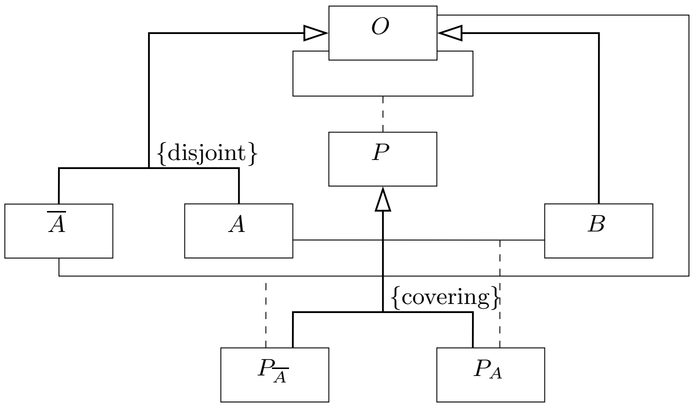

Figure 19: UML encoding of the assertion $A \sqsubseteq \forall P.B$

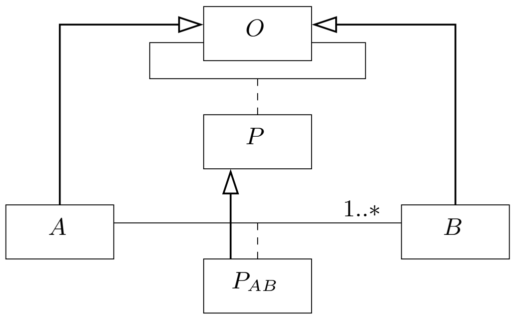

Figure 20: UML encoding of the assertion $A \sqsubseteq \exists P.B$

4. For each assertion of the form $A \sqsubseteq \forall P.B$, we introduce a new class $\overline{A}$ and two new binary associations P_A and $P_{\overline{A}}$ (with their associated classes) and we construct the portion of diagram in Figure 19, where A and $\overline{A}$ are disjoint and there is a generalization with covering constraint between P and its children P_A and $P_{\overline{A}}$. Note that A and B are the components of P_A, whereas $\overline{A}$ and O are the components of $P_{\overline{A}}$. Intuitively, the diagram enforces that each instance of A participating to P is in fact participating to P_A, and hence associated via P to an instance of B.

5. For each assertion of the form $A \sqsubseteq \exists P.B$, we introduce a new binary association P_{AB}, with its associated class, and we construct the portion of diagram shown in Figure 20. Note the proper multiplicity constraint $1..*$ on the participation of A to P_{AB}[14]. Intuitively, this implies that for each instance of A, there exists an instance of B related to it through P_{AB}, and hence through P.

Lemma 5.4 *Given a primitive $\mathcal{ALC}^-$ KB $\mathcal{K}'$, the size of the UML class diagram $\mathcal{D}$ constructed as above is linear in the size of $\mathcal{K}'$.*

Proof. By construction. □

Lemma 5.5 *An atomic concept A is satisfiable in an $\mathcal{ALC}^-$ KB $\mathcal{K}'$ if and only if the class A is consistent in the UML class diagram $\mathcal{D}$ constructed as above.*

Proof. "$\Leftarrow$" Let $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be an instantiation for $\mathcal{D}$ (i.e., a model of the corresponding FOL assertions). We show that $\mathcal{J}$ is also a model of all assertions in $\mathcal{K}'$.

1. For each assertion of the form $A \sqsubseteq B$ in $\mathcal{K}'$, there is a generalization in $\mathcal{D}$ between the child class A and the parent class B. Hence, $\mathcal{J}$ populates A and B so that $A^{\mathcal{J}} \subseteq B^{\mathcal{J}}$.

2. For each assertion of the form $A \sqsubseteq \neg B$ in $\mathcal{K}'$, we have in $\mathcal{D}$ the hierarchy shown in Fig. 17, characterized by a disjointness constraint between A and B. $\mathcal{J}$ assigns to the classes A, B and O the sets $A^{\mathcal{J}}, B^{\mathcal{J}}, O^{\mathcal{J}}$ so that $A^{\mathcal{J}} \subseteq O^{\mathcal{J}}$, $B^{\mathcal{J}} \subseteq O^{\mathcal{J}}$ and $A^{\mathcal{J}} \cap B^{\mathcal{J}} = \emptyset$. From the latter we have that $A^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \setminus B^{\mathcal{J}}$.

3. Each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$ in $\mathcal{K}'$ corresponds in $\mathcal{D}$ to the hierarchy shown in Fig. 18, characterized by a covering constraint among B and its children B_1 and B_2. $\mathcal{J}$ populates the classes A, B, B_1 and B_2, so that $A^{\mathcal{J}} \subseteq B^{\mathcal{J}}$, and $B^{\mathcal{J}} = B_2^{\mathcal{J}} \cup B_2^{\mathcal{J}}$. Hence we get $A^{\mathcal{J}} \subseteq B_1^{\mathcal{J}} \cup B_2^{\mathcal{J}}$.

4. Each assertion of the form $A \sqsubseteq \forall P.B$ in $\mathcal{K}'$ corresponds, in $\mathcal{D}$, to the sub-diagram in Fig. 19. $\mathcal{J}$

[14]In fact, in the case where we also have the assertion $A \sqsubseteq \forall P.B$ for some B, instead of proceeding as in Figure 20, we can simply add the cardinality constraint $1..*$ to the association P_{AB} in Figure 19.

populates it according to the following constraints:

$$\begin{array}{rcl} A^{\mathcal{J}} & \subseteq & O^{\mathcal{J}} \\ \overline{A}^{\mathcal{J}} & \subseteq & O^{\mathcal{J}} \\ A^{\mathcal{J}} \cap \overline{A}^{\mathcal{J}} & = & \emptyset \\ B^{\mathcal{J}} & \subseteq & O^{\mathcal{J}} \\ P^{\mathcal{J}} & \subseteq & O^{\mathcal{J}} \times O^{\mathcal{J}} \\ P_{\overline{A}}^{\mathcal{J}} & \subseteq & \overline{A}^{\mathcal{J}} \times O^{\mathcal{J}} \\ P_{A}^{\mathcal{J}} & \subseteq & A^{\mathcal{J}} \times B^{\mathcal{J}} \\ P_{A}^{\mathcal{J}} & \subseteq & P^{\mathcal{J}} \\ P_{\overline{A}}^{\mathcal{J}} & \subseteq & P^{\mathcal{J}} \\ P^{\mathcal{J}} & \subseteq & P_{A}^{\mathcal{J}} \cup P_{\overline{A}}^{\mathcal{J}} \end{array}$$

From the constraints above, we get that $P_{A}^{\mathcal{J}} \cap P_{\overline{A}}^{\mathcal{J}} = \emptyset$. Therefore, if $x \in A^{\mathcal{J}}$ then for all $x' \in O^{\mathcal{J}}$ if $(x, x') \in P^{\mathcal{J}}$ then $(x, x') \in P_{A}^{\mathcal{J}}$ and therefore $x' \in B^{\mathcal{J}}$, i.e., $A^{\mathcal{J}} \subseteq \{x \in O^{\mathcal{J}} \mid \forall x' \in O^{\mathcal{J}}.\, (x, x') \in P^{\mathcal{J}} \supset x' \in B^{\mathcal{J}}\}$.

5. Each assertion of the form $A \sqsubseteq \exists P.B$ in $\mathcal{K}'$ corresponds, in $\mathcal{D}$, to the sub-diagram shown in Figure 20. $\mathcal{J}$ populates it and satisfies the constraints $A^{\mathcal{J}} \subseteq O^{\mathcal{J}}$, $B^{\mathcal{J}} \subseteq O^{\mathcal{J}}$, $P^{\mathcal{J}} \subseteq O^{\mathcal{J}} \times O^{\mathcal{J}}$, $P_{AB}^{\mathcal{J}} \subseteq P^{\mathcal{J}}$ and $P_{AB}^{\mathcal{J}} \subseteq A^{\mathcal{J}} \times B^{\mathcal{J}}$, and for each $x \in A^{\mathcal{J}}$ we have that $\sharp\{x' \in \Delta^{\mathcal{I}} \mid (x, x') \in P_{AB}^{\mathcal{J}}\} \geq 1$ (mandatory participation constraint). From these we get that for each $x \in A^{\mathcal{J}}$ there exists $x' \in O^{\mathcal{J}}$ such that $(x, x') \in P^{\mathcal{J}}$ and $x' \in B^{\mathcal{J}}$, i.e., $A^{\mathcal{J}} \subseteq \{x \in O^{\mathcal{J}} \mid \exists x' \in O^{\mathcal{J}} (x, x') \in P^{\mathcal{J}} \wedge x' \in B^{\mathcal{J}}\}$.

"$\Rightarrow$" Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of $\mathcal{K}'$ with $A^{\mathcal{I}} \neq \emptyset$. We show that it can be seen as an instantiation of $\mathcal{D}$, once we assign a suitable extension to the auxiliary classes and roles introduced in the construction of $\mathcal{D}$. First, we define $O^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

1. For each assertion of the form $A \sqsubseteq B$ in $\mathcal{K}'$, we have a generalization between classes A and B in $\mathcal{D}$. $\mathcal{I}$ assigns to concepts A and B in $\mathcal{K}'$ the subsets $A^{\mathcal{I}}$ and $B^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, such that $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$, and hence correctly captures the generalization between classes A and B in $\mathcal{D}$.

2. For each assertion of the form $A \sqsubseteq \neg B$ in $\mathcal{K}'$, we have a fragment of $\mathcal{D}$ as in Fig. 17. $\mathcal{I}$ assigns to concepts A and B the subsets $A^{\mathcal{I}}$ and $B^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, such that $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$. Then we have that $A^{\mathcal{I}} \subseteq O^{\mathcal{I}}, B^{\mathcal{I}} \subseteq O^{\mathcal{I}}$ and $A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$, thus correctly capturing the fragment of $\mathcal{D}$.

3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$ in $\mathcal{K}'$, we have a fragment of $\mathcal{D}$ as in Fig. 18. $\mathcal{I}$ assigns to concepts B_1 and B_2 the subsets $B_1^{\mathcal{I}}$ and $B_2^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, respectively, and to A a subset of their union, i.e., $A^{\mathcal{I}} \subseteq B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$. Let us define $B^{\mathcal{I}} = B_1^{\mathcal{I}} \cup B_2^{\mathcal{I}}$. Then $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$, thus correctly capturing the fragment of $\mathcal{D}$.

4. For each assertion of the form $A \sqsubseteq \forall P.B$ in $\mathcal{K}'$, we have a fragment of $\mathcal{D}$ as in Fig. 19. Let us define:

   - $\overline{A}^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
   - $P_{A}^{\mathcal{I}} = \{(x, x') \in P^{\mathcal{I}} \mid x \in A^{\mathcal{I}}\}$
   - $P_{\overline{A}}^{\mathcal{I}} = \{(x, x') \in P^{\mathcal{I}} \mid x \in \overline{A}^{\mathcal{I}}\}$

Then, by $A^{\mathcal{I}} \subseteq \{x \in \Delta^{\mathcal{I}} \mid \forall x' \in \Delta^{\mathcal{I}}. \, (x, x') \in P^{\mathcal{I}} \supset x' \in B^{\mathcal{I}}\}$, we get:

$$\begin{array}{rcl}
A^{\mathcal{I}} & \subseteq & O^{\mathcal{I}} \\
\overline{A}^{\mathcal{I}} & \subseteq & O^{\mathcal{I}} \\
A^{\mathcal{I}} \cap \overline{A}^{\mathcal{I}} & = & \emptyset \\
B^{\mathcal{I}} & \subseteq & O^{\mathcal{I}} \\
P^{\mathcal{I}} & \subseteq & O^{\mathcal{I}} \times O^{\mathcal{I}} \\
P^{\mathcal{I}}_{A} & \subseteq & A^{\mathcal{I}} \times B^{\mathcal{I}} \\
P^{\mathcal{I}} & \subseteq & P^{\mathcal{I}}_{A} \cup P^{\mathcal{I}}_{\overline{A}} \\
P^{\mathcal{I}}_{A} & \subseteq & P^{\mathcal{I}} \\
P^{\mathcal{I}}_{\overline{A}} & \subseteq & P^{\mathcal{I}}
\end{array}$$

thus correctly capturing the fragment of $\mathcal{D}$.

5. For each assertion of the form $A \sqsubseteq \exists P.B$ in $\mathcal{K}'$, we have a fragment of $\mathcal{D}$ as in Fig. 20. Let us define $P^{\mathcal{I}}_{AB} = \{(x, x') \in P^{\mathcal{I}} \mid x \in A^{\mathcal{I}}\}$. Then, by $A^{\mathcal{I}} \subseteq \{x \in \Delta^{\mathcal{I}} \mid \exists x' \in \Delta^{\mathcal{I}}. (x, x') \in P^{\mathcal{I}} \wedge x' \in B^{\mathcal{I}}\}$, we get that for each $x \in A^{\mathcal{I}}$ we have $\sharp\{x' \in \Delta^{\mathcal{I}} \mid (x, x') \in P^{\mathcal{I}}_{AB}\} \geq 1$, and we have that such an instantiation is correct for the fragment of $\mathcal{D}$.

$\square$

By Lemmata 5.1, 5.2, 5.3, 5.4, 5.5, and EXPTIME-hardness of reasoning in $\mathcal{ALC}$ knowledge bases [1], we get our hardness result.

Theorem 5.6 *Class consistency in UML class diagrams is EXPTIME-hard.*

6 Complexity of Reasoning on UML Class Diagrams

The FOL formalization of class diagrams presented in Section 3, allows one to exploit FOL theorem provers to perform the various reasoning tasks on class diagrams. However, using the full power of a FOL theorem prover may be an overkill. Moreover, one incurs in the undecidability of FOL, which does not guarantee that the theorem prover will always terminate providing the answer to the requested task. Instead, one can resort to less expressive logics, in particular DLs, which provide enough expressive power to represent UML class diagrams, while keeping reasoning decidable, with known complexity bounds.

In this section we show that reasoning on UML class diagrams in decidable, and in fact EXPTIME-complete. To do so we show that we can polynomially encode UML class diagrams in $\mathcal{DLR}_{ifd}$ knowledge bases [12, 6] and that such an encoding precisely captures the FOL semantics of UML class diagrams. Hence, reasoning on such diagrams is reduced to reasoning on $\mathcal{DLR}_{ifd}$ knowledge bases, which is in EXPTIME.

6.1 Encoding of UML Class Diagrams in $\mathcal{DLR}_{ifd}$

We now illustrate the encoding of UML class diagrams in $\mathcal{DLR}_{ifd}$, discussing each construct separately.

6.1.1 Classes

An UML class is represented by a $\mathcal{DLR}_{ifd}$ concept. Indeed, both UML classes and $\mathcal{DLR}_{ifd}$ concepts denote *sets of objects*.

To capture an attribute a of type T for a class C we use a $\mathcal{DLR}_{ifd}$ binary relation a, and we specify the type of the attribute with the assertion:

$$C \sqsubseteq \forall[1](a \Rightarrow (2 : T))$$

Such an assertion specifies that, for each instance c of the concept C, all objects related to c by a, are instances of T. Note that an attribute name is not necessarily unique in the whole diagram, and hence two different classes could have the same attribute, possibly of different types. This situation is correctly captured by the formalization in $\mathcal{DLR}_{ifd}$. To specify a multiplicity $[i..j]$ associated to the attribute we add the assertion:

$$C \sqsubseteq (\geq i\,[1]a) \sqcap (\leq j\,[1]a)$$

Such an assertion specifies that each instance of C participates at least i times and at most j times to relation a via component 1. If $i = 0$, i.e., the attribute is *optional*, we omit the first conjunct, and if $j = *$ we omit the second one. Observe that, for attributes with multiplicity $[0..*]$, we omit the whole assertion, and that, when the multiplicity is missing (i.e., $[1..1]$ is assumed) the above assertion becomes:

$$C \sqsubseteq \exists[1]a \sqcap (\leq 1\,[1]a)$$

Let

$$f(P_1, \ldots, P_m) : R$$

be an operation of a class C that has m parameters belonging to the classes $P_1, \ldots, P_m$ respectively and a result belonging to R. We formalize such an operation as a $\mathcal{DLR}_{ifd}$ relation, named $\mathsf{op}_{f(P_1,\ldots,P_m)}$, of arity $1{+}m{+}1$ among instances of the $\mathcal{DLR}_{ifd}$ concepts $C, P_1, \ldots, P_m, R$. On such a relation we enforce the following assertions.

- An assertion imposing the correct types to the parameters:

$$\mathsf{op}_{f(P_1,\ldots,P_m)} \sqsubseteq (2 : P_1) \sqcap \cdots \sqcap (m+1 : P_m)$$

- An assertion imposing that invoking the operation on a given object with given parameters determines in a unique way the result (i.e., the relation corresponding to the operation is in fact a function from the invocation object and the parameters to the result):

$$(\mathbf{fd}\ \mathsf{op}_{f(P_1,\ldots,P_m)}\ 1, \ldots, m+1 \rightarrow m+2)$$

  In case the operation has no parameters (i.e., $m = 0$), instead of the above functional dependency we make use of the assertion:

$$\top_1 \sqsubseteq (\leq 1\,[1]\mathsf{op}_{f()})$$

  These assertions are determined only by the number of parameters, and not by the specific class for which the operation is defined, nor by the types of parameters and of the result.

- An assertion imposing the correct type of the result, when the operation is invoked on instances of the class C:

$$C \sqsubseteq \forall[1](\mathsf{op}_{f(P_1,\ldots,P_m)} \Rightarrow (m+2 : R))$$

As discussed in Section 3, the chosen way of naming relations corresponding to operations does not pose any difficulty in the formalization of *overloading* of operations within the same class, since an operation is represented in $\mathcal{DLR}_{ifd}$ by a relation having as name the signature of the operation, which consists not only of the operation name but also of the parameter types. Observe that the formalization of operations in $\mathcal{DLR}_{ifd}$ allows one to have operations with the same name or even with the same signature in two different classes. As discussed in Section 3, *overriding* of operations may show up as a restriction on the return type.

Example 6.1 The $\mathcal{DLR}_{ifd}$ assertions that capture the attributes of class phone in Figure 2 are:

$$\begin{array}{lcl} \mathsf{phone} & \sqsubseteq & \forall[1](\mathsf{number} \Rightarrow (2 : \mathsf{String})) \\ \mathsf{phone} & \sqsubseteq & (\geq 1\,[1]\mathsf{number}) \\ \mathsf{phone} & \sqsubseteq & \forall[1](\mathsf{brand} \Rightarrow (2 : \mathsf{String})) \end{array}$$

Operation lastDialed() is captured by the $\mathcal{DLR}_{ifd}$ assertions:

$$\begin{array}{rcl} \mathsf{phone} & \sqsubseteq & \forall[1](\mathsf{op}_{\mathsf{lastDialed()}}) \Rightarrow (2 : \mathsf{String}) \\ \top_1 & \sqsubseteq & (\leq 1\,[1]\mathsf{op}_{\mathsf{lastDialed()}}) \end{array}$$

Operation lengthCall(String) is captured by the $\mathcal{DLR}_{ifd}$ assertions:

$$\begin{array}{rcl}
\mathsf{op}_{\mathsf{lengthCall(String)}} & \sqsubseteq & (2:\mathsf{String}) \\
(\mathbf{fd}\ \mathsf{op}_{\mathsf{lengthCall(String)}}\ 1,2 \rightarrow 3) & & \\
\mathsf{phone} & \sqsubseteq & \forall[1](\mathsf{op}_{\mathsf{lengthCall(String)}} \Rightarrow (3:\mathsf{int}))
\end{array}$$

6.1.2 Associations

We have to distinguish between associations not having an association class and those having one. In the former case, we can encode an n-ary association A between classes $C_1, \ldots, C_n$ (see Figure 4) simply as a $\mathcal{DLR}_{ifd}$ n-ary relation A, together with the following typing assertion:

$$A \sqsubseteq (1:C_1) \sqcap (2:C_2) \sqcap \cdots \sqcap (n:C_n)$$

An n-ary association A with an association class (see Figure 6) is formalized in $\mathcal{DLR}_{ifd}$ by reifying A into a $\mathcal{DLR}_{ifd}$ concept A with n *binary* relations $r_1, \ldots, r_n$, one for each component of the association A. We enforce the following assertion:

$$\begin{array}{rcl}
A & \sqsubseteq & \exists[1]r_1 \sqcap (\leq 1\,[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2:C_1)) \sqcap \\
& & \exists[1]r_2 \sqcap (\leq 1\,[1]r_2) \sqcap \forall[1](r_2 \Rightarrow (2:C_2)) \sqcap \\
& & \vdots \\
& & \exists[1]r_n \sqcap (\leq 1\,[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2:C_n))
\end{array}$$

where $\exists[1]r_i$ (with $i \in \{1, \ldots, n\}$) specifies that the concept A must have all components $r_1, \ldots, r_n$ of the association A, $(\leq 1\,[1]r_i)$ (with $i \in \{1, \ldots, n\}$) specifies that each such component is single-valued, and $\forall[1](r_i \Rightarrow (2:C_i))$ (with $i \in \{1, \ldots, n\}$) specifies the class each component has to belong to. Finally, in order to faithfully represent the association by a class, we assert

$$(\mathbf{id}\ A\ [1]r_1, \ldots, [1]r_n)$$

which specifies that each instance of A represents a *distinct* tuple in $C_1 \times \cdots \times C_n$.

We can easily represent in $\mathcal{DLR}_{ifd}$ a multiplicity on a binary association. If the association has no related association class, we capture multiplicities by the following $\mathcal{DLR}_{ifd}$ assertions (referring to Figure 3):

$$\begin{array}{rcl}
\top_1 & \sqsubseteq & (\geq n_\ell\,[1]A) \sqcap (\leq n_u\,[1]A) \\
\top_1 & \sqsubseteq & (\geq m_\ell\,[2]A) \sqcap (\leq m_u\,[2]A)
\end{array}$$

Example 6.2 The $\mathcal{DLR}_{ifd}$ assertions that capture the aggregation[15] in Figure 9, are:

$$\begin{array}{rcl}
\mathsf{reference} & \sqsubseteq & (1:\mathsf{phone\ bill}) \sqcap (2:\mathsf{phone\ call}) \\
\top_1 & \sqsubseteq & (\geq 1\,[1]\mathsf{reference}) \\
\top_1 & \sqsubseteq & (\geq 1\,[2]\mathsf{reference}) \sqcap (\leq 1\,[2]\mathsf{reference})
\end{array}$$

If, instead, the association has a related class, we can impose a number restriction on the relations modeling the components of the association. Since the names of such relations (which correspond to roles) are unique with respect to the association only, and not with respect to the entire diagram, we have to state such constraints in $\mathcal{DLR}_{ifd}$ as follows (referring to Figure 5):

$$\begin{array}{rcl}
\top_1 & \sqsubseteq & (\geq n_\ell\,[2](r_1 \sqcap (1:A))) \sqcap (\leq n_u\,[2](r_1 \sqcap (1:A))) \\
\top_1 & \sqsubseteq & (\geq m_\ell\,[2](r_2 \sqcap (1:A))) \sqcap (\leq m_u\,[2](r_2 \sqcap (1:A)))
\end{array}$$

Example 6.3 The $\mathcal{DLR}_{ifd}$ assertions modeling the association in Figure 7 are:

$$\begin{array}{rcl}
\mathsf{origin} & \sqsubseteq & \forall[1](\mathsf{call} \Rightarrow (2:\mathsf{phone_call})) \sqcap \exists[1]\mathsf{call} \sqcap (\leq 1\,[1]\mathsf{call}) \sqcap \\
& & \forall[1](\mathsf{from} \Rightarrow (2:\mathsf{phone})) \sqcap \exists[1]\mathsf{from} \sqcap (\leq 1\,[1]\mathsf{from}) \\
\top_1 & \sqsubseteq & (\geq 1\,[2](\mathsf{call} \sqcap (1:\mathsf{origin}))) \sqcap (\leq 1\,[2](\mathsf{call} \sqcap (1:\mathsf{origin}))) \\
\mathsf{origin} & \sqsubseteq & \forall[1](\mathsf{place} \Rightarrow (2:\mathsf{String})) \\
\mathsf{origin} & \sqsubseteq & \exists[1]\mathsf{place} \sqcap (\leq 1\,[1]\mathsf{place})
\end{array}$$

[15]Recall that an aggregation is a special case of binary association without association class.

6.1.3 Generalizations and hierarchies

Generalization is naturally supported in $\mathcal{DLR}_{ifd}$. If an UML class C_2 generalizes a class C_1, we can express this by the $\mathcal{DLR}_{ifd}$ assertion:

$$C_1 \sqsubseteq C_2.$$

Inheritance between $\mathcal{DLR}_{ifd}$ concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of $\sqsubseteq$, which is based on sub-setting. Observe that the encoding in $\mathcal{DLR}_{ifd}$ also captures correctly inheritance among association classes and multiple inheritance between classes.

A class hierarchy as the one in Figure 10 can be represented by the assertions

$$C_i \sqsubseteq C \qquad \text{for each } i \in \{1, \ldots, n\}$$

A disjointness constraint among classes $C_1, \ldots, C_n$ can be formalized as

$$C_i \sqsubseteq \bigsqcap_{j=i+1}^{n} \neg C_j \qquad \text{for each } i \in \{1, \ldots, n\}$$

while a covering constraint can be expressed as

$$C \sqsubseteq \bigsqcup_{j=1}^{n} C_j$$

Example 6.4 The hierarchy in Figure 11 can be formalized by means of the following $\mathcal{DLR}_{ifd}$ assertions:

$$\begin{array}{rcl}
\mathsf{cell_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{fixed_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{cell_phone} & \sqsubseteq & \neg\mathsf{fixed_phone} \\
\mathsf{phone} & \sqsubseteq & \mathsf{cell_phone} \sqcup \mathsf{fixed_phone}
\end{array}$$

If needed, one can easily add $\mathcal{DLR}_{ifd}$ assertions to state that all classes that are not in the same hierarchy are a priori disjoint, and that objects in the same hierarchy must belong to a most specific class.

Example 6.5 Finally, we show how the overall UML class diagram in Figure 13 can be modeled in $\mathcal{DLR}_{ifd}$:

$$\begin{array}{rcl}
\mathsf{origin} & \sqsubseteq & \forall[1](\mathsf{place} \Rightarrow (2:\mathsf{String})) \\
\mathsf{origin} & \sqsubseteq & \exists[1]\mathsf{place} \sqcap (\leq 1\,[1]\mathsf{place}) \\
\mathsf{origin} & \sqsubseteq & \forall[1](\mathsf{call} \Rightarrow (2:\mathsf{phone_call})) \sqcap \exists[1]\mathsf{call} \sqcap (\leq 1\,[1]\mathsf{call}) \sqcap \\
& & \forall[1](\mathsf{from} \Rightarrow (2:\mathsf{phone})) \sqcap \exists[1]\mathsf{from} \sqcap (\leq 1\,[1]\mathsf{from}) \\
\mathsf{m_origin} & \sqsubseteq & \forall[1](\mathsf{call} \Rightarrow (2:\mathsf{mobile_call})) \sqcap \exists[1]\mathsf{call} \sqcap (\leq 1\,[1]\mathsf{call}) \sqcap \\
& & \forall[1](\mathsf{from} \Rightarrow (2:\mathsf{cell_phone})) \sqcap \exists[1]\mathsf{from} \sqcap (\leq 1\,[1]\mathsf{from}) \\
\top_1 & \sqsubseteq & (\geq 1\,[2](\mathsf{call} \sqcap (1:\mathsf{origin}))) \sqcap (\leq 1\,[2](\mathsf{call} \sqcap (1:\mathsf{origin}))) \\
\mathsf{reference} & \sqsubseteq & (1:\mathsf{phone_bill}) \sqcap (2:\mathsf{phone_call}) \\
\top_1 & \sqsubseteq & (\geq 1\,[1]\mathsf{reference}) \\
\top_1 & \sqsubseteq & (\geq 1\,[2]\mathsf{reference}) \sqcap (\leq 1\,[2]\mathsf{reference}) \\
\mathsf{mobile_call} & \sqsubseteq & \mathsf{phone_call} \\
\mathsf{m_origin} & \sqsubseteq & \mathsf{origin} \\
\mathsf{cell_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{fixed_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{cell_phone} & \sqsubseteq & \neg\mathsf{fixed_phone} \\
\mathsf{phone} & \sqsubseteq & \mathsf{cell_phone} \sqcup \mathsf{fixed_phone}
\end{array}$$

6.2 Correctness of the Encoding

We now show that the encoding presented above is indeed correct. In particular, we show that there is a direct correspondence between instantiations of the UML class diagram and models of the corresponding $\mathcal{DLR}_{ifd}$ knowledge base. This is captured by the following theorem.

Theorem 6.6 *Let $\mathcal{D}$ be an UML class diagram and $\mathcal{K}_\mathcal{D}$ the $\mathcal{DLR}_{ifd}$ knowledge base constructed as described above. Then every instantiation of $\mathcal{D}$ is a model of $\mathcal{K}_\mathcal{D}$, and vice-versa.*

Proof. First of all, we observe that both (the FOL formalization of) the UML class diagram $\mathcal{D}$ and the $\mathcal{DLR}_{ifd}$ knowledge base $\mathcal{K}_\mathcal{D}$ are over the same alphabet. So interpretations are compatible. Considering each UML class diagram construct separately, it is easy to see that an interpretation satisfies its FOL formalization if and only if it satisfies the corresponding $\mathcal{DLR}_{ifd}$ assertions. We show this in some detail below, also to make apparent the very close correspondence between the two formalizations.

- *Class attributes.* An attribute a of type T of the class C with multiplicity $[i..j]$ is captured in $\mathcal{D}$ by the FOL assertions:

$$\forall x,y.\ (C(x) \wedge a(x,y)) \supset T(y)$$
$$\forall x,y.\ C(x) \supset i \leq \sharp\{y \mid a(x,y)\} \leq j$$

  The corresponding $\mathcal{DLR}_{ifd}$ assertions in $\mathcal{K}_\mathcal{D}$ are

$$\begin{array}{rcl} C & \sqsubseteq & \forall[1](a \Rightarrow (2:T)) \\ C & \sqsubseteq & (\geq i\,[1]a) \sqcap (\leq j\,[1]a) \end{array}$$

  Now, given an instantiation $\mathcal{I}$ for $\mathcal{D}$, each $x \in C^\mathcal{I}$ is such that x is connected through the binary relation $a^\mathcal{I}$ only to elements of $T^\mathcal{I}$, and x participates at least i and at most j times to $a^\mathcal{I}$. Hence $\mathcal{I}$ satisfies the $\mathcal{DLR}_{ifd}$ assertions above. Conversely, given a model $\mathcal{I}$ of $\mathcal{K}_\mathcal{D}$, it is easy to see that each $x \in C^\mathcal{I}$ is connected through the binary relation $a^\mathcal{I}$ only to elements of $T^\mathcal{I}$, and x participates at least i and at most j times to $a^\mathcal{I}$. Therefore, $\mathcal{I}$ satisfies the FOL formulas above.

- *Class operations.* An operation $f(P_1,\ldots,P_m) : R$ of class C is expressed by the FOL assertions[16]:

$$\forall x,p_1,\ldots,p_m,r.\ f(x,p_1,\ldots,p_m,r) \supset \textstyle\bigwedge_{i=1}^n P_i(p_i)$$
$$\forall x,p_1,\ldots,p_m,r,r'.\ f(x,p_1,\ldots,p_m,r) \wedge f(x,p_1,\ldots,p_m,r') \supset r = r'$$
$$\forall x,p_1,\ldots,p_m,r.\ C(x) \wedge f(x,p_1,\ldots,p_m,r) \supset R(r)$$

  The corresponding $\mathcal{DLR}_{ifd}$ assertions in $\mathcal{K}_\mathcal{D}$ are

$$\begin{array}{l} \mathtt{op}_{f(P_1,\ldots,P_m)} \sqsubseteq (2:P_1) \sqcap \cdots \sqcap (m+1:P_m) \\ (\mathbf{fd}\ \mathtt{op}_{f(P_1,\ldots,P_m)}\ 1,\ldots,m+1 \rightarrow m+2) \\ C \sqsubseteq \forall[1](\mathtt{op}_{f(P_1,\ldots,P_m)} \Rightarrow (m+2:R)) \end{array}$$

  Given an instantiation $\mathcal{I}$ for $\mathcal{D}$, it is such that for each $x \in C^\mathcal{I}$, if x participates to $y \in f^\mathcal{I}$ as first component, the components $2,\ldots,m+2$ of y belong to $P_1^\mathcal{I},\ldots,P_m^\mathcal{I},R$ respectively, and the component $m+2$ is univocally determined by the first $m+1$. Hence $\mathcal{I}$ satisfies the $\mathcal{DLR}_{ifd}$ assertions above. Conversely, by the inclusion assertion, a model $\mathcal{I}$ of $\mathcal{K}_\mathcal{D}$ is such that for any $x \in C^\mathcal{I}$, if x participates to $y \in \mathtt{op}^\mathcal{I}_{f(P_1,\ldots,P_m)}$, the components $2,\ldots,m+2$ of y belong to $P_1^\mathcal{I},\ldots,P_m^\mathcal{I},R$ respectively. Moreover, by the functional dependency, the component $m+2$ is univocally determined by the first $m+1$. Therefore, $\mathcal{I}$ satisfies the FOL formulas above.

- *Associations without association class.* Typing of an n-ary association A without association class is captured in $\mathcal{D}$ by the FOL assertion:

$$\forall x_1,\ldots,x_n.\ A(x_1,\ldots,x_n) \supset C_1(x_1) \wedge \ldots \wedge C_n(x_n).$$

  The corresponding $\mathcal{DLR}_{ifd}$ assertion in $\mathcal{K}_\mathcal{D}$ is

$$A \sqsubseteq (1:C_1) \sqcap (2:C_2) \sqcap \ldots \sqcap (n:C_n),$$

  Given an instantiation $\mathcal{I}$ for $\mathcal{D}$, we have that for any $x \in A^\mathcal{I}$, the components of x belong to $C_1^\mathcal{I},\ldots,C_n^\mathcal{I}$ respectively. Hence $\mathcal{I}$ satisfies the $\mathcal{DLR}_{ifd}$ assertion above. Conversely, given an interpretation $\mathcal{I}$ for

[16] To simplify the notation, we again denote $f_{P_1,\ldots,P_m}$ simply by f.

$\mathcal{K}_\mathcal{D}$, the $\mathcal{DLR}_{ifd}$ assertion above requires that for any $x \in A^\mathcal{I}$, the components of x belong to $C_1^\mathcal{I}, \ldots, C_n^\mathcal{I}$ respectively. Therefore $\mathcal{I}$ satisfies the FOL formula above.

Multiplicities of a binary association without association class are expressed by the FOL assertions:

$$\begin{array}{l} \forall x.\ (n_\ell \leq \sharp\{y \mid A(x,y)\} \leq n_u) \\ \forall y.\ (m_\ell \leq \sharp\{x \mid A(x,y)\} \leq m_u) \end{array}$$

The corresponding $\mathcal{DLR}_{ifd}$ assertions in $\mathcal{K}_\mathcal{D}$ are

$$\begin{array}{lcl} \top_1 & \sqsubseteq & (\geq n_\ell\,[1]A) \sqcap (\leq n_u\,[1]A) \\ \top_1 & \sqsubseteq & (\geq m_\ell\,[2]A) \sqcap (\leq m_u\,[2]A) \end{array}$$

Again, considering the semantics of the assertions in FOL and in $\mathcal{DLR}_{ifd}$, it is immediate to verify that they are satisfied by exactly the same models.

- *Associations with association class.* An n-ary association A with association class is formalized in $\mathcal{D}$ by the following FOL assertions:

$$\begin{array}{l} \forall x, y.\ r_i(x,y) \wedge A(x) \supset C_i(y) \quad \text{for } i = 1, \ldots, n \\ \forall x.\ A(x) \supset \exists y.\ r_i(x,y) \quad \text{for } i = 1, \ldots, n \\ \forall x, y, y'.\ A(x) \wedge r_i(x,y) \wedge r_i(x,y') \supset y = y' \quad \text{for } i = 1, \ldots, n \\ \forall y_1, \ldots, y_n, x, x'.\ A(x) \wedge A(x') \wedge \bigwedge_{i=1}^n (r_i(x,y_i) \wedge r_i(x',y_i)) \supset x = x' \end{array}$$

The corresponding $\mathcal{DLR}_{ifd}$ assertions in $\mathcal{K}_\mathcal{D}$ are

$$\begin{array}{lcl} A & \sqsubseteq & \exists[1]r_1 \sqcap (\leq 1\,[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2\!:\!C_1)) \sqcap \\ & & \exists[1]r_2 \sqcap (\leq 1\,[1]r_2) \sqcap \forall[1](r_2 \Rightarrow (2\!:\!C_2)) \sqcap \\ & & \quad\vdots \\ & & \exists[1]r_n \sqcap (\leq 1\,[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2\!:\!C_n)) \\ (\mathbf{id}\ A\ [1]r_1, \ldots, [1]r_n) & & \end{array}$$

Given an instantiation $\mathcal{I}$ for $\mathcal{D}$, by the FOL assertion above, we have that for each $x \in A^\mathcal{I}$, x participates exactly once as first component to each of the binary relations $r_i^\mathcal{I}$, and x is connected through $r_1^\mathcal{I}, \ldots, r_n^\mathcal{I}$ to elements of $C_1^\mathcal{I}, \ldots, C_n^\mathcal{I}$ respectively; moreover, no two instances of $A^\mathcal{I}$ can agree on the participation to $r_1^\mathcal{I}, \ldots, r_n^\mathcal{I}$. Hence $\mathcal{I}$ satisfies all the $\mathcal{DLR}_{ifd}$ assertions above. Similarly, it is easy to see that a model $\mathcal{I}$ of $\mathcal{K}_\mathcal{D}$, which has to satisfy the $\mathcal{DLR}_{ifd}$ assertions above, satisfies the corresponding FOL assertions as well.

Multiplicities of a binary association A with association class are expressed by the FOL assertions:

$$\begin{array}{l} \forall y.\ (n_\ell \leq \sharp\{y \mid A(x) \wedge r_1(x,y)\} \leq n_u) \\ \forall y.\ (m_\ell \leq \sharp\{y \mid A(x) \wedge r_2(x,y)\} \leq m_u) \end{array}$$

The corresponding $\mathcal{DLR}_{ifd}$ assertions in $\mathcal{K}_\mathcal{D}$ are

$$\begin{array}{lcl} \top_1 & \sqsubseteq & (\geq n_\ell\,[2](r_1 \sqcap (1\!:\!A))) \sqcap (\leq n_u\,[2](r_1 \sqcap (1\!:\!A))) \\ \top_1 & \sqsubseteq & (\geq m_\ell\,[2](r_2 \sqcap (1\!:\!A))) \sqcap (\leq m_u\,[2](r_2 \sqcap (1\!:\!A))) \end{array}$$

Again, considering the semantics of the assertions in FOL and in $\mathcal{DLR}_{ifd}$, it is immediate to verify that they are satisfied by exactly the same models.

- *Generalizations.* The generalization between a more general class C a more specific class C_1 is formalized by the FOL assertion:

$$\forall x.\ C_1(x) \supset C(x)$$

The corresponding $\mathcal{DLR}_{ifd}$ assertion in $\mathcal{K}_\mathcal{D}$ is:

$$C_1 \ \sqsubseteq\ C$$

Considering the semantics of such assertions, it is immediate to verify that they are satisfied by exactly the same models. It is also immediate to verify that the FOL and $\mathcal{DLR}_{ifd}$ assertions expressing covering constraints and disjointness constraints on class hierarchies are satisfied by exactly the same models.

□

A consequence of the above result is that reasoning on UML class diagrams can be performed by reasoning on $\mathcal{DLR}_{ifd}$ knowledge bases. In particular, the following result holds.

Theorem 6.7 *Let $\mathcal{D}$ be an UML class diagram and $\mathcal{K}_\mathcal{D}$ the $\mathcal{DLR}_{ifd}$ knowledge base constructed as described above. Then a class C is consistent in $\mathcal{D}$ if and only if the concept C is satisfiable in $\mathcal{K}_\mathcal{D}$.*

Proof. The claim is a straightforward consequence of Theorem 6.6. □

Since we can reduce reasoning on UML class diagrams to reasoning on $\mathcal{DLR}_{ifd}$ knowledge bases, from the results about reasoning in $\mathcal{DLR}_{ifd}$ [12, 15] we get an EXPTIME upper bound for reasoning on UML class diagrams.

Theorem 6.8 *Class consistency in UML class diagrams is EXPTIME-complete.*

Proof. Theorem 5.6 gives us the EXPTIME-hardness. The completeness follows from Theorem 6.7, by considering that the size of $\mathcal{K}_\mathcal{D}$ is polynomial in $\mathcal{D}$ and that concept satisfiability in $\mathcal{DLR}_{ifd}$ knowledge bases is EXPTIME-complete [12, 15]. □

7 Reasoning in $\mathcal{ALCQI}$

The results in the previous section show that we can exploit reasoning tools developed for DLs to reason on UML class diagrams. However, current state-of-the-art DL based reasoning systems do not support yet all constructs of $\mathcal{DLR}_{ifd}$. In particular, this holds for functional dependencies and identification constraints, since their implementation requires very advanced forms of reasoning on individuals [15]. In this section we show that, as far as reasoning on UML class diagrams is concerned (cf. Section 4), we can resort to a less expressive DL, namely $\mathcal{ALCQI}$, which is accepted by state-of-the-art DL based reasoning system [41, 31]. In this way, we can exploit such reasoning systems as core engines for advanced UML CASE tools [30].

Notably, $\mathcal{ALCQI}$ does not include functional dependencies and identification constraints, which play a special role, since they allow us to correctly capture the FOL semantics of n-ary associations represented in $\mathcal{DLR}_{ifd}$ in a reified way, and of operations.

Interestingly, when we do not want to specifically reason about functional dependencies or identification constraints, which is the case for the UML reasoning tasks, we can drop such constraints from $\mathcal{DLR}_{ifd}$ knowledge bases, while still preserving soundness and completeness of reasoning on concepts and relations [15]. This is due to the tree model property $\mathcal{DLR}_{ifd}$ [14, 15]: intuitively, if a knowledge base admits a model, it admits one having a tree structure, and since in such models the tuples cannot have more than one component in common, functional dependencies and identification constraints are trivially satisfied. Another potential difficulty is that, in $\mathcal{ALCQI}$, relations are only binary, while $\mathcal{DLR}_{ifd}$ admits relations of arbitrary arity. We overcome this difficulty by translating a $\mathcal{DLR}_{ifd}$ relation of arity $n > 2$ in a *reified* way, by introducing a concept, denoting the tuples of the relation, and n $\mathcal{ALCQI}$ (binary) functional roles, one for each component of the relation. Observe that the tree-model property guarantees that such a translation is faithful, in the sense that there will be no two instances of the concept representing the same tuple of the relation [16].

Building on these observations we now present an encoding of UML class diagrams directly in $\mathcal{ALCQI}$ that, although it does not preserve models, is sound and complete with respect to reasoning on UML class diagrams.

7.1 Encoding of UML Class Diagrams in $\mathcal{ALCQI}$

We show how to construct from an UML class diagram an $\mathcal{ALCQI}$ knowledge base that preserves reasoning.

7.1.1 Classes

An UML class C is represented by an atomic concept C. Each attribute a of type T for class C is represented by an atomic role a, together with an inclusion assertion encoding the typing of the attribute a for the class C:

$$C \sqsubseteq \forall a.T$$

We formalize the multiplicity $[i..j]$ of attribute a as

$$C \sqsubseteq (\geq i\, a.\top) \sqcap (\leq j\, a.\top)$$

expressing that for each instance of the concept C there are at least i and at most j role fillers for role a. As we did for $\mathcal{DLR}_{ifd}$, for attributes with multiplicity $[0..*]$ we omit the whole assertion, and when the multiplicity is missing (i.e., $[1..1]$ is assumed) the above assertion becomes:

$$C \sqsubseteq \exists a.\top \sqcap (\leq 1\, a.\top)$$

An operation $f() : R$ without parameters for class C is modeled directly as a (binary) role $\mathtt{R}_{f()}$, for which the following assertion holds:

$$C \sqsubseteq \forall \mathtt{R}_{f()}.R \sqcap (\leq 1\, \mathtt{R}_{f()}.\top)$$

Instead, an operation with one or more parameters $f(P_1, \ldots, P_m) : R$ of class C, which formally corresponds to an $(m+2)$-ary relation that is functional on the last component, cannot be directly expressed in $\mathcal{ALCQI}$. Therefore, we make use of reification, and introduce an atomic concept $\mathtt{C}_{f(P_1,\ldots,P_m)}$, $m+2$ $\mathcal{ALCQI}$ roles $r_1, \ldots, r_{m+2}$ and the following assertions, which type the input parameters and the return value:

$$\begin{array}{rcl}
\mathtt{C}_{f(P_1,\ldots,P_m)} & \sqsubseteq & \exists r_1.\top \sqcap (\leq 1\, r_1.\top) \sqcap \\
 & & \vdots \\
 & & \exists r_{m+1}.\top \sqcap (\leq 1\, r_{m+1}.\top) \\
\mathtt{C}_{f(P_1,\ldots,P_m)} & \sqsubseteq & \forall r_2.P_1 \sqcap \cdots \sqcap \forall r_{m+1}.P_m \\
C & \sqsubseteq & \forall r_1^-.(\mathtt{C}_{f(P_1,\ldots,P_m)} \Rightarrow \forall r_{m+2}.R)
\end{array}$$

The first assertion states that each instance of $\mathtt{C}_{f(P_1,\ldots,P_m)}$, representing a tuple, correctly is connected to exactly one object for each of the roles $r_1, \ldots, r_{m+1}$. Instead, note that in general there may be two instances of $\mathtt{C}_{f(P_1,\ldots,P_m)}$ representing the same tuple. However, this cannot be the case in a tree-like model (cf., tree-model property). The other two assertions impose the correct typing of the parameters, depending only on the name of the operation, and of the return value, depending also on the class.

7.1.2 Associations

Each binary association (or aggregation) A between a class C_1 and a class C_2 is represented by the atomic role A, together with the inclusion assertion

$$\top \sqsubseteq \forall A.C_2 \sqcap \forall A^-.C_1$$

encoding the typing of A. The multiplicities of A (see Figure 3) are formalized by the assertions

$$\begin{array}{rcl}
\top & \sqsubseteq & (\geq n_\ell\, A.\top) \sqcap (\leq n_u\, A.\top) \\
\top & \sqsubseteq & (\geq m_\ell\, A^-.\top) \sqcap (\leq m_u\, A^-.\top)
\end{array}$$

Binary associations with association class, and n-ary (with $n > 2$) associations, with or without association class, are modeled through reification. More precisely, each association A relating classes $C_1, \ldots, C_n$ is represented by an atomic concept A together with the inclusion assertion

$$A \sqsubseteq \exists r_1.C_1 \sqcap \cdots \sqcap \exists r_n.C_n \sqcap (\leq 1\, r_1) \sqcap \cdots \sqcap (\leq 1\, r_n)$$

If the association A has explicit role names in the UML class diagram, then $r_1, \ldots, r_n$ above are such names. Otherwise, they are arbitrary names used to denote the components of A. As we did for operations, we are not requiring that each instance of the concept A denotes a distinct tuple, but again this is the case in tree-like models.

Multiplicities on binary associations with association class (see Figure 5) are represented by

$$\begin{array}{rcl}\top & \sqsubseteq & (\geq n_\ell\, r_1^- .A) \sqcap (\leq n_u\, r_1^- .A) \\ \top & \sqsubseteq & (\geq m_\ell\, r_2^- .A) \sqcap (\leq m_u\, r_2^- .A)\end{array}$$

7.1.3 Generalizations

Generalizations between classes, and disjointness and covering constraints on hierarchies are immediately expressed in $\mathcal{ALCQI}$, as for $\mathcal{DLR}_{ifd}$. In particular, a generalization between a class C and its child class C_1 can be represented using the $\mathcal{ALCQI}$ inclusion assertion $C_1 \sqsubseteq C$. A class hierarchy as the one in Figure 10 can be represented by the assertions $C_1 \sqsubseteq C, \ldots, C_n \sqsubseteq C$. A disjointness constraint among classes $C_1, \ldots, C_n$ can be modeled as $C_i \sqsubseteq \sqcap_{j=i+1}^n \neg C_j$, with $1 \leq i \leq n-1$, while a covering constraint can be expressed as $C \sqsubseteq \sqcup_{i=1}^n C_i$.

Example 7.1 Considering the UML class diagram shown in Figure 13, the corresponding $\mathcal{ALCQI}$ knowledge base, defined as above, is the following:

$$\begin{array}{rcl}
\mathsf{origin} & \sqsubseteq & \forall \mathsf{place}.\mathsf{String} \\
\mathsf{origin} & \sqsubseteq & \exists \mathsf{place}.\top \sqcap (\leq 1\, \mathsf{place}) \\
\mathsf{origin} & \sqsubseteq & \exists \mathsf{call}.\mathsf{phone_call} \sqcap (\leq 1\, \mathsf{call}) \sqcap \\
 & & \exists \mathsf{from}.\mathsf{phone} \sqcap (\leq 1\, \mathsf{from}) \\
\mathsf{m_origin} & \sqsubseteq & \exists \mathsf{m_call}.\mathsf{mobile_call} \sqcap (\leq 1\, \mathsf{m_call}) \sqcap \\
 & & \exists \mathsf{m_from}.\mathsf{cell_phone} \sqcap (\leq 1\, \mathsf{m_from}) \\
\top & \sqsubseteq & (\geq 1\, \mathsf{call}^-.\mathsf{origin}) \sqcap (\leq 1\, \mathsf{call}^-.\mathsf{origin}) \\
\top & \sqsubseteq & \forall \mathsf{reference}^-.\mathsf{phone_bill} \sqcap \forall \mathsf{reference}.\mathsf{phone_call} \\
\top & \sqsubseteq & (\geq 1\, \mathsf{reference}^-) \\
\top & \sqsubseteq & (\geq 1\, \mathsf{reference}) \sqcap (\leq 1\, \mathsf{reference}) \\
\mathsf{mobile_call} & \sqsubseteq & \mathsf{phone_call} \\
\mathsf{m_origin} & \sqsubseteq & \mathsf{origin} \\
\mathsf{cell_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{fixed_phone} & \sqsubseteq & \mathsf{phone} \\
\mathsf{cell_phone} & \sqsubseteq & \neg \mathsf{fixed_phone} \\
\mathsf{phone} & \sqsubseteq & \mathsf{cell_phone} \sqcup \mathsf{fixed_phone}
\end{array}$$

7.2 Correctness of the Encoding

We now show that the encoding of an UML class diagram into an $\mathcal{ALCQI}$ knowledge base is *correct*, in the sense that it preserves the reasoning services over UML class diagrams. Formally, the following result holds.

Theorem 7.2 *Let $\mathcal{D}$ be an UML class diagram and $\mathcal{K}_\mathcal{D}$ the $\mathcal{ALCQI}$ knowledge base constructed as specified above. Then a class C is consistent in $\mathcal{D}$ if and only if the concept C is satisfiable in $\mathcal{K}_\mathcal{D}$.*

Proof. "$\Rightarrow$" Let $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ be an instantiation of $\mathcal{D}$ (i.e., a model of the corresponding FOL assertions) such that $C^\mathcal{I} \neq \emptyset$. Then we can build a model $\mathcal{J} = (\Delta^\mathcal{J}, \cdot^\mathcal{J})$ of $\mathcal{K}_\mathcal{D}$ such that $C^\mathcal{J} \neq \emptyset$ as follows.

- $\Delta^\mathcal{J} = \Delta^\mathcal{I} \cup \bigcup_{A \in \mathcal{A}} \{t_{(d_1,\ldots,d_n)} \mid (d_1, \ldots, d_n) \in A^\mathcal{I}\} \cup \bigcup_{F \in \mathcal{F}} \{t_{(d_1,\ldots,d_n)} \mid (d_1, \ldots, d_n) \in F^\mathcal{I}\}$, where $\mathcal{A}$ denotes the set of all non-binary associations without association class in $\mathcal{D}$, and $\mathcal{F}$ denotes all functional relations that model class operations.
- $C^\mathcal{J} = C^\mathcal{I}$ for all concepts C corresponding to classes C in $\mathcal{D}$.
- $R^\mathcal{J} = R^\mathcal{I}$ for all $\mathcal{ALCQI}$ roles R corresponding to attributes, operations without parameters, aggregations, binary associations without association class, and association class roles in $\mathcal{D}$.

- For each operation $f(P_1,\ldots,P_m) : R$ with one or more parameters, we define $\mathtt{C}^{\mathcal{J}}_{f(P_1,\ldots,P_m)} = \{t_{(d_0,d_1,\ldots,d_{m+1})} \mid (d_0,d_1,\ldots,d_{m+1}) \in f^{\mathcal{I}}_{(P_1,\ldots,P_m)}\}$, and for each $\mathcal{ALCQI}$ role r_i modeling the i-th component of the relation $f_{(P_1,\ldots,P_m)}$, we define $r_i^{\mathcal{J}} = \{(t_{(d_0,d_1,\ldots,d_{m+1})}, d_i) \mid (d_0,d_1,\ldots,d_{m+1}) \in f^{\mathcal{I}}_{(P_1,\ldots,P_m)}\}$.
- Finally, for each n-ary association A with arity $n > 2$ and without association class, we define $A^{\mathcal{J}} = \{t_{(d_1,\ldots,d_n)} \mid d_1,\ldots,d_n \in A^{\mathcal{I}}\}$ and for each $\mathcal{ALCQI}$ role r_i modeling the i-th component of the association A, we define $r_i^{\mathcal{J}} = \{(t_{(d_1,\ldots,d_n)}, d_i) \mid (d_1,\ldots,d_n) \in A^{\mathcal{I}}\}$.

Trivially $C^{\mathcal{J}} = C^{\mathcal{I}} \neq \emptyset$. It is also immediate to check that $\mathcal{J}$ satisfies all the assertions in $\mathcal{K}_{\mathcal{D}}$. Again one can proceed by focusing on the assertions that each kind of UML class diagram construct gives rise to.

As an example, we explicit the proof for operations, since proving the statement in this case is less straightforward than in all the others cases. An operation $f(P_1,\ldots,P_m) : R$ of a class C in the diagram $\mathcal{D}$ is represented by the FOL formulas:

$$\begin{array}{l}
\forall x, p_1,\ldots,p_m, r.\ f(x,p_1,\ldots,p_m,r) \supset \bigwedge_{i=1}^{m} P_i(p_i) \\
\forall x, p_1,\ldots,p_m, r, r'.\ f(x,p_1,\ldots,p_m,r) \wedge f(x,p_1,\ldots,p_m,r') \supset r = r' \\
\forall x, p_1,\ldots,p_m, r.\ C(x) \wedge f(x,p_1,\ldots,p_m,r) \supset R(r)
\end{array}$$

that correspond to the $\mathcal{ALCQI}$ assertions:

$$\begin{array}{rcl}
\mathtt{C}_{f(P_1,\ldots,P_m)} & \sqsubseteq & \exists r_1.\top \sqcap (\leq 1\, r_1.\top) \sqcap \\
& & \vdots \\
& & \exists r_{m+2}.\top \sqcap (\leq 1\, r_{m+2}.\top) \\
\mathtt{C}_{f(P_1,\ldots,P_m)} & \sqsubseteq & \forall r_2.P_1 \sqcap \cdots \sqcap \forall r_{m+1}.P_m \\
C & \sqsubseteq & \forall r_1^-.(\mathtt{C}_{f(P_1,\ldots,P_m)} \Rightarrow \forall r_{m+2}.R)
\end{array}$$

Given an instantiation $\mathcal{I}$ of $\mathcal{D}$, for all $y \in f^{\mathcal{I}}$, the components $2,\ldots,m+1$ of y belong to $P_1^{\mathcal{I}},\ldots,P_m^{\mathcal{I}}$, and for all $x \in C^{\mathcal{I}}$ if x participates as first component to $y \in f^{\mathcal{I}}$, the component $m+2$ of y belongs to $R^{\mathcal{I}}$. Additionally, the first $m+1$ components univocally determine the component $m+2$.

The interpretation $\mathcal{J}$, built from $\mathcal{I}$ as shown above, instantiates the concept $\mathtt{C}_{f(P_1,\ldots,P_m)}$, which models $f(P_1,\ldots,P_m) : R$, with the $(m+2)$-tuples of $f^{\mathcal{I}}_{(P_1,\ldots,P_m)}$, and instantiates each role r_i with pairs where the first component is a tuple $(d_0,d_1,\ldots,d_{m+1})$ of $f^{\mathcal{I}}_{(P_1,\ldots,P_m)}$ and the second one is the component d_i of $(d_0,d_1,\ldots,d_{m+1})$ to which r_i refers. In particular, each tuple of r_i connects each element of $\mathtt{C}_{f(P_1,\ldots,P_m)}$ to the element of the correct type (and only to it) and no two elements of $\mathtt{C}_{f(P_1,\ldots,P_m)}$ represent the same tuple. Hence, $\mathcal{J}$ it satisfies the above $\mathcal{ALCQI}$ assertions.

"$\Leftarrow$" By the tree model property we know that if C is satisfiable in the $\mathcal{ALCQI}$ knowledge base $\mathcal{K}_{\mathcal{D}}$ then there exists a tree-like model $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ of $K_{\mathcal{D}}$, such that $C^{\mathcal{J}} \neq \emptyset$. From such a tree-like model we can build an instantiation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{D}$ such that $C^{\mathcal{I}} \neq \emptyset$ as follows.

- $\Delta^{\mathcal{J}} = \bigcup_{C \in \mathcal{C}} C^{\mathcal{J}}$, where $\mathcal{C}$ denotes the set of all classes in $\mathcal{D}$.
- $C^{\mathcal{I}} = C^{\mathcal{J}}$ for all classes C in $\mathcal{D}$.
- $R^{\mathcal{I}} = R^{\mathcal{J}}$ for all attributes, operations without parameters, aggregations, binary associations without association class, and association class roles in $\mathcal{D}$.
- For each operation $f(P_1,\ldots,P_m) : R$ with one or more parameters, we define $f^{\mathcal{I}}_{(P_1,\ldots,P_m)} = \{(d_0,d_1,\ldots,d_{m+1}) \mid \exists t \in \mathtt{C}^{\mathcal{J}}_{f(P_1,\ldots,P_m)}.\ \bigwedge_{i=0}^{m+1} (t,d_i) \in r_i^{\mathcal{J}}\}$.
- Finally for each n-ary association A with arity $n > 2$ and without association class, we define $A^{\mathcal{I}} = \{(d_1,\ldots,d_n) \mid \exists t \in A^{\mathcal{J}}.\ \bigwedge_{i=0}^{m+1} (t,d_i) \in r_i^{\mathcal{J}}\}$.

Observe that, since $\mathcal{J}$ is a tree-like model, it is guaranteed that there is only one object t in $\mathtt{C}_{f(P_1,\ldots,P_m)}$ that represents a given tuple, similarly for the concepts A representing n-ary associations with or without association class. Hence tuples of n-ary associations, tuples of relations corresponding to class operations, as well as key constraints for association classes and uniqueness of the operation results is guaranteed. Keeping such an observation in mind is easy to check that $\mathcal{I}$ is indeed an instantiation of $\mathcal{K}$ with $C^{\mathcal{I}} \neq \emptyset$.

Analogously to the previous case, we detail the proof for operations.

Given a model $\mathcal{J}$ for $\mathcal{K}_{\mathcal{D}}$, each $y \in \mathtt{C}^{\mathcal{J}}_{f(P_1,\ldots,P_m)}$ is connected to elements of $C^{\mathcal{J}}, P_1^{\mathcal{J}}, \ldots, P_m^{\mathcal{J}}, R^{\mathcal{J}}$ via roles $r_1^{\mathcal{J}}, \ldots, r_{m+2}^{\mathcal{J}}$, respectively; y participates to each $r_i^{\mathcal{J}}$ exactly once, as first component.

The instantiation $\mathcal{I}$, built from $\mathcal{J}$ as shown above, populates $f^{\mathcal{I}}_{(P_1,\ldots,P_m)}$ with m+2-tuples $(d_0, d_1, \ldots, d_{m+1})$ that correspond to the elements of $\mathtt{C}^{\mathcal{J}}_{f(P_1,\ldots,P_m)}$, and such that each d_i is the second component of $r_i^{\mathcal{J}}$. In particular, each parameter and return value of $f_{(P_1,\ldots,P_m)}$ is correctly typed and, from $\mathcal{J}$ and the tree model property, $f_{(P_1,\ldots,P_m)}$ is a function from the invocation object and the parameters to the result value. Hence, $\mathcal{I}$ correctly instantiates $f_{(P_1,\ldots,P_m)}$.

□

Note that the notion of correctness that can be adopted for the encoding in $\mathcal{ALCQI}$ is the one that results from Theorem 7.2. Such a notion is much weaker than the one for the encoding in $\mathcal{DLR}_{ifd}$ given by Theorem 6.6. Indeed, differently from the encoding in $\mathcal{DLR}_{ifd}$, the encoding in $\mathcal{ALCQI}$ does not preserve models since $\mathcal{ALCQI}$ is not equipped with means to express n-ary relations and identification and functional dependency constrains, which are needed to fully express UML class diagrams. However, as Theorem 7.2 shows, the encoding in $\mathcal{ALCQI}$ preserves enough semantics to carry out sound and complete reasoning on UML class diagrams.

Finally, note that the size of the $\mathcal{ALCQI}$ knowledge base $\mathcal{K}_{\mathcal{D}}$, obtained by encoding a UML class diagram $\mathcal{D}$ in $\mathcal{ALCQI}$, is linear in the size of $\mathcal{D}$. Hence the EXPTIME upper bound for reasoning on UML class diagrams is preserved by the encoding in $\mathcal{ALCQI}$.

7.3 Description Logics Reasoners

Current state-of-the-art DL reasoning systems [34] support arbitrary complex $\mathcal{ALCQI}$ knowledge bases and implement sound and complete reasoning algorithms. These algorithms are based on tableaux techniques and are in fact not optimal from the computational complexity point of view. Indeed, they are NEXPTIME [22], while reasoning over $\mathcal{ALCQI}$ knowledge bases is EXPTIME-complete. However, state-of-the-art DL reasoning systems implement highly optimized versions of the standard tableaux based algorithms [39], and exhibit good average case performance [44, 5].

Two of the best-known systems are FaCT[17], developed at the University of Manchester [38, 41, 40] and Racer[18], developed at the University of Hamburg [33, 32]. Both these systems perform a preliminary *classification* (see [2]) of the concepts of the $\mathcal{ALCQI}$ knowledge base. Classification iteratively computes, by subsequent subsumption tests, a taxonomy of classes, making explicit all subsumption relationships among the concepts of the knowledge base. Once this classification step is performed, reasoning services can take advantage of it to speed up inferences.

Encoding a UML class diagram in an $\mathcal{ALCQI}$ knowledge base allows the designer of the diagram for exlpoiting the reasoning services offered by the DL reasoners. In such a way, relevant properties of the diagram can be formally verified, as discussed in Section 4. Indeed, classification builds a hierarchy of the (concepts corresponding to the) UML classes belonging to the diagram. This hierarchy reflects the various constraints that the diagram enforces on the classes, as well as their properties and the relations among them. In the next section, we apply these ideas to an application domain of industrial interest.

8 A Case Study

In the previous section we showed that UML class diagrams can be encoded as $\mathcal{ALCQI}$ knowledge bases preserving enough semantics to keep reasoning sound and complete. On the other hand $\mathcal{ALCQI}$ is the

[17] `http://www.cs.man.ac.uk/~horrocks/FaCT/`
[18] `http://kogs-www.informatik.uni-hamburg.de/~race`

DL implemented in current state-of-the-art DL-based systems. Hence these systems could serve as a core reasoning engine in advanced CASE tools equipped with automated reasoning capabilities on UML class diagrams. In order to verify such an idea, we did some experimentation on both UML class diagrams developed for educational purposes, and on UML class diagrams of industrial interest [6, 5]. In this section we give a brief overview of our experience, with an industrial scale example, namely, the UML class diagrams forming CIM.

Common Information Model (CIM)[19] is a model defined by the Distributed Management Task Force (DMTF), with the purpose of providing a rigorous approach for modeling systems and networks using the object-oriented paradigm. CIM has a *Meta Schema*, expressed as a set of UML class diagrams that form the basis of a sort of vocabulary for analyzing and describing managed systems. According to the particular needs of a given application, such schemas can be extended through subclassing to include aspects specific to the application. CIM offers three main conceptual schemas, each expressed as a UML class diagram: the *Core Model*, the *Common Model* and the *Extension Schemas*. The Core Model and the Common Model together form the *CIM Schema*.

- The Core Model is an information model capturing basic notions that are applicable to all areas of management (e.g., logical device or system component).
- The Common Model is an information model that expresses concepts related to specific management areas, but still independent of a particular technology or implementation. The common areas defined in the Common Model are: *Systems*, *Devices*, *Applications*, *Networks*, and *Physical*.
- Extension Schemas are made up of classes that represent managed objects that are *technology specific additions to the Common Model*.

Such schemas are constituted by UML class diagrams of substantial size (hundreds of classes and of associations) and include multiplicity constraints on binary association and aggregations, class and association hierarchies, covering and disjointness constraints. Such diagrams are written in MOF (Meta Object Facility) format, so as to be easily used in applications such as meta-information repositories, software development management systems, information management systems, and data warehousing.

We developed a translator that reads a MOF file and generates an $\mathcal{ALCQI}$ knowledge base that corresponds to the UML class diagram described in the MOF file. We run such translator and then we use a DL-based system, namely FaCT or Racer, to classify the resulting $\mathcal{ALCQI}$ knowledge base. Observe that this step, although exponential in theory, takes just a few seconds for each of the CIM models above on both FaCT and Racer. Once these preliminary steps are done, we are ready to ask for interesting properties of the UML class diagrams, making use of reasoning provided by the DL-based systems on the $\mathcal{ALCQI}$ counterpart of the diagrams. Typically, these properties can be verified in fractions of seconds.

The UML class diagrams forming CIM are very well designed making most interesting properties explicitly available or verifiable by scanning the diagram, and avoiding as much as possible redundancy. However, by automated reasoning, we were able to show, in few cases indeed, that it is possible to refine the diagrams in order to make explicit some properties otherwise hidden in the interaction of the various classes and associations. Here we illustrate one of such cases.

Example 8.1 We focus on the CIM Core Model and, in particular, on the sub-schema shown in Figure 21.

This sub-schema models the relation between managed system elements and their statistical information. Since there may be different kinds of statistical information, depending on the managed system element it refers to, the class Statistical_Information and the association class related to association Statistics have several sub-classes[20]. Observe that there is an implicit covering constraint and a disjointness constraint on each ISA hierarchy. Therefore, each child of Statistics contains tuples that are made up of elements from *one* sub-class of Managed System Element and *the* suitable sub-class of Statistical_Information. Additionally, each element of the sub-classes of Statistical_Information participates exactly once to the suitable association sub-classes of Statistics.

[19] `http://www.dmtf.org/standards/cim_schema_v26.php`

[20] The latter information is not shown in the fragment of the CIM Core Model in Figure 21.

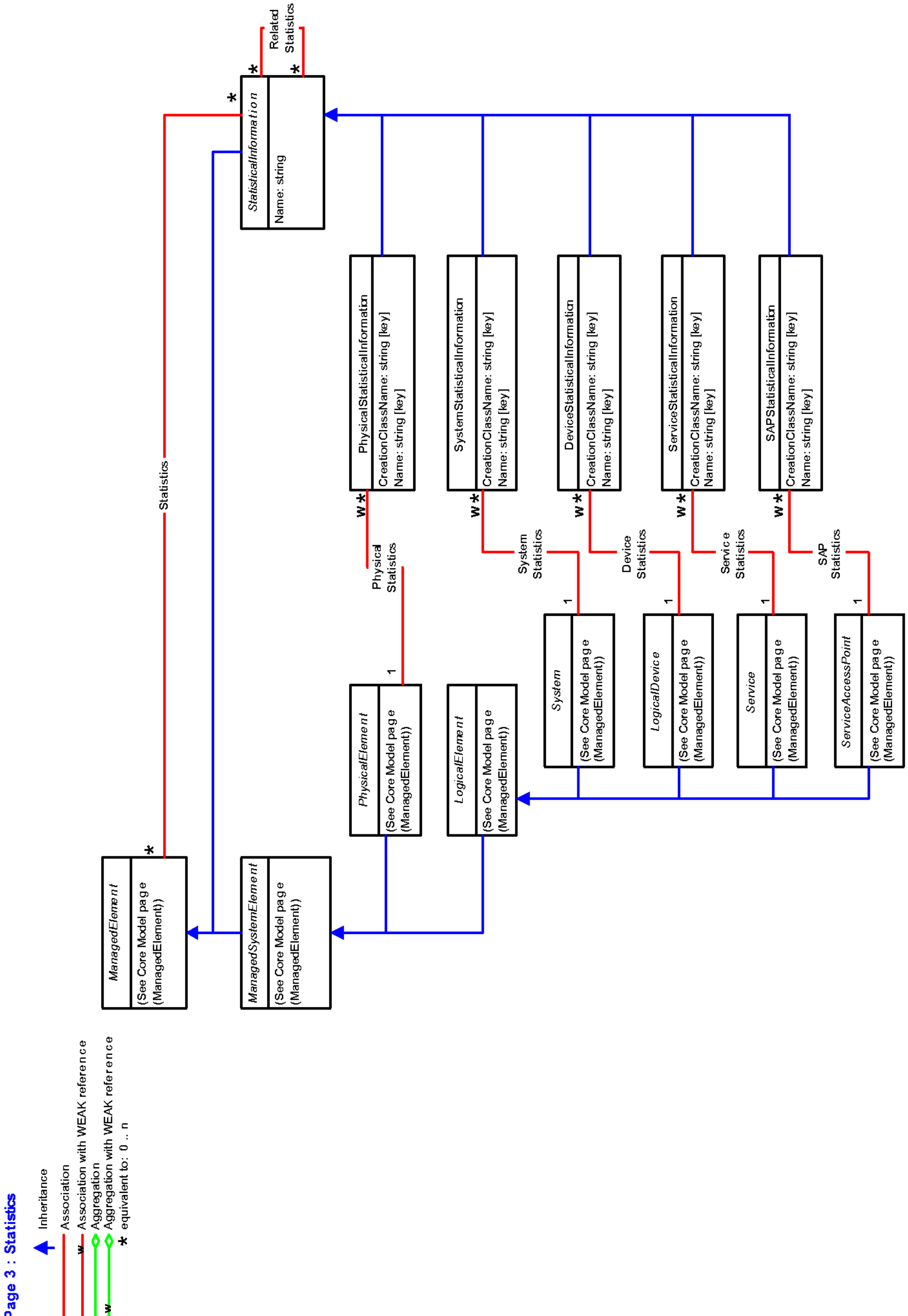

Figure 21: A fragment of the CIM Core Model UML class diagram

Now we can wonder whether an instance of Statistical_Information has to participate exactly once to the association Statistics; observe that this is not explicitly written in the diagram. Let $\mathcal{K}_{cm}$ be the $\mathcal{ALCQI}$ knowledge base corresponding to the UML class diagram of the CIM Core Model. What we want to know can be checked by asking for the satisfiability of the concept $(\geq 2\, r_1^- .\mathsf{Statistics}) \sqcup \neg\exists r_1^- .\mathsf{Statistics}$ with respect to $\mathcal{K}_{cm}$, where we are assuming that class Statistical_Information participates to association Statistics via role r_1.

The answer to this inference query is "No". Let us explain why. The covering and disjointness constraints impose that each tuple of Statistics belongs to exactly one of its sub-classes and that each element of Statistical_Information belongs to exactly one of its children. Hence, if an instance of Statistical_Information participates twice to association Statistics, since it belongs to exactly one of the sub-classes of Statistical_Information, then the maximal multiplicity related to it is violated. On the other hand, if an instance does not participate at all in the association Statistics, then, by the same reason, the minimal multiplicity is violated.

As a result we can refine the multiplicity of the participation of instances of the class Statistical_Information to the association Statistics and state that such a multiplicity is 1..1, instead of just 0..*[21].

Observe that there may be several reasons for the designers of CIM to leave out such a refinement. The point here is not to detect a bug on the CIM Core Model, but to show that automated tools can point out situations that arise due to bugs.

9 Conclusions

In this paper we have shown that reasoning on UML class diagrams can be quite a complex task. Indeed we have proved that it is EXPTIME-complete, without considering arbitrary OCL constraints (which would lead to undecidability). This result suggests that it is highly desirable to provide automated reasoning support for detecting relevant properties of the diagram. With respect to this, we have shown that the DL $\mathcal{ALCQI}$, implemented in current state-of-the-art DL-based systems, is already equipped with the capabilities necessary to reason on UML class diagrams. The experimentation we did, while certainly limited and not providing a definitive answer, indicate that current state-of-the-art DL-based systems are ready to serve as a core reasoning engine in advanced CASE tools.

Various issues remain to be addressed. First of all, the reasoning tasks we have analyzed in this paper do not include reasoning on keys and identification constraints. While these are not among the basic reasoning services that should be supported, they may be of interest for dealing with class diagrams in which keys are introduced for classes, together with complex class hierarchies. Such forms of reasoning can be directly supported in $\mathcal{DLR}_{ifd}$ [15]. Instead, DL-based systems need to be substantially enhanced to fully implement $\mathcal{DLR}_{ifd}$ (in particular sophisticated abilities to deal with individuals need to be added). Another aspect that deserves further treatment are multiplicities on associations of arbitrary arity, which UML defines to be look-across [47, 27]. Reasoning on look-across multiplicity constraints is largely unexplored. While multiplicities on n-ary associations appear rarely in UML class diagrams, more work needs to be done to understand their interaction with other constructs in order to take them into account during reasoning. It is also of interest to characterize interesting fragments of OCL constraints that do not lead to undecidability. Although we did not treat it in this paper, $\mathcal{DLR}_{ifd}$ (and even $\mathcal{ALCQI}$) can express interesting forms of OCL constraints, such as rich typing restrictions on associations and refinement of properties along class hierarchies.

Finally, it is worth noting that the results presented here hold also for other conceptual modeling formalisms typically used in software engineering and databases. In particular, the EXPTIME-completeness result applies to the Entity-Relationship model enhanced with ISA on entities and relationships [4].

References

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge Uni-

[21] Note that, as indicated in the CIM Core Model, all classes in the same hierarchy participate to associations in the same hierarchy with the same role.

versity Press, 2002. To appear.

[2] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. In *Proc. of the 3rd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'92)*, pages 270–281. Morgan Kaufmann, Los Altos, 1992.

[3] Leo Bachmair and Haral Ganzinger. Resolution theorem proving. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

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

[5] Daniela Berardi. Using DLs to reason on UML class diagrams. In *Proc. of the KI'2002 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-63/`, 2002.

[6] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams using description logic based systems. In *Proc. of the KI'2001 Workshop on Applications of Description Logics*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-44/`, 2001.

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

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

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

[10] Alexander Borgida, Maurizio Lenzerini, and Riccardo Rosati. Description logics for data bases. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 16. Cambridge University Press, 2002. To appear.

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

[12] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning on UML class diagrams in description logics. In *Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001)*, 2001.

[13] 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)*, volume 1013 of *Lecture Notes in Computer Science*, pages 229–246. Springer, 1995.

[14] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158, 1998.

[15] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, chapter 23, pages 1581–1634. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[17] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2–13, 1998.

[18] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998.

[19] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

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

[21] Tony Clark and Andy S. Evans. Foundations of the Unified Modeling Language. In David Duke and Andy Evans, editors, *Proc. of the 2nd Northern Formal Methods Workshop*. Springer, 1997.

[22] Francesco M. Donini. Complexity of reasoning. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 3. Cambridge University Press, 2002. To appear.

[23] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

[24] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. The UML as a formal modeling notation. In Haim Kilov, Bernhard Rumpe, and Ian Simmonds, editors, *Proc. of the OOPSLA'97 Workshop on Object-oriented Behavioral Semantics*, pages 75–81. Technische Universität München, TUM-I9737, 1997.

[25] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. Meta-modelling semantics of UML. In H. Kilov, editor, *Behavioural Specifications for Businesses and Systems*, chapter 2. Kluwer Academic Publisher, 1999.

[26] Andy S. Evans. Reasoning with UML class diagrams. In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT'98)*. IEEE Computer Society Press, 1998.

[27] 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.

[28] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[29] Martin Fowler and Kendall Scott. *UML Distilled – Applying the Standard Object Modeling Laguage*. Addison Wesley Publ. Co., Reading, Massachussetts, 1997.

[30] Enrico Franconi and Gary Ng. The i.com tool for intelligent conceptual modeling. In *Proc. of the 7th Int. Workshop on Knowledge Representation meets Databases (KRDB 2000)*, pages 45–53. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-29/`, 2000.

[31] Volker Haarslev and Ralf Möller. Expressive ABox reasoning with number restrictions, role hierarchies, and transitively closed roles. In *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 273–284, 2000.

[32] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 161–168, 2001.

[33] Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

[34] Volker Haarslev and Ralf Möller. Description logics systems. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 8. Cambridge University Press, 2002. To appear.

[35] Reiner Hähnle. Tableaux and related methods. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 100–178. Elsevier Science Publishers (North-Holland), Amsterdam, 2001.

[36] David Harel and Bernhard Rumpe. Modeling languages: Syntax, semantics and all that stuff. Technical Report MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.

[37] Ian Horrocks. The FaCT system. In Harrie de Swart, editor, *Proc. of the 2nd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'98)*, volume 1397 of *Lecture Notes in Artificial Intelligence*, pages 307–312. Springer, 1998.

[38] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[39] Ian Horrocks. Daml+oil: a description logic for the semantic web. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1):4–9, 2002.

[40] Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

[41] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[42] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The Information Manifold. In *Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments*, pages 85–91, 1995.

[43] Deborah L. McGuinness and Jon R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.

[44] Neil V. Murray, editor. *Proc. of the 3rd Int. Conf. on Analytic Tableaux and Related Methods (TABLEAUX'99)*, volume 1617 of *Lecture Notes in Computer Science*. Springer, 1999.

[45] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Addison Wesley Publ. Co., Reading, Massachussetts, 1998.

[46] Ulrike Sattler. *Terminological Knowledge Representation Systems in a Process Engineering Application*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 1998.

[47] 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, 1992.